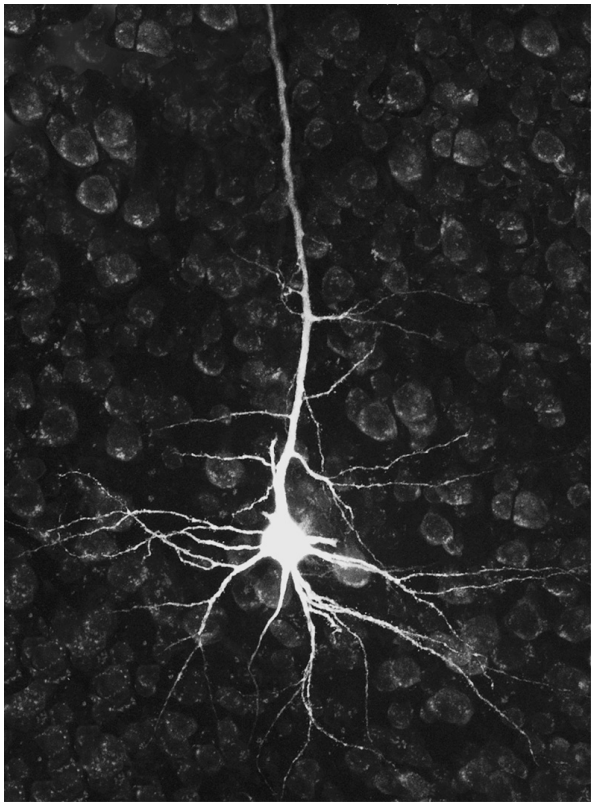
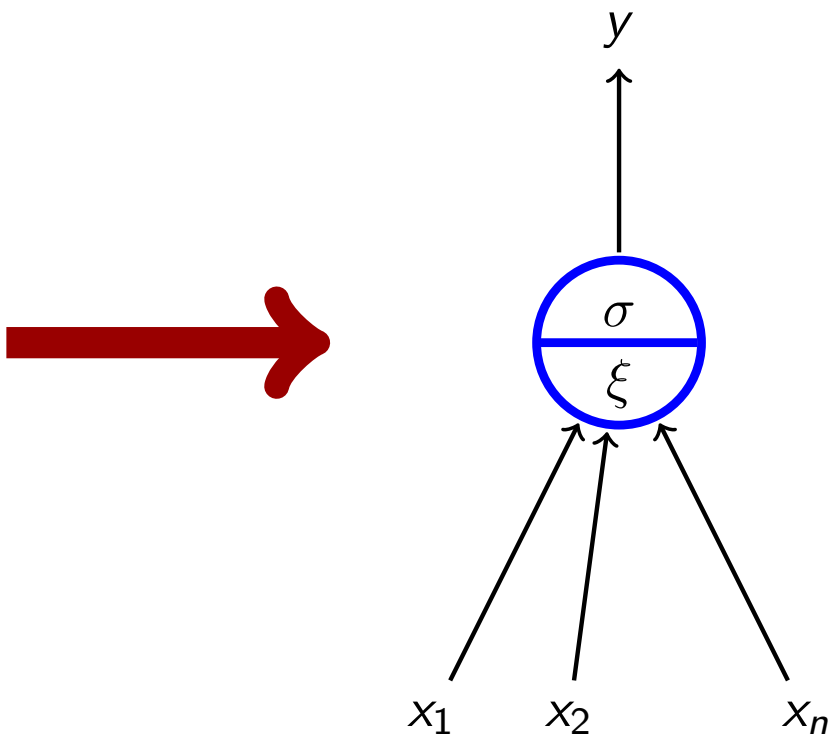
Neural Networks

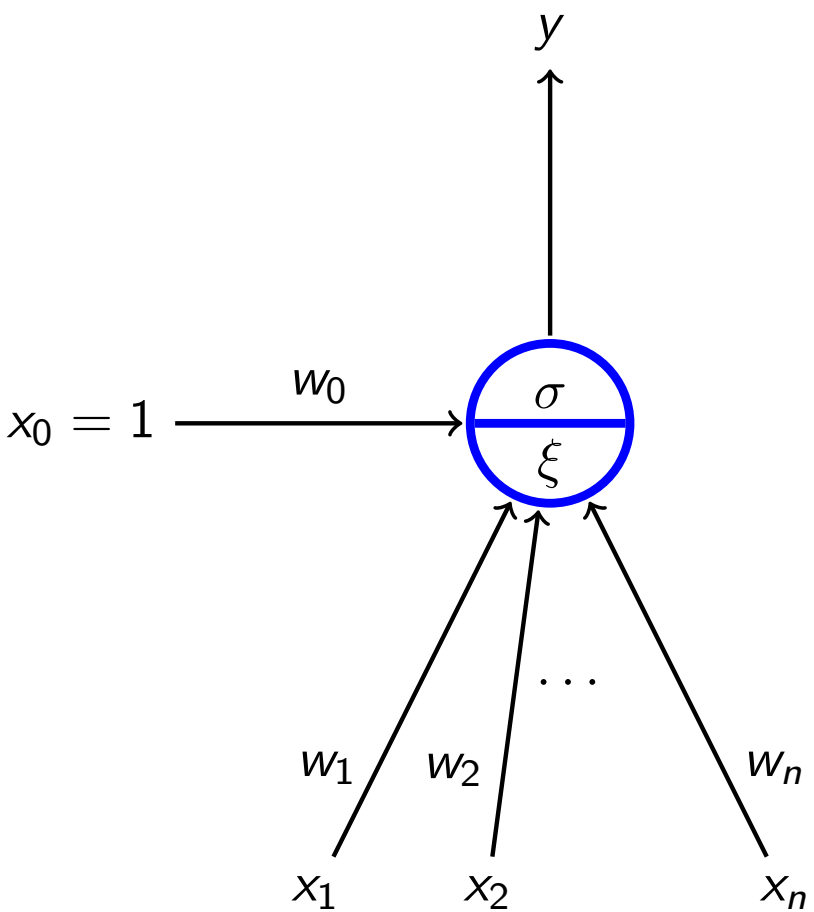
(Primitive) Mathematical Model of Neuron



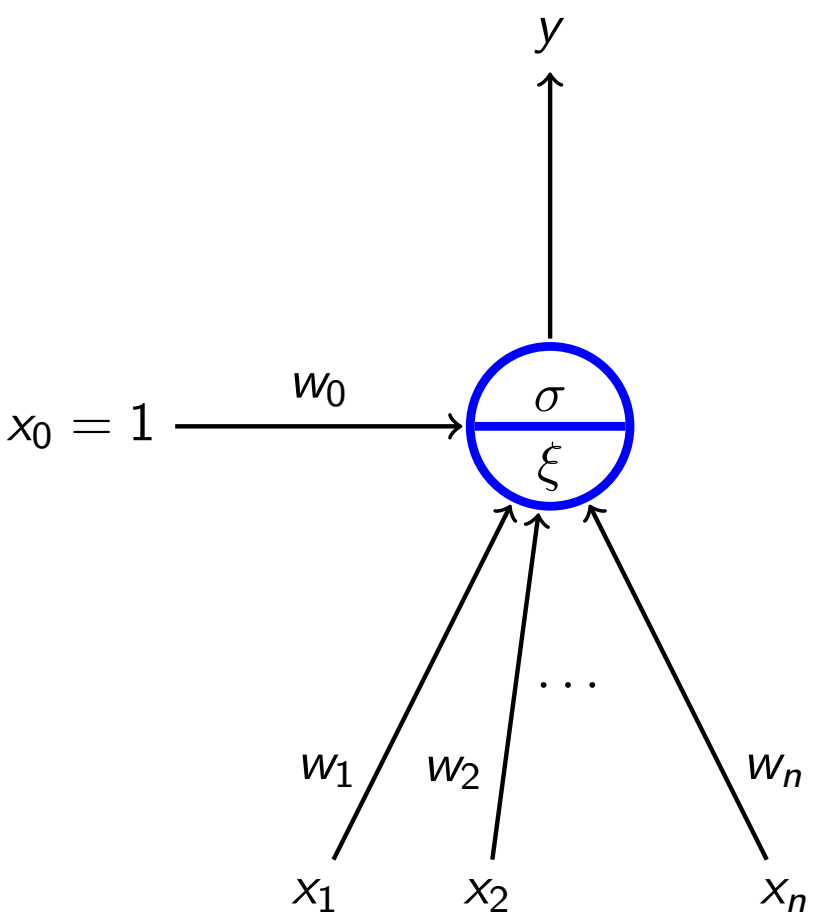


Formal neuron

- $x_1, \ldots, x_n$ real *inputs*

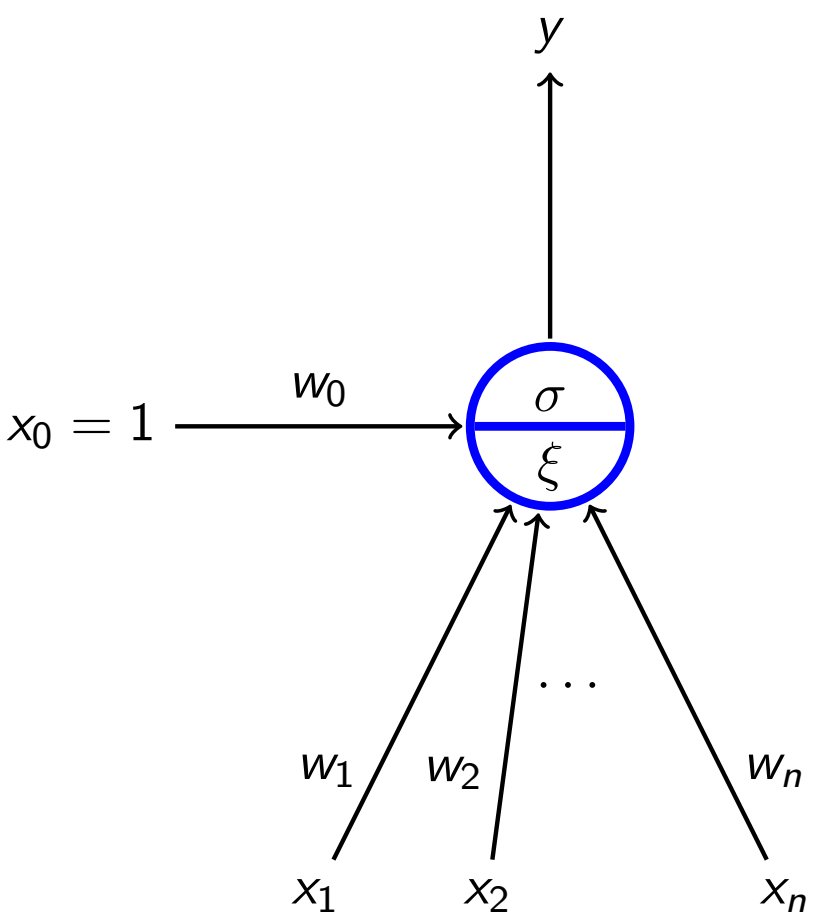


Formal neuron



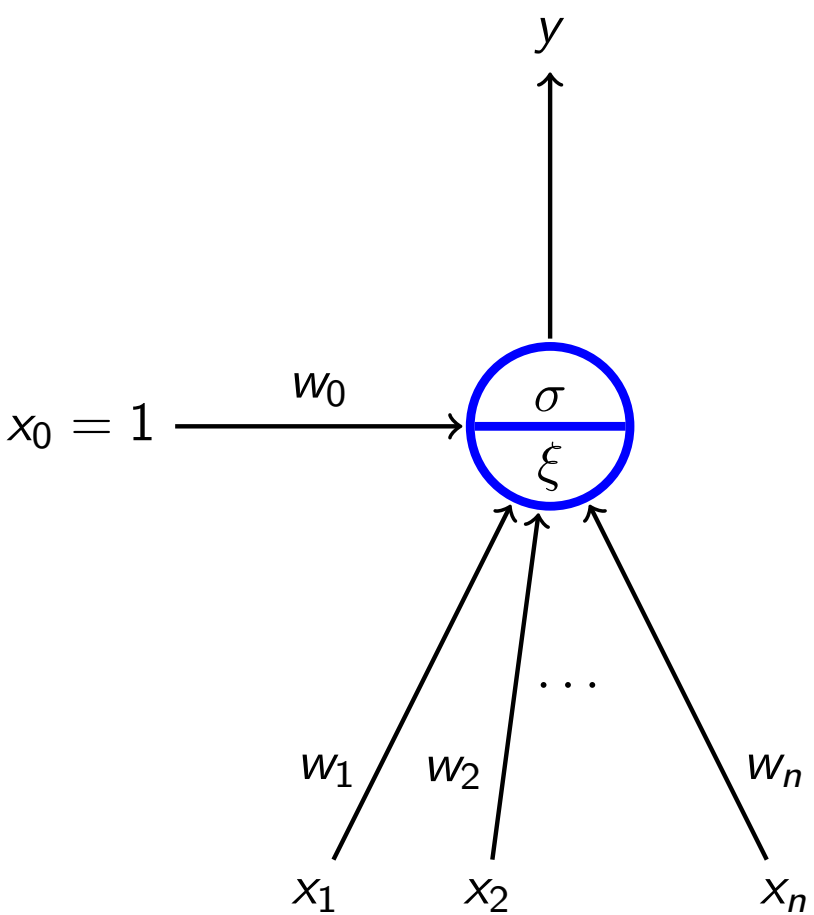
- $x_1, \ldots, x_n$ real *inputs*
- x_0 special input, always 1

Formal neuron



- $x_1, \ldots, x_n$ real *inputs*
- x_0 special input, always 1
- $w_0, w_1, \ldots, w_n$ real *weights*

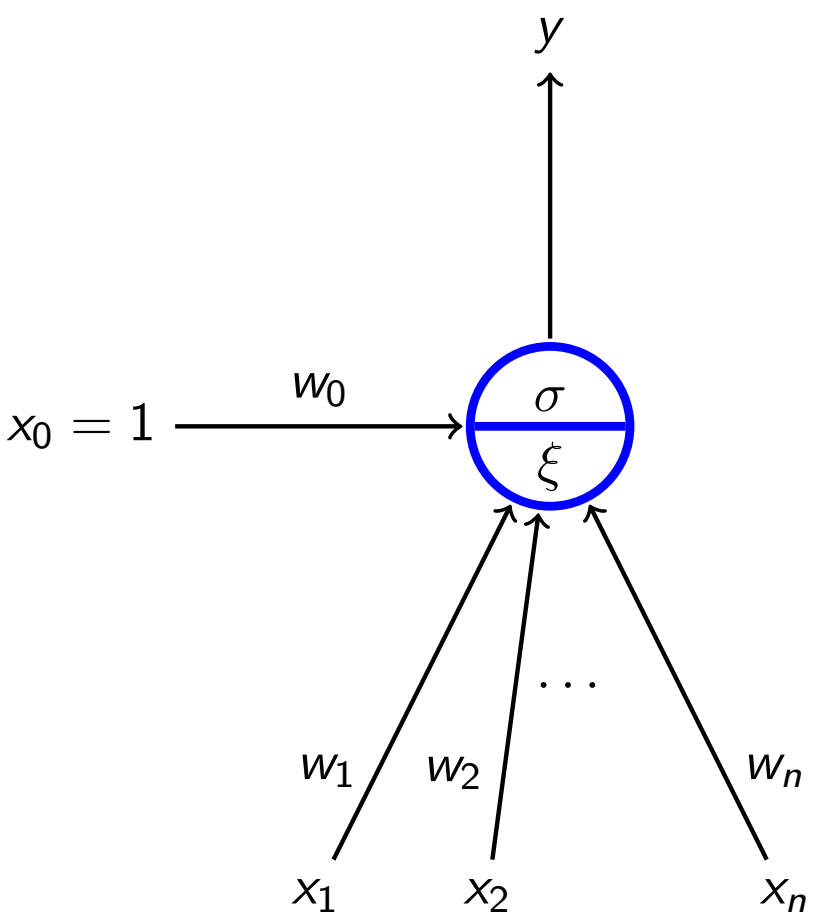
Formal neuron



- $x_1, \ldots, x_n$ real *inputs*
- x_0 special input, always 1
- $w_0, w_1, \ldots, w_n$ real *weights*
- $\xi = w_0 + \sum_{i=1}^{n} w_i x_i$ *inner potential*;

  In general, other potentials are considered (e.g. Gaussian), more on this in PV021.

Formal neuron



- $x_1, \ldots, x_n$ real *inputs*
- x_0 special input, always 1
- $w_0, w_1, \ldots, w_n$ real *weights*
- $\xi = w_0 + \sum_{i=1}^{n} w_i x_i$ *inner potential*;

  In general, other potentials are considered (e.g. Gaussian), more on this in PV021.
- y *output* defined by $y = \sigma(\xi)$ where σ is an *activation function*.

  We consider several activation functions.

  e.g., *linear threshold function*

$$\sigma(\xi) = sgn(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

Formal Neuron vs Linear Models

- If σ is a linear threshold function

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

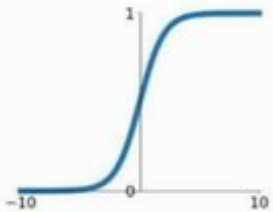
  We obtained a linear classifier.
- If σ is identity, i.e., $\sigma(\xi) = \xi$, we obtain a linear (affine) function.
- If $\sigma(\xi) = 1/(1 + e^{-\xi})$ we obtain the logistic regression.

Also, other activation functions are used in neural networks!

Activation Functions

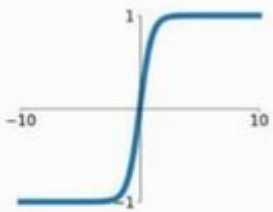
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



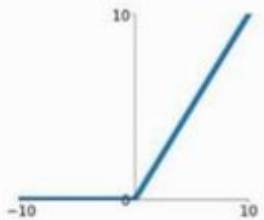
tanh

$$\tanh(x)$$



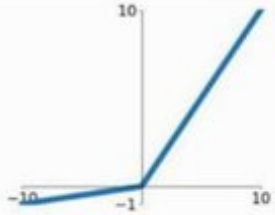
ReLU

$$\max(0, x)$$



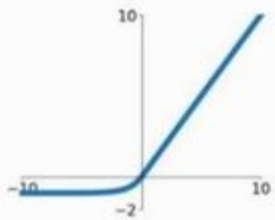
Leaky ReLU

$$\max(0.1x, x)$$

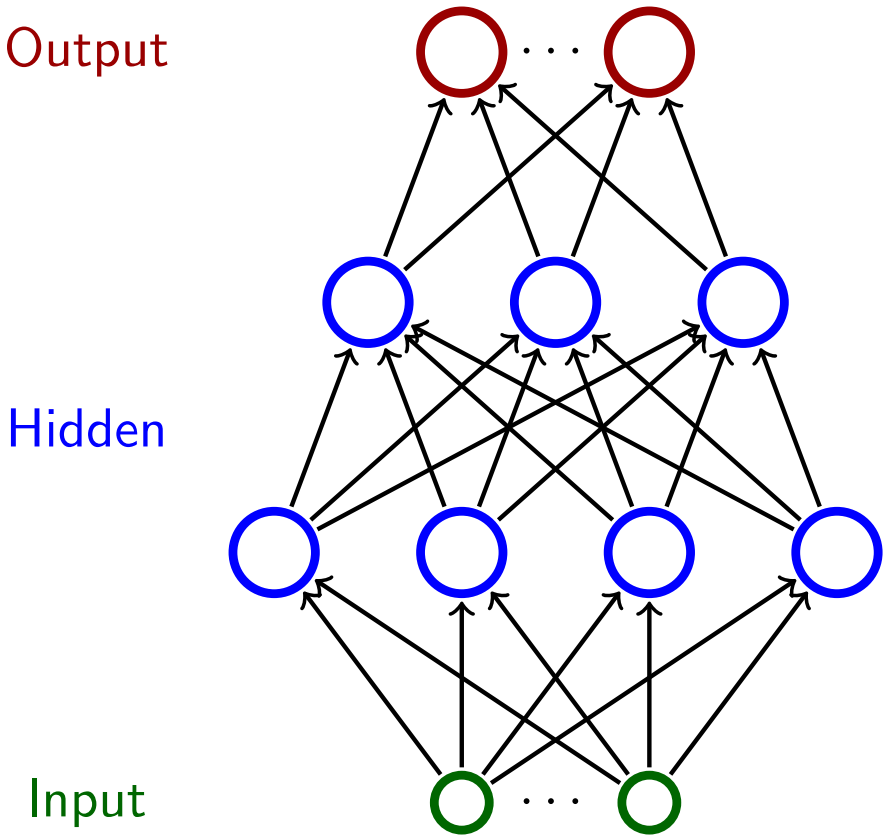


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Multilayer Perceptron (MLP)

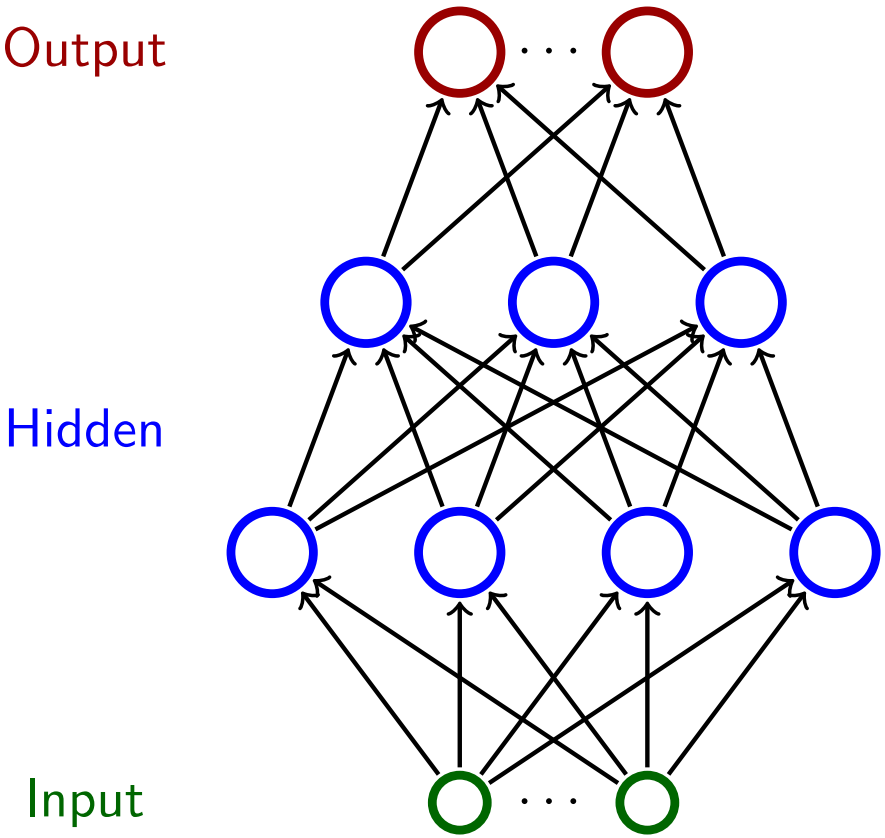


- Neurons are organized in *layers*

  (input layer, output layer, possibly several hidden layers)
- Layers are numbered from 0;

  The input is 0-th
- Neurons in the ℓ-th layer are connected with all neurons in the $\ell + 1$-th layer

Multilayer Perceptron (MLP)



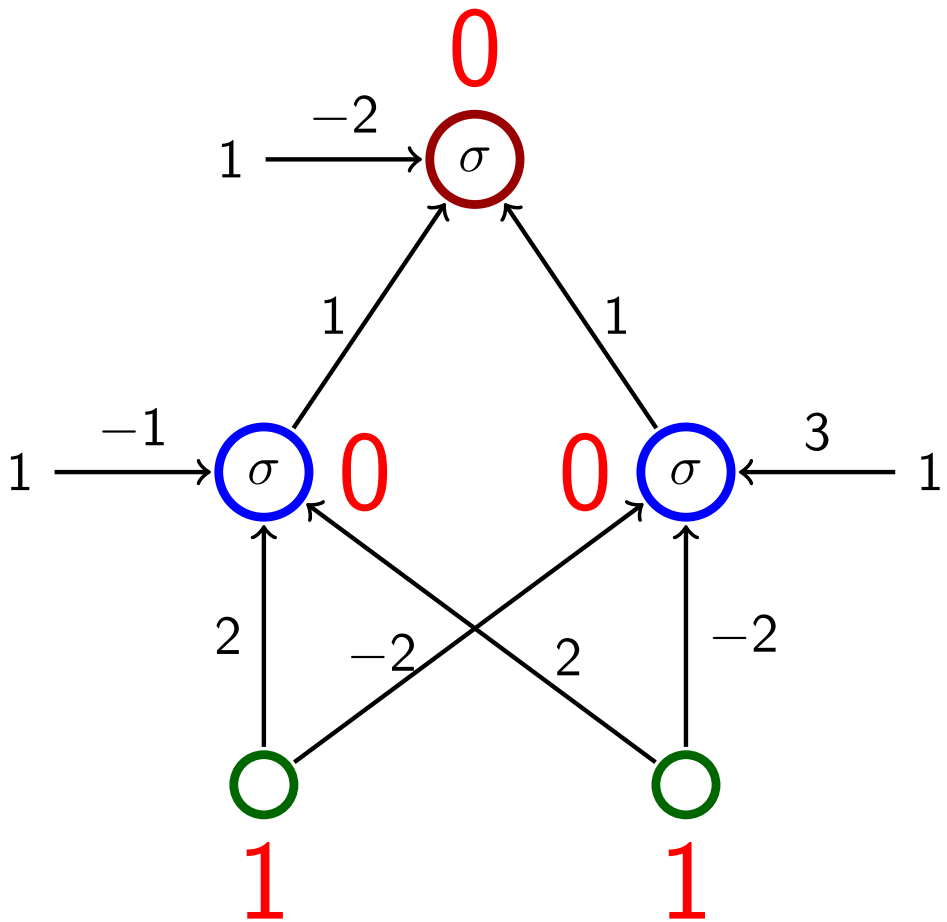
- Neurons are organized in *layers*

  (input layer, output layer, possibly several hidden layers)
- Layers are numbered from 0;

  The input is 0-th
- Neurons in the ℓ-th layer are connected with all neurons in the $\ell + 1$-th layer

Intuition: The network computes a function: Assign input values to the input neurons and 0 to the rest. Proceed upwards through the layers, one layer per step. In the ℓ-th step consider output values of neurons in $\ell - 1$-th layer as inputs to neurons of the ℓ-th layer. Compute output values of neurons in the ℓ-th layer.

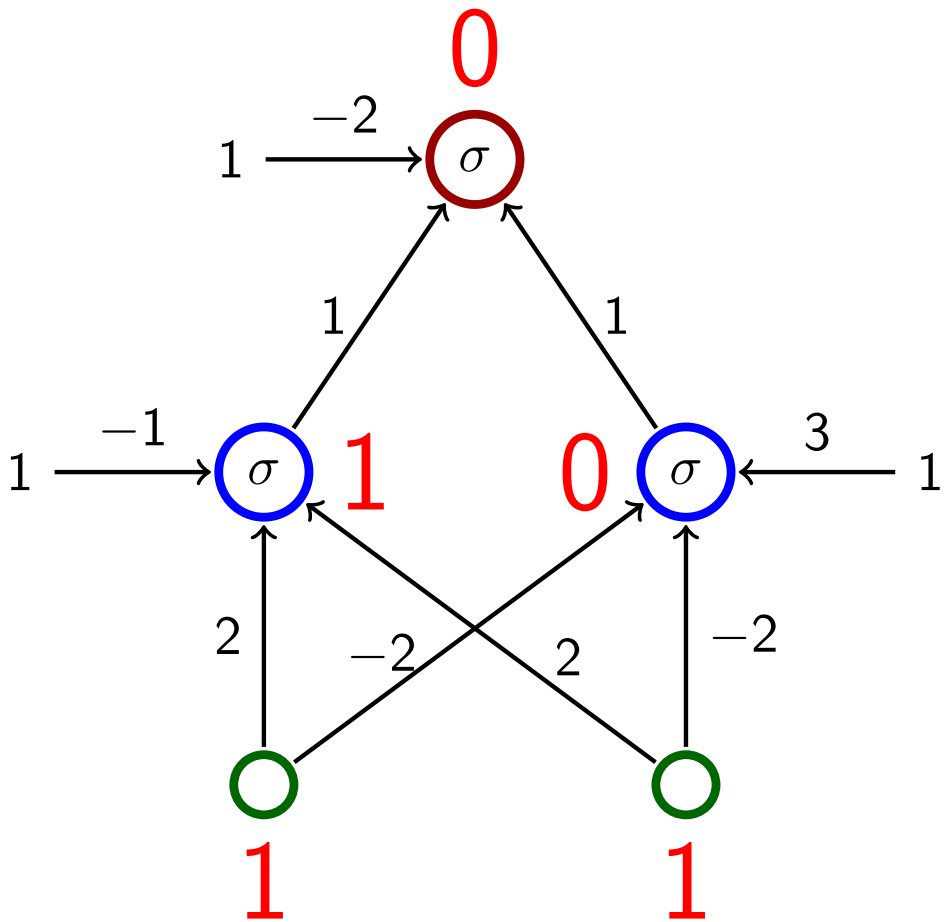
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

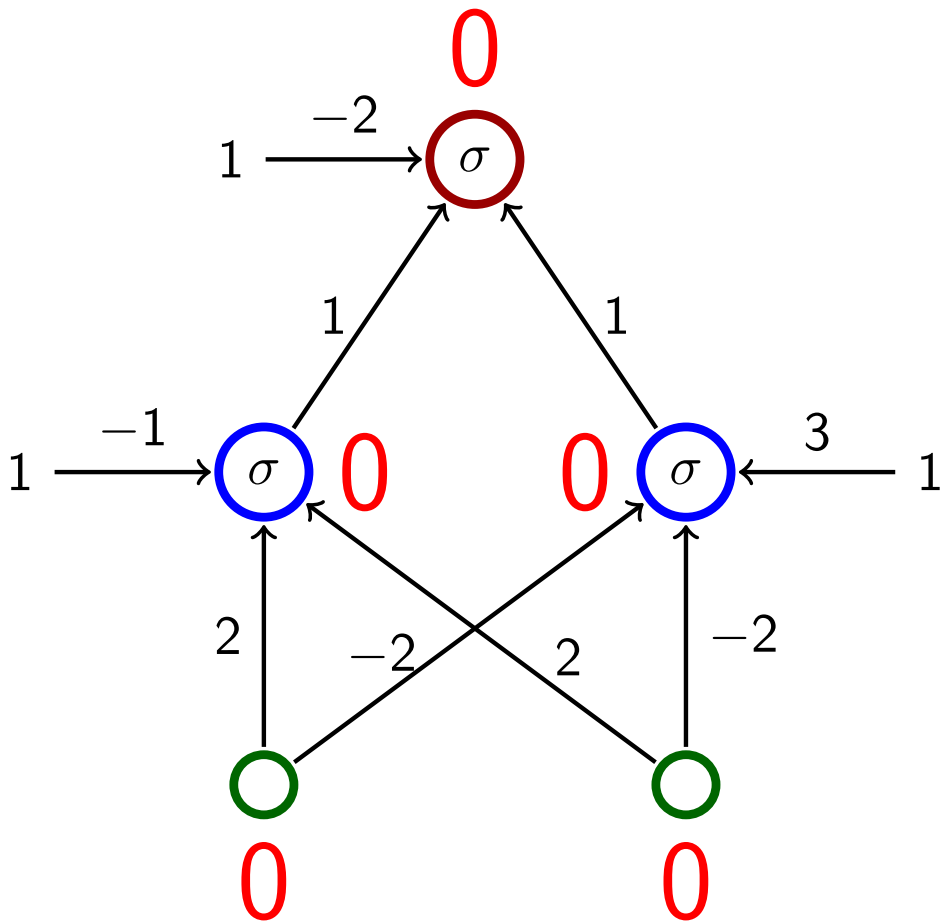
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

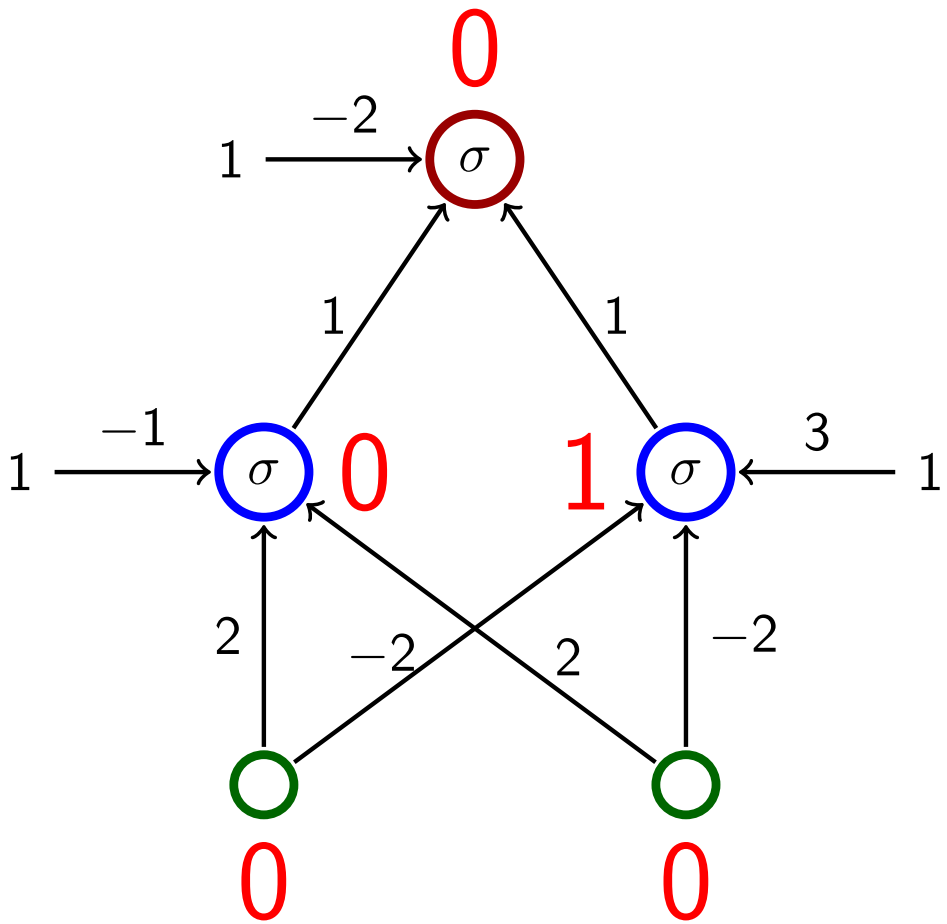
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

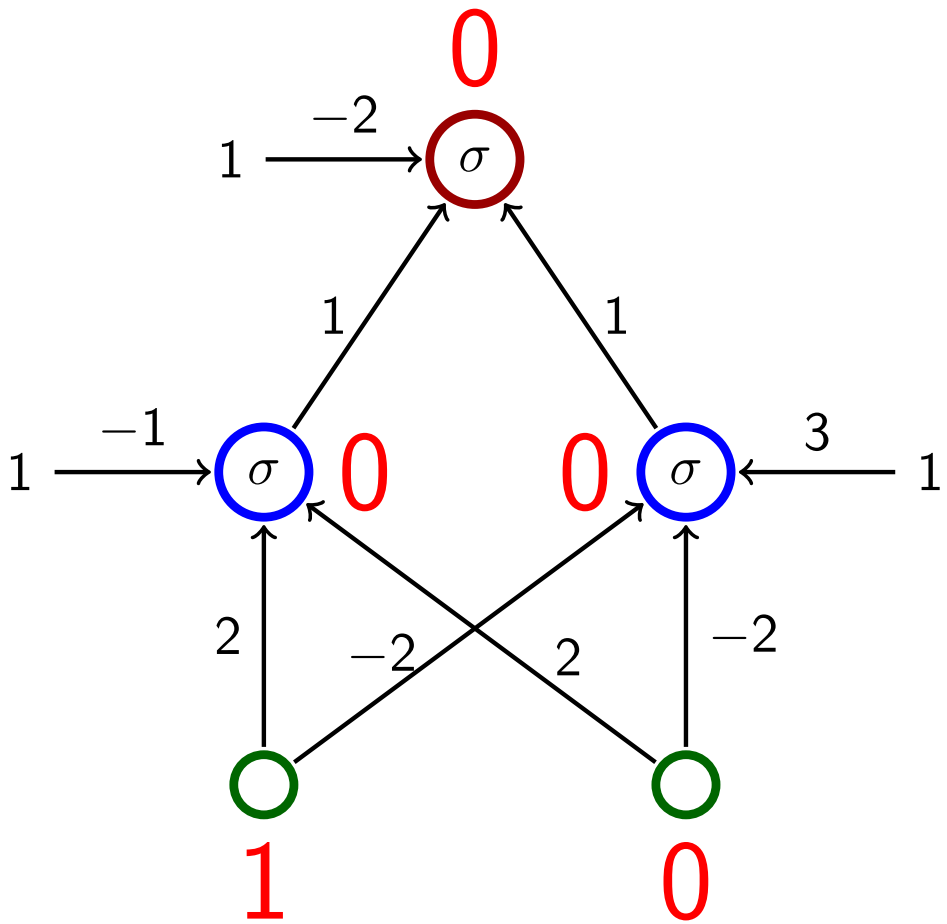
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

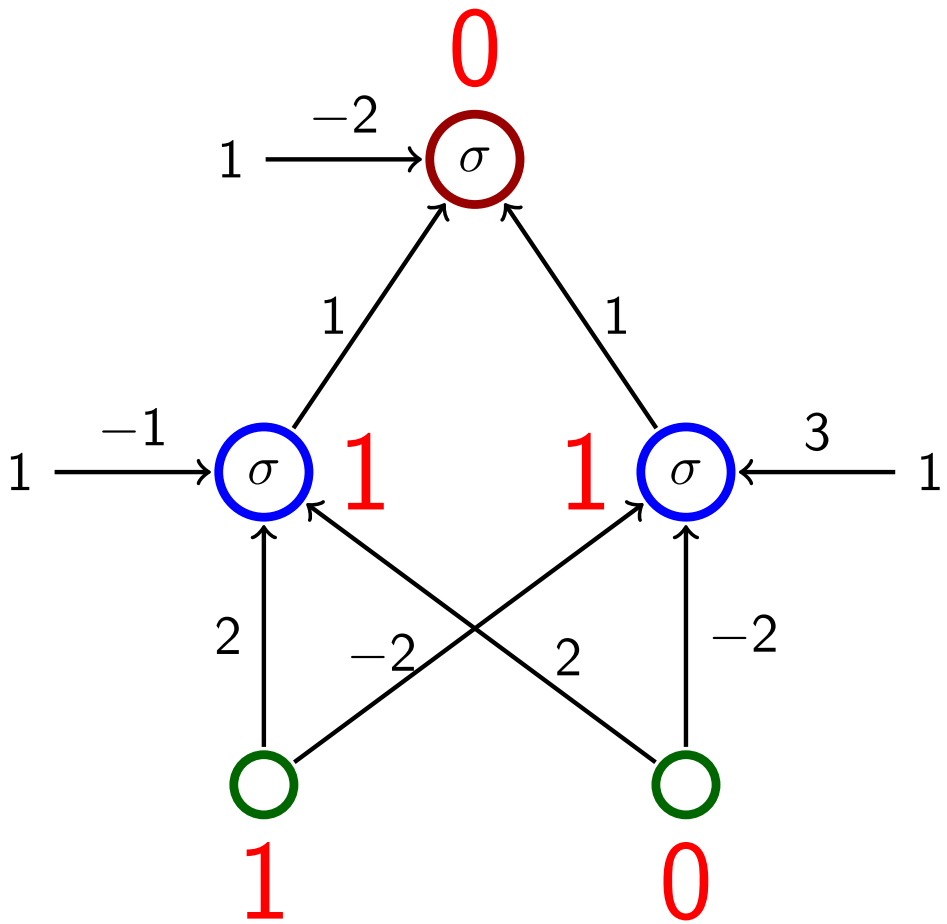
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

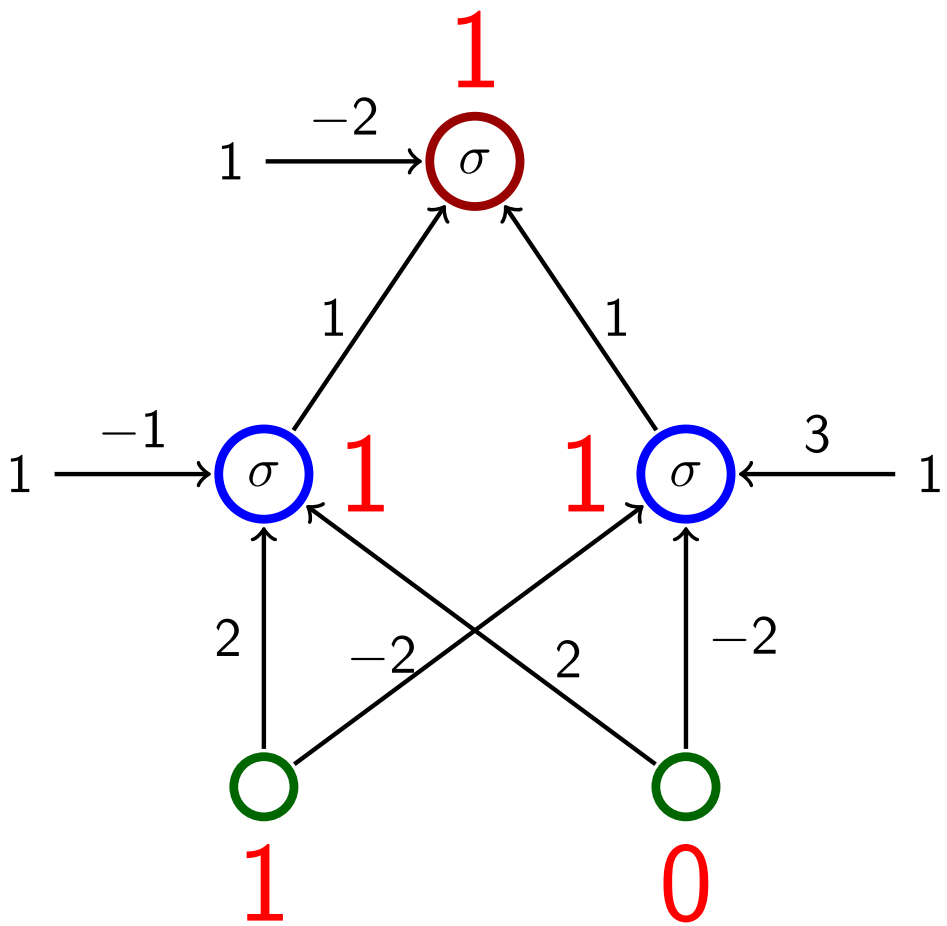
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

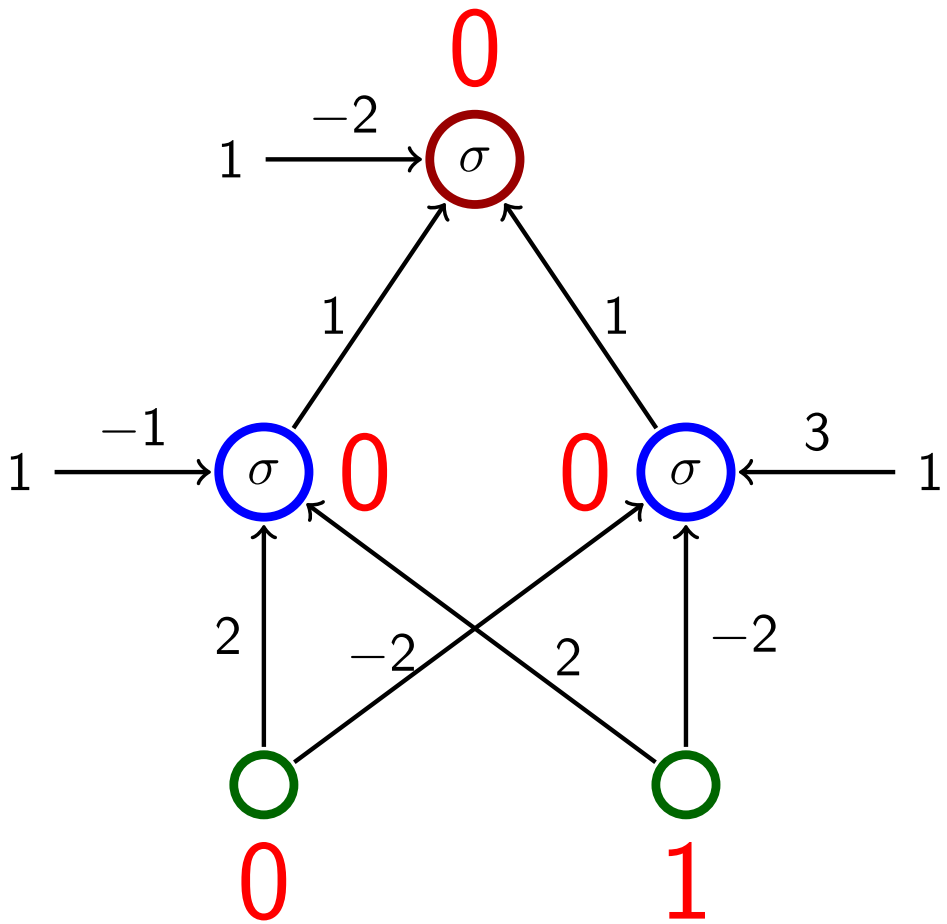
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

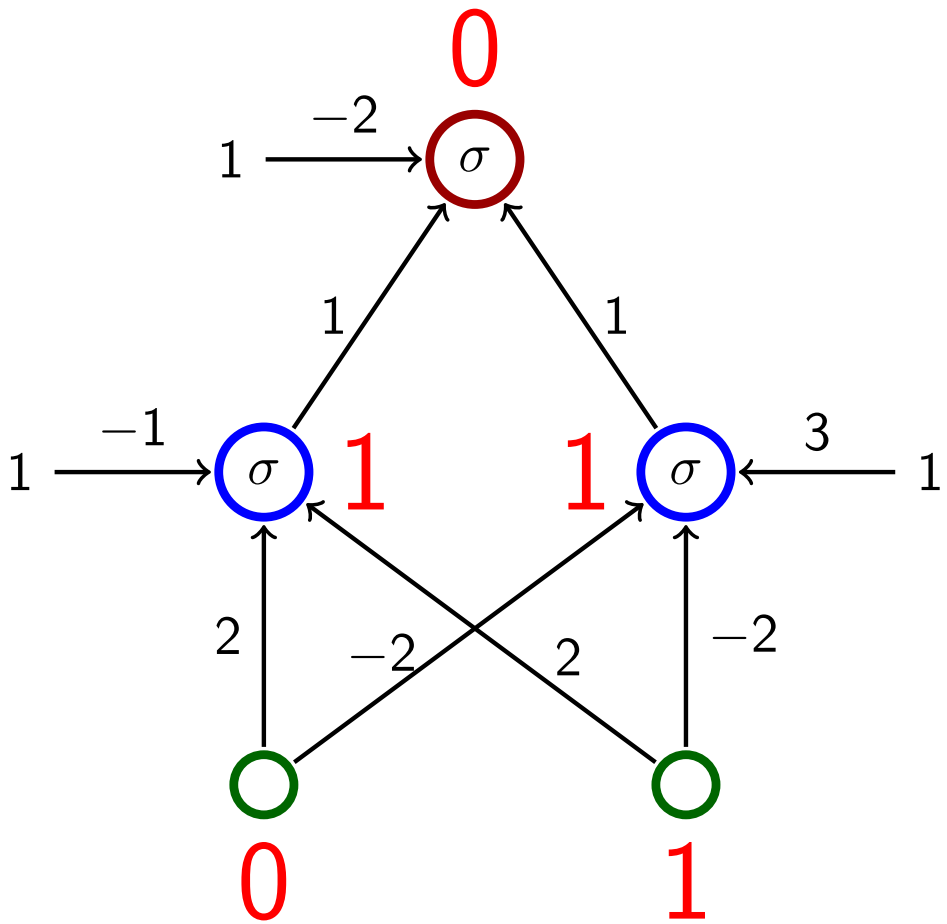
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

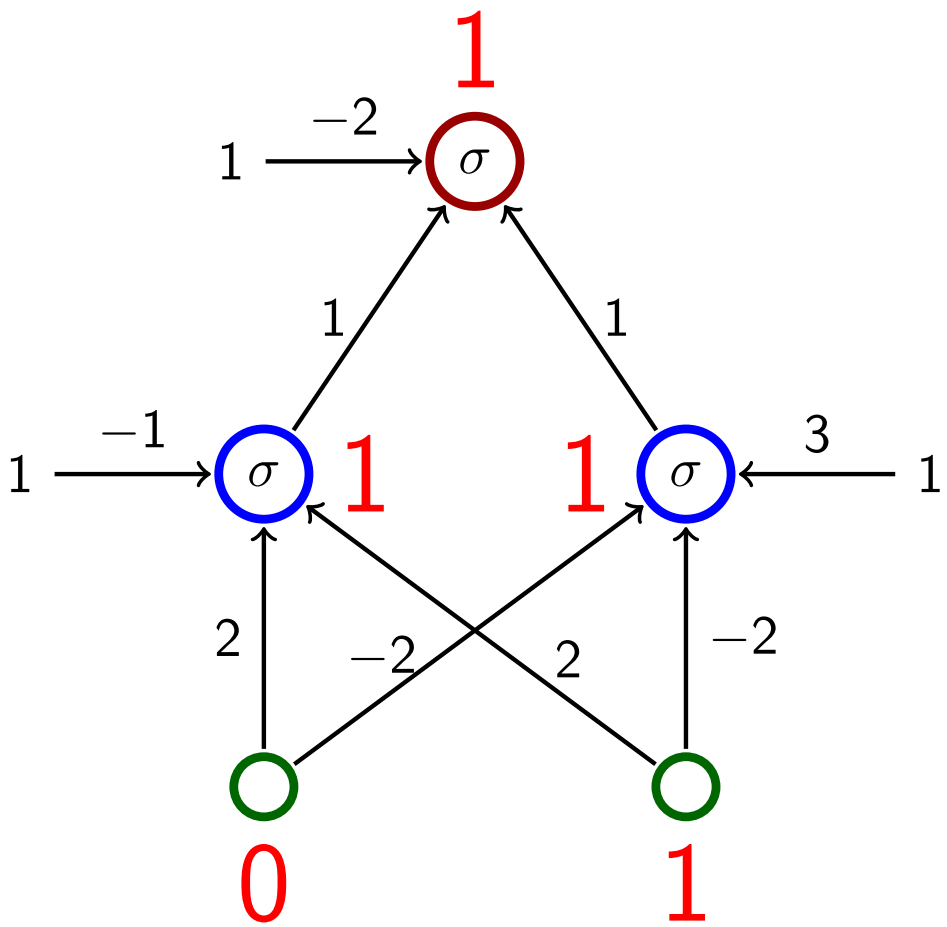
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

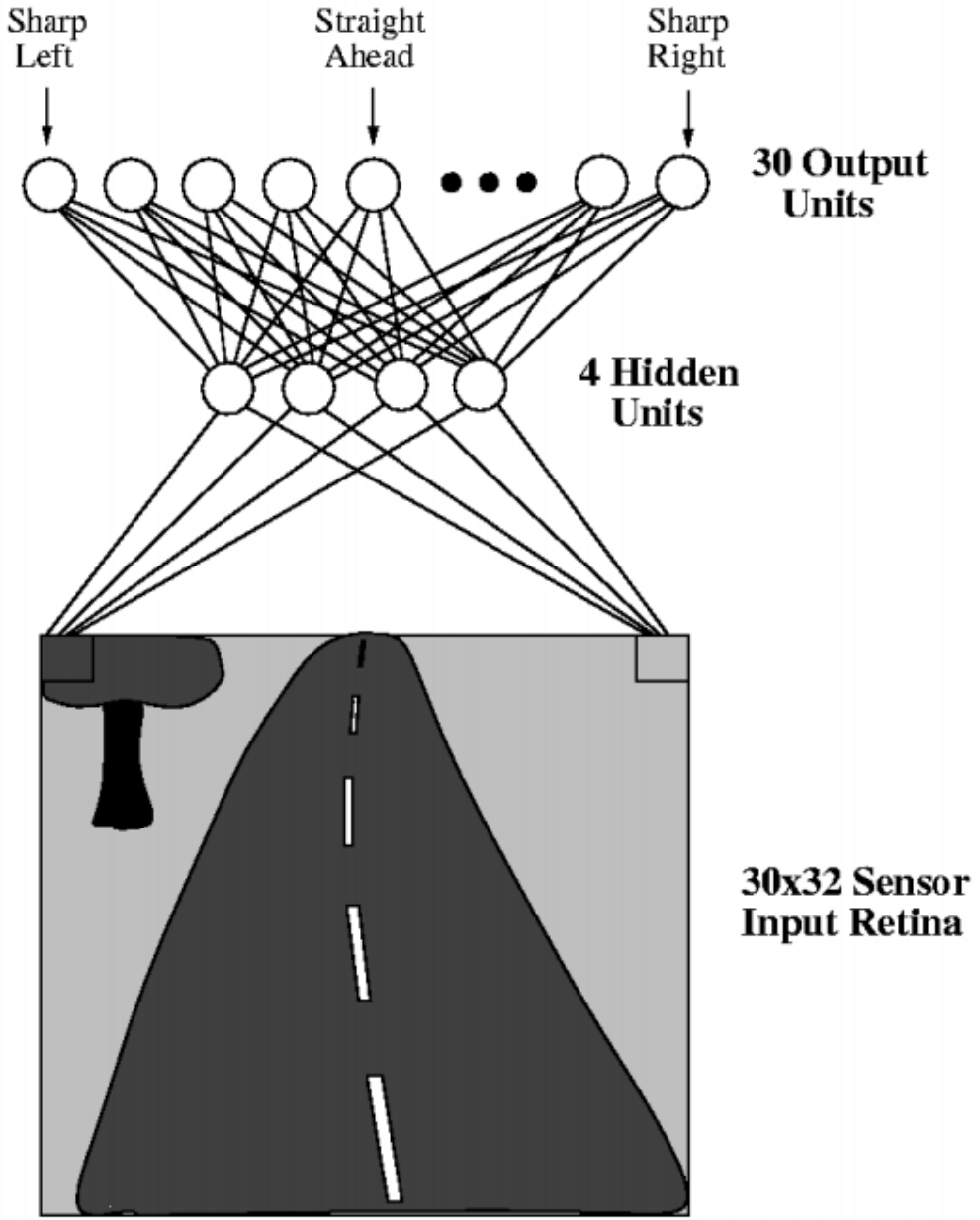
Example



- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

Classical Example – ALVINN

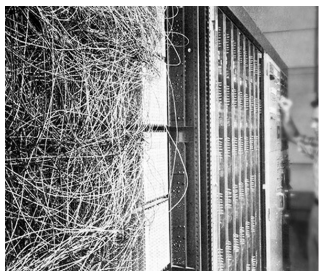


Source: `http://jmvidal.cse.sc.edu/talks/ann/alvin.html`

- One of the first autonomous car driving systems (in the 90s)
- ALVINN drives a car
- The net has $30 \times 32 = 960$ input neurons (the input space is $\mathbb{R}^{960}$).
- The value of each input captures the shade of gray of the corresponding pixel.
- Output neurons indicate where to turn (to the center of gravity).

A Bit of History

- Perceptron (Rosenblatt et al., 1957)

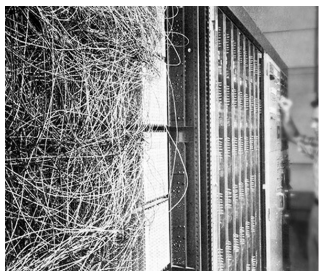


- Single layer (i.e., no hidden layers), the activation function *linear threshold*

  (i.e., a bit more general linear classifier)
- Perceptron learning algorithm
- Used to recognize digits

A Bit of History

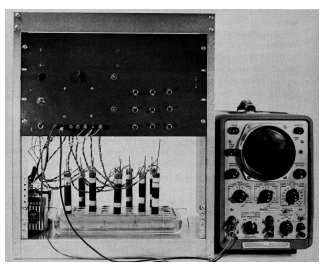
- Perceptron (Rosenblatt et al., 1957)



  - Single layer (i.e., no hidden layers), the activation function *linear threshold*

    (i.e., a bit more general linear classifier)
  - Perceptron learning algorithm
  - Used to recognize digits

- Adaline (Widrow & Hof, 1960)



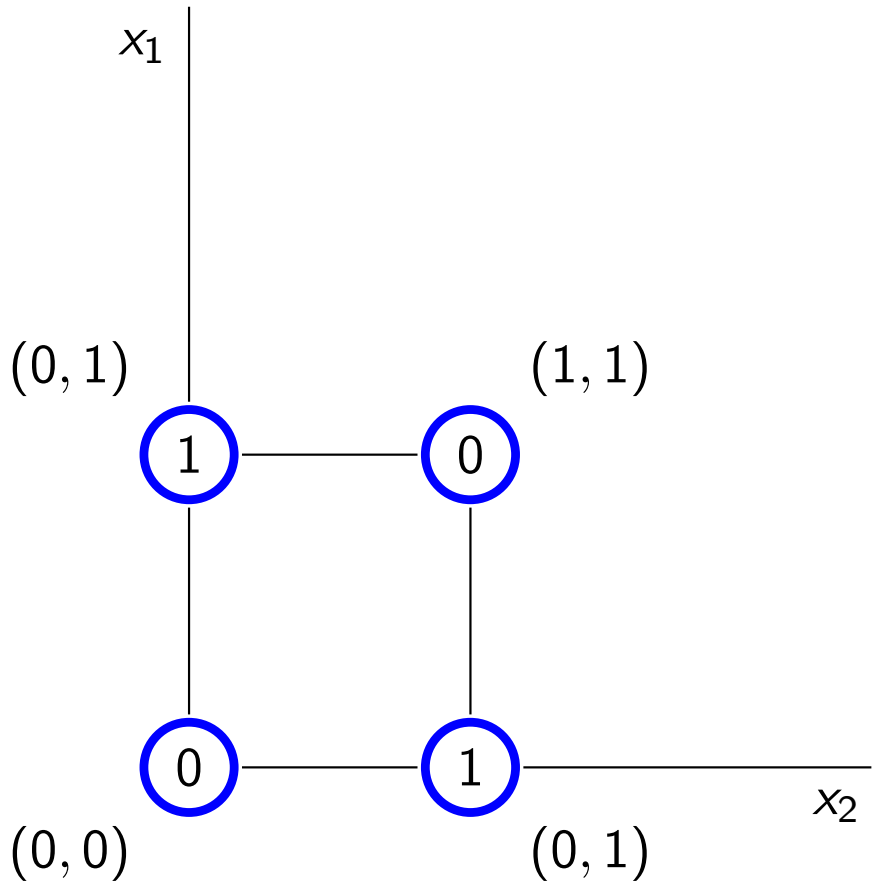
  - Single layer, the activation function *identity*

    (i.e., a bit more linear function)
  - Online version of the gradient descent
  - Used a new circuitry element called *memristor* which was able to "remember" history of current in form of resistance

In both cases, the expressive power is somewhat limited- it can only express linear decision boundaries and linear (affine) functions.

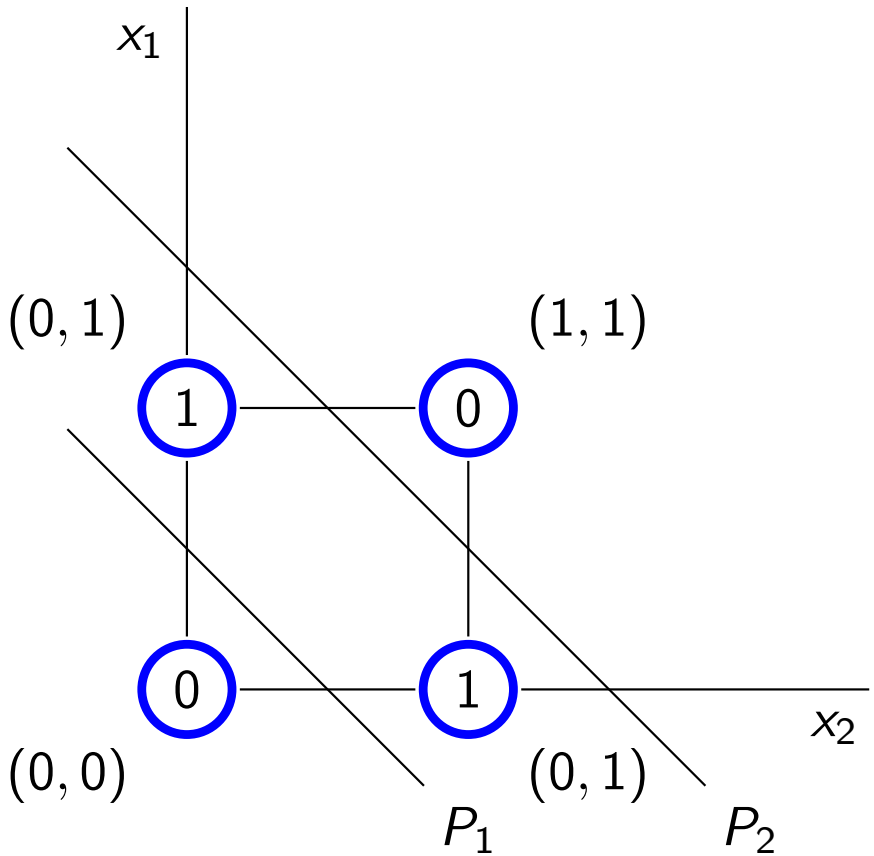
A Bit of History

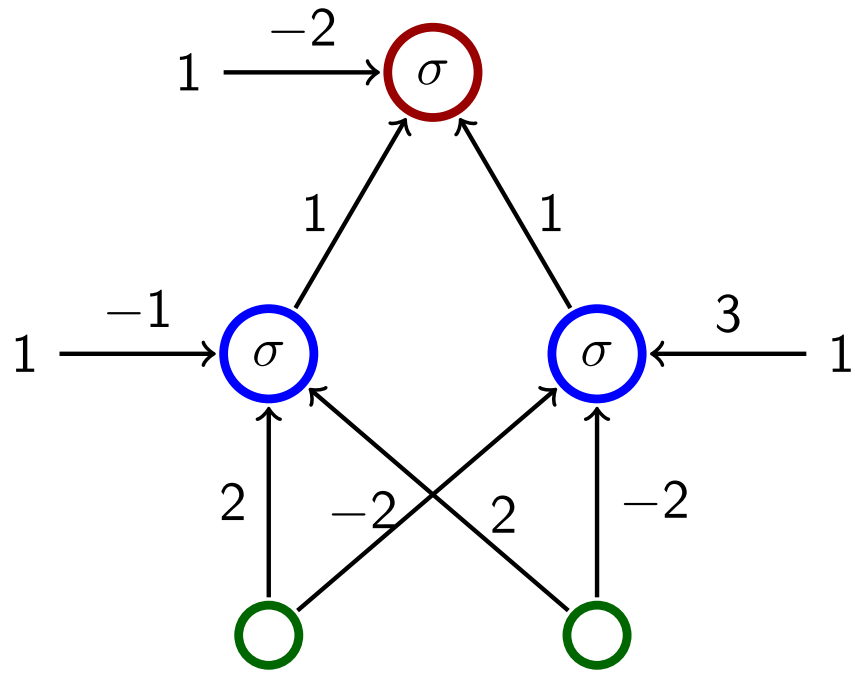
One of the famous (counter)-examples: XOR



No perceptron can distinguish between ones and zeros.

XOR vs Multilayer Perceptron





(Here, σ is a linear threshold function.)

$$P_1 : -1 + 2x_1 + 2x_2 = 0 \qquad P_2 : 3 - 2x_1 - 2x_2 = 0$$

The output neuron performs an intersection of half-spaces.

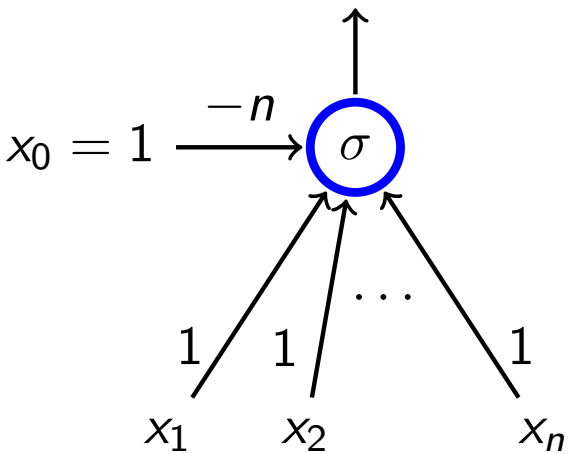
Boolean functions

Activation function: *unit step function* $\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$

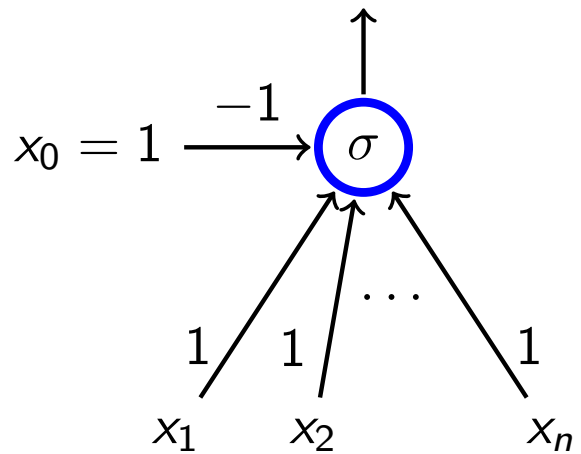
Boolean functions

Activation function: *unit step function* $\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$

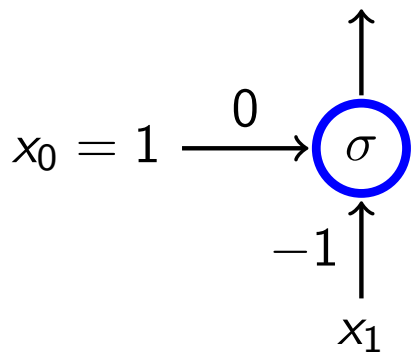
$y = AND(x_1, \ldots, x_n)$



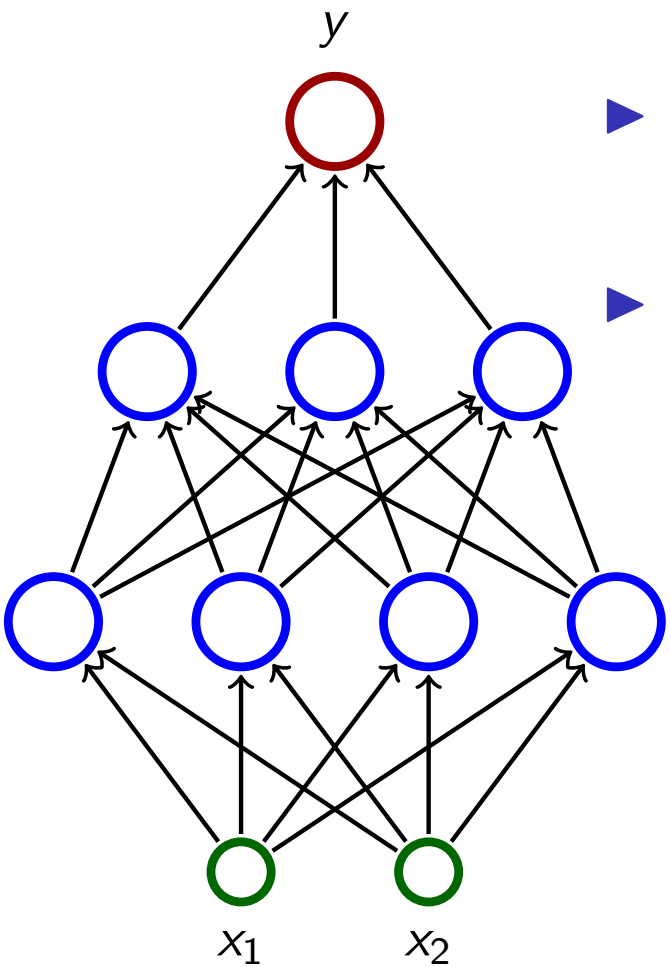
$y = OR(x_1, \ldots, x_n)$



$y = NOT(x_1)$

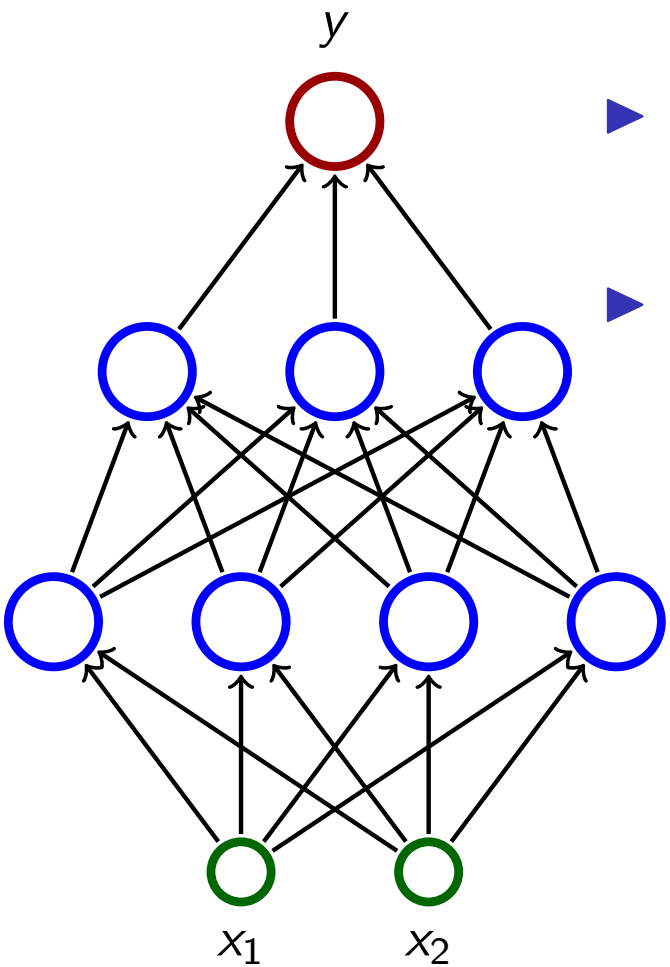


Non-linear separation



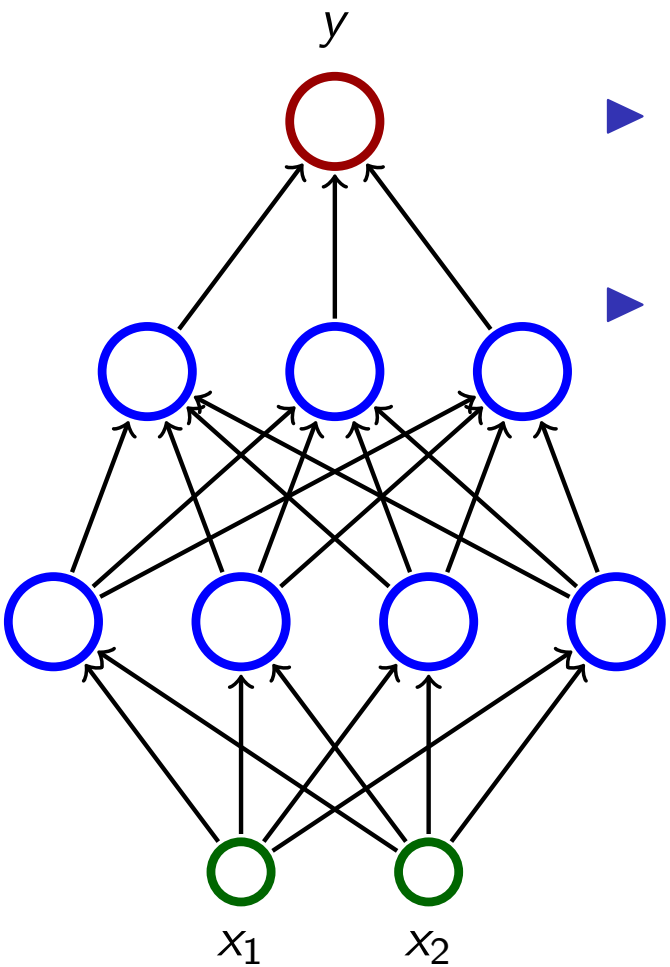
- Consider a three-layer network; each neuron has the unit step activation function.
- The network divides the input space in two subspaces according to the output (0 or 1).

Non-linear separation



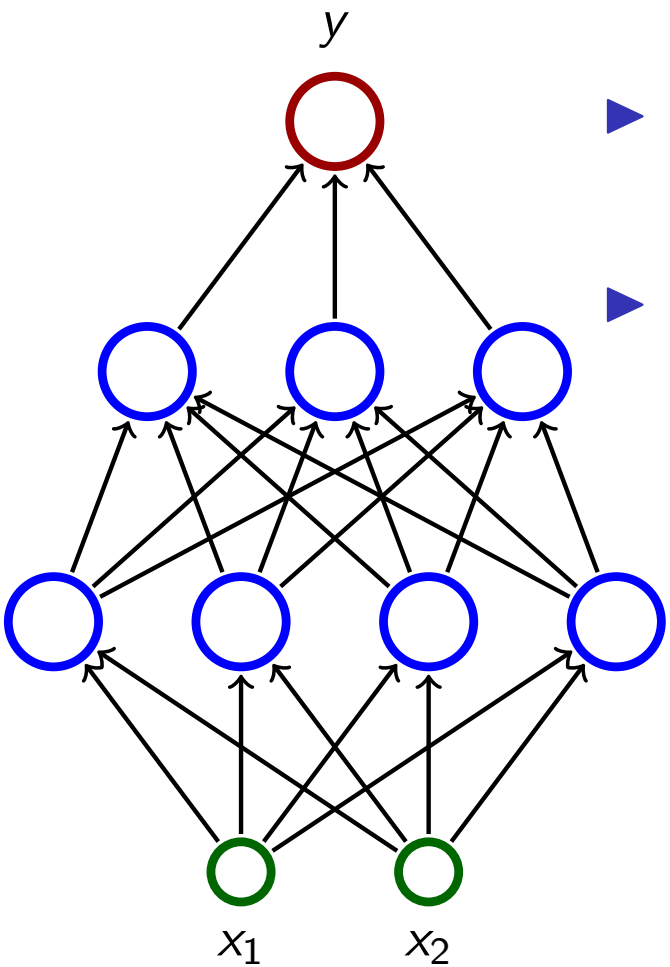
- Consider a three-layer network; each neuron has the unit step activation function.
- The network divides the input space in two subspaces according to the output (0 or 1).
  - The first (hidden) layer divides the input space into half-spaces.

Non-linear separation



- Consider a three-layer network; each neuron has the unit step activation function.
- The network divides the input space in two subspaces according to the output (0 or 1).
  - The first (hidden) layer divides the input space into half-spaces.
  - The second layer may, e.g., make intersections of the half-spaces $\Rightarrow$ convex sets.

Non-linear separation



- Consider a three-layer network; each neuron has the unit step activation function.
- The network divides the input space in two subspaces according to the output (0 or 1).
  - The first (hidden) layer divides the input space into half-spaces.
  - The second layer may, e.g., make intersections of the half-spaces $\Rightarrow$ convex sets.
  - The third layer may, e.g., make unions of some convex sets.

Example

Consider a triangle T in $\mathbb{R}^2$ determined by three vertices $(-1,-1),(1,-1),(-1,2)$.

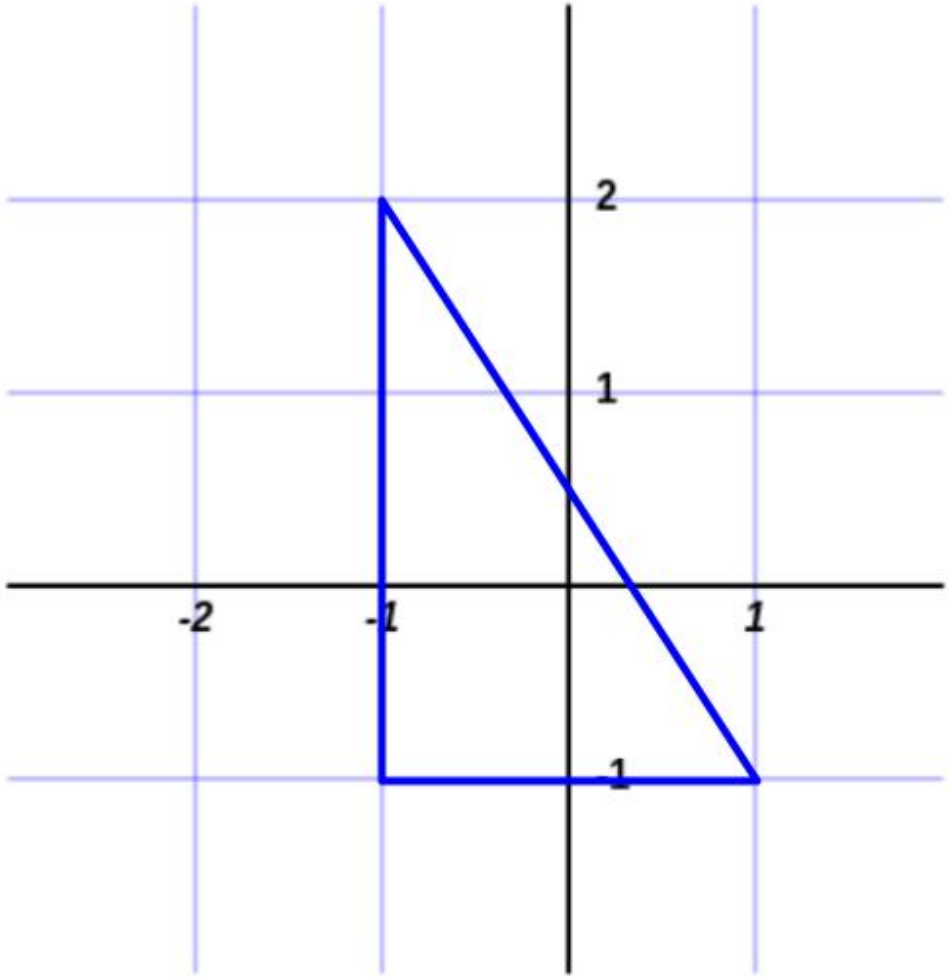
Give an example of a multilayer perceptron (MLP) with two input neurons and a single output neuron computing the function $F:\mathbb{R}^2 \to \{0,1\}$ defined as follows:

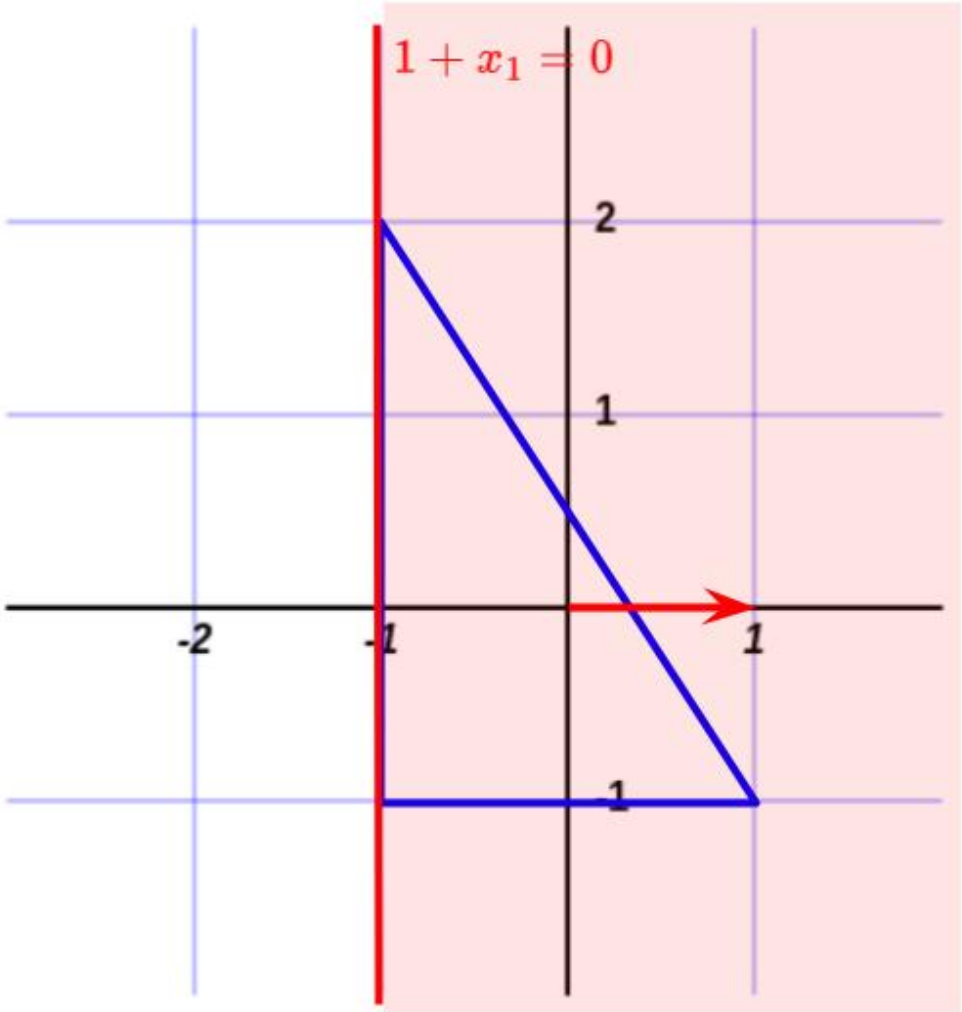
$F(x_1,x_2)=1$ iff (x_1,x_2) lies either inside, or on the border of T

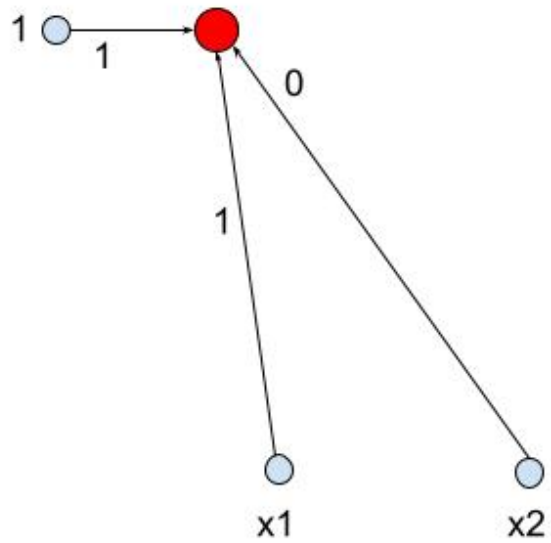
All activation functions in the network should be

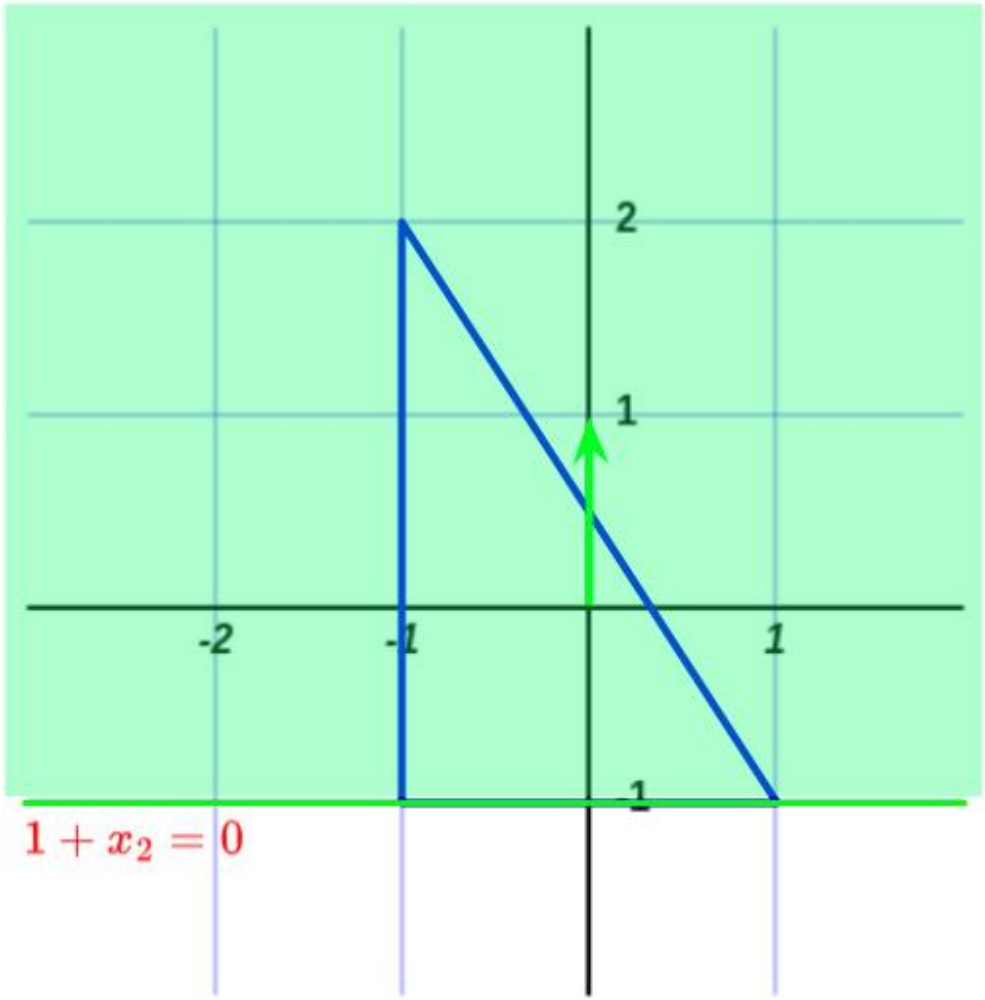
$$\sigma(\xi) = sgn(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

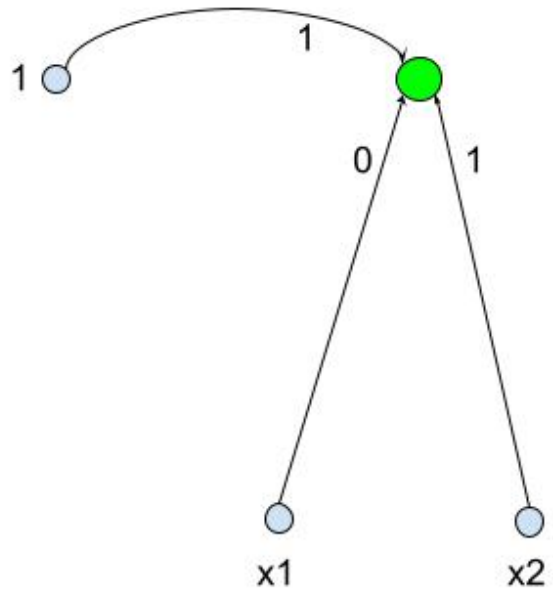
Homework: Consider $F(x_1,x_2)=1$ iff (x_1,x_2) lies inside of T (but *not* on the border)

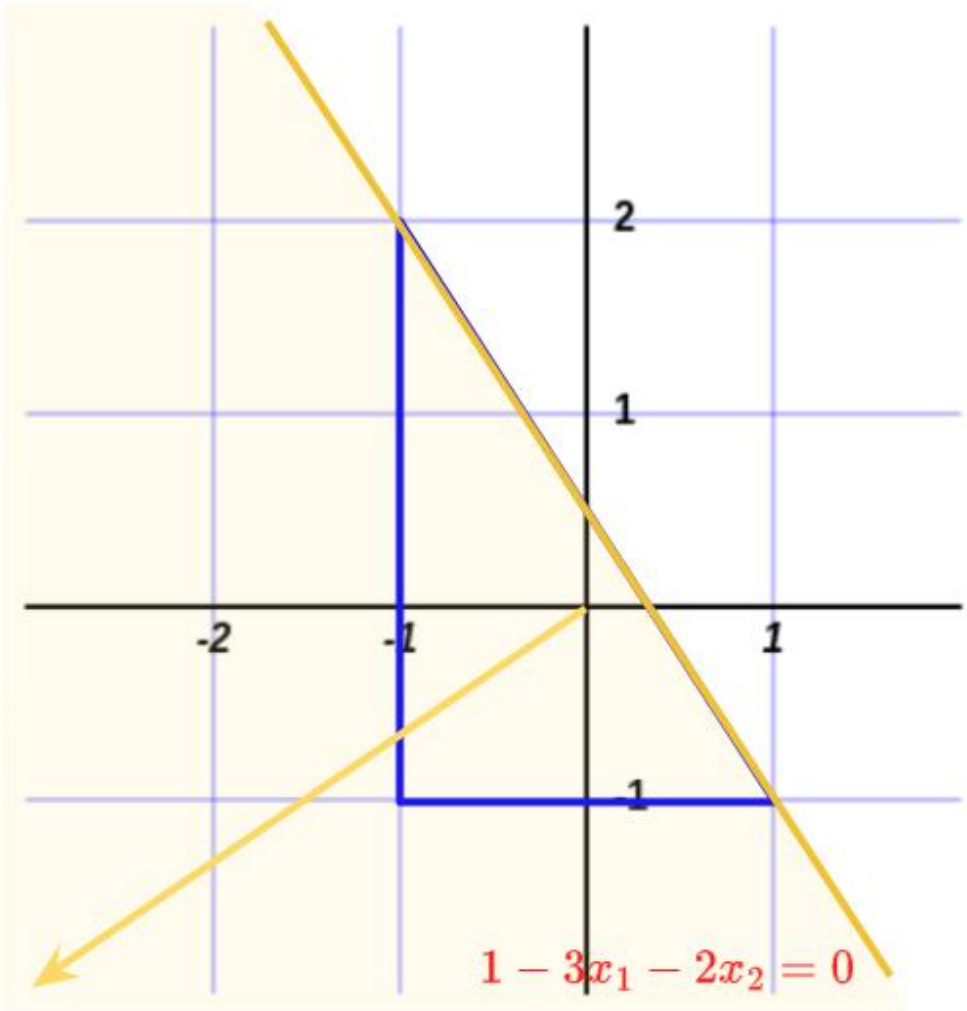

2
1
-2
-1
1
-1

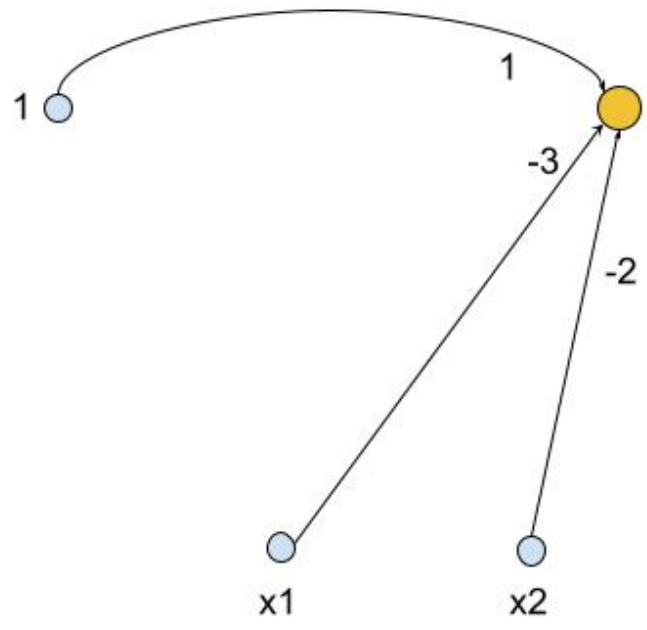

$1 + x_1 = 0$
2
1
-2
-1
1
1

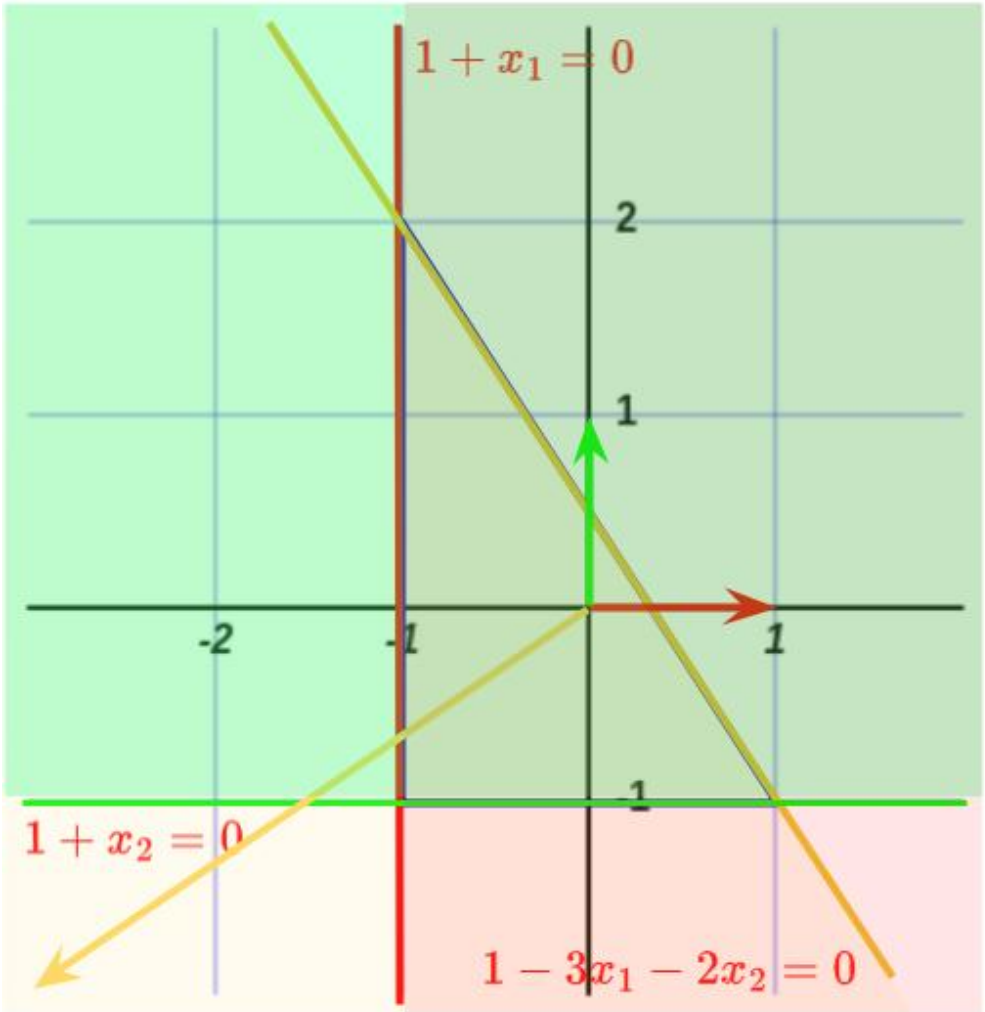

1
1
0
1
x1
x2

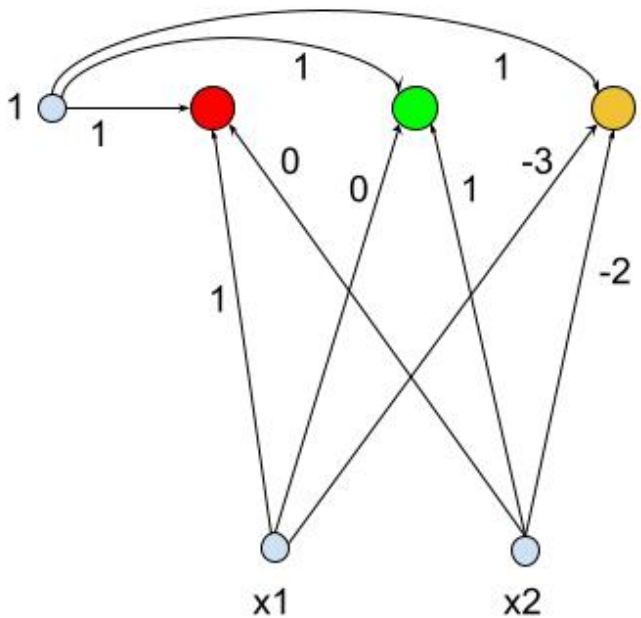

2
1
-2
-1
1
-1
$1 + x_2 = 0$

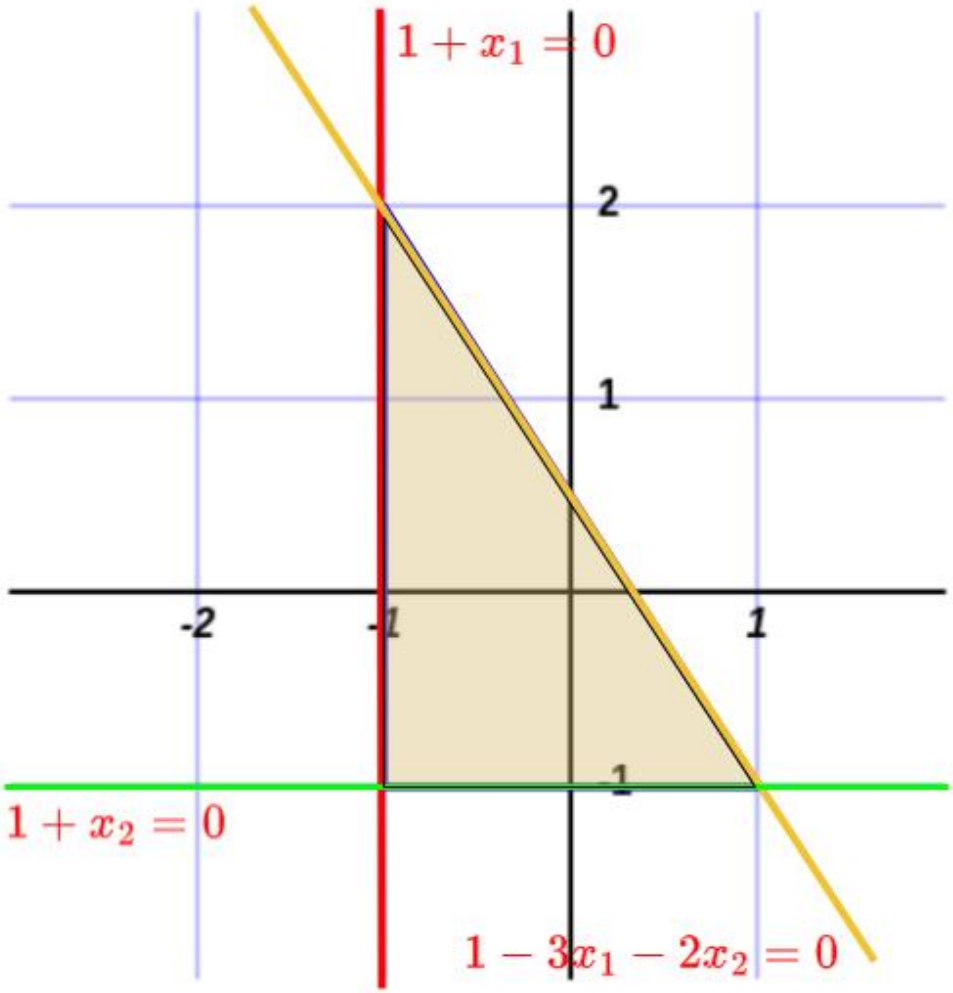

1
1
0
1
x1
x2

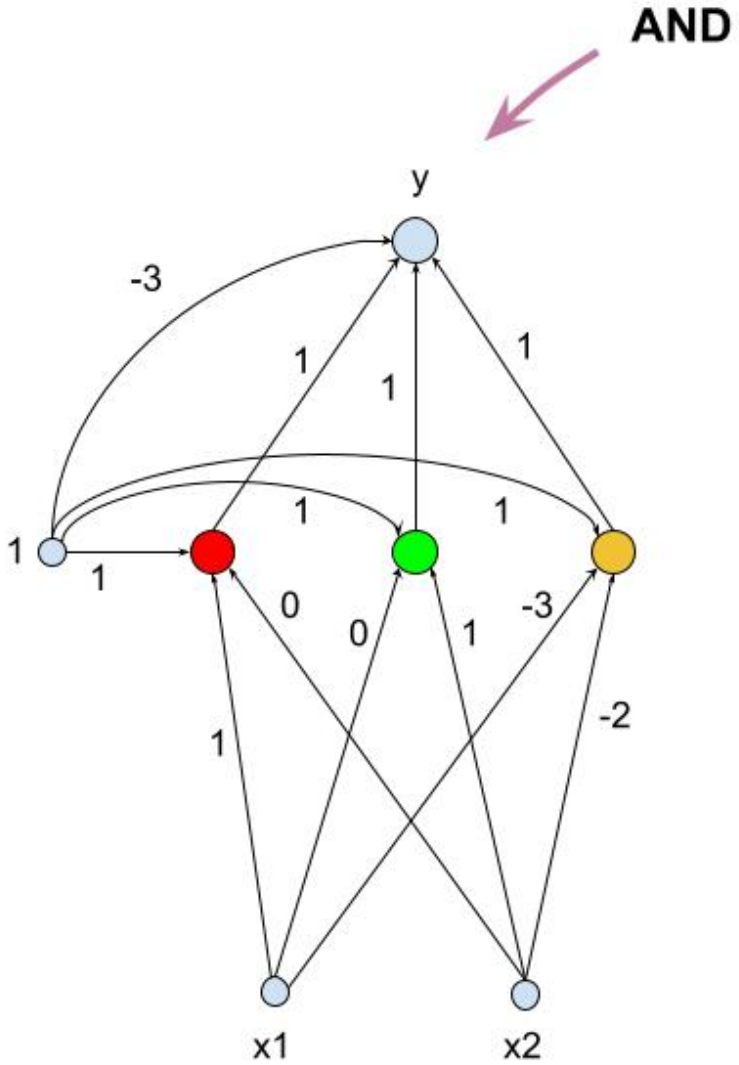

2
1
-2
-1
1
1
$1 - 3x_1 - 2x_2 = 0$


1
1
-3
-2
x1
x2


$1 + x_1 = 0$
2
1
-2
-1
1
-1
$1 + x_2 = 0$
$1 - 3x_1 - 2x_2 = 0$


1
1
1
1
0
0
1
-3
1
-2
x1
x2


$1 + x_1 = 0$
2
1
-2
-1
1
-1
$1 + x_2 = 0$
$1 - 3x_1 - 2x_2 = 0$


AND
y
-3
1
1
1
1
1
1
1
1
0
0
1
-3
-2
1
x1
x2

Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can

Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can
  - approximate with arbitrarily small error any "reasonable" boundary
    a given input is classified as 1 iff the output value of the network is $\geq 1/2$.

Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can
  - approximate with arbitrarily small error any "reasonable" boundary
    a given input is classified as 1 iff the output value of the network is $\geq 1/2$.
  - approximate with arbitrarily small error any "reasonable" function from $[0, 1]$ to $(0, 1)$.

  Here, "reasonable" means that it is pretty tough to find a function that is not reasonable.

Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can
  - approximate with arbitrarily small error any "reasonable" boundary
    a given input is classified as 1 iff the output value of the network is $\geq 1/2$.
  - approximate with arbitrarily small error any "reasonable" function from $[0, 1]$ to $(0, 1)$.

  Here, "reasonable" means that it is pretty tough to find a function that is not reasonable.

So, multilayer perceptrons are sufficiently powerful for any application.

Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can
  - approximate with arbitrarily small error any "reasonable" boundary
    a given input is classified as 1 iff the output value of the network is $\geq 1/2$.
  - approximate with arbitrarily small error any "reasonable" function from $[0, 1]$ to $(0, 1)$.

  Here, "reasonable" means that it is pretty tough to find a function that is not reasonable.

So, multilayer perceptrons are sufficiently powerful for any application.

But for a long time, at least throughout the 60s and 70s, nobody well-known knew any efficient method for training multilayer networks!

Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can
  - approximate with arbitrarily small error any "reasonable" boundary
    a given input is classified as 1 iff the output value of the network is $\geq 1/2$.
  - approximate with arbitrarily small error any "reasonable" function from $[0, 1]$ to $(0, 1)$.

  Here, "reasonable" means that it is pretty tough to find a function that is not reasonable.

So, multilayer perceptrons are sufficiently powerful for any application.

But for a long time, at least throughout the 60s and 70s, nobody well-known knew any efficient method for training multilayer networks!

An efficient way of using the gradient descent was published in 1986!

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)
- individual neurons are denoted by indices, e.g., i, j.

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)

- individual neurons are denoted by indices, e.g., i, j.
- ξ_j is the inner potential of the neuron j when the computation is finished.

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)

- individual neurons are denoted by indices, e.g., i, j.
- ξ_j is the inner potential of the neuron j when the computation is finished.
- y_j is the output value of the neuron j when the computation is finished.

  (we formally assume $y_0 = 1$)

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)

- individual neurons are denoted by indices, e.g., i, j.
- ξ_j is the inner potential of the neuron j when the computation is finished.
- y_j is the output value of the neuron j when the computation is finished.

  (we formally assume $y_0 = 1$)
- w_{ji} is the weight of the arc **from** the neuron i **to** the neuron j.

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)

- individual neurons are denoted by indices, e.g., i, j.
- ξ_j is the inner potential of the neuron j when the computation is finished.
- y_j is the output value of the neuron j when the computation is finished.

  (we formally assume $y_0 = 1$)
- w_{ji} is the weight of the arc **from** the neuron i **to** the neuron j.

- $j_{\leftarrow}$ is the set of all neurons from which there are edges to j

  (i.e. $j_{\leftarrow}$ is the layer directly below j)

MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)

- individual neurons are denoted by indices, e.g., i, j.
- ξ_j is the inner potential of the neuron j when the computation is finished.
- y_j is the output value of the neuron j when the computation is finished.

  (we formally assume $y_0 = 1$)
- w_{ji} is the weight of the arc **from** the neuron i **to** the neuron j.

- $j_{\leftarrow}$ is the set of all neurons from which there are edges to j

  (i.e. $j_{\leftarrow}$ is the layer directly below j)
- $j^{\rightarrow}$ is the set of all neurons with edges from j.

  (i.e. $j^{\rightarrow}$ is the layer directly above j)

MLP – Notation

- Inner potential of a neuron j:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

MLP – Notation

- Inner potential of a neuron j:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- A value of a non-input neuron $j \in Z \setminus X$ when the computation is finished is

$$y_j = \sigma_j(\xi_j)$$

Here σ_j is an activation function of the neuron j.

(y_j is determined by weights $\vec{w}$ and a given input $\vec{x}$, so it's sometimes written as $y_j[\vec{w}](\vec{x})$)

MLP – Notation

- Inner potential of a neuron j:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- A value of a non-input neuron $j \in Z \setminus X$ when the computation is finished is

$$y_j = \sigma_j(\xi_j)$$

  Here σ_j is an activation function of the neuron j.

  (y_j is determined by weights $\vec{w}$ and a given input $\vec{x}$, so it's sometimes written as $y_j[\vec{w}](\vec{x})$)

- Fixing weights of all neurons, the network computes a function $F[\vec{w}] : \mathbb{R}^{|X|} \to \mathbb{R}^{|Y|}$ as follows: Assign values of a given vector $\vec{x} \in \mathbb{R}^{|X|}$ to the input neurons, evaluate the network, then $F[\vec{w}](\vec{x})$ is the vector of values of the output neurons.

  Here, we implicitly assume a fixed ordering on input and output vectors.

MLP – Learning

- Given a set D of training examples:

$$D = \left\{ \left(\vec{x}_k, \vec{d}_k \right) \quad | \quad k = 1, \ldots, p \right\}$$

Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write d_{kj} to denote the value in $\vec{d}_k$ corresponding to the output neuron j.

MLP – Learning

- Given a set D of training examples:

$$D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \quad | \quad k = 1, \ldots, p \right\}$$

  Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write d_{kj} to denote the value in $\vec{d}_k$ corresponding to the output neuron j.

- **Error Function:** $E(\vec{w})$ where $\vec{w}$ is a vector of all weights in the network. The choice of E depends on the solved task (classification vs regression etc.).
  Example (Squared error):

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

  where

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} (y_j[\vec{w}](\vec{x}_k) - d_{kj})^2$$

MLP – Batch Gradient Descent

The algorithm computes a sequence of weights $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- weights $\vec{w}^{(0)}$ are initialized randomly close to 0

MLP – Batch Gradient Descent

The algorithm computes a sequence of weights $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- weights $\vec{w}^{(0)}$ are initialized randomly close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$) is $\vec{w}^{(t+1)}$ computed as follows:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

MLP – Batch Gradient Descent

The algorithm computes a sequence of weights $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- weights $\vec{w}^{(0)}$ are initialized randomly close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$) is $\vec{w}^{(t+1)}$ computed as follows:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

where

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

is the weight change w_{ji} and $0 < \varepsilon(t) \leq 1$ is the learning rate in the step $t+1$.

MLP – Batch Gradient Descent

The algorithm computes a sequence of weights $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- weights $\vec{w}^{(0)}$ are initialized randomly close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$) is $\vec{w}^{(t+1)}$ computed as follows:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

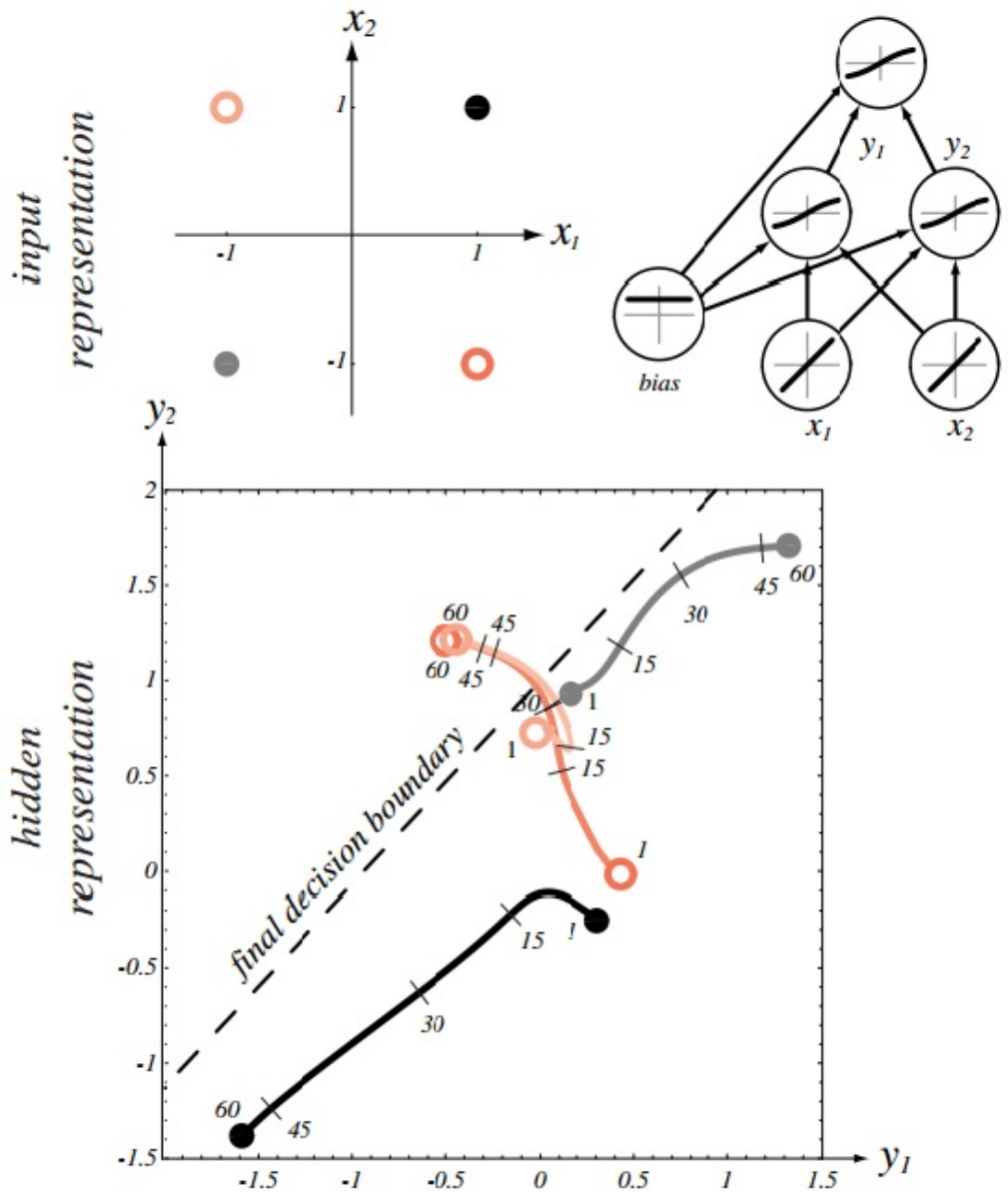
where

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

is the weight change w_{ji} and $0 < \varepsilon(t) \leq 1$ is the learning rate in the step $t+1$.

Note that $\frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$ is a component of ∇E, i.e., the weight change in the step $t+1$ can be written as follows: $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon(t) \cdot \nabla E(\vec{w}^{(t)})$.

Illustration of Gradient Descent – XOR



Source: Pattern Classification (2nd Edition); Richard O. Duda, Peter E. Hart, David G. Stork

Stochastic Gradient Descent (SGD)

Assume that $E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$ where $E_k(\vec{w})$ is an error w.r.t. the single training example $(\vec{x}_k, \vec{d}_k)$.

- weights in $\vec{w}^{(0)}$ are randomly initialized to values close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$), weights $\vec{w}^{(t+1)}$ are computed as follows:
  - Choose (randomly) a set of training examples $T \subseteq \{1, \ldots, p\}$
  - Compute

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} + \Delta\vec{w}^{(t)}$$

where

$$\Delta\vec{w}^{(t)} = -\varepsilon(t) \cdot \sum_{k \in T} \nabla E_k(\vec{w}^{(t)})$$

- $0 < \varepsilon(t) \leq 1$ is a *learning rate* in step $t+1$
- $\nabla E_k(\vec{w}^{(t)})$ is the gradient of the error of the example k

Note that the random choice of the minibatch is typically implemented by randomly shuffling all data and then choosing minibatches sequentially.

Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to a local minimum or oscillate indefinitely.

Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to a local minimum or oscillate indefinitely.
- In practice, does converge to low error for many large networks on (big) real data.

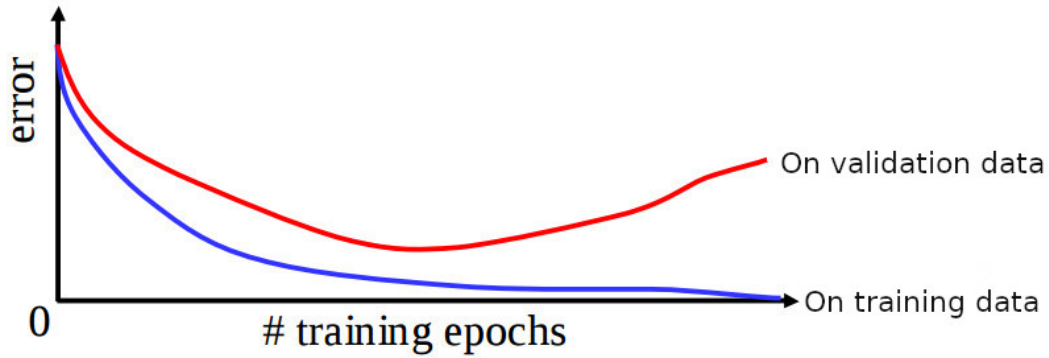
Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to a local minimum or oscillate indefinitely.
- In practice, does converge to low error for many large networks on (big) real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.

There are many issues concerning learning efficiency (data normalization, selection of activation functions, weight initialization, learning rate, efficiency of the gradient descent itself, etc.) – see PV021.

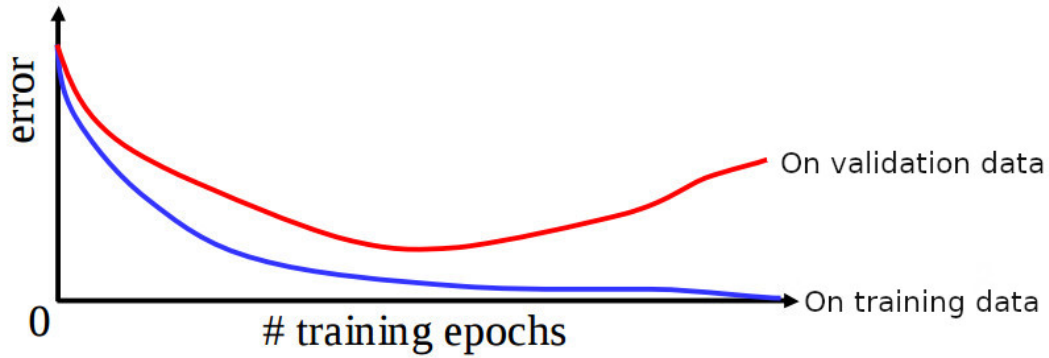
Overfitting

- Due to their expressive power, neural networks are quite sensitive to overfitting.



Overfitting

- Due to their expressive power, neural networks are quite sensitive to overfitting.



- Keep a hold-out validation set and test the error of the network on this set after every epoch. Stop training when additional epochs increase the validation error.

  The validation error can be measured by completely different means than the training error E.

Hidden Neurons Representations

Trained hidden neurons can be seen as newly constructed features.

E.g., in a two-layer network used for classification, the hidden layer transforms the input so that important features become explicit (and hence the result may become linearly separable).

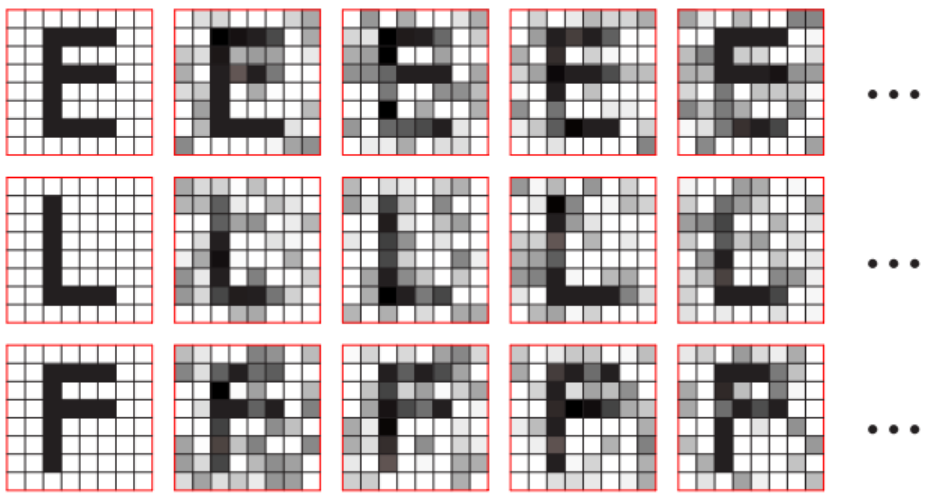
Hidden Neurons Representations

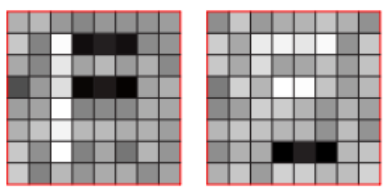
Trained hidden neurons can be seen as newly constructed features.

E.g., in a two-layer network used for classification, the hidden layer transforms the input so that important features become explicit (and hence the result may become linearly separable).

Consider a two-layer MLP, 64-2-3, for classifying letters (three output neurons, each corresponding to one of the letters).

sample training patterns





learned input-to-hidden weights

Optimal Architecture?

- For MLP: Too few hidden neurons prevent the network from adequately fitting the data. Too many hidden units can result in overfitting.

  (There are advanced methods that prevent overfitting even for rich models, such as regularization, where the error function penalizes overfitting – see PV021.)

Optimal Architecture?

- For MLP: Too few hidden neurons prevent the network from adequately fitting the data. Too many hidden units can result in overfitting.

  (There are advanced methods that prevent overfitting even for rich models, such as regularization, where the error function penalizes overfitting – see PV021.)
- There are (almost) infinitely many types of architectures of neural networks (convolutional, recurrent, transformers, adversarial, etc.) suitable for various tasks.

Optimal Architecture?

- For MLP: Too few hidden neurons prevent the network from adequately fitting the data. Too many hidden units can result in overfitting.

  (There are advanced methods that prevent overfitting even for rich models, such as regularization, where the error function penalizes overfitting – see PV021.)
- There are (almost) infinitely many types of architectures of neural networks (convolutional, recurrent, transformers, adversarial, etc.) suitable for various tasks.
- **Transfer learning:** Start with a known solution to a related problem.

  Simplified view: Preserve lower parts of the network trained to solve the related problem (feature extractors). Add your top part and then train only the new top part (or train the whole network but carefully).

How to Choose Activation Functions & Error

- **Hidden neurons:** "Almost" linear activations such as (leaky) ReLU ($y = \max(0, \xi)$)

  Better than sigmoidal that saturate more often.
- **Output neurons:** Single output:
  - Regression: Typically "linear" output, i.e., no activation on the output neuron.
  - Binary classification: Logistic sigmoid $y = 1/(1 + e^{-\xi})$
- **Error:** Single output:
  - Regression: (Mean) squared error
  - Binary classification: Binary cross-entropy

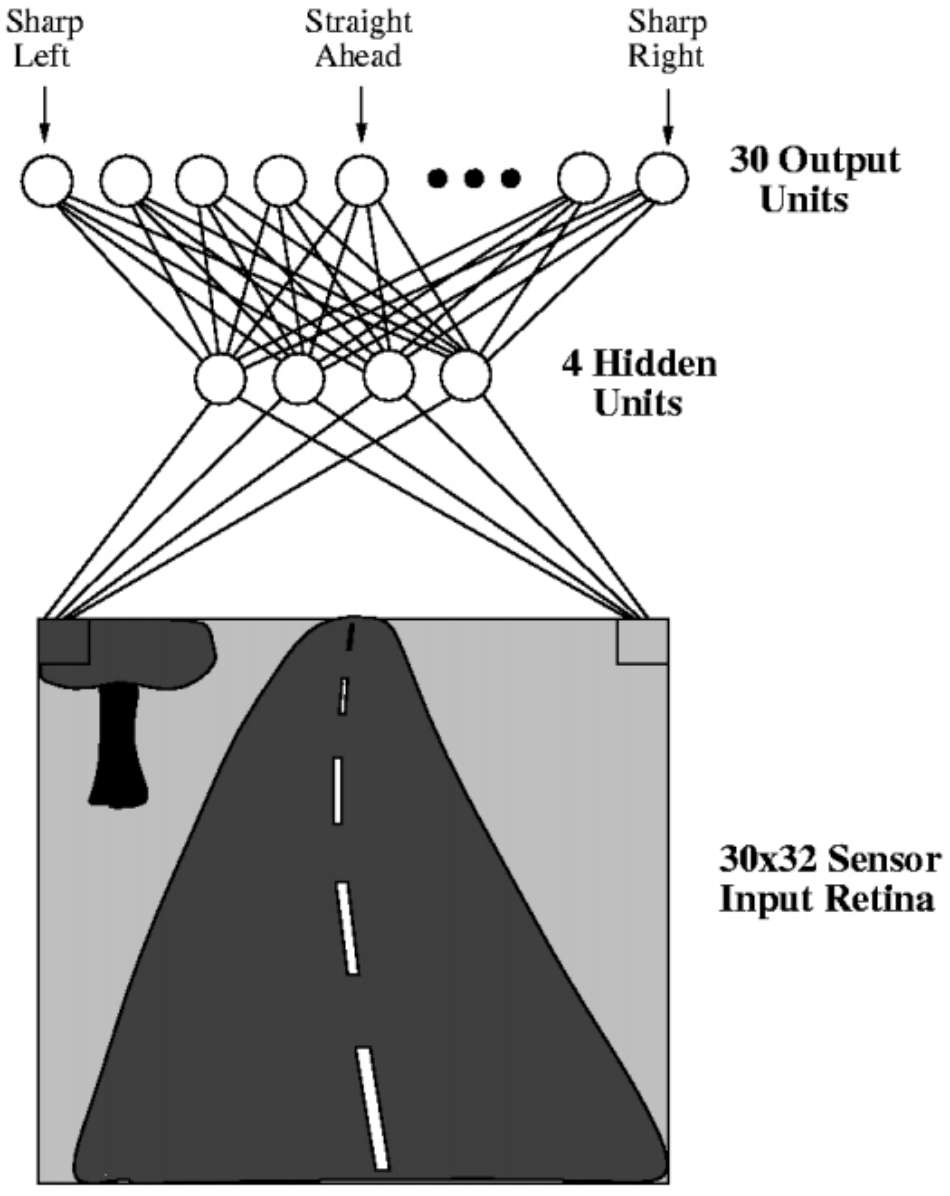
For multiple outputs and classification, use softmax output and cross-entropy.

Applications

- Image recognition, segmentation, etc.
- Machine translation and other text processing
- Text generation, image generation, movie generation, theatre plays generation
- Text to Speech and vice versa
- Finance, business predictions, fraud detection
- Game playing (backgammon is a classic example, AlphaGo is the famous one, computer games are the big ones, **bridge** is the hard one)
- (artificial brain and intelligence)
- ...

Text and image processing are possibly the most advanced deep learning applications.

ALVINN


Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina

ALVINN

- Two layer MLP, $960 - 4 - 30$ (sometimes $960 - 5 - 30$)

ALVINN

- Two layer MLP, $960 - 4 - 30$ (sometimes $960 - 5 - 30$)
- Inputs correspond to pixels.
- Sigmoidal activation function (logistic sigmoid).

ALVINN

- Two layer MLP, $960 - 4 - 30$ (sometimes $960 - 5 - 30$)
- Inputs correspond to pixels.
- Sigmoidal activation function (logistic sigmoid).
- Direction corresponds to the center of gravity.

  , I.e., output neurons are considered as points of mass evenly distributed along a line. The weight of each neuron corresponds to its value.

ALVINN – Training

Trained while driving.

ALVINN – Training

Trained while driving.

- A camera captured the road from the front window, approx. 25 pictures per second

ALVINN – Training

Trained while driving.

- A camera captured the road from the front window, approx. 25 pictures per second
- Training examples $(\vec{x}_k, \vec{d}_k)$ where

ALVINN – Training

Trained while driving.

- A camera captured the road from the front window, approx. 25 pictures per second
- Training examples $(\vec{x}_k, \vec{d}_k)$ where
  - $\vec{x}_k$ = image of the road

ALVINN – Training

Trained while driving.

- A camera captured the road from the front window, approx. 25 pictures per second
- Training examples $(\vec{x}_k, \vec{d}_k)$ where
  - $\vec{x}_k$ = image of the road
  - $\vec{d}_k \approx$ corresponding direction of the steering wheel set by the driver

ALVINN – Training

Trained while driving.

- A camera captured the road from the front window, approx. 25 pictures per second
- Training examples $(\vec{x}_k, \vec{d}_k)$ where
  - $\vec{x}_k$ = image of the road
  - $\vec{d}_k \approx$ corresponding direction of the steering wheel set by the driver
- the values $\vec{d}_k$ computed using Gaussian distribution:

$$d_{ki} = e^{-D_i^2/10}$$

Where D_i is the distance between the i-th output from the one corresponding to the steering wheel's real direction.

(The authors claimed this approach is better than the binary output because similar road directions induce similar driver reactions.)

Selection of Training Examples

Naive approach: just take images from the camera.

Selection of Training Examples

Naive approach: just take images from the camera.

Problems:

Selection of Training Examples

Naive approach: just take images from the camera.

Problems:

- A too-good driver never teaches the network how to solve deviations from the right track. A couple of harsh solutions:

Selection of Training Examples

Naive approach: just take images from the camera.

Problems:

- A too-good driver never teaches the network how to solve deviations from the right track. A couple of harsh solutions:
  - turn the learning off momentarily, deviate from the right track, then turn on the learning and let the network learn how to solve the situation.

Selection of Training Examples

Naive approach: just take images from the camera.

Problems:

- A too-good driver never teaches the network how to solve deviations from the right track. A couple of harsh solutions:
  - turn the learning off momentarily, deviate from the right track, then turn on the learning and let the network learn how to solve the situation.
  - let the driver go crazy! (a bit dangerous, expensive, unreliable)

Selection of Training Examples

Naive approach: just take images from the camera.

Problems:

- A too-good driver never teaches the network how to solve deviations from the right track. A couple of harsh solutions:
  - turn the learning off momentarily, deviate from the right track, then turn on the learning and let the network learn how to solve the situation.
  - let the driver go crazy! (a bit dangerous, expensive, unreliable)
- Images are very similar (the network sees the road from the right lane), overfitting.

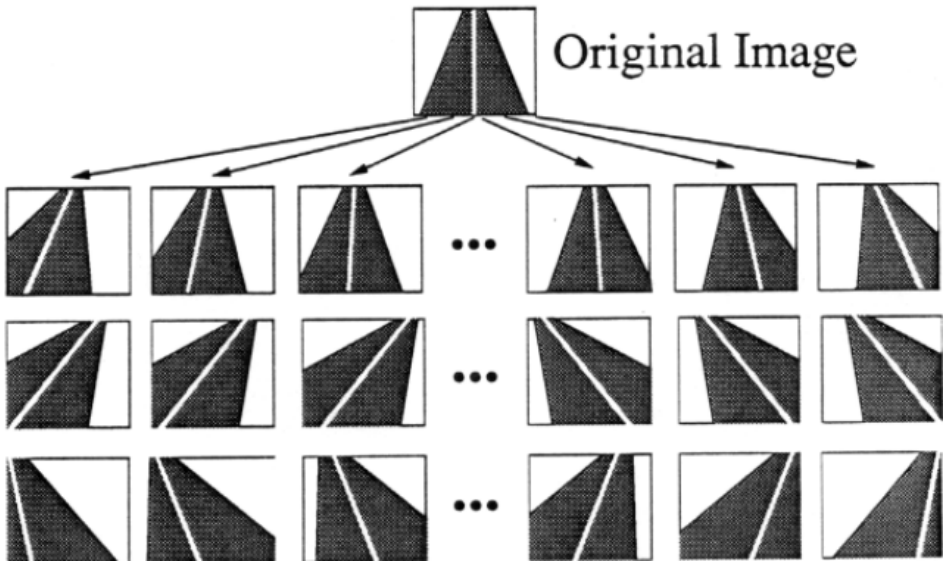
Selection of Training Examples

Problems with too good driver were solved as follows:

Selection of Training Examples

Problems with too good driver were solved as follows:

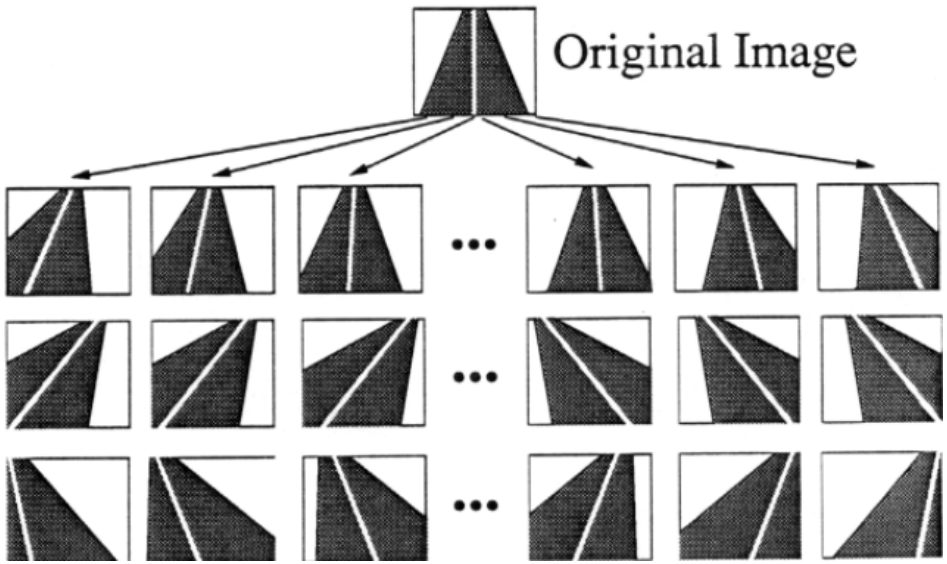
- every image of the road has been has been transformed to 15 slightly different copies



Selection of Training Examples

Problems with too good driver were solved as follows:

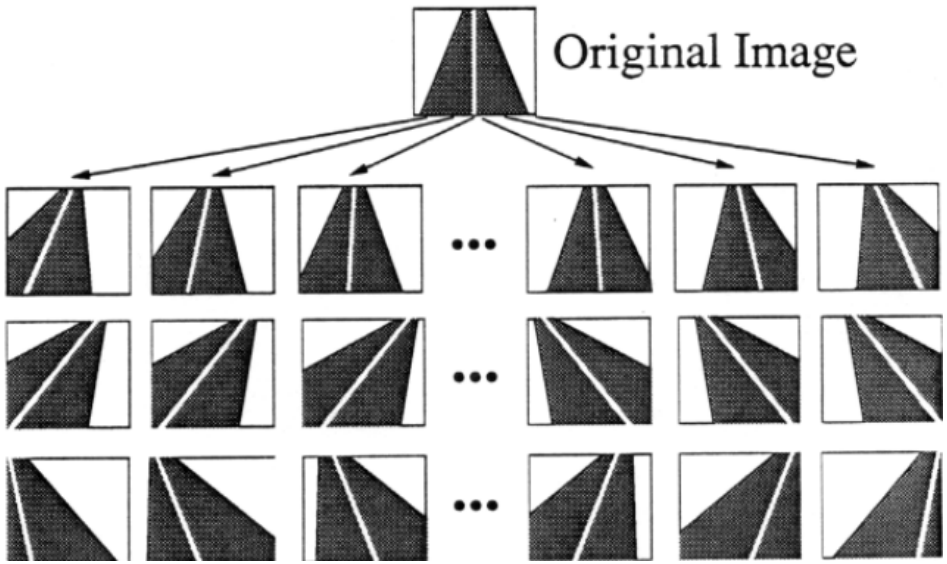
- every image of the road has been has been transformed to 15 slightly different copies



Selection of Training Examples

Problems with too good driver were solved as follows:

- every image of the road has been has been transformed to 15 slightly different copies

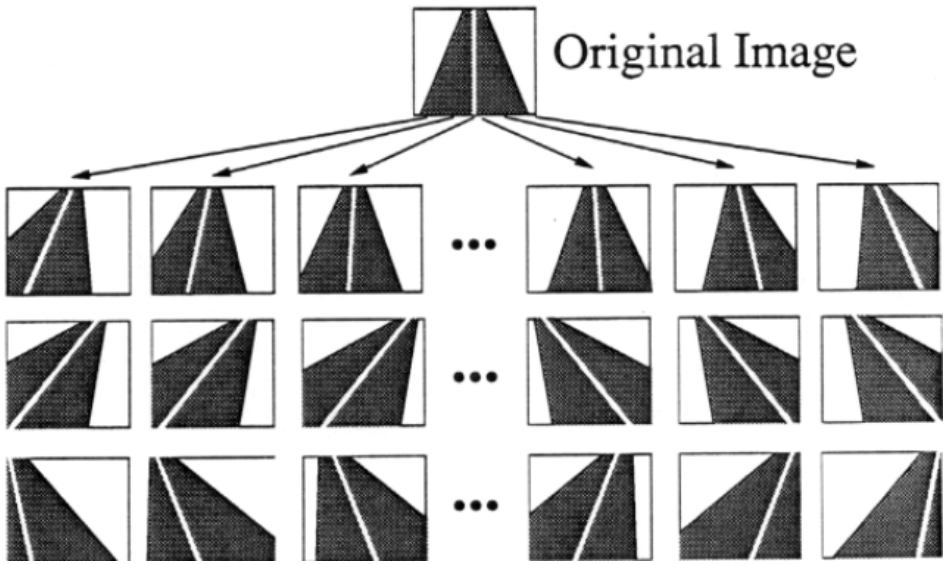


The repetitiveness of images was solved as follows:

Selection of Training Examples

Problems with too good driver were solved as follows:

- every image of the road has been has been transformed to 15 slightly different copies



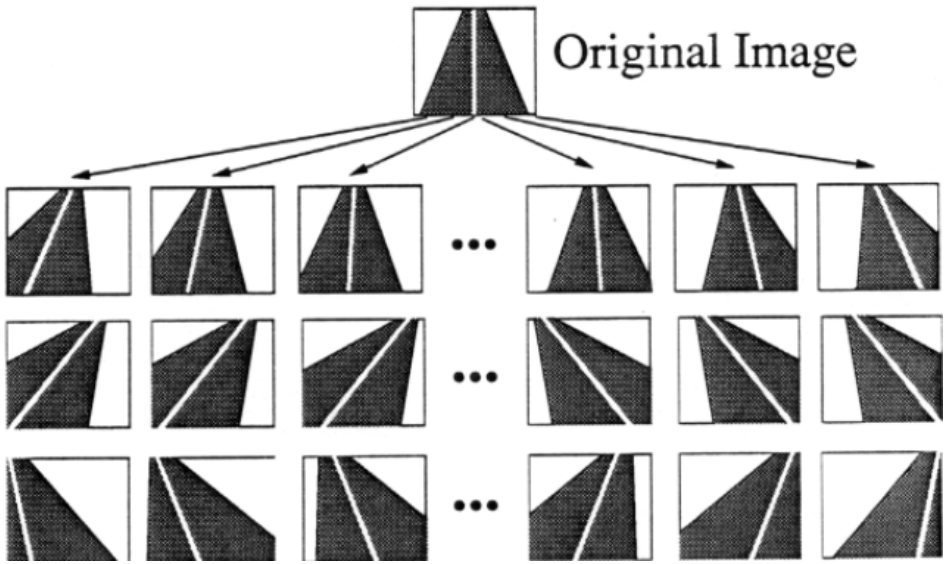
The repetitiveness of images was solved as follows:

- the system has a buffer of 200 images (including the 15 copies of the current one), in every round trains on these images

Selection of Training Examples

Problems with too good driver were solved as follows:

- every image of the road has been has been transformed to 15 slightly different copies



The repetitiveness of images was solved as follows:

- the system has a buffer of 200 images (including the 15 copies of the current one), in every round trains on these images
- afterward, a new image is captured, 15 copies made, and these new 15 substitute 15 selected from the buffer (10 with the smallest training error, five randomly)

ALVINN – Training

- gradient descent

ALVINN – Training

- gradient descent
- constant learning rate (possibly different for each neuron – see PV021)

ALVINN – Training

- gradient descent
- constant learning rate (possibly different for each neuron – see PV021)
- some other optimizations (see PV021)

ALVINN – Training

- gradient descent
- constant learning rate (possibly different for each neuron – see PV021)
- some other optimizations (see PV021)

The result:

- Training took 5 minutes, and the speed was 4 miles per hour
  (The speed was limited by the hydraulic controller of the steering wheel, not the learning algorithm.)

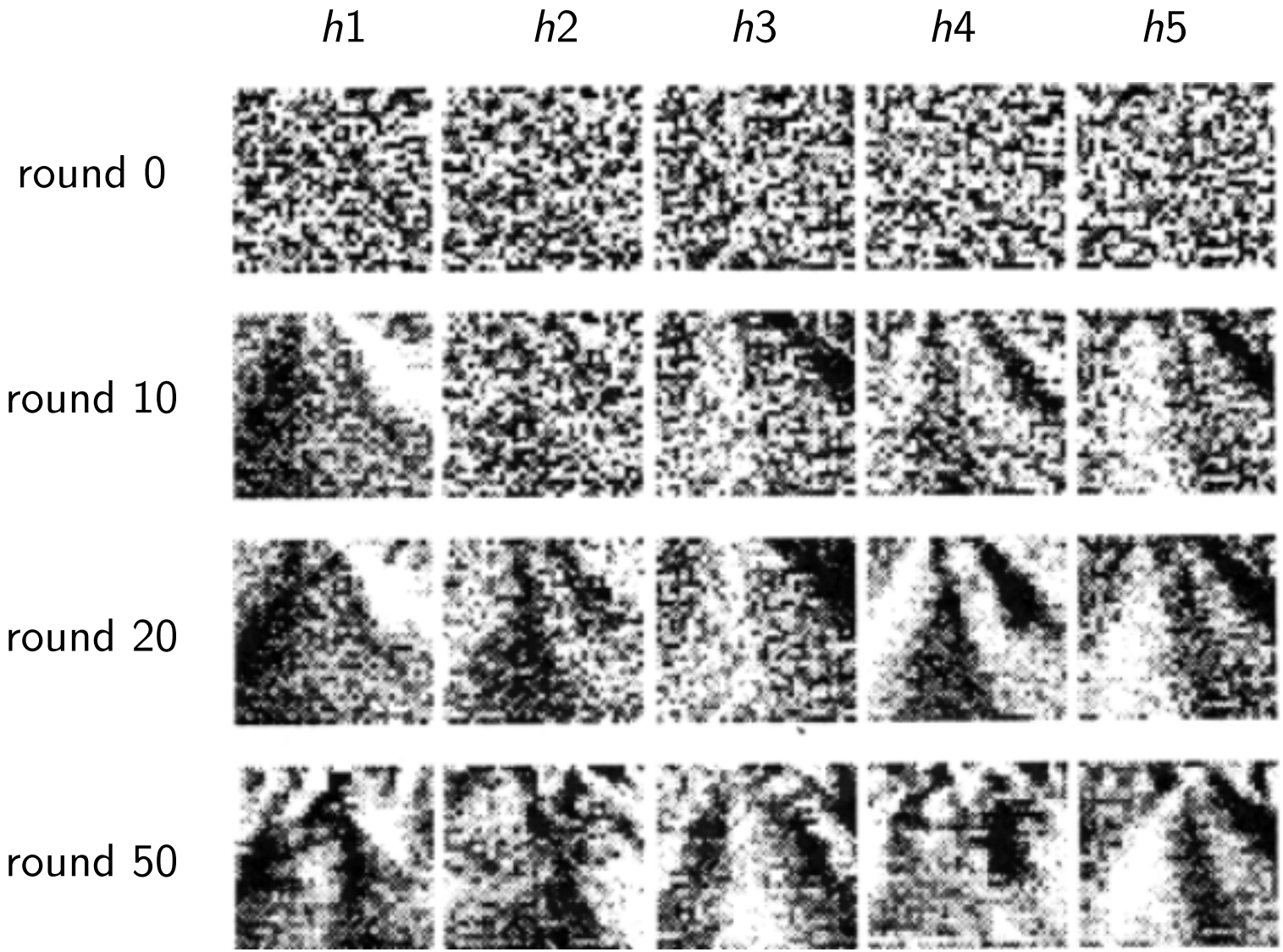
ALVINN – Training

- gradient descent
- constant learning rate (possibly different for each neuron – see PV021)
- some other optimizations (see PV021)

The result:

- Training took 5 minutes, and the speed was 4 miles per hour
  (The speed was limited by the hydraulic controller of the steering wheel, not the learning algorithm.)
- ALVINN was able to go through roads it had never "seen" and in different weather

ALVINN – Weight Learning



Here $h1, \ldots, h5$ are values of hidden neurons.

Backpropagation

How to Compute the Gradient?

To implement a single step of the gradient descent, we need to compute the partial derivatives $\frac{\partial E}{\partial w_{ij}}$ of E w.r.t. all weights w_{ij}.

How to Compute the Gradient?

To implement a single step of the gradient descent, we need to compute the partial derivatives $\frac{\partial E}{\partial w_{ij}}$ of E w.r.t. all weights w_{ij}.

To simplify consider for now a restricted case:

- Single element training set

$$D = \{(x, d)\}$$

where $x \in \mathbb{R}$ and $d \in \mathbb{R}$ are numbers
(i.e., not vectors as in the general case).

How to Compute the Gradient?

To implement a single step of the gradient descent, we need to compute the partial derivatives $\frac{\partial E}{\partial w_{ij}}$ of E w.r.t. all weights w_{ij}.

To simplify consider for now a restricted case:

- Single element training set

$$D = \{(x, d)\}$$

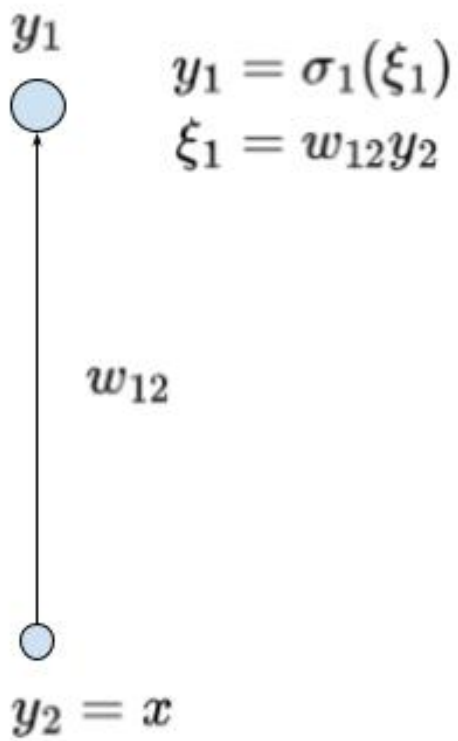
where $x \in \mathbb{R}$ and $d \in \mathbb{R}$ are numbers
(i.e., not vectors as in the general case).

- The error function is the squared error.
  - Assume that 1 is the output neuron,
  - which means that $y_1 = y_1[\vec{w}](x)$ is the output of the network with weights $\vec{w}$ and the input x,
  - and the error on D is then

$$E(\vec{w}) = \frac{1}{2}(y_1 - d)^2$$

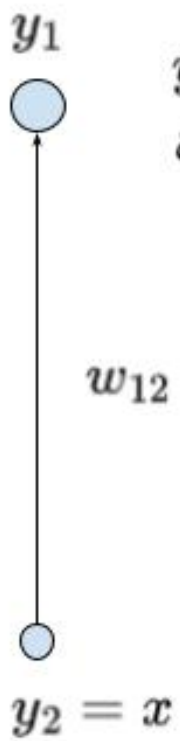
$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

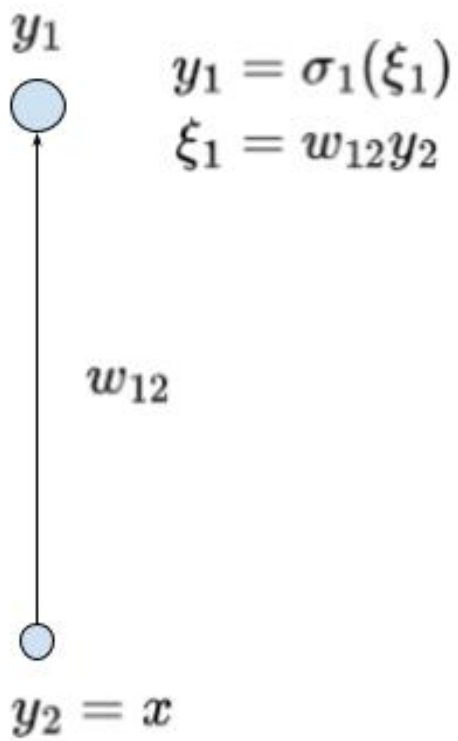


$$y_1 = \sigma_1(\xi_1)$$
$$\xi_1 = w_{12} y_2$$

$$\frac{\partial E}{\partial w_{12}}$$

$$D = \{(x, d)\}$$

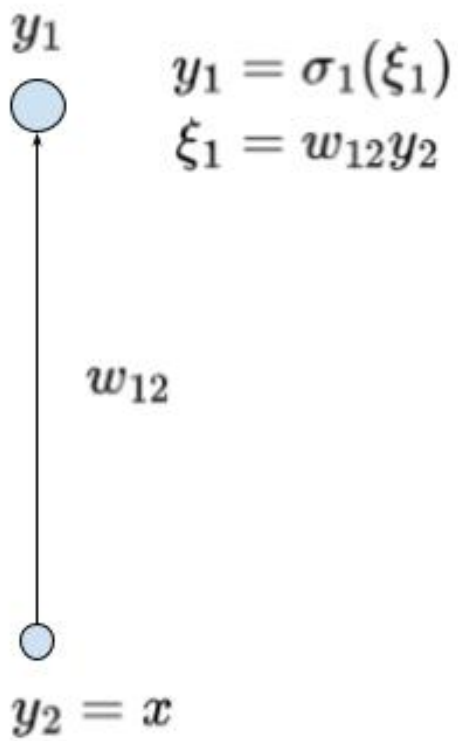
$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1} \frac{\partial y_1}{\partial w_{12}}$$

$$D = \{(x, d)\}$$

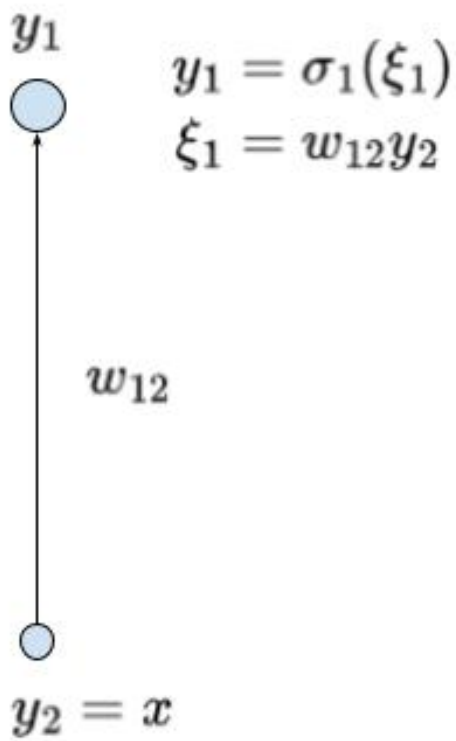
$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



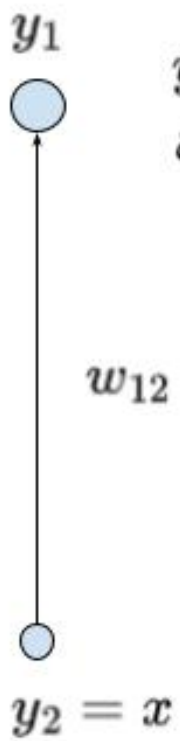
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}}$$
$$= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2$$

Here σ_1' is just the plain derivative of σ' as a function of a single variable

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}}$$
$$= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$



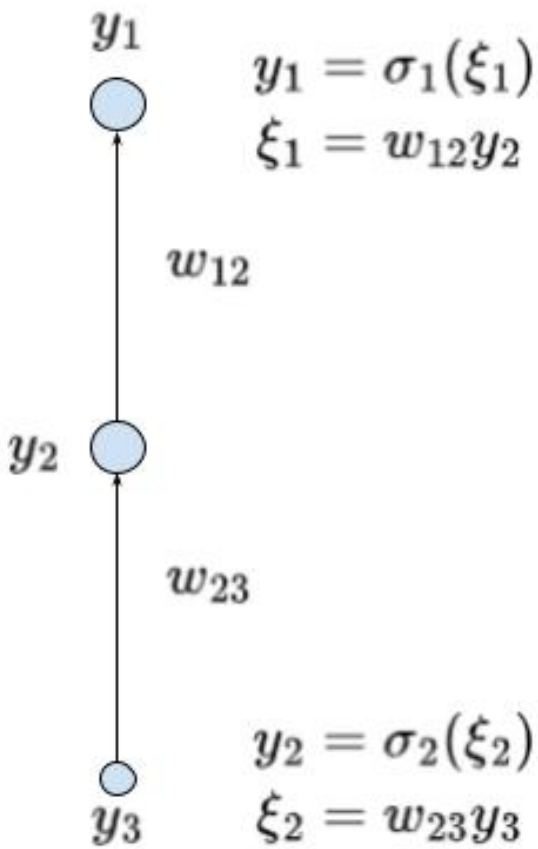
Here σ_1' is just the plain derivative of σ' as a function of a single variable

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

Note that if σ_1 is identity, we obtain exactly the gradient from the linear regression method. Considering σ_1 equal to the logistic sigmoid and E the cross-entropy, we get the logistic regression gradient.

$$D = \{(x, d)\}$$

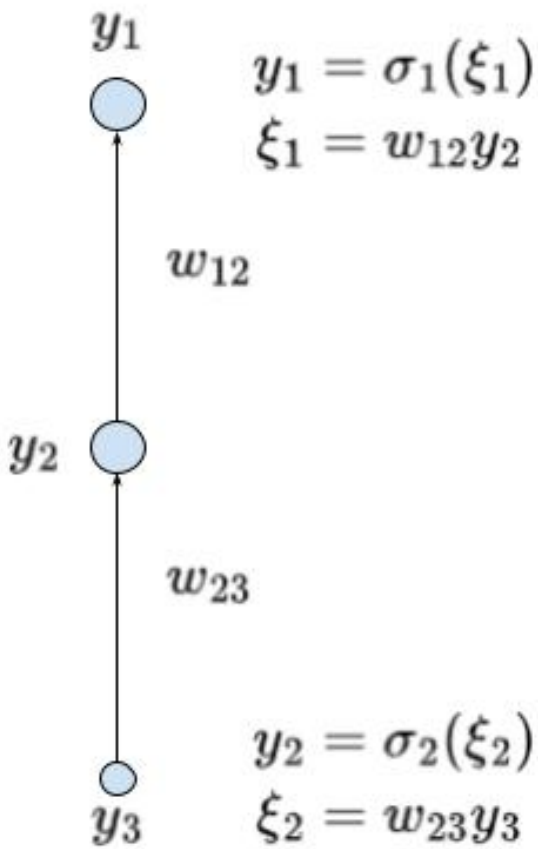
$$E = \frac{1}{2}(y_1 - d)^2$$



$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

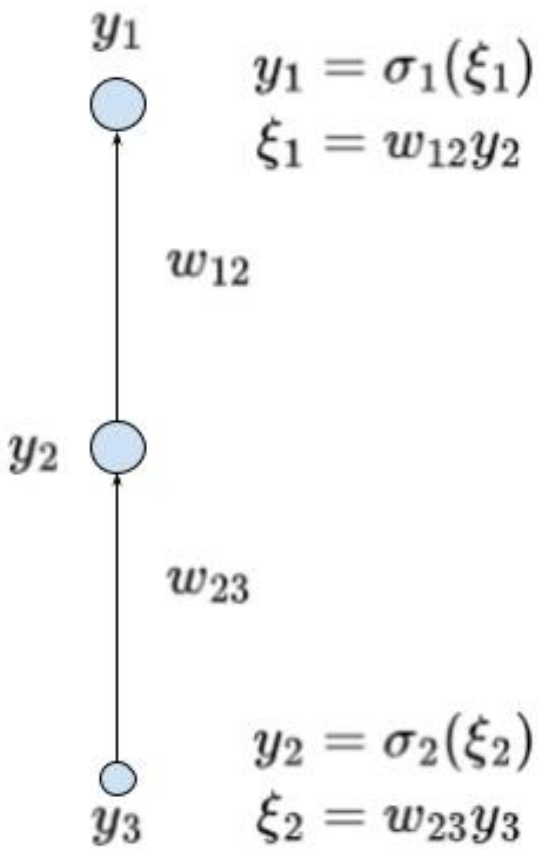
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$



$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

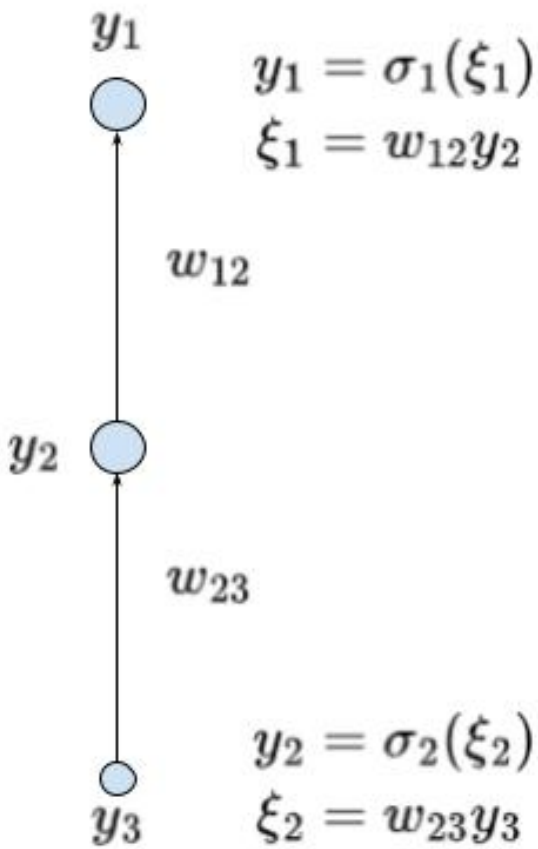
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$



$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_3$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



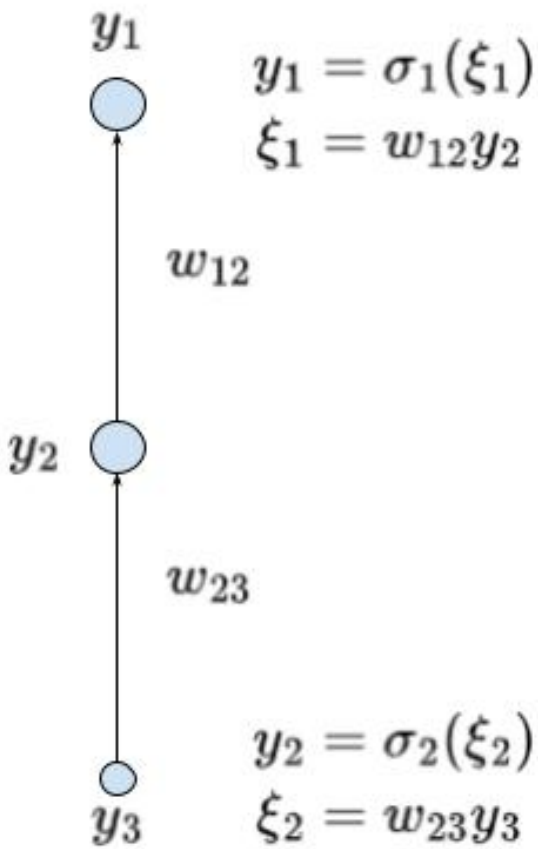
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) y_3$$

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

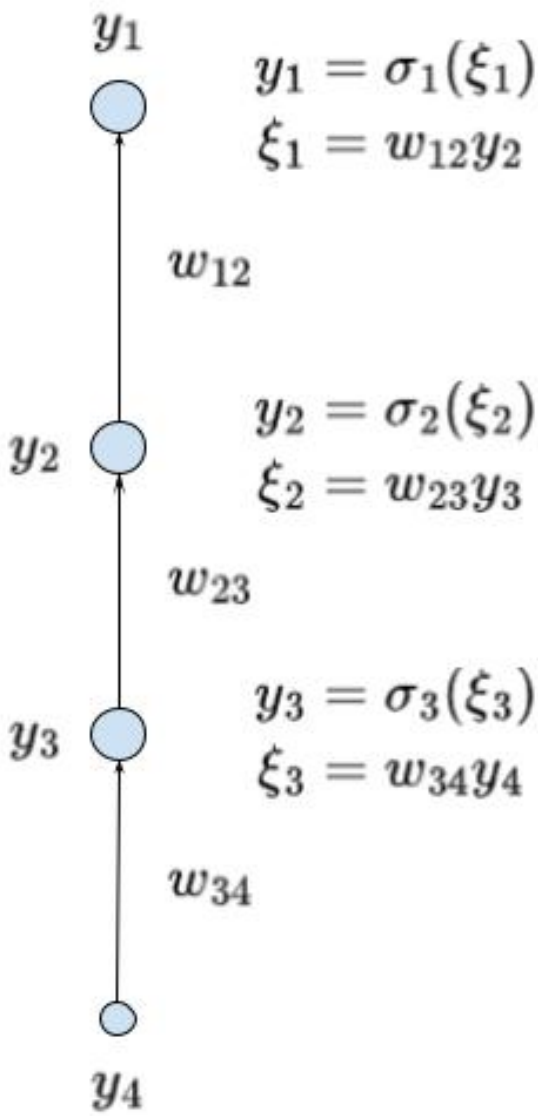
$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_3$$

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) y_3$$

$$\frac{\partial E}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3) y_4$$

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) w_{12}$$

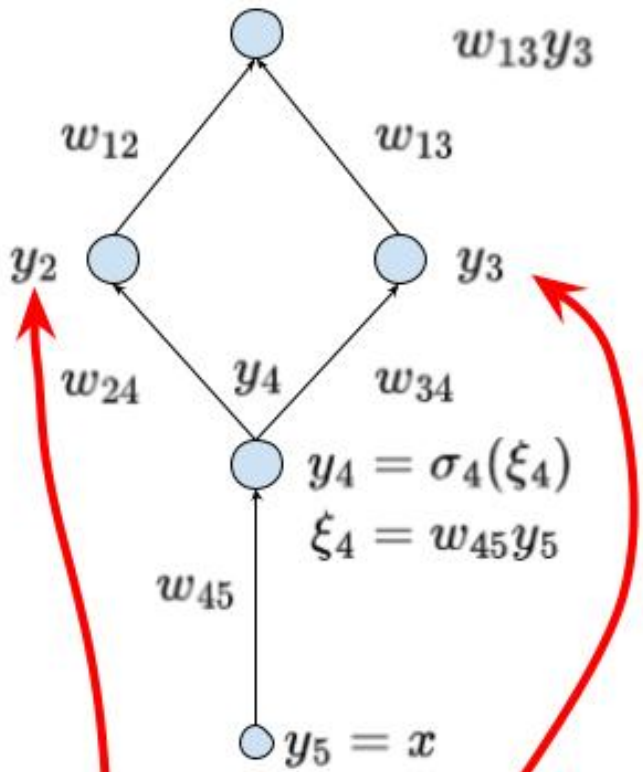
$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial y_3} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) w_{23}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

$$y_1 = \sigma_1(\xi_1)$$

$$\xi_1 = w_{12}y_2 + w_{13}y_3$$



$$y_2 = \sigma_2(\xi_2)$$

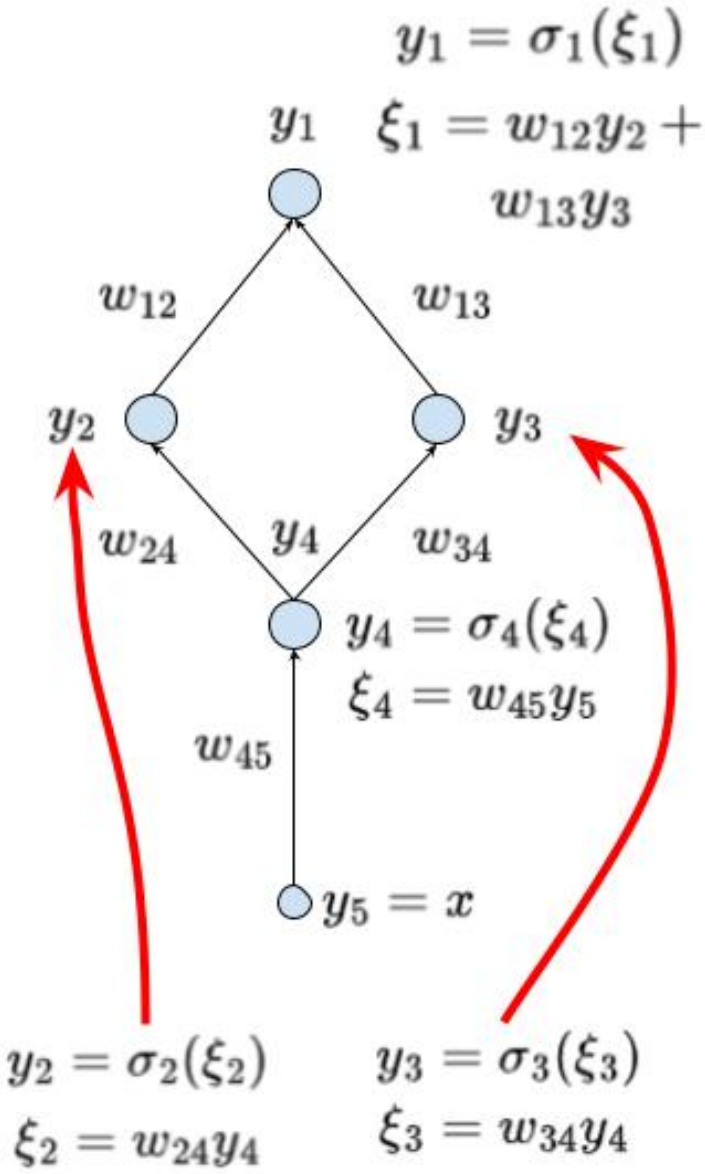
$$\xi_2 = w_{24}y_4$$

$$y_3 = \sigma_3(\xi_3)$$

$$\xi_3 = w_{34}y_4$$

$$D = \{(x, d)\}$$

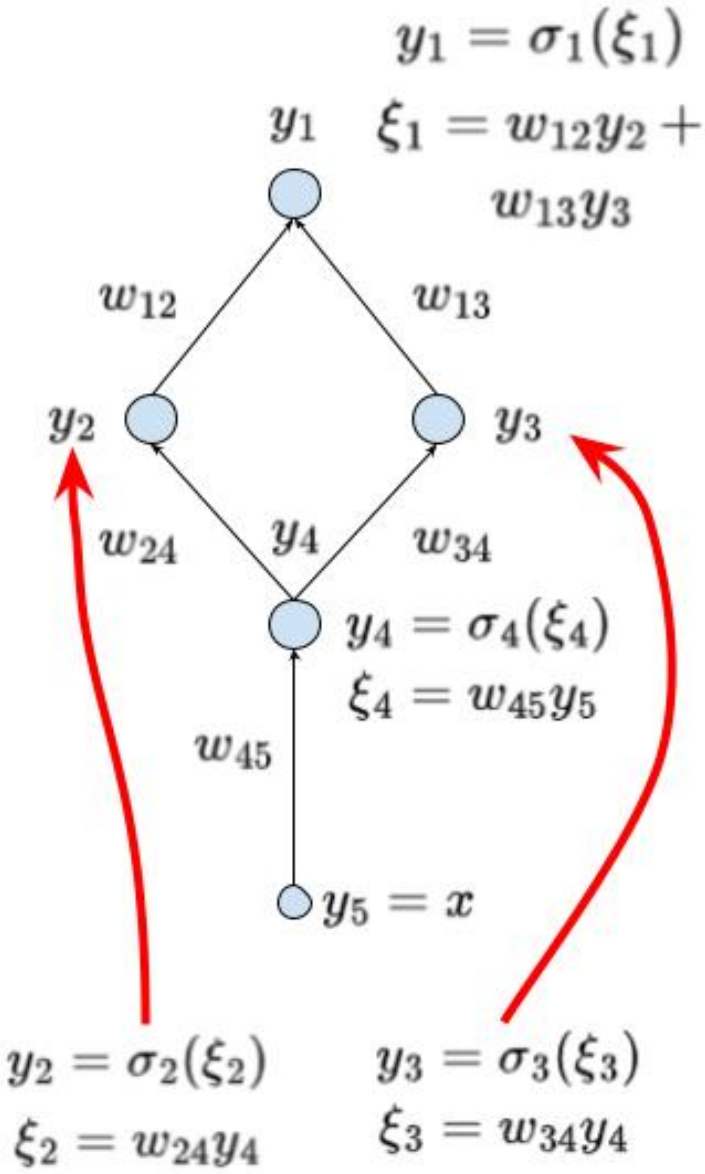
$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

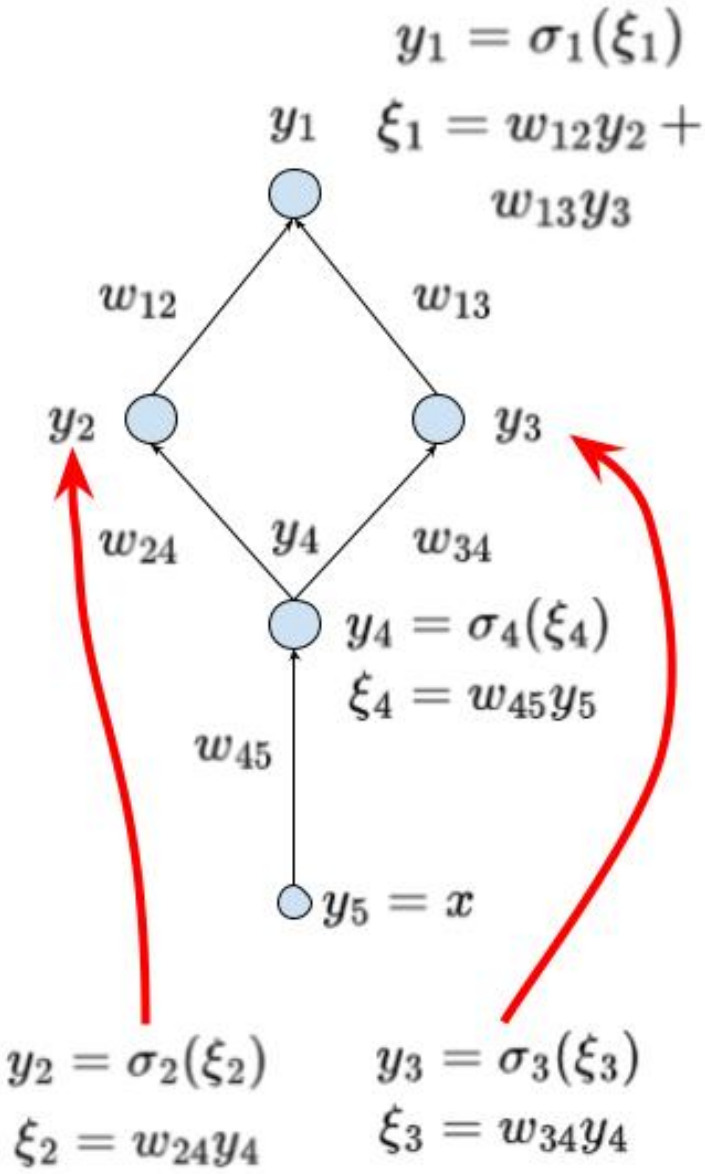


$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\sigma_2'(\xi_2)y_3$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



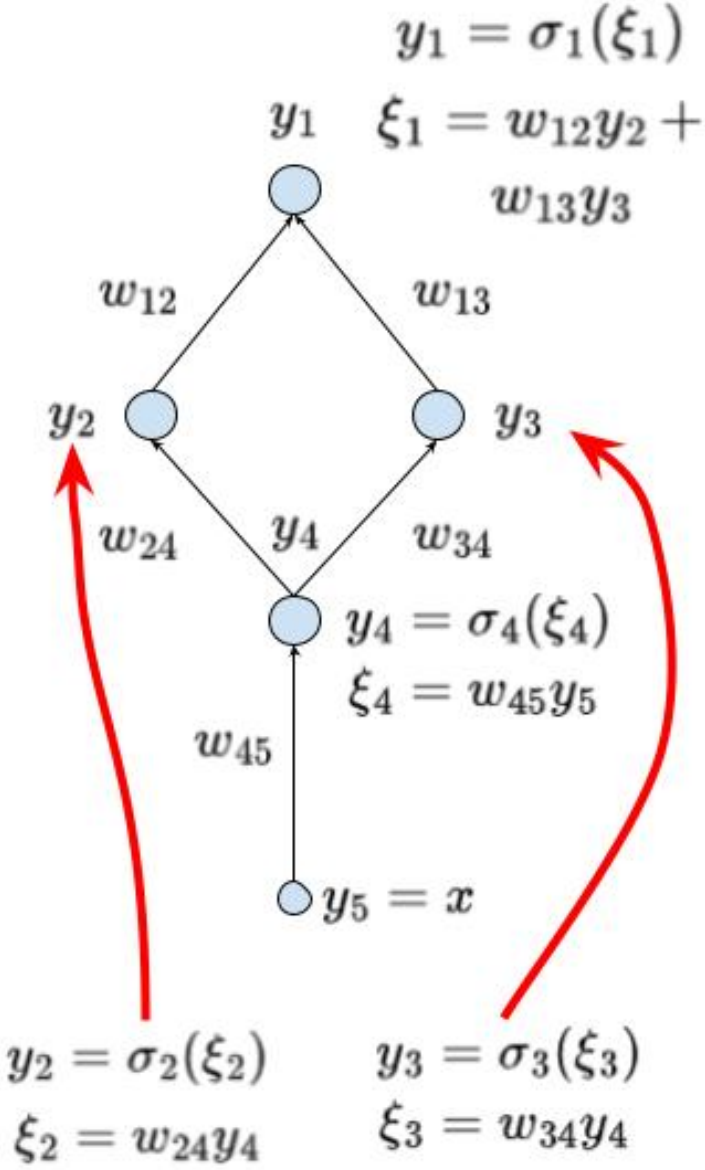
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\sigma_2'(\xi_2)y_3$$

$$\frac{\partial E}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_4$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

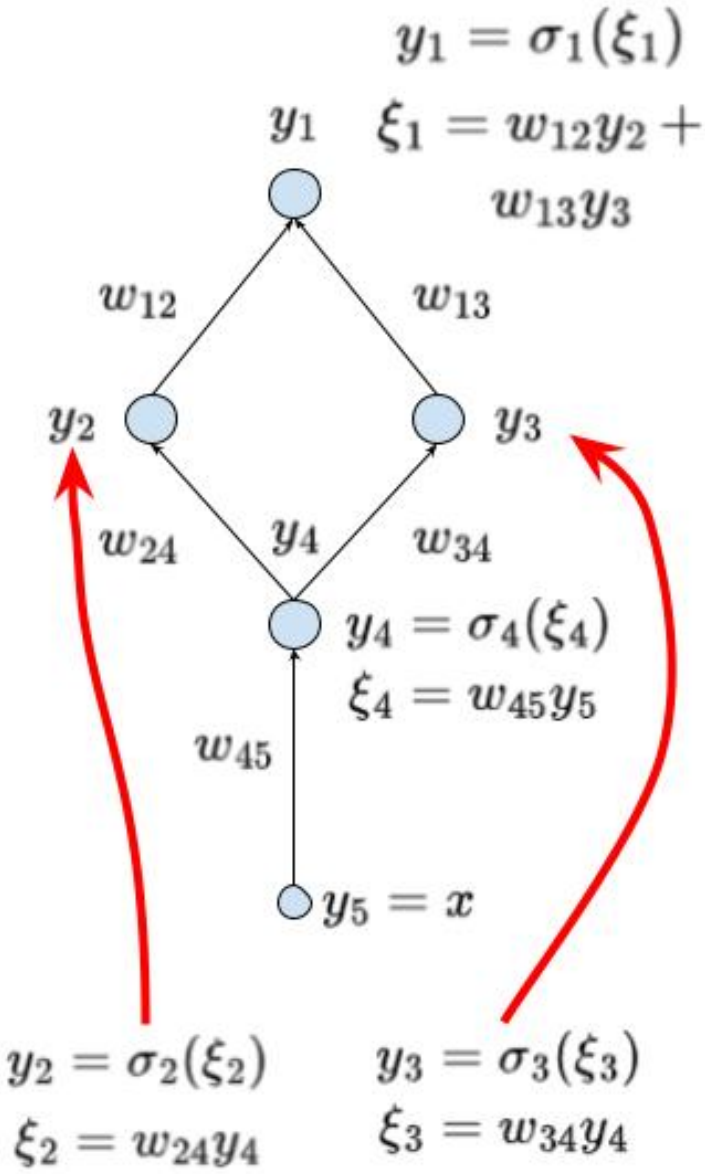
$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\sigma_2'(\xi_2)y_3$$

$$\frac{\partial E}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_4$$

$$\frac{\partial E}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3)y_4$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\sigma_2'(\xi_2)y_3$$

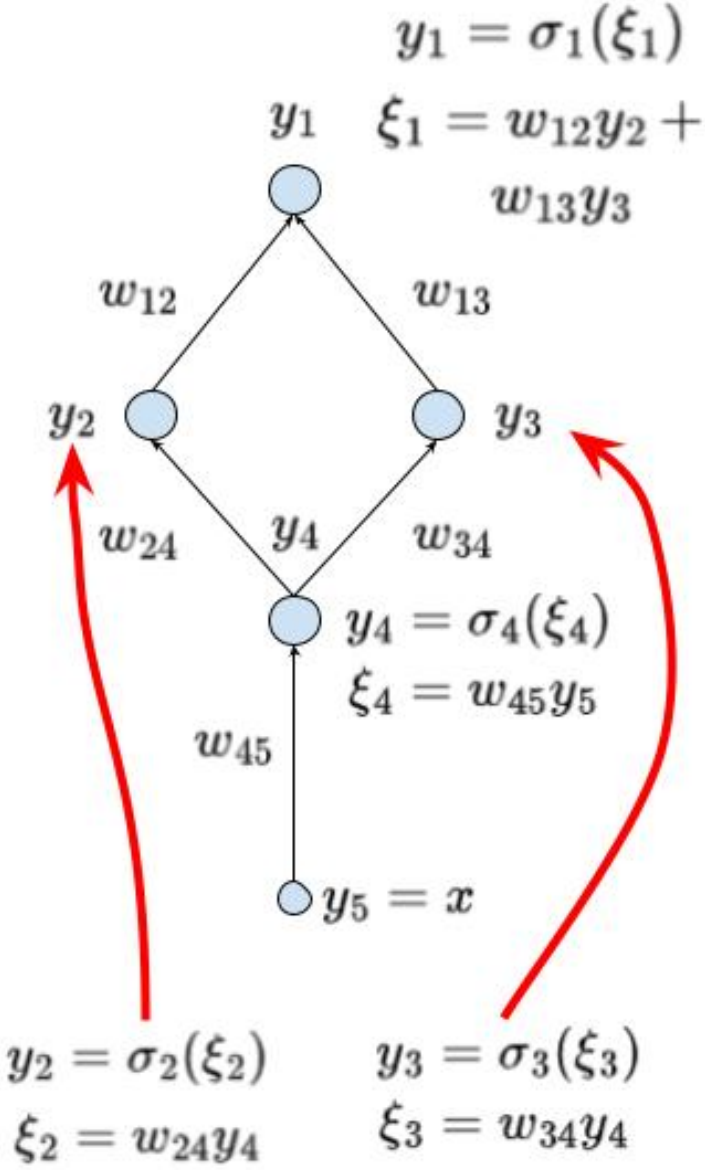
$$\frac{\partial E}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_4$$

$$\frac{\partial E}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3)y_4$$

$$\frac{\partial E}{\partial w_{45}} = \frac{\partial E}{\partial y_4}\frac{\partial y_4}{\partial \xi_4}\frac{\partial \xi_3}{\partial w_{45}} = \frac{\partial E}{\partial y_4}\sigma_4'(\xi_4)y_5$$

$$D = \{(x, d)\}$$

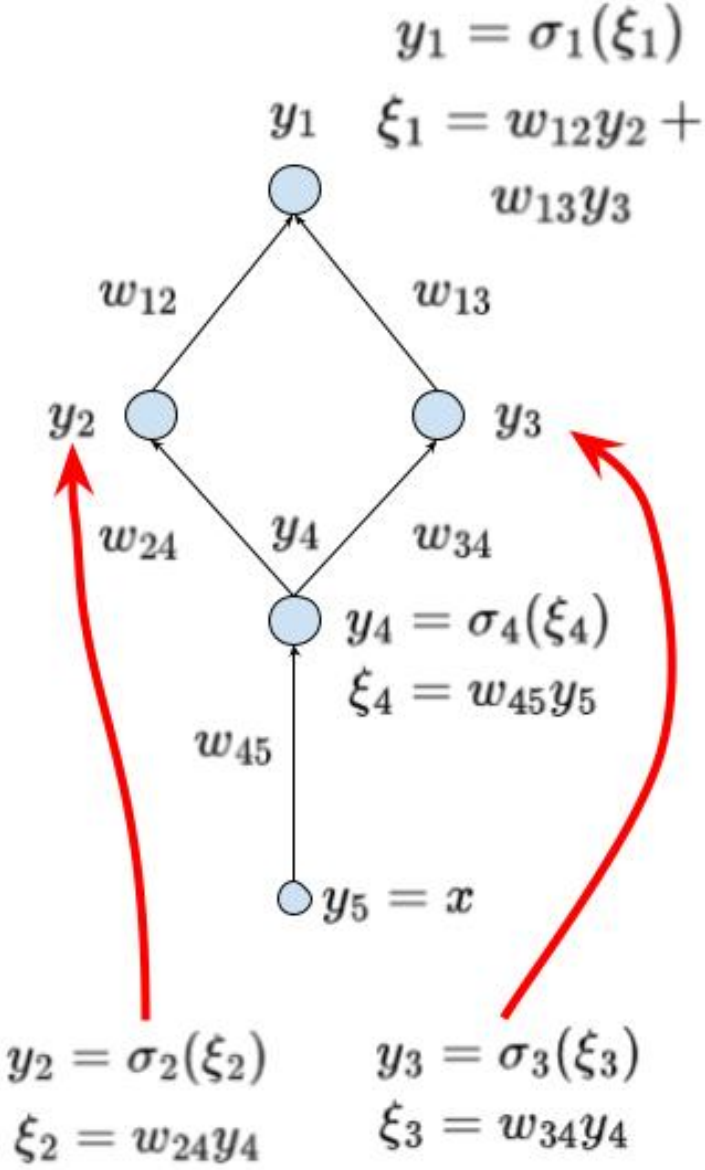
$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

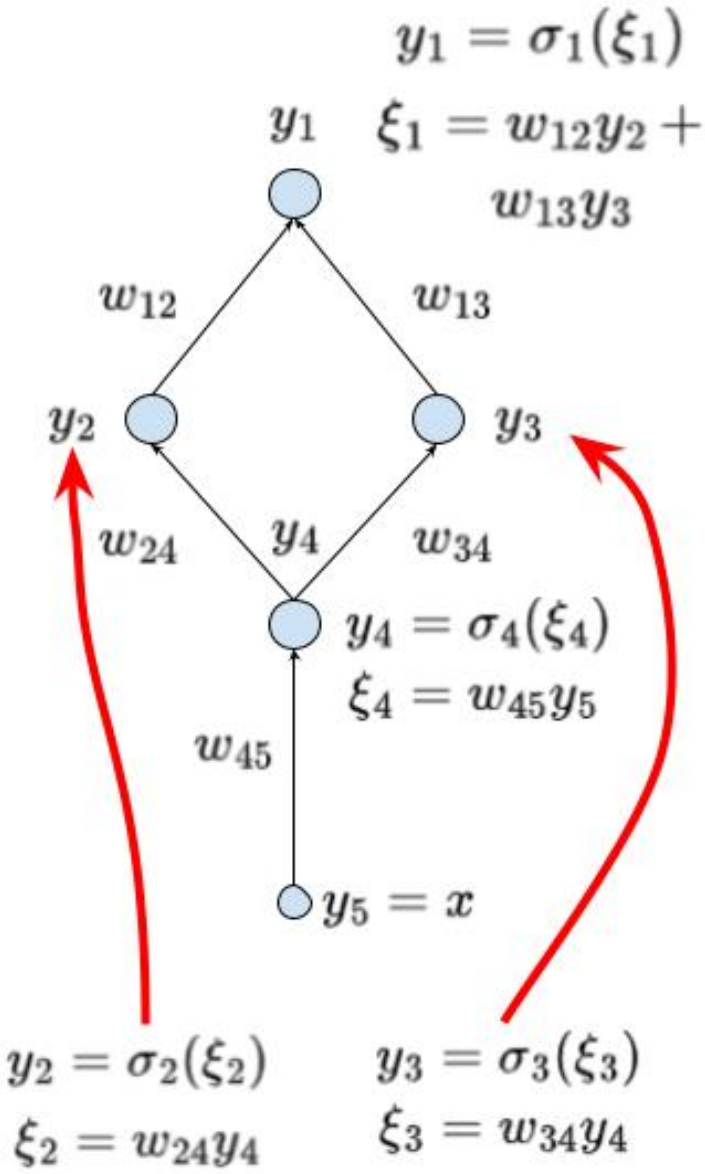


$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



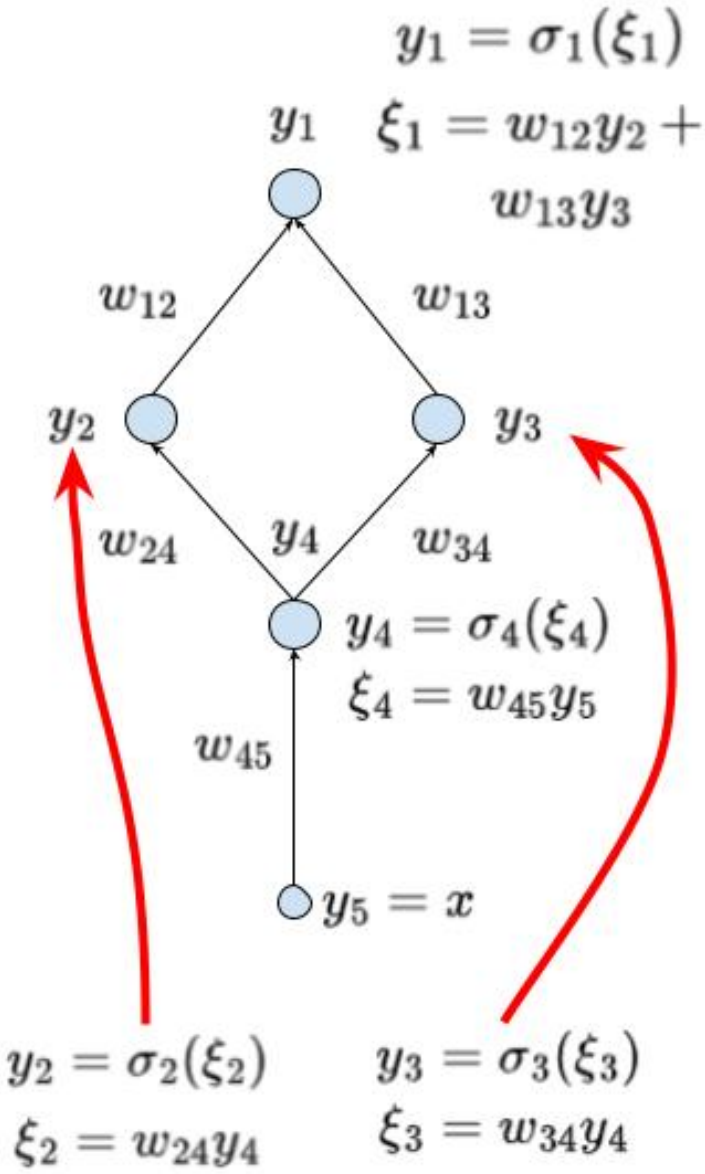
$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



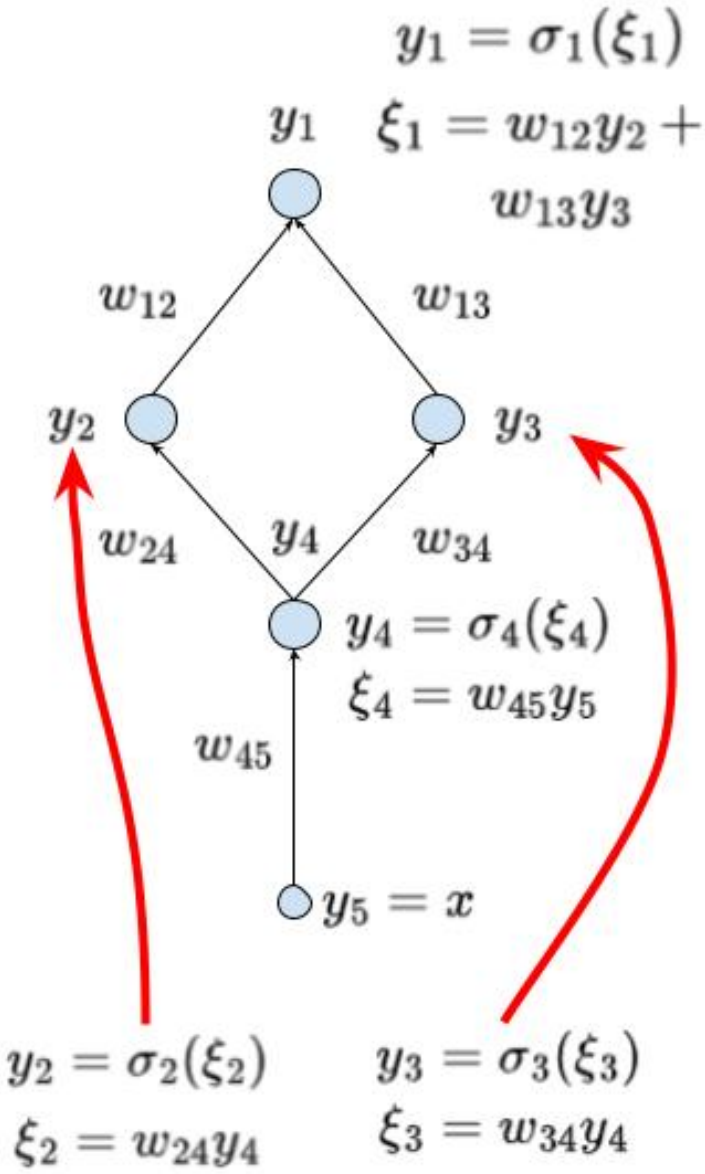
$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial y_1} = y_1 - d$$

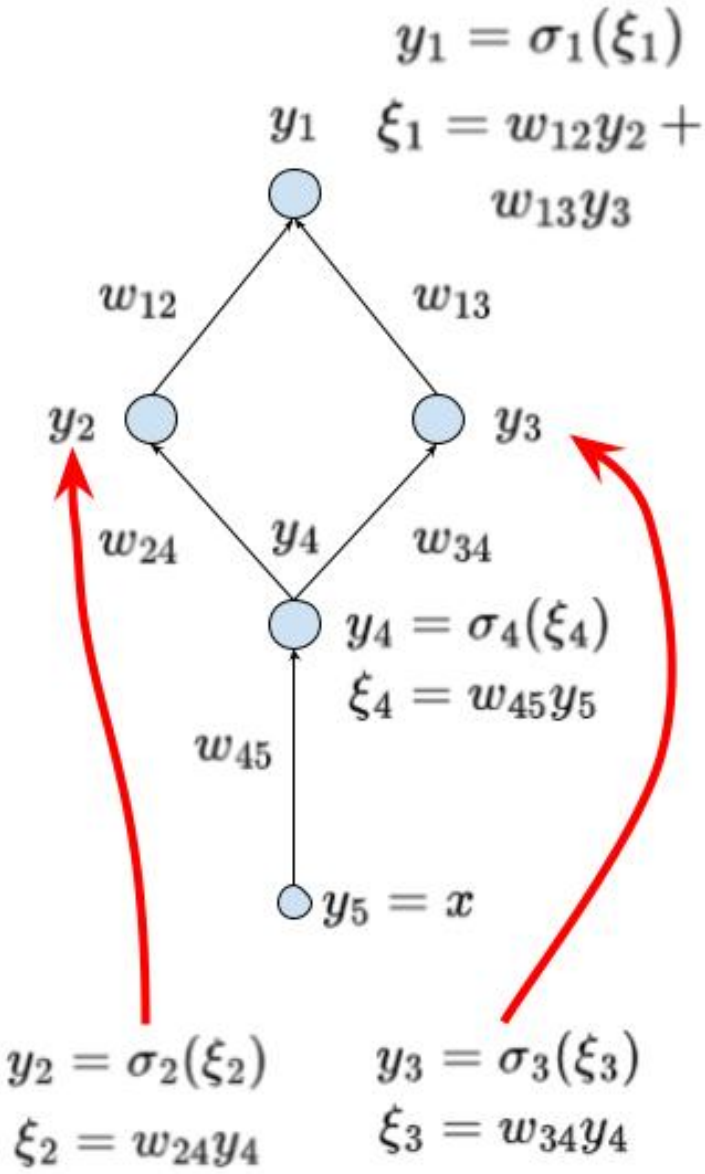
$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\frac{\partial E}{\partial y_4}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial y_1} = y_1 - d$$

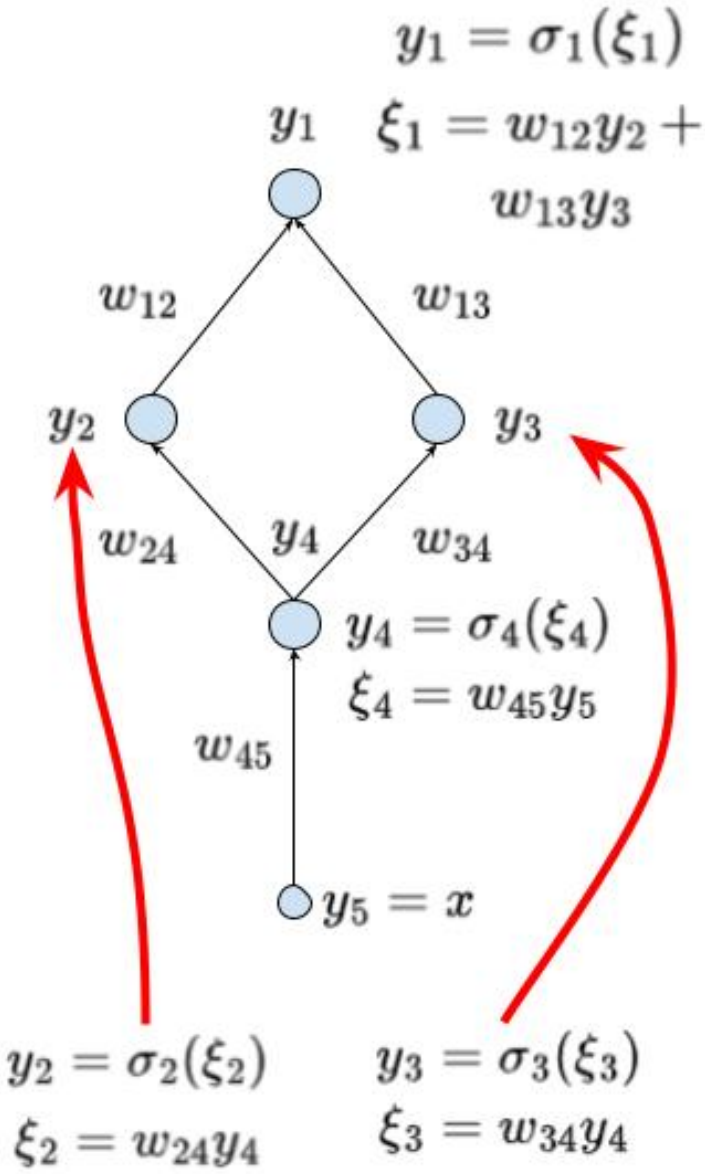
$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\frac{\partial E}{\partial y_4} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_4}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

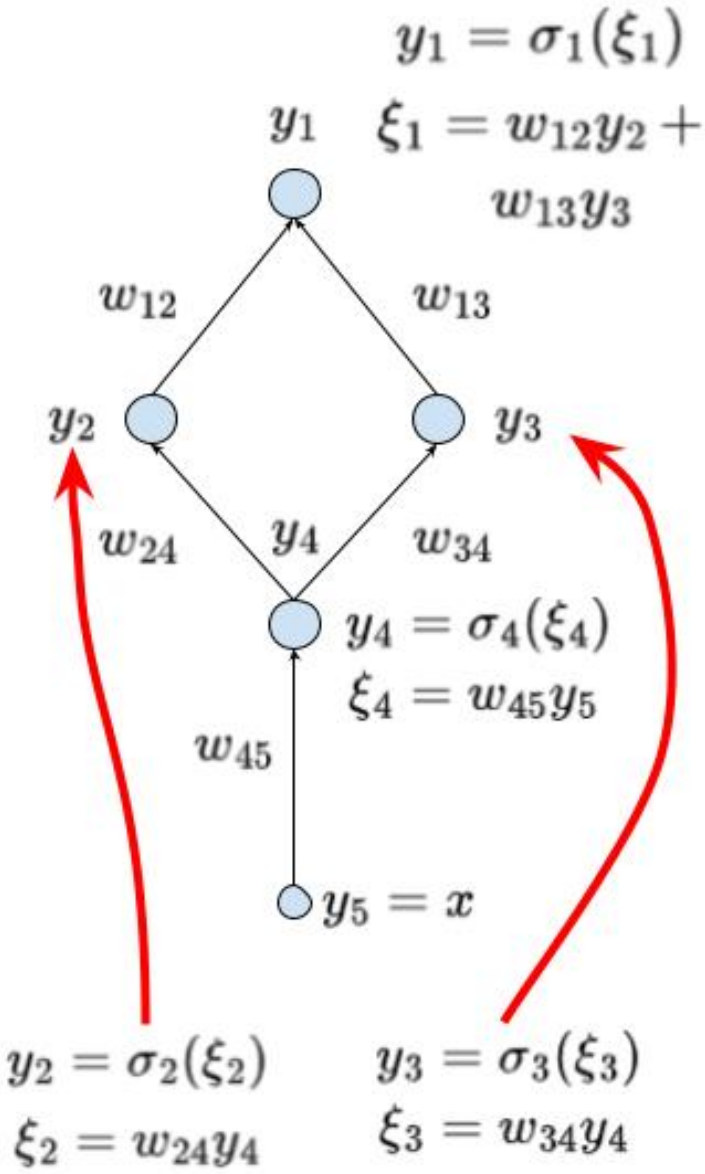
$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\frac{\partial E}{\partial y_4} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_4}$$

$$= \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial y_4}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\begin{aligned}\frac{\partial E}{\partial y_4} &= \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_4} \\ &= \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial y_4} \\ &= \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)w_{24} + \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3)w_{34}\end{aligned}$$

MLP – Gradient Computation

Under our simplifying assumptions $D = \{(x, d)\}$ and $E = (y_1 - d)^2$ the gradient computaton proceeds as follows:

Applying the chain rule, we obtain

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$

where (after more applications of the chain rule)

$$\frac{\partial E_k}{\partial y_1} = y_1 - d$$

Keep in mind that 1 is the only output neuron which means that y_1 is the value of the network.

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj} \qquad \text{for } j \in Z \setminus (Y \cup X)$$

Here $y_r = y[\vec{w}](x)$ where $\vec{w}$ are the current weights and x is the example input.

MLP – Gradient Computation – General!

Let us drop our simplifying assumptions!

- Given a set D of training examples:

$$D = \left\{ \left(\vec{x}_k, \vec{d}_k \right) \quad | \quad k = 1, \ldots, p \right\}$$

Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write d_{kj} to denote the value in $\vec{d}_k$ corresponding to the output neuron j.

MLP – Gradient Computation – General!

Let us drop our simplifying assumptions!

- Given a set D of training examples:

$$D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \quad | \quad k = 1, \ldots, p \right\}$$

  Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write d_{kj} to denote the value in $\vec{d}_k$ corresponding to the output neuron j.

- **Error Function:** $E(\vec{w})$ where $\vec{w}$ is a vector of all weights in the network. The choice of E depends on the solved task (classification vs regression, etc.).
  Example (Squared error):

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

  where

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} (y_j[\vec{w}](\vec{x}_k) - d_{kj})^2$$

MLP – Gradient Computation

For every weight w_{ji} we have (obviously)

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$$

So now it suffices to compute $\frac{\partial E_k}{\partial w_{ji}}$, that is the error for a fixed training example $(\vec{x}_k, \vec{d}_k)$.

MLP – Gradient Computation

For every weight w_{ji} we have (obviously)

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$$

So now it suffices to compute $\frac{\partial E_k}{\partial w_{ji}}$, that is the error for a fixed training example $(\vec{x}_k, \vec{d}_k)$.

Applying the chain rule, we obtain

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$

where (more applications of the chain rule)

$\frac{\partial E_k}{\partial y_j}$ is computed directly for the output neurons $j \in Y$

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj} \qquad \text{for } j \in Z \setminus (Y \cup X)$$

(Here $y_r = y[\vec{w}](\vec{x}_k)$ where $\vec{w}$ are the current weights and $\vec{x}_k$ is the input of the k-th training example.)

MLP – Backpropagation

Input: A training set $D = \left\{ \left(\vec{x}_k, \vec{d}_k \right) \quad | \quad k = 1, \ldots, p \right\}$ and the current vector of weights $\vec{w}$.

Note that the backprop. is repeated in every iteration of the gradient descent!

- Evaluate all values y_i of neurons using the standard bottom-up procedure with the input $\vec{x}_k$.

MLP – Backpropagation

Input: A training set $D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \quad | \quad k = 1, \ldots, p \right\}$ and the current vector of weights $\vec{w}$.

Note that the backprop. is repeated in every iteration of the gradient descent!

- Evaluate all values y_i of neurons using the standard bottom-up procedure with the input $\vec{x}_k$.
- For every training example $(\vec{x}_k, \vec{d}_k)$ compute $\frac{\partial E_k}{\partial y_j}$ using *backpropagation* through layers top-down :
  - For all $j \in Y$ compute $\frac{\partial E_k}{\partial y_j}$ by taking the derivative of the error.

    e.g., in the case of the squared error, we have $\frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$.

MLP – Backpropagation

Input: A training set $D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \quad | \quad k = 1, \ldots, p \right\}$ and the current vector of weights $\vec{w}$.

Note that the backprop. is repeated in every iteration of the gradient descent!

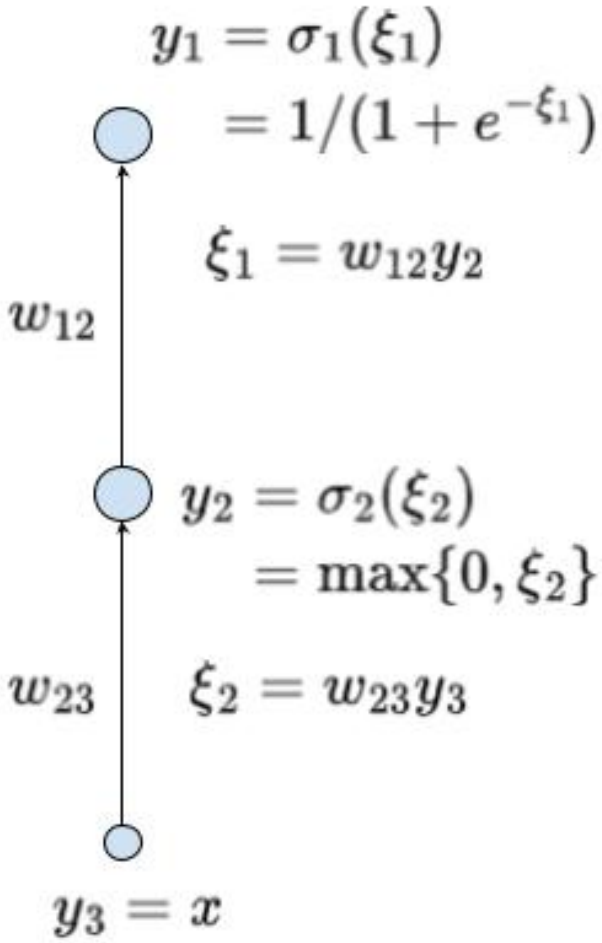
- Evaluate all values y_i of neurons using the standard bottom-up procedure with the input $\vec{x}_k$.
- For every training example $(\vec{x}_k, \vec{d}_k)$ compute $\frac{\partial E_k}{\partial y_j}$ using *backpropagation* through layers top-down :
  - For all $j \in Y$ compute $\frac{\partial E_k}{\partial y_j}$ by taking the derivative of the error.
    e.g., in the case of the squared error, we have $\frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$.
  - In the layer ℓ, assuming that $\frac{\partial E_k}{\partial y_r}$ has been computed for all neurons r in the layer $\ell + 1$, compute

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$

    for all j from the ℓ-th layer. Here σ_r' is the derivative of σ_r.
  - Put $\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$

Output: $\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$.

MLP Learning Example



Training set:

$$D = \{(x, d)\} = \{(1, 1)\}$$

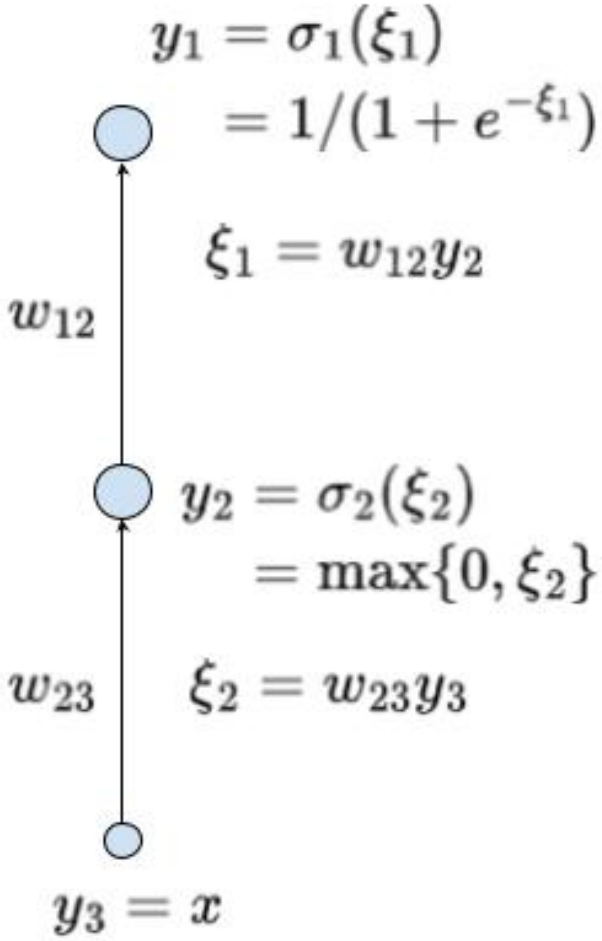
That is

$$y_3 = x = 1$$
$$d = 1$$

Error cross-entropy:

$$E(\vec{w}) = -(d \log(y_1) + (1 - d) \log(1 - y_1))$$
$$= -\log(y_1)$$

MLP Learning Example



Training set:

$$D = \{(x, d)\} = \{(1, 1)\}$$

That is

$$y_3 = x = 1$$
$$d = 1$$

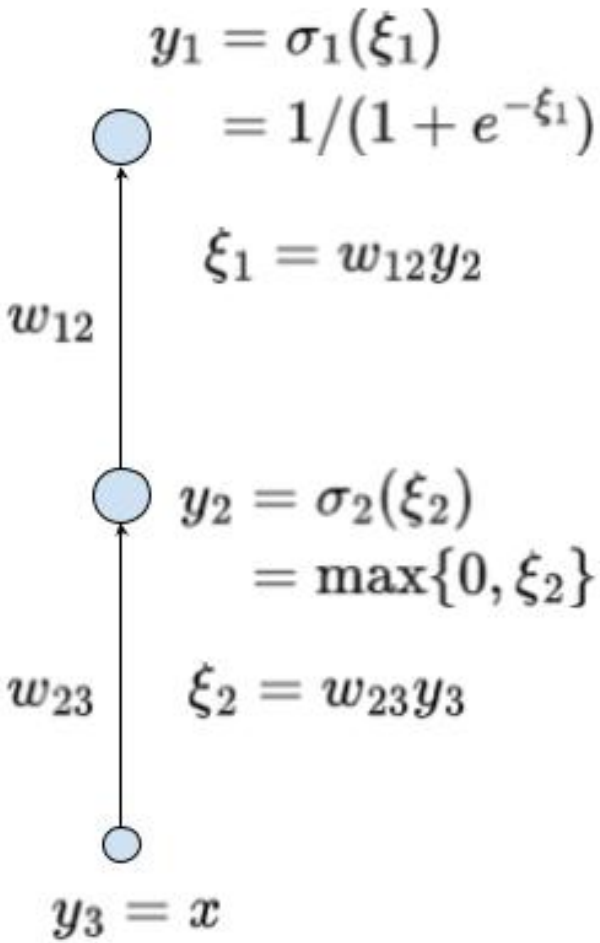
Error cross-entropy:

$$E(\vec{w}) = -(d \log(y_1) + (1 - d) \log(1 - y_1))$$
$$= -\log(y_1)$$

Assume the initial weight vector
$\vec{w}^{(0)} = (w_{12}^{(0)}, w_{23}^{(0)}) = (\frac{1}{4}, 2)$.

Consider the learning rate $\varepsilon = 0.5$.

MLP Learning Example – Gradient Descent



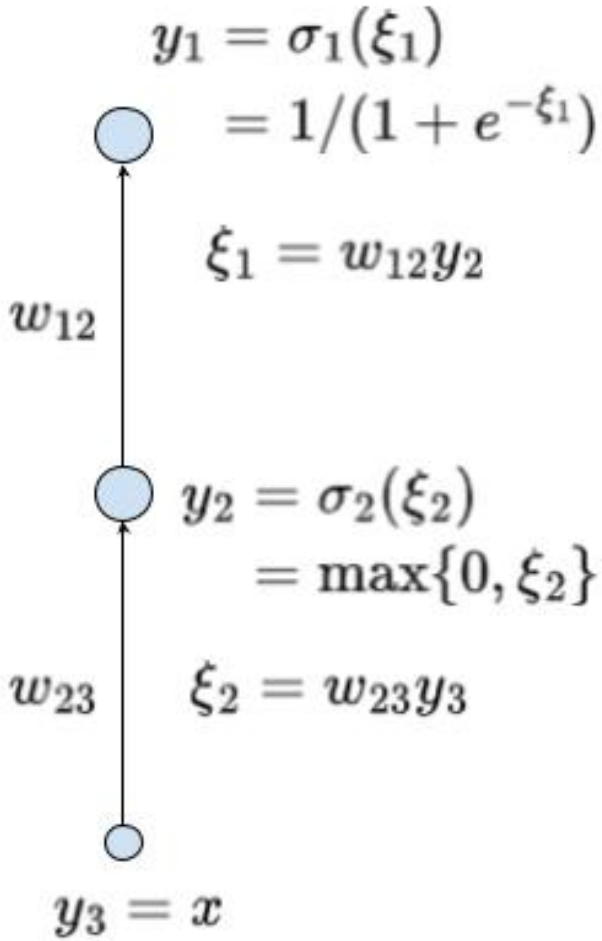
To make the gradient descent step:

$$w_{12}^{(1)} = w_{12}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{12}}(\vec{w}^{(0)})$$

$$w_{23}^{(1)} = w_{23}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{23}}(\vec{w}^{(0)})$$

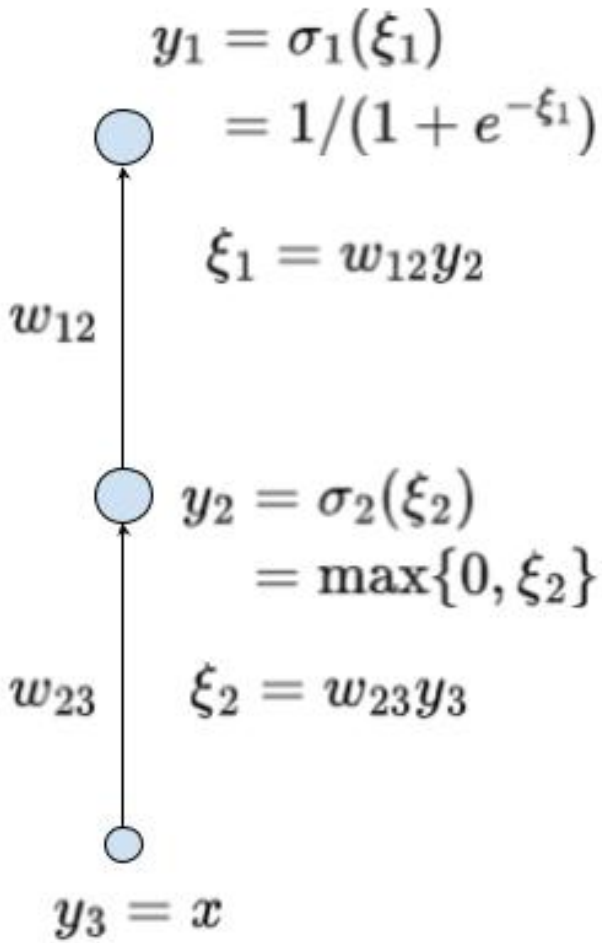
we need to compute the partial derivatives $\frac{\partial E}{\partial w_{12}}$ and $\frac{\partial E}{\partial w_{23}}$.

MLP Learning Example – Forward Pass



We have $x = 1$, $w_{12}^{(0)} = 1/4$, $w_{23}^{(0)} = 2$

MLP Learning Example – Forward Pass



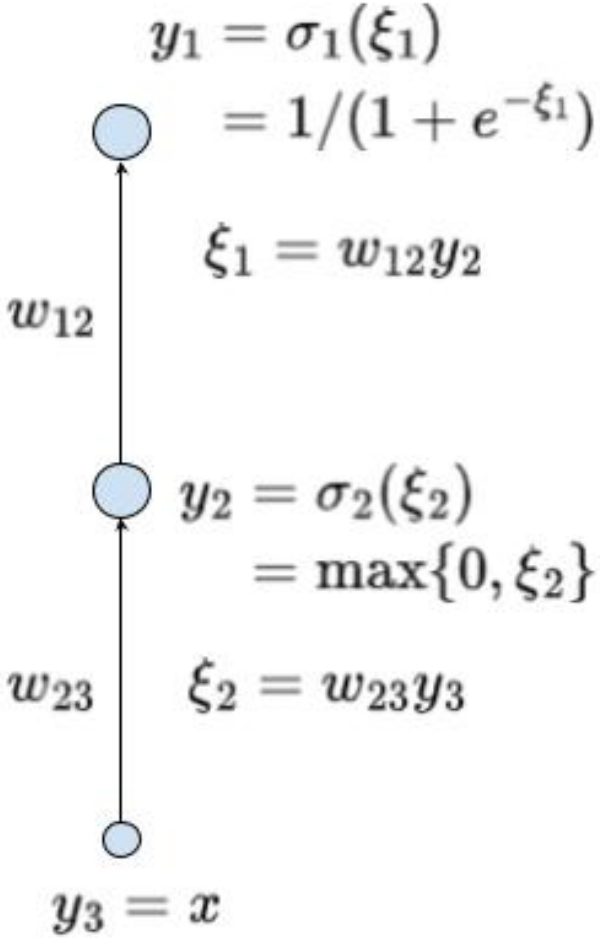
We have $x = 1, w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2$

First, compute the **forward pass**

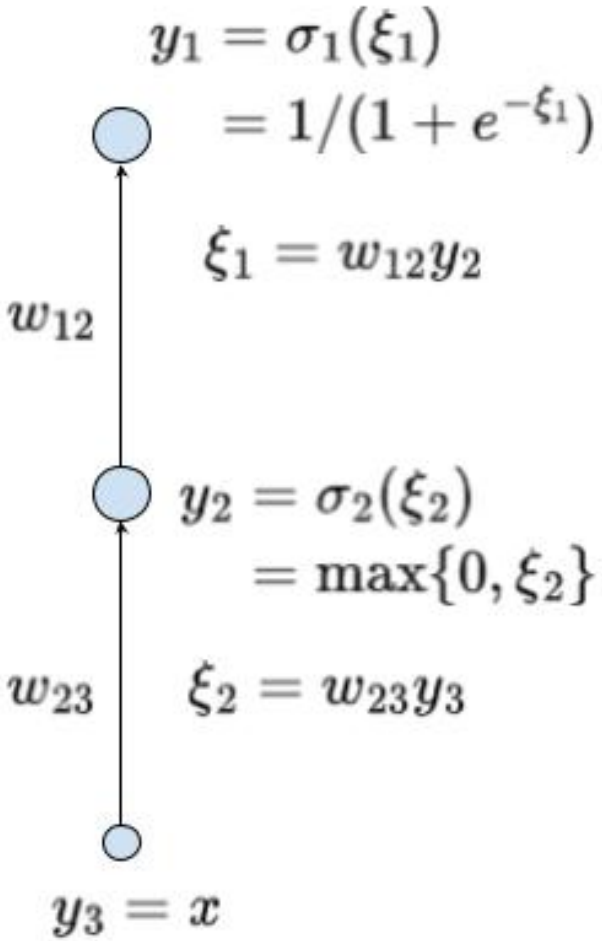
$$y_3 = x = 1$$
$$\xi_2 = w_{23}^{(0)} y_3 = 2y_3 = 2$$
$$y_2 = \max\{0, \xi_2\} = 2$$
$$\xi_1 = w_{12}^{(0)} y_2 = \frac{1}{4} 2 = \frac{1}{2}$$
$$y_1 = 1/(1+e^{-\xi_1}) = 1/(1+e^{-(1/2)})$$
$$= 0.6225$$

MLP Learning Example – Backward Pass

We have $w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1$.



MLP Learning Example – Backward Pass

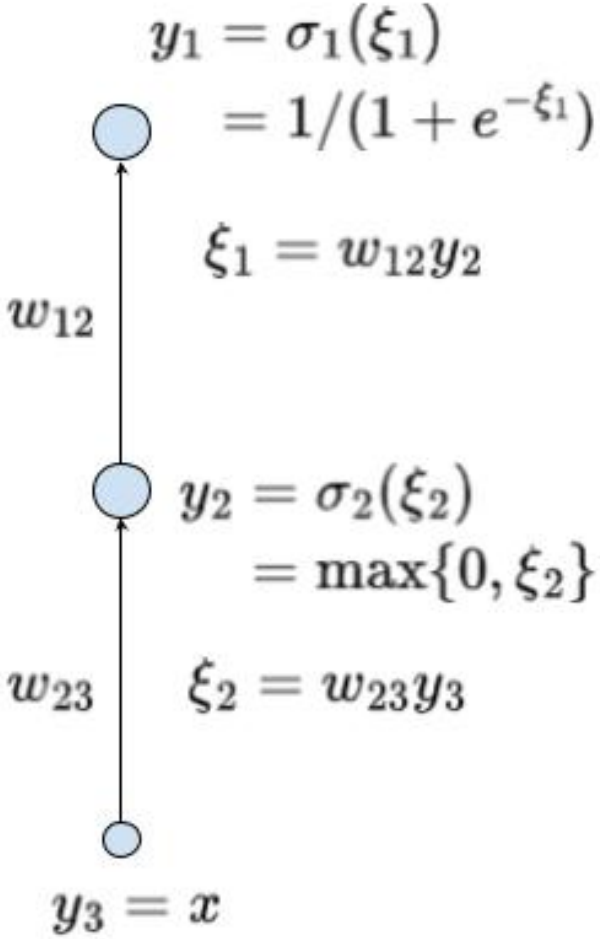


We have $w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1.$

Proceed with the backward pass:

$$\frac{\partial E}{\partial y_1} = \frac{\partial(-\log(y_1))}{\partial y_1} = -\frac{1}{y_1} = -1.6065$$

MLP Learning Example – Backward Pass



We have $w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1$.

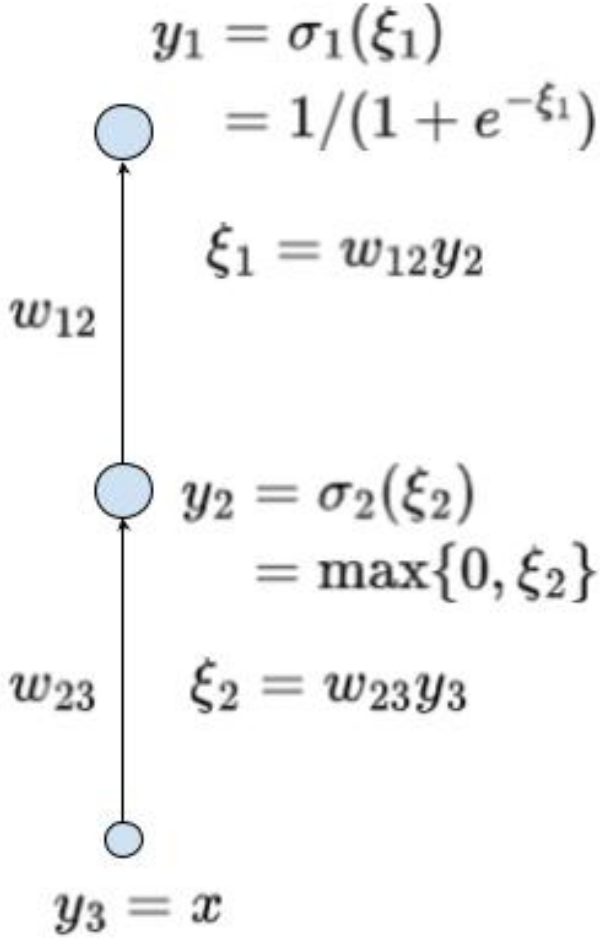
Proceed with the backward pass:

$$\frac{\partial E}{\partial y_1} = \frac{\partial(-\log(y_1))}{\partial y_1} = -\frac{1}{y_1} = -1.6065$$

Since $\sigma_1' = \sigma_1(1-\sigma_1)$

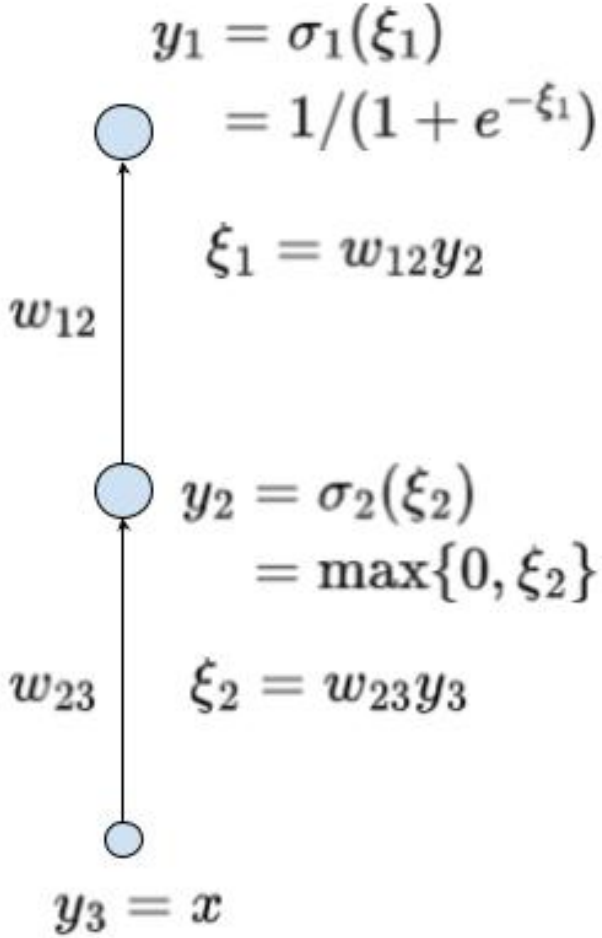
$$\begin{aligned}
\frac{\partial E}{\partial y_2} &= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}^{(0)} \\
&= \frac{\partial E}{\partial y_1}\sigma_1(\xi_1)(1-\sigma_1(\xi_1))w_{12}^{(0)} \\
&= \frac{\partial E}{\partial y_1}y_1(1-y_1)w_{12}^{(0)} \\
&= -1.6065 \cdot 0.6225 \cdot 0.3775 \cdot (1/4) \\
&= -0.09438
\end{aligned}$$

MLP Learning Example – The Gradient



We have
$w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1, \frac{\partial E}{\partial y_1} = -1.6065, \frac{\partial E}{\partial y_2} = -0.09438.$

MLP Learning Example – The Gradient

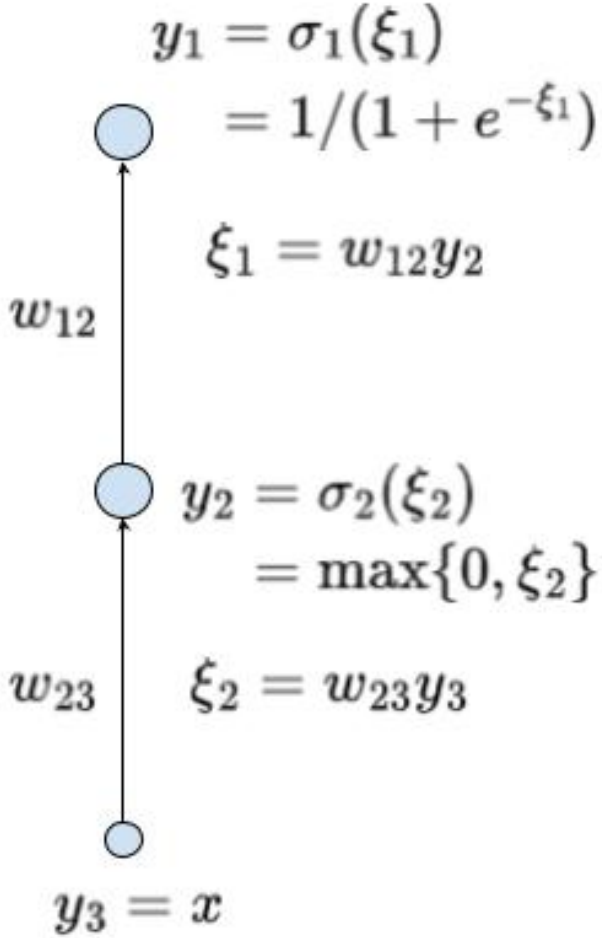


We have
$w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1, \frac{\partial E}{\partial y_1} = -1.6065, \frac{\partial E}{\partial y_2} = -0.09438.$

Compute derivatives of E w.r.t. weights:

$$\begin{aligned}\frac{\partial E}{\partial w_{12}} &= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2 \\ &= \frac{\partial E}{\partial y_1} y_1(1-y_1) y_2 \\ &= -0.755\end{aligned}$$

MLP Learning Example – The Gradient



We have
$w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1, \frac{\partial E}{\partial y_1} = -1.6065, \frac{\partial E}{\partial y_2} = -0.09438.$

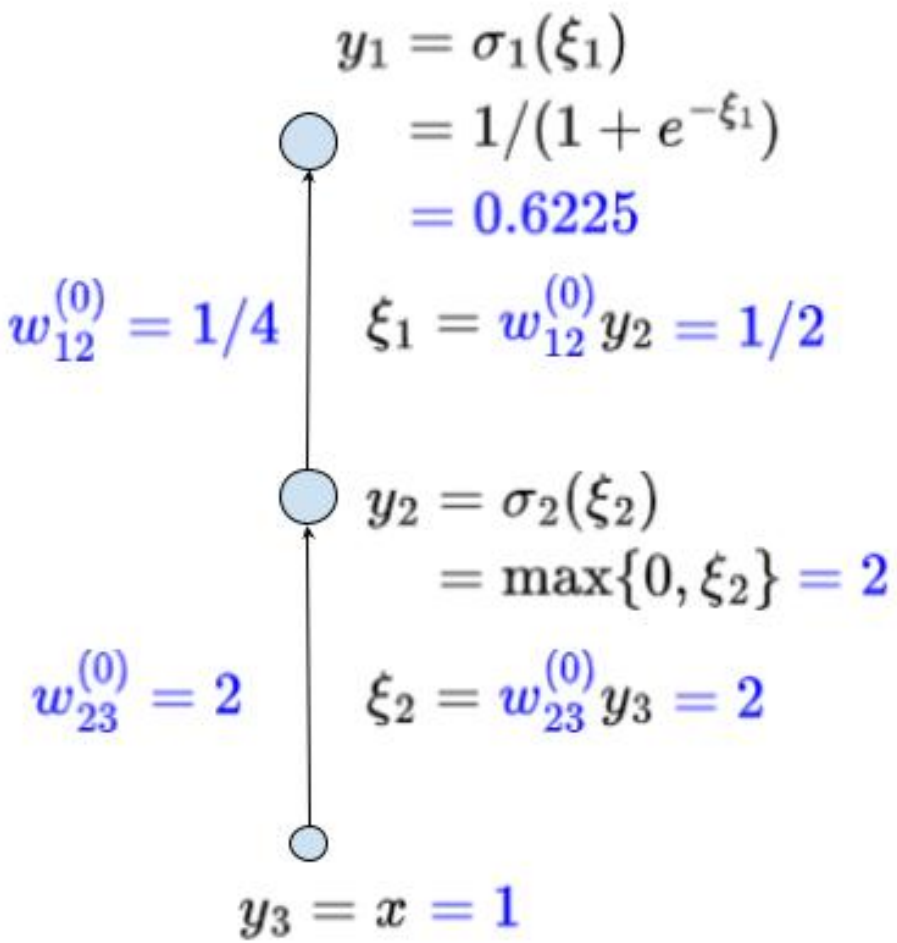
Compute derivatives of E w.r.t. weights:

$$\begin{aligned}\frac{\partial E}{\partial w_{12}} &= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2 \\ &= \frac{\partial E}{\partial y_1} y_1(1-y_1)y_2 \\ &= -0.755\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{23}} &= \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_3 \\ &= \frac{\partial E}{\partial y_2} y_3 \\ &= -0.09438\end{aligned}$$

Backpropagation – Example – Summary

Forward pass (bottom up)



Backpropagation – Example – Summary

Forward pass (bottom up)

$$y_1 = \sigma_1(\xi_1) = 1/(1+e^{-\xi_1}) = 0.6225$$

$$w_{12}^{(0)} = 1/4 \qquad \xi_1 = w_{12}^{(0)} y_2 = 1/2$$

$$y_2 = \sigma_2(\xi_2) = \max\{0, \xi_2\} = 2$$

$$w_{23}^{(0)} = 2 \qquad \xi_2 = w_{23}^{(0)} y_3 = 2$$

$$y_3 = x = 1$$

Backward pass (top down)

$$\frac{\partial E}{\partial y_1} = 1/y_1 = -1.6065$$

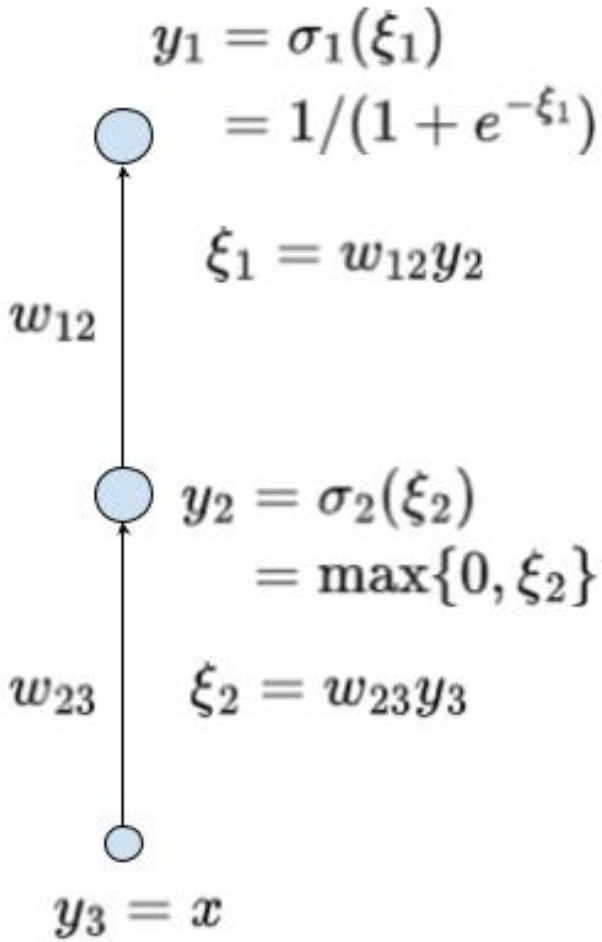
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1} y_1(1-y_1) y_2 = -0.755$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1} y_1(1-y_1) w_{12}^{(0)} = -0.09438$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2} 1 y_3 = -0.09438$$

Note that **WE HAVE NOT YET CHANGED ANY WEIGHTS!**

MLP Learning Example – Gradient Descent Step

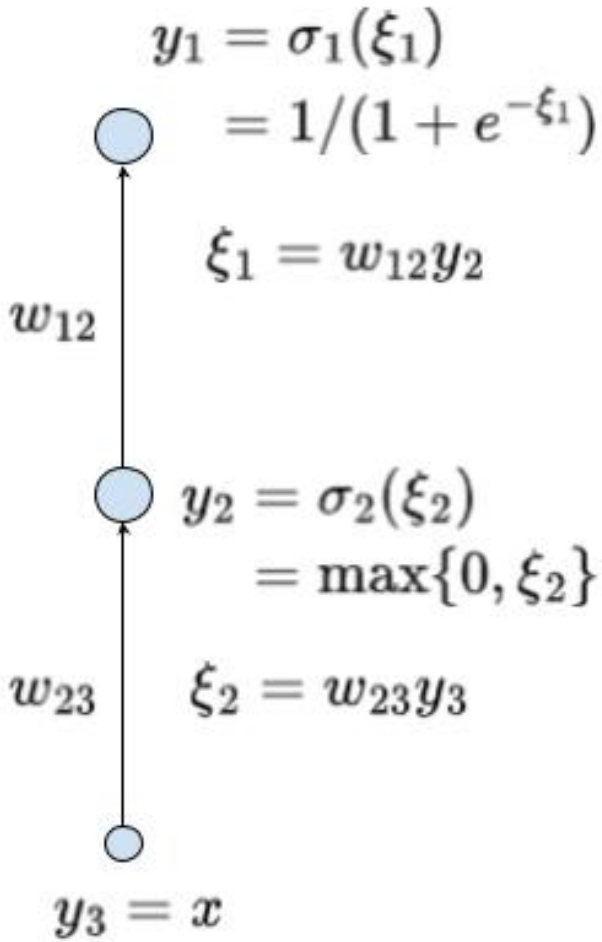


So **ONLY NOW** we can make the **learning step** and change the weights:

$$
\begin{aligned}
w_{12}^{(1)} &= w_{12}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{12}}(\vec{w}^{(0)}) \\
&= \frac{1}{4} - 0.5 \cdot (-0.755) \\
&= 0.627
\end{aligned}
$$

$$
\begin{aligned}
w_{23}^{(1)} &= w_{23}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{23}}(\vec{w}^{(0)}) \\
&= 2 - 0.5 \cdot (-0.09438) \\
&= 2.047
\end{aligned}
$$

MLP Learning Example – Gradient Descent Step



So **ONLY NOW** we can make the **learning step** and change the weights:

$$
\begin{aligned}
w_{12}^{(1)} &= w_{12}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{12}}(\vec{w}^{(0)}) \\
&= \frac{1}{4} - 0.5 \cdot (-0.755) \\
&= 0.627
\end{aligned}
$$

$$
\begin{aligned}
w_{23}^{(1)} &= w_{23}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{23}}(\vec{w}^{(0)}) \\
&= 2 - 0.5 \cdot (-0.09438) \\
&= 2.047
\end{aligned}
$$

We have made just a **single** gradient descent step!

MLP Training Summary

- MLP are trained using the **gradient descent** algorithm

  In practice, modifications of GD are typically used, but most of them have strong roots in GD.

MLP Training Summary

- MLP are trained using the **gradient descent** algorithm

  In practice, modifications of GD are typically used, but most of them have strong roots in GD.
- The gradients are computed using the backpropagation algorithm

  the backpropagation is a universal method for automatic differentiation, surpassing any use in neural networks.

MLP Training Summary

- MLP are trained using the **gradient descent** algorithm

  In practice, modifications of GD are typically used, but most of them have strong roots in GD.
- The gradients are computed using the backpropagation algorithm

  the backpropagation is a universal method for automatic differentiation, surpassing any use in neural networks.
- Training of neural networks in practice is tricky due to many reasons comprising in particular
  - highly complex non-linear shape of the error function,
  - tendency to overfit very quickly,
  - black-box nature, hard to see what the network does,
  - huge hype around deep learning (which many people confuse with AI) results in high expectations even in cases where no (learning) algorithm may solve the given problem!

MLP Training Summary

- MLP are trained using the **gradient descent** algorithm

  In practice, modifications of GD are typically used, but most of them have strong roots in GD.
- The gradients are computed using the backpropagation algorithm

  the backpropagation is a universal method for automatic differentiation, surpassing any use in neural networks.
- Training of neural networks in practice is tricky due to many reasons comprising in particular
  - highly complex non-linear shape of the error function,
  - tendency to overfit very quickly,
  - black-box nature, hard to see what the network does,
  - huge hype around deep learning (which many people confuse with AI) results in high expectations even in cases where no (learning) algorithm may solve the given problem!

An advice: Always concentrate the main effort on the solved problem formulation and, afterward, on the data you have at your disposal (and honestly separate the Test set right at the beginning).

Deep Learning

- Cybenko's theorem shows that two-layer networks are omnipotent – such results nearly killed NN when support vector machines were found to be easier to train in 00's.

Deep Learning

- Cybenko's theorem shows that two-layer networks are omnipotent – such results nearly killed NN when support vector machines were found to be easier to train in 00's.
- Later, it has been shown (experimentally) that deep networks (with many layers) have better representational properties.

Deep Learning

- Cybenko's theorem shows that two-layer networks are omnipotent – such results nearly killed NN when support vector machines were found to be easier to train in 00's.
- Later, it has been shown (experimentally) that deep networks (with many layers) have better representational properties.
- ... but how to train them? The gradient descent suffers from a so-called vanishing gradient; intuitively, updates of weights in lower layers are *very* slow.

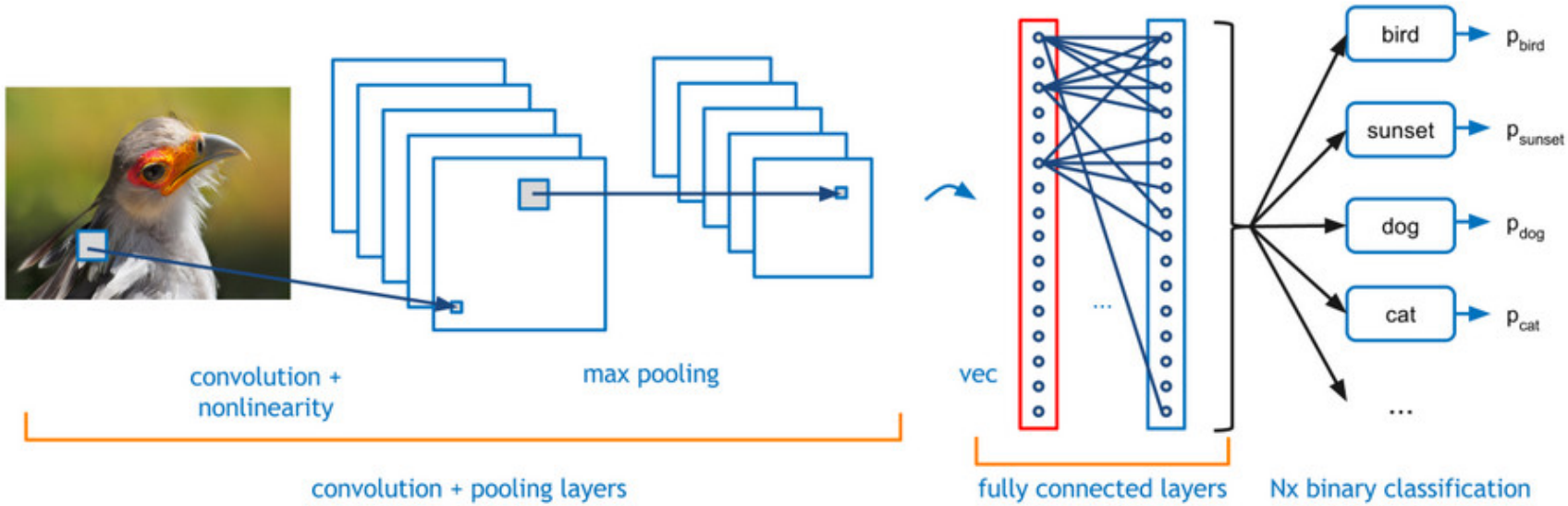
Deep Learning

- Cybenko's theorem shows that two-layer networks are omnipotent – such results nearly killed NN when support vector machines were found to be easier to train in 00's.
- Later, it has been shown (experimentally) that deep networks (with many layers) have better representational properties.
- ... but how to train them? The gradient descent suffers from a so-called vanishing gradient; intuitively, updates of weights in lower layers are *very* slow.
- In 2006, a solution was found by Hinton et al.:
  - Use *unsupervised* methods to initialize the weights layer by layer to capture important data features.
    More precisely: The lowest hidden layer learns patterns in data, the second lowest learns patterns in data transformed through the first layer, and so on.

Deep Learning

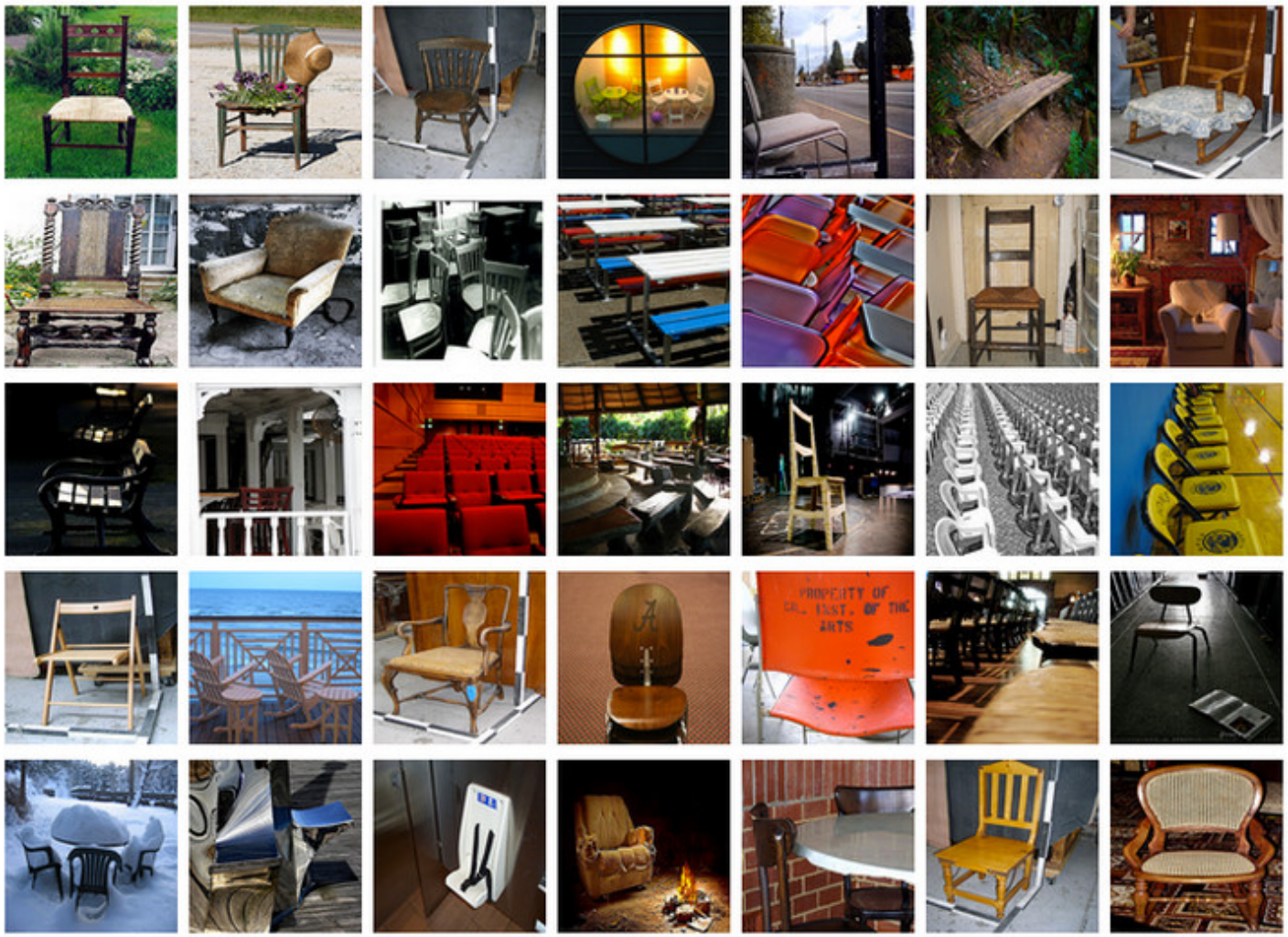
- Cybenko's theorem shows that two-layer networks are omnipotent – such results nearly killed NN when support vector machines were found to be easier to train in 00's.
- Later, it has been shown (experimentally) that deep networks (with many layers) have better representational properties.
- ... but how to train them? The gradient descent suffers from a so-called vanishing gradient; intuitively, updates of weights in lower layers are *very* slow.
- In 2006, a solution was found by Hinton et al.:
  - Use *unsupervised* methods to initialize the weights layer by layer to capture important data features.
    More precisely: The lowest hidden layer learns patterns in data, the second lowest learns patterns in data transformed through the first layer, and so on.
  - Then use a supervised learning algorithm to only *fine tune* the weights to the desired input-output behavior.
- ... but the actual revolution started with convolutional networks trained on several GPUs.

Convolutional network



ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

ImageNet database (16,000,000 color images, 20,000 categories)



ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

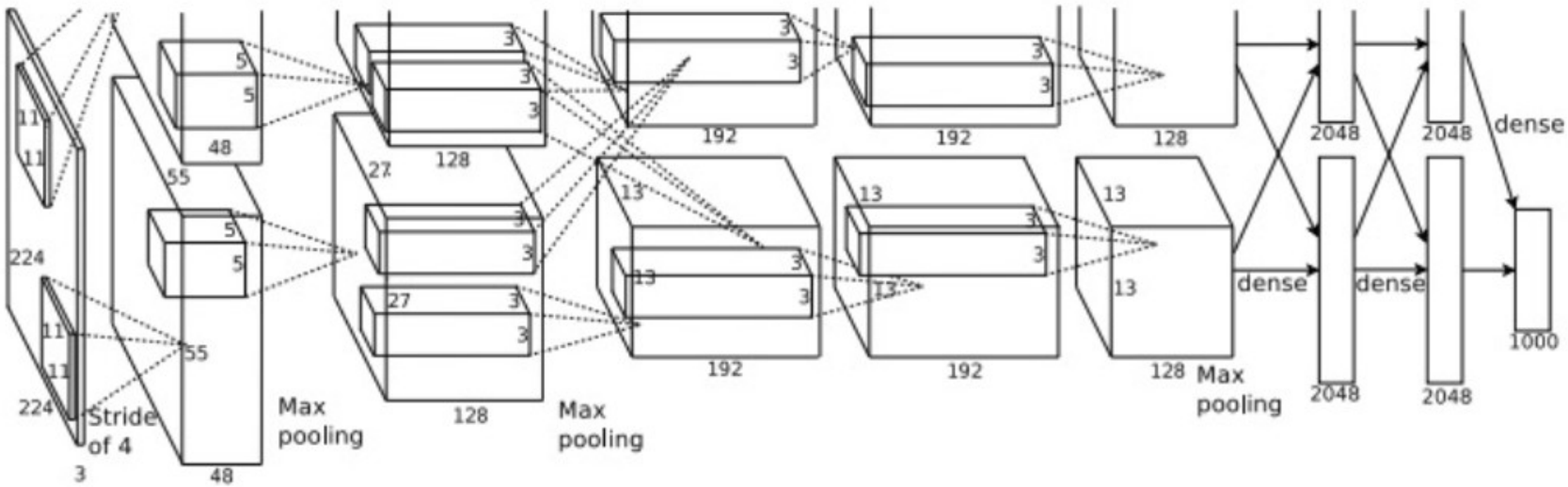
Competition in classification over a subset of images from ImageNet.

In 2012, training saw 1,200,000 images and 1000 categories. Validation set 50,000, Test set 150,000.

Many images contain several objects $\rightarrow$ typical rule is top-5 highest probability assigned by the net.

KSH síť

ImageNet classification with deep convolutional neural networks, by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012).



Trained on two GPUs (NVIDIA GeForce GTX 580)

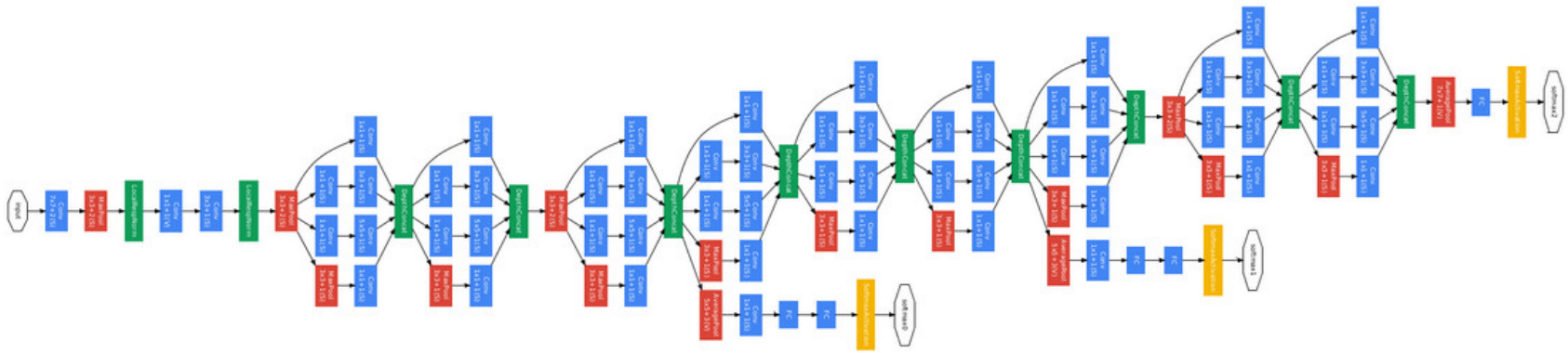
Results:

- Accuracy 84.7% in top-5 (second best alg. at the time: 73.8%)
- 63.3% in "perfect" classification (top-1)

ILSVRC 2014

The same set of images as in 2012, top-5 criterium.

GoogLeNet: deep convolutional net, 22 layers



Results:

- 93.33% in top-5

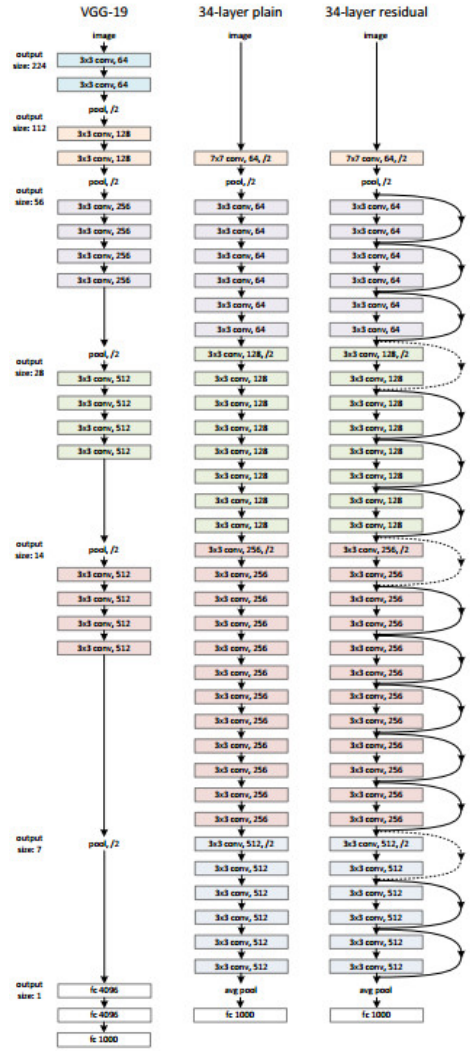
Superhuman power?

Superhuman GoogLeNet?!

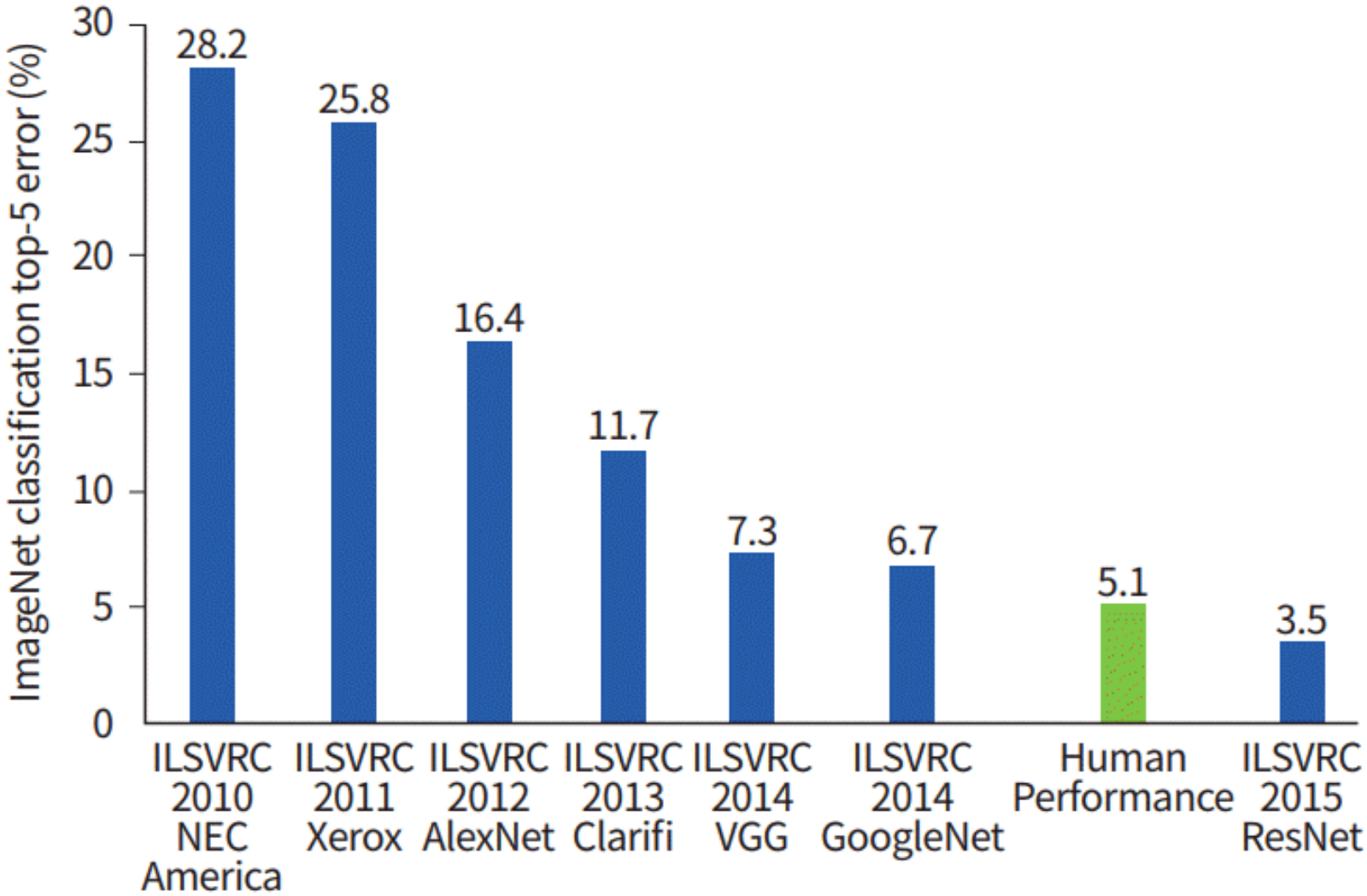
Andrej Karpathy: ...the task of labeling images with 5 out of 1000 categories quickly turned out to be highly challenging, even for some friends in the lab who have been working on ILSVRC and its classes for a while. First, we thought we would put it up on [Amazon Mechanical Turk]. Then, we thought we could recruit paid undergrads. Then, I organized a labeling party of intense labeling effort only among the (expert labelers) in our lab. Then, I developed a modified interface that used GoogLeNet predictions to prune the number of categories from 1000 to only about 100. It was still too hard - people kept missing categories and getting up to ranges of 13-15% error rates. In the end, I realized that to get anywhere competitively close to GoogLeNet, it would be most efficient if I sat down and went through the painfully long training process and the subsequent careful annotation process myself. The labeling happened at a rate of about 1 per minute, but this decreased over time... Some images are easily recognized, while some pictures (such as those of fine-grained breeds of dogs, birds, or monkeys) can require multiple minutes of concentrated effort. I became very good at identifying breeds of dogs... Based on the sample of images I worked on, the GoogLeNet classification error turned out to be 6.8%... In the end, my error turned out to be 5.1%

ILSVRC 2015

- Microsoft network ResNet: 152 layers, complex architecture
- Trained on 8 GPUs
- 96.43% **accuracy** in top-5



ILSVRC


ImageNet classification top-5 error (%)
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
5.1
3.5
ILSVRC 2010 NEC America
ILSVRC 2011 Xerox
ILSVRC 2012 AlexNet
ILSVRC 2013 Clarifi
ILSVRC 2014 VGG
ILSVRC 2014 GoogleNet
Human Performance
ILSVRC 2015 ResNet

ILSVRC 2016

Trumps-Soushen (The Third Research Institute of Ministry of Public Security)

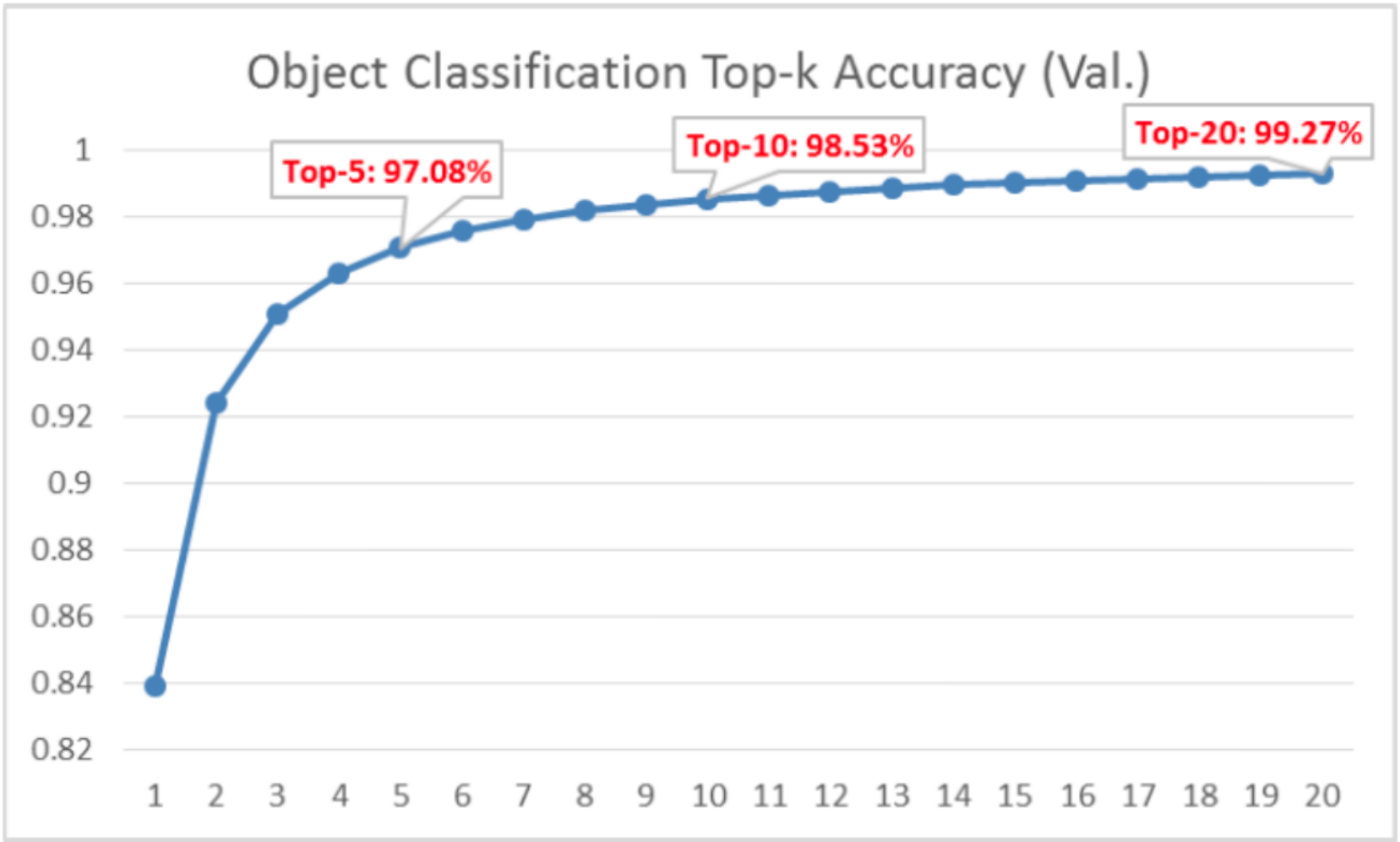
There is no new innovative technology or novelty by Trimps-Soushen.

Ensemble of the pre-trained models from previous years.

Each model is strong at classifying some categories but weak at categorizing others.

Test error: 2.99%

Top-k accuracy analyzed



https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

Top-20 typical errors

Out of 1458 misclassified images in Top-20:

Error Categories	Numbers	Percentages(%)
Label May Wrong	221	15.16
Multiple Objects (>5)	118	8.09
Non-Obvious Main Object	355	24.35
Confusing Label	206	14.13
Fine-grained Label	258	17.70
Obvious Wrong	234	16.05
Partial Object	66	4.53

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

Top-k accuracy analyzed

Predict:
1 *pencil box*
2 *diaper*
3 *bib*
4 *purse*
5 *running shoe*

Ground Truth:
sleeping bag

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

Top-k accuracy analyzed

Predict:
1 *dock*
2 *submarine*
3 *boathouse*
4 *breakwater*
5 *lifeboat*

Ground Truth:
paper towel

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

Top-k accuracy analyzed

Predict:
1 *bolete*
2 *earthstar*
3 *gyromitra*
4 *hen of the woods*
5 *mushroom*

Ground Truth:
stinkhorn

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

Top-k accuracy analyzed

Predict:
1 *apron*
2 *plastic bag*
3 *sleeping bag*
4 *umbrella*
5 *bulletproof vest*

Ground Truth:
poncho

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

Anomaly Detection

What is an Anomaly?

Anomalies are hard to define in general.

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

This lecture presents algorithms detecting specific data instances that may be anomalous w.r.t. the used algorithm.

It is up to the context-knowing user to decide whether such data are anomalous.

What is the Anomaly Detection Good For?

There are two main sources of motivation for anomaly detection in machine learning:

- *Modeling perspective*: Many models are sensitive to anomalous data.

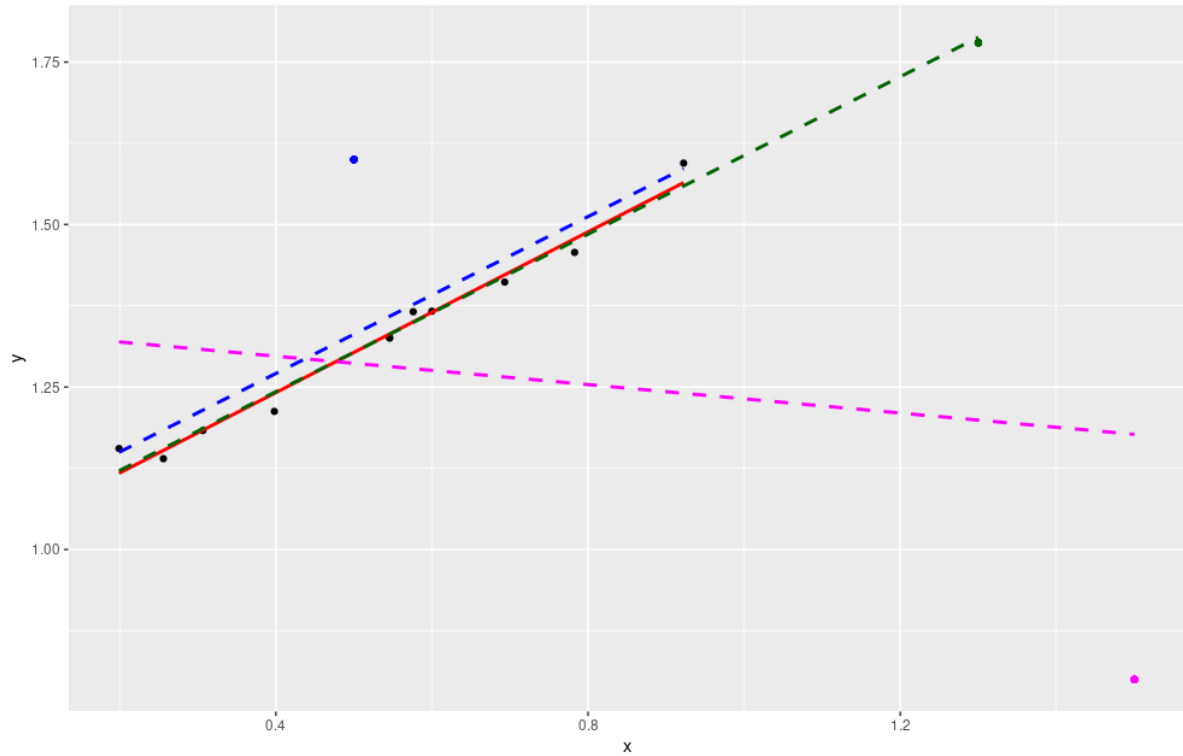
What is the Anomaly Detection Good For?

There are two main sources of motivation for anomaly detection in machine learning:

- *Modeling perspective*: Many models are sensitive to anomalous data.
- *Application perspective*: Many tasks are based on searching for anomalies in data.

Modeling Perspective

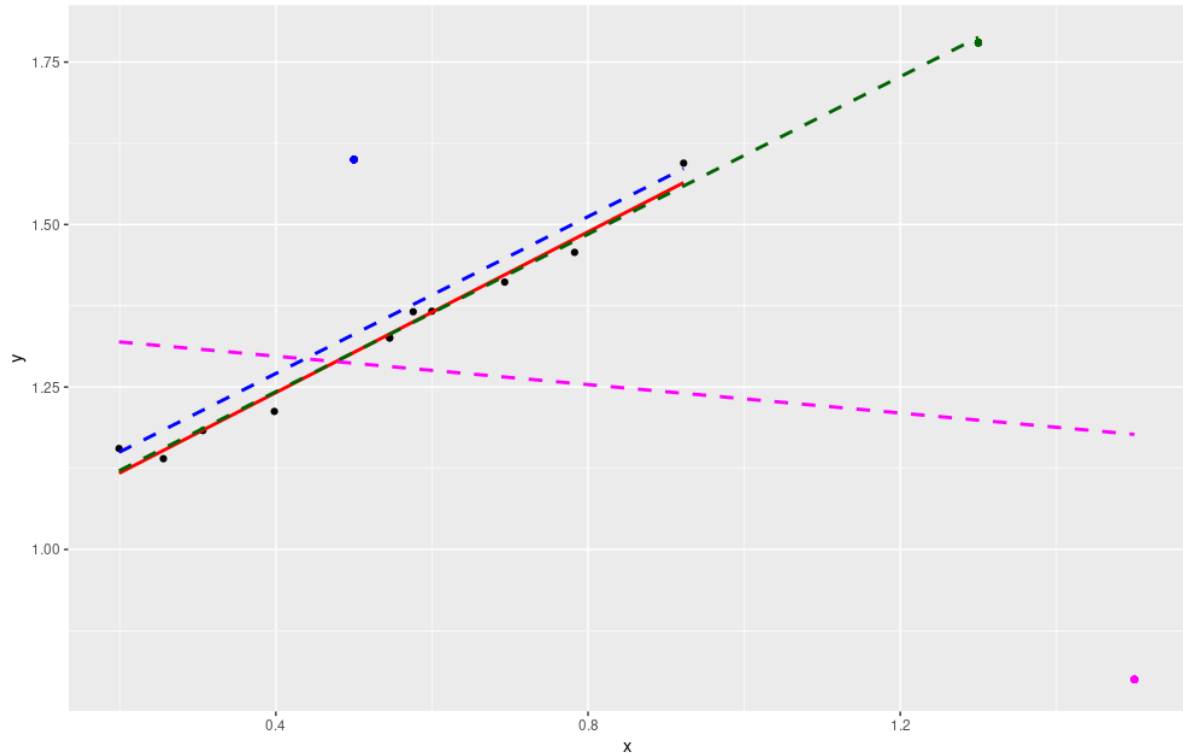
Recall the behavior of the linear model.



To train such a model properly, the outliers should be removed.

Modeling Perspective

Recall the behavior of the linear model.



To train such a model properly, the outliers should be removed.

On the other hand, we will see that the sensitivity of some models can be used to *detect* the anomalies.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.
- **Ecosystem Disturbances**
  - Detect atypical natural world events.
  - Prediction and understanding of hurricanes, floods, droughts, heat waves, and fires.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.
- **Ecosystem Disturbances**
  - Detect atypical natural world events.
  - Prediction and understanding of hurricanes, floods, droughts, heat waves, and fires.
- **Medicine**
  - Unusual patient symptoms or test results may indicate health problems.
  - Balancing the need for further tests with the potential costs and risks.
- ...

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.
- Data measurement and collection error:
  - Thermometer positioned on the direct sun (possibly a measurement error)
  - 40 kg infant is not usual among humans (possibly a data collection error)

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.
- Data measurement and collection error:
  - Thermometer positioned on the direct sun (possibly a measurement error)
  - 40 kg infant is not usual among humans (possibly a data collection error)

The anomalies mentioned above should be inspected by domain experts and possibly corrected/removed.

Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).

Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).
- Proximity-based techniques: Given distance between objects, anomalies are objects distant from others.

Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).
- Proximity-based techniques: Given distance between objects, anomalies are objects distant from others.
- Density-based techniques: Estimate the density distribution of the objects. Anomalies would probably be outside of dense regions.

I would count the most recent deep learning-based anomaly detection among the model-based techniques even though a combination of the above approaches is usually used.

We will make the above ideas more precise in the rest of this lecture.

Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.

  As anomalies are rare, approaches based on supervised learning are rare.

Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.

  As anomalies are rare, approaches based on supervised learning are rare.
- Semi-supervised:
  - We may know what instances are normal.

    A tumor detection system trained on healthy people detects tumors as anomalies.

    In this case, we detect anomalies that are dissimilar to normal instances. We may train a model representing the normal instances and detect instances that do not fit the model.
  - We may have information about (some) normal and some anomalous instances. Here, we can use methods for semi-supervised classification.

Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.

  As anomalies are rare, approaches based on supervised learning are rare.
- Semi-supervised:
  - We may know what instances are normal.

    A tumor detection system trained on healthy people detects tumors as anomalies.

    In this case, we detect anomalies that are dissimilar to normal instances. We may train a model representing the normal instances and detect instances that do not fit the model.
  - We may have information about (some) normal and some anomalous instances. Here, we can use methods for semi-supervised classification.
- Unsupervised: Create a model of all instances and hope that anomalous instances will still be dissimilar to instances typical for the model.

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).
- Degree of anomaly: Usually, anomalies are reported in a binary fashion. However, we often want to measure how anomalous an object is, similar to normal objects.

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).
- Degree of anomaly: Usually, anomalies are reported in a binary fashion. However, we often want to measure how anomalous an object is, similar to normal objects.

Another issue is evaluation: How can we determine how good an anomaly detector is?

In the supervised case, we have a ground truth, but usually, we just observe detected anomalies.

Various Approaches to Anomaly Detection

We shall have a look at five approaches to the unsupervised anomaly detection:

- Statistical
- Proximity-based
- Density-based
- Cluster-based
- Autoencoders

The above approaches are also used in the semi-supervised setting where we have a dataset of normal instances. However, some issues discussed further will not appear in this case.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Statistical

Definition 1

An outlier *is an object with a low probability in the probability distribution of the data.*

Suppose we knew the "true" probability distribution P on objects. In that case, we may set up a threshold t (using an appropriate training set) and say that every object A with $P(A) < t$ is anomalous.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Suppose we knew the "true" probability distribution P on objects. In that case, we may set up a threshold t (using an appropriate training set) and say that every object A with $P(A) < t$ is anomalous.

However, in most cases, we do not know P and have to resort to a model of P created using a dataset of feature vectors.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Suppose we knew the "true" probability distribution P on objects. In that case, we may set up a threshold t (using an appropriate training set) and say that every object A with $P(A) < t$ is anomalous.

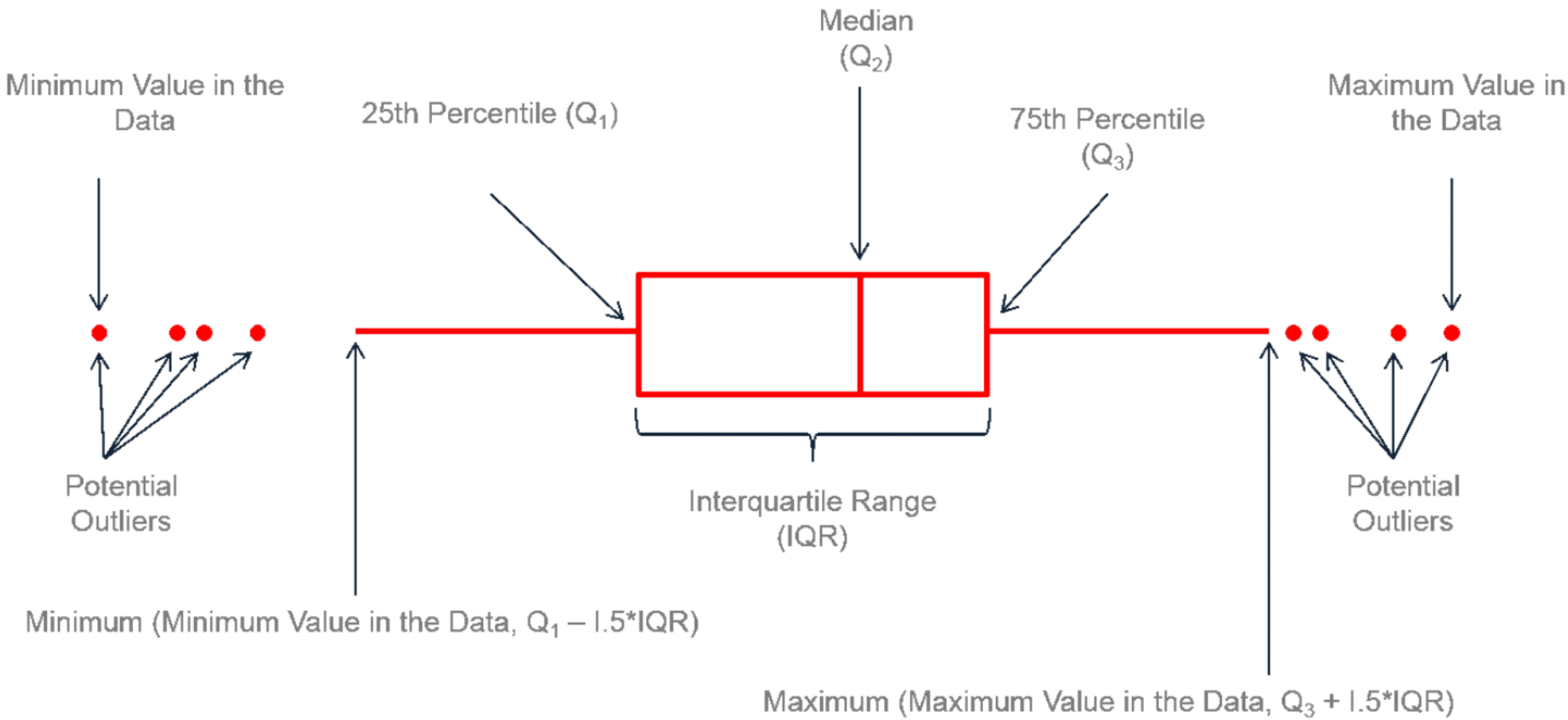
However, in most cases, we do not know P and have to resort to a model of P created using a dataset of feature vectors.

There are many more or less sophisticated tests based on models of P in the literature.

Some use rather advanced statistical methods.

Let us have a look at a few simple examples.

Box Plot



A simple method for outlier detection in the *univariate* case, that is, for values of a single attribute.

Univariate Gaussian Model

Consider a numeric attribute whose values x are normally distributed with mean μ and standard deviation σ.

We write $N(\mu, \sigma^2)$.

Univariate Gaussian Model

Consider a numeric attribute whose values x are normally distributed with mean μ and standard deviation σ.

We write $N(\mu, \sigma^2)$.

Given an attribute value x, we compute the z-score:

$$z = (x - \mu)/\sigma$$

Then z has the distribution $N(0, 1)$.

Univariate Gaussian Model

Consider a numeric attribute whose values x are normally distributed with mean μ and standard deviation σ.

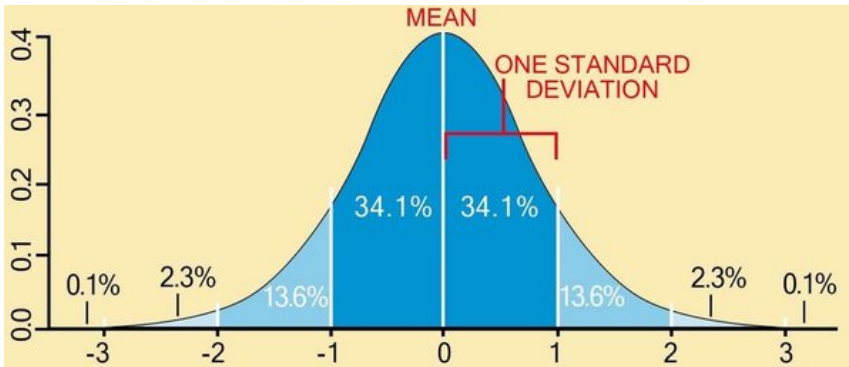
We write $N(\mu, \sigma^2)$.

Given an attribute value x, we compute the z-score:

$$z = (x - \mu)/\sigma$$

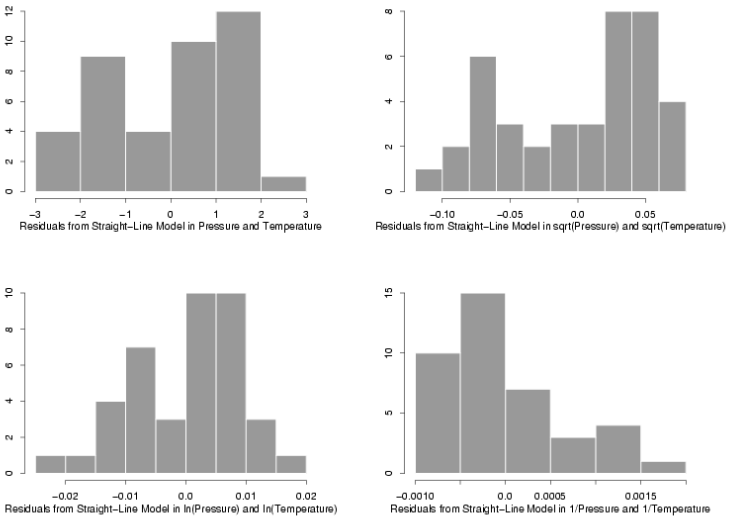
Then z has the distribution $N(0, 1)$.

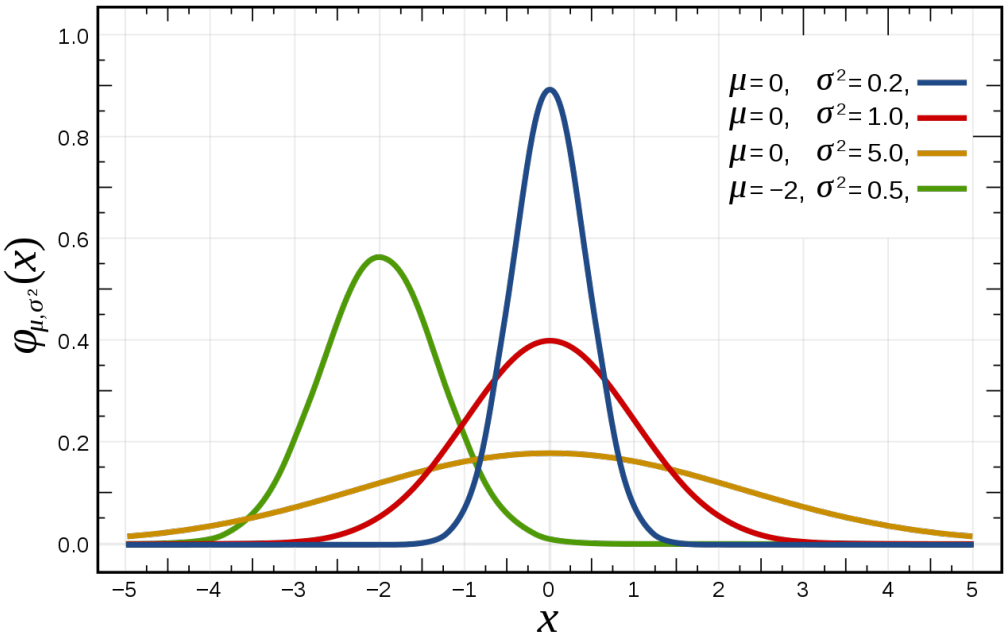
Now, choose c so the probability $P(|z| \geq c)$ is small enough for z to be an outlier w.r.t. $N(0, 1)$.



Given an attribute value x, we may decide whether x is an outlier by deciding whether $|z| = |(x - \mu)/\sigma| \geq c$.

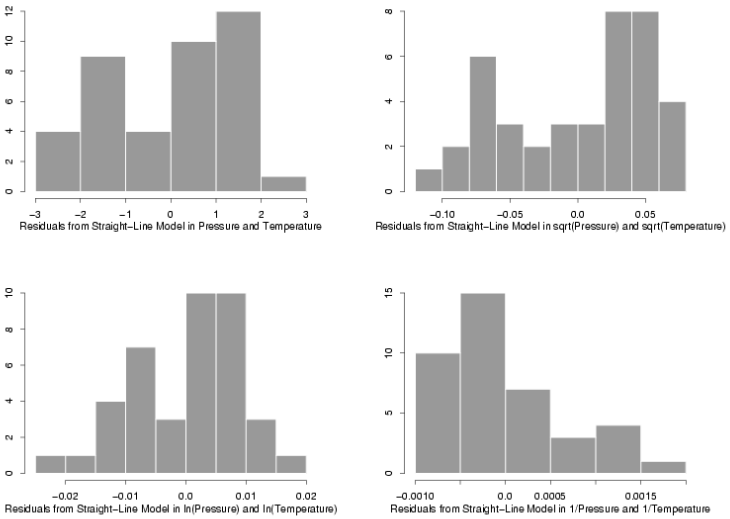
Univariate Gaussian

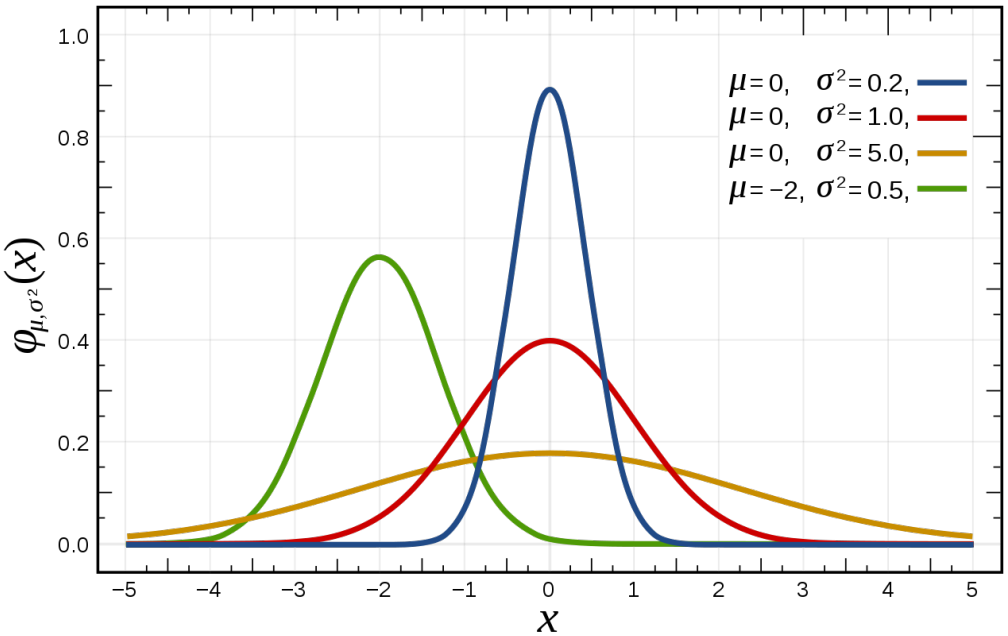




Problem 1: The attribute does not have a normal distribution.

Univariate Gaussian





Problem 1: The attribute does not have a normal distribution.

Before starting, use a normality test (observe the histogram, use a specialized test such as Shapiro-Wilk).

Some transformations may sometimes succeed in normalizing an attribute (Box-Cox transformation, etc.)

Different distributions can be used to model the attribute (Log Normal, Weibull, etc.), but be aware of the assumptions of anomaly detection tests!

Univariate Gaussian

Problem 2: We typically do not know the population mean μ and the population variance σ^2.

Univariate Gaussian

Problem 2: We typically do not know the population mean μ and the population variance σ^2.

They can be estimated from a dataset by the sample mean and the sample variance:

$$\bar{\mu} = \frac{1}{p}\sum_{i=1}^{p} x_i \qquad s^2 = \frac{1}{p-1}\sum_{i=1}^{p}(x_i - \bar{\mu})^2$$

However, then $z = (x - \bar{\mu})/s$ does no longer have the distribution $N(0, 1)$.

The distribution of z is normal if x is sampled independently of the data yielding $\bar{\mu}$ and s^2.

Univariate Gaussian

Problem 2: We typically do not know the population mean μ and the population variance σ^2.

They can be estimated from a dataset by the sample mean and the sample variance:

$$\bar{\mu} = \frac{1}{p}\sum_{i=1}^{p} x_i \qquad s^2 = \frac{1}{p-1}\sum_{i=1}^{p}(x_i - \bar{\mu})^2$$

However, then $z = (x - \bar{\mu})/s$ does no longer have the distribution $N(0, 1)$.

The distribution of z is normal if x is sampled independently of the data yielding $\bar{\mu}$ and s^2.

Note that $\bar{\mu}$ and s^2 are unbiased estimates of μ and σ^2. Also, $\bar{\mu}$ converges to μ and s^2 converges to σ^2 with growing sample size.

Univariate Gaussian Model

Problem 3: The outliers distort the estimates $\bar{\mu}$ and s^2 of the mean and the standard deviation.

For example, a millionaire would not look like an outlier in a group containing a billionaire.

There is a circularity here: To get outliers using the normal distribution model, we need to remove the outliers to have a good model.

Univariate Gaussian Model

Problem 3: The outliers distort the estimates $\bar{\mu}$ and s^2 of the mean and the standard deviation.

For example, a millionaire would not look like an outlier in a group containing a billionaire.

There is a circularity here: To get outliers using the normal distribution model, we need to remove the outliers to have a good model.

One possible heuristic is to remove the outliers one by one, starting from the most extreme, hoping the most extreme outlier will be detected in every step.

One such heuristic is the Grubb's test.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and s sample standard deviation of D.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and s sample standard deviation of D.

Now $P(G \geq c_{\alpha,p}) = \alpha$ if

$$c_{\alpha,p} = \frac{p-1}{\sqrt{p}} \sqrt{\frac{t^2}{p-2+t}}$$

Here t is such a value that makes $P(T \geq t) = \alpha/(2p)$ for T with the t-distribution with $p-2$ degrees of freedom.

For finding t we used to use tables.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and s sample standard deviation of D.

Now $P(G \geq c_{\alpha,p}) = \alpha$ if

$$c_{\alpha,p} = \frac{p-1}{\sqrt{p}} \sqrt{\frac{t^2}{p-2+t}}$$

Here t is such a value that makes $P(T \geq t) = \alpha/(2p)$ for T with the t-distribution with $p-2$ degrees of freedom.

For finding t we used to use tables.

Grubb's test simply iteratively tests whether $P(G \geq c_{\alpha,p})$ and, if yes, removes an x_i maximizing $|x_i - \bar{x}|$ from D.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Mahalanobis distance:

$$\text{mahalanobis}(\vec{x}, \vec{z}) = (\vec{x} - \vec{z})\Sigma^{-1}(\vec{x} - \vec{z})^{\top}$$

Here Σ is the covariance matrix of the data.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Mahalanobis distance:

$$\text{mahalanobis}(\vec{x}, \vec{z}) = (\vec{x} - \vec{z})\Sigma^{-1}(\vec{x} - \vec{z})^{\top}$$

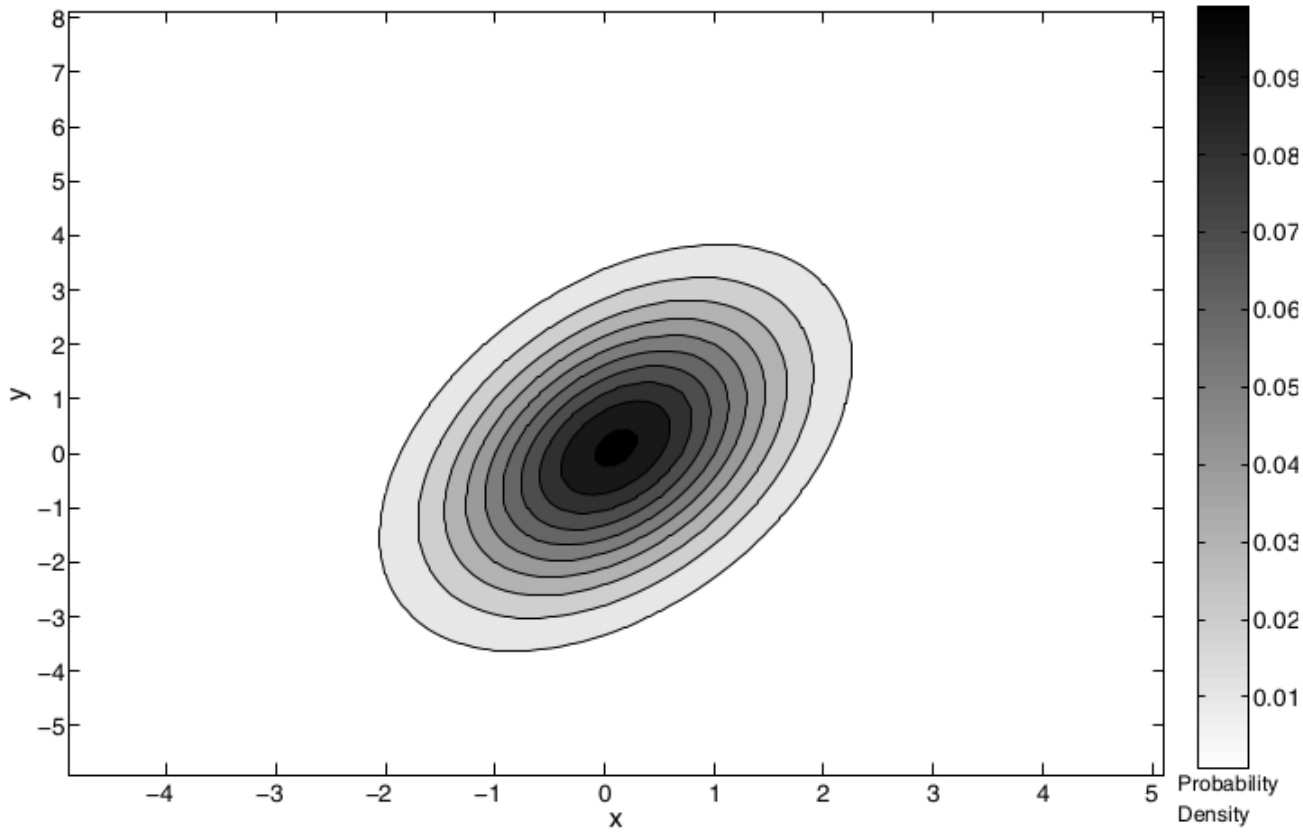
Here Σ is the covariance matrix of the data.

Intuitively, the Mahalanobis distance generalizes the squared Euclidean distance:

$$(\vec{x} - \vec{z})(\vec{x} - \vec{z})^{\top}$$

By taking into account the "spread" of the data.

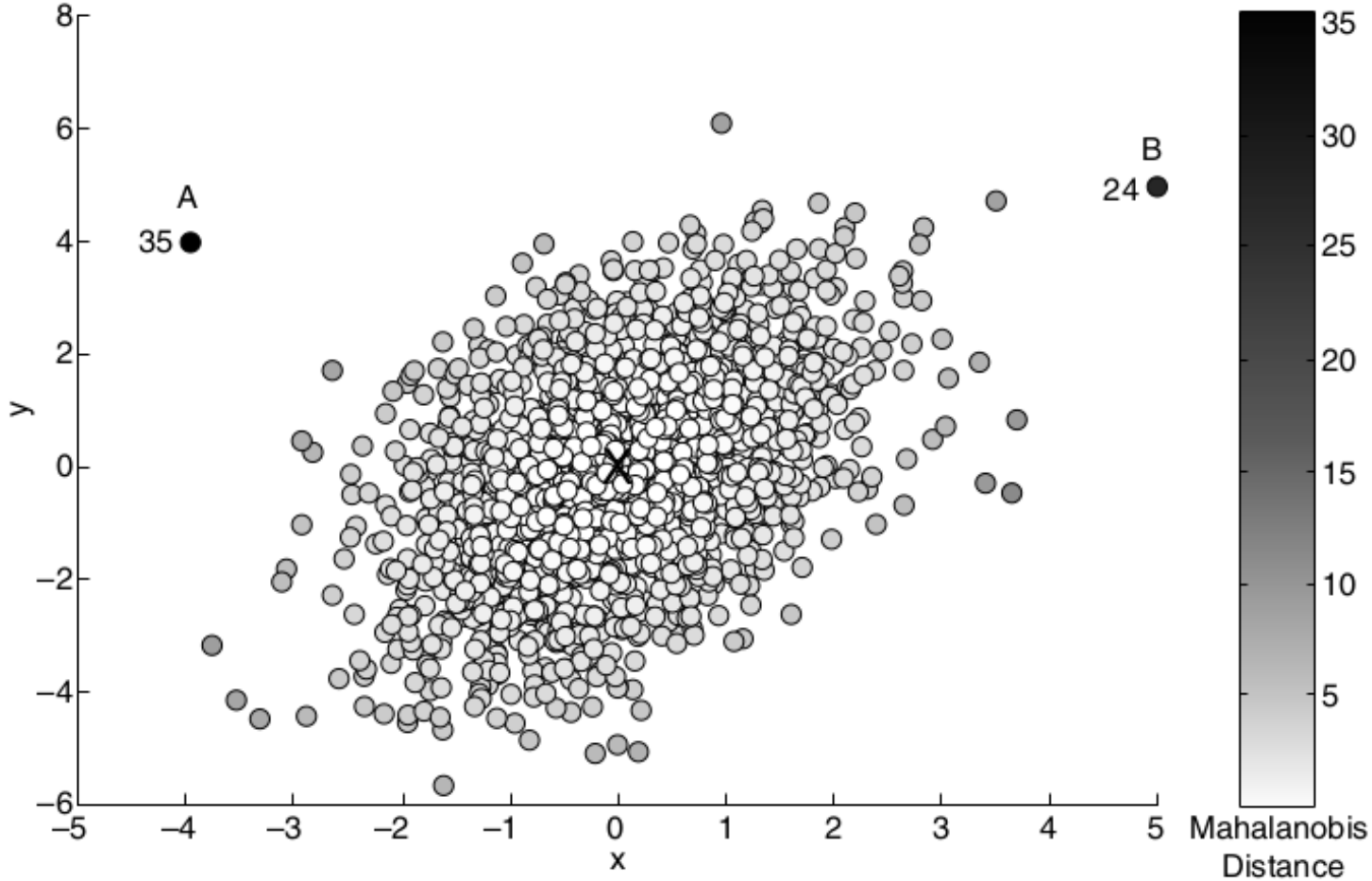
Mahalanobis Distance



Here, we assume multivariate normal data distribution. The covariance matrix is

$$\mathbf{\Sigma} = \begin{pmatrix} 1.00 & 0.75 \\ 0.75 & 3.00 \end{pmatrix}$$

Mahalanobis Distance



Notice that A has a larger Mahalanobis distance from the cluster's center than B even though it is closer in the Euclidean distance.

The reason is that the density function falls more rapidly in the direction of A than in the direction of B.

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Let us assume that we have M and A models to compute $P_M(x)$ and $P_A(x)$ the likelihoods of generating x.

A is often considered uniform, and M is learned from the data (yes, distorted by the outliers).

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Let us assume that we have M and A models to compute $P_M(x)$ and $P_A(x)$ the likelihoods of generating x.

A is often considered uniform, and M is learned from the data (yes, distorted by the outliers).

Now, we compute the total likelihood of the dataset before and after removing each element. If the likelihood changes significantly, we have probably eliminated an anomaly.

Given a partition U, V of D we define

$$L(U, V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i) \right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i) \right)$$

Given a partition U, V of D we define

$$L(U, V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i) \right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i) \right)$$

This is the likelihood of generating D using the following random process:

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is α.

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right)\left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is α.
- If normal has been chosen, choose x_i from the distribution M.
- If anomaly has been chosen, choose x_i from the distribution A.

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right)\left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is α.
- If normal has been chosen, choose x_i from the distribution M.
- If anomaly has been chosen, choose x_i from the distribution A.

To eliminate the "large" product, we consider the log-likelihood:

$$\begin{aligned} LL(U,V) &= \log(L(U,V)) \\ &= |U| \log(1-\alpha) + \sum_{x_i \in U} \log P_M(x_i) \\ &\quad + |V| \log \alpha + \sum_{x_i \in V} \log P_A(x_i) \end{aligned}$$

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

- Consider i maximizing Δ_i. If $\Delta_i \geq c$, then

$$M_{k+1} = M_k \setminus \{x_i\} \qquad A_{k+1} = A_k \cup \{x_i\}$$

Else, stop.

Here, c is a threshold we must set up.

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

- Consider i maximizing Δ_i. If $\Delta_i \geq c$, then

$$M_{k+1} = M_k \setminus \{x_i\} \qquad A_{k+1} = A_k \cup \{x_i\}$$

  Else, stop.

  Here, c is a threshold we must set up.

Ultimately, the A_k will contain the anomalies the algorithm detects.

Note that we may also move all anomalies detected in a single iteration from M_k to A_k. This would result in a different method (also valid).

Summary of Statistical Anomaly Detection

- Build on strong statistical foundations.
- Tests are very effective if the dataset is sufficiently large (informative).
- Lots of tests for univariate data, the area has developed for a long time.
- Fewer options for multivariate data and problematic for highly dimensional data (curse of dimensionality).

The likelihood-based approach does not assume a particular distribution shape: It can be used with arbitrary models of P_M and P_A, including deep learning ones.

Proximity-Based Methods

Proximity-Based Outlier Detection

Assume a distance measure d. That is, given two feature vectors $\vec{x}, \vec{z}$ their distance is $d(\vec{x}, \vec{z})$.

We consider the Euclidean distance for simplicity.

Definition 2

The outlier score of an object is given by the distance to its k-nearest neighbor.

Proximity-Based Outlier Detection

Assume a distance measure d. That is, given two feature vectors $\vec{x}, \vec{z}$ their distance is $d(\vec{x}, \vec{z})$.

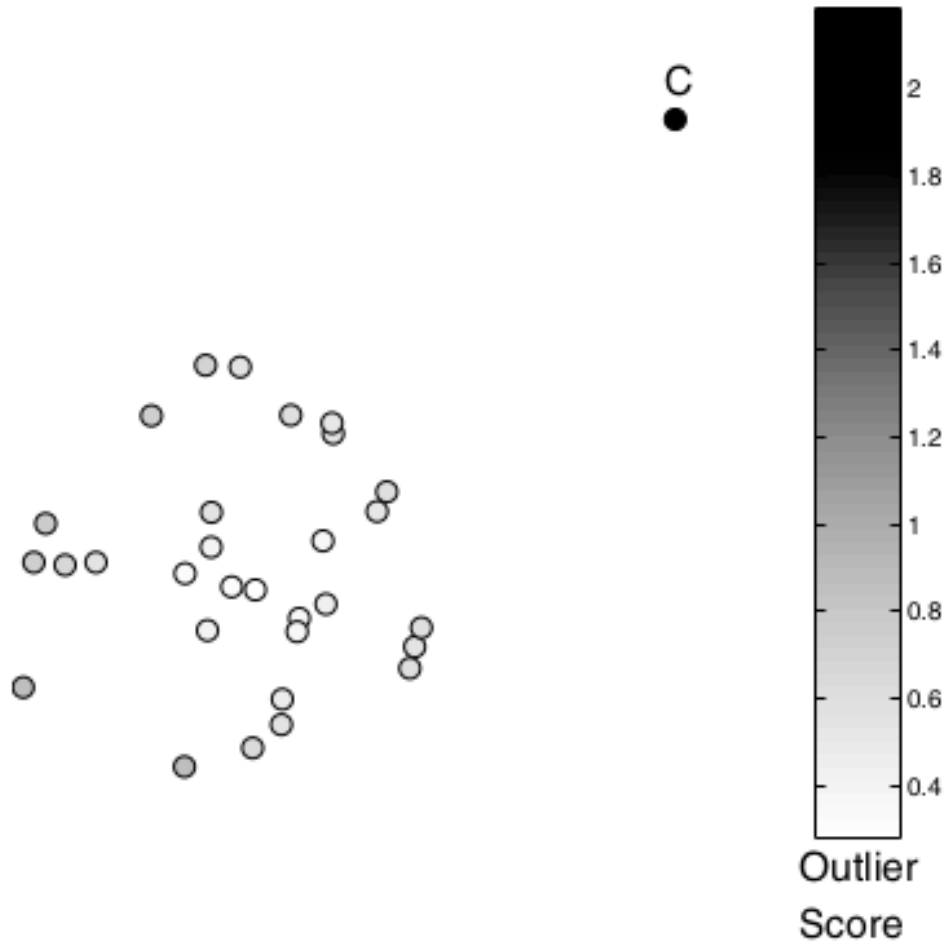
We consider the Euclidean distance for simplicity.

Definition 2

The outlier score of an object is given by the distance to its k-nearest neighbor.

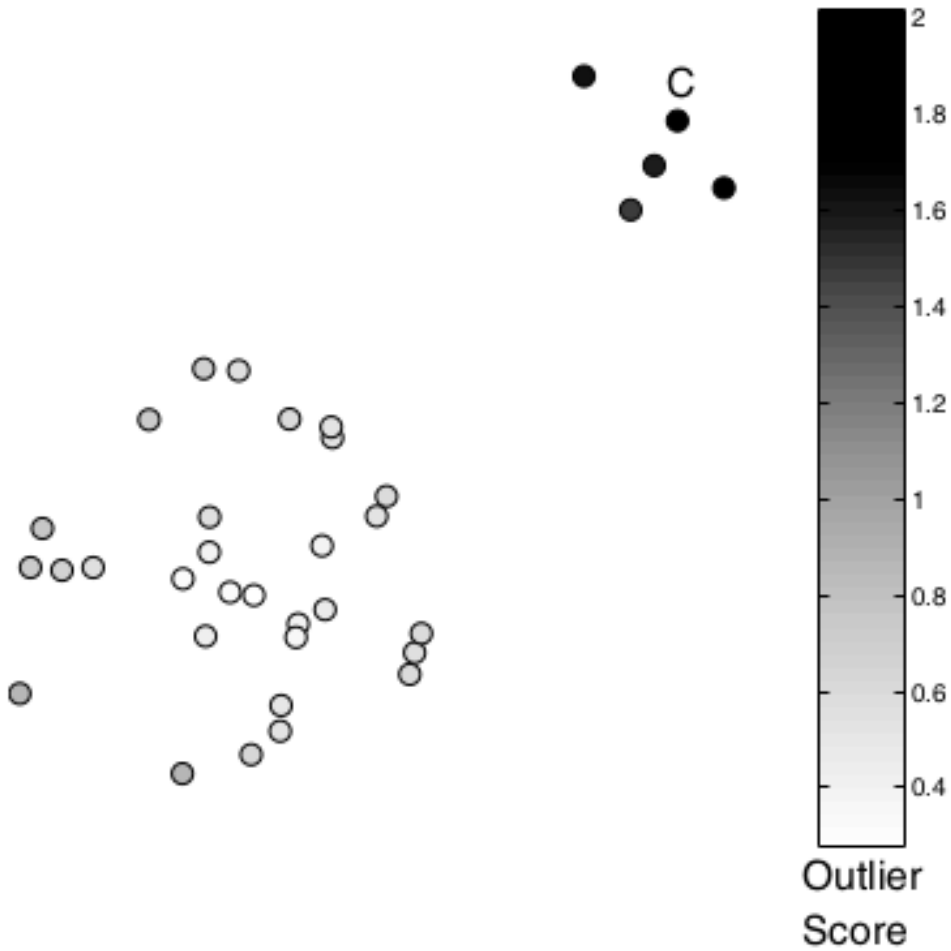
A threshold on the minimum distance of an outlier can be set on a training set.

Outlier Score



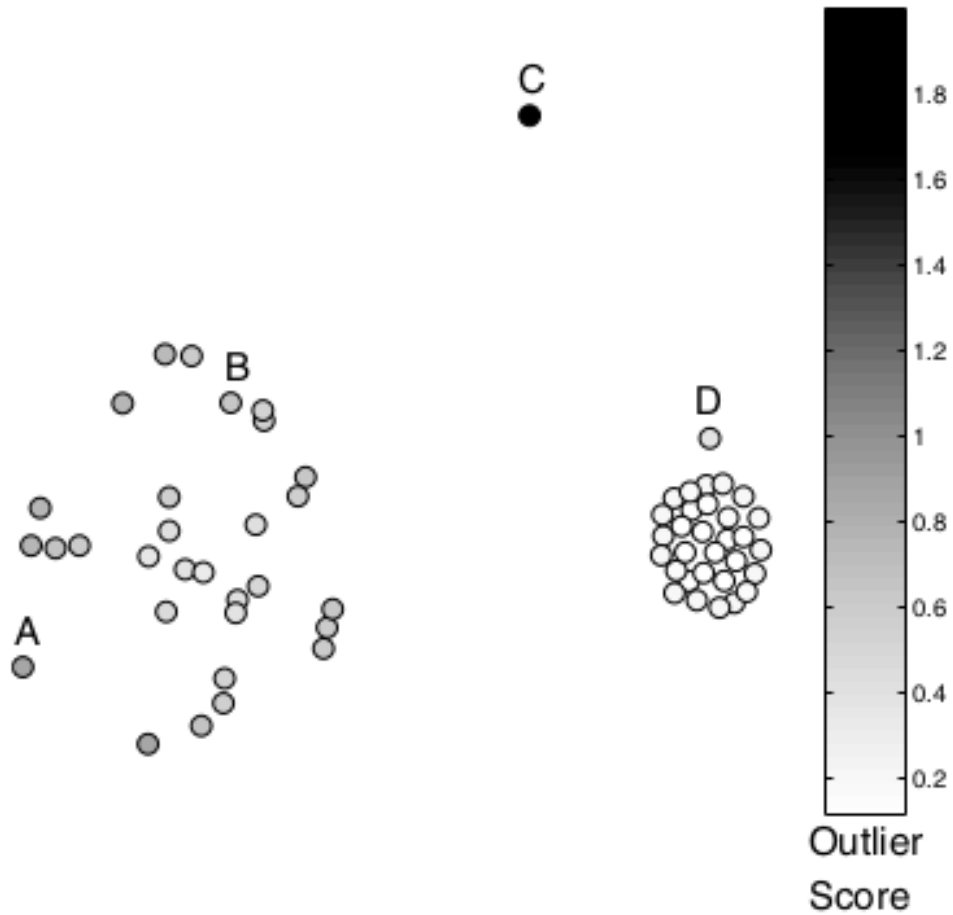
The outlier score is based on the distance to the fifth nearest neighbor.

Outlier Score



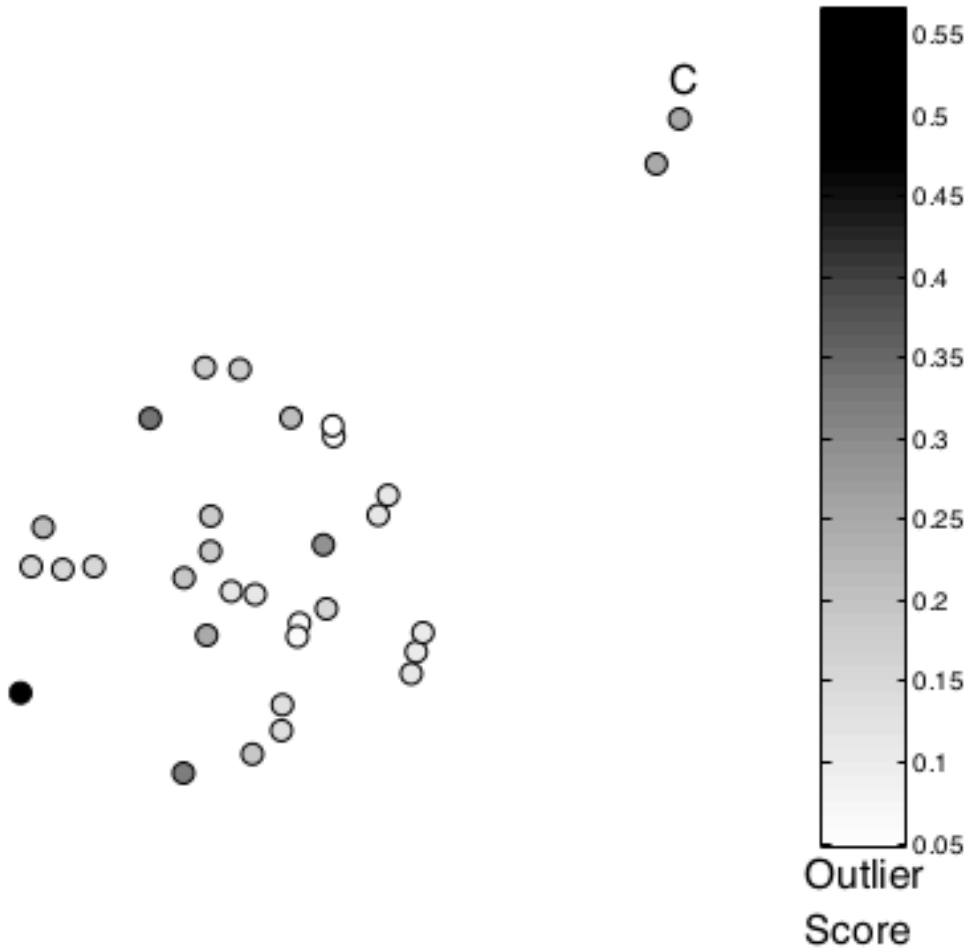
The outlier score is based on the distance to the fifth nearest neighbor.

Outlier Score



The outlier score is based on the distance to the fifth nearest neighbor.

Outlier Score



The outlier score is based on the distance to the first nearest neighbor.

Proximity-Based Approaches

- Conceptually simple and easy to implement.
- Time complexity typically $\mathcal{O}(p^2)$ where p is the number of feature vectors in the dataset.
- Sensitivity to the choice of parameters (the number of neighbors).
- Density/compactness not taken explicitly into account.

Density-Based Methods

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,
- computing the average $\bar{D}$ of the local densities of k-nearest neighbors of A,

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,
- computing the average $\bar{D}$ of the local densities of k-nearest neighbors of A,
- dividing the average local density with the local density of A, that is, compute $\bar{D}/D_A$.

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

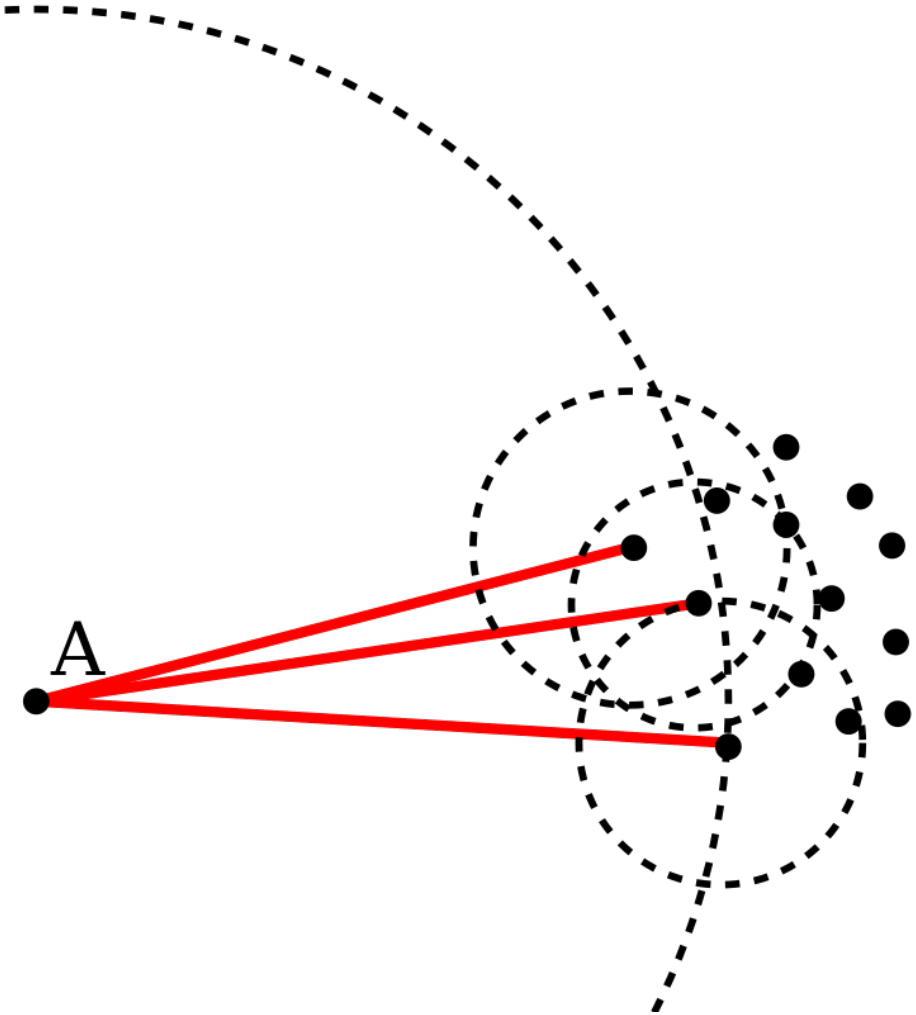
A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,
- computing the average $\bar{D}$ of the local densities of k-nearest neighbors of A,
- dividing the average local density with the local density of A, that is, compute $\bar{D}/D_A$.

The question is, how exactly can the local density be defined?

LOF - Illustration



Here, the density around A is smaller than the densities around the other points.

LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define k-distance(A) as the distance of the k-th nearest neighbor.

LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define k-distance(A) as the distance of the k-th nearest neighbor.

Denote by $N_k(A)$ the set of all objects in the distance from A up to k-distance(A).

Note that if there are objects in the same distance from A, we may have more than k elements in $N_k(A)$.

LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define k-distance(A) as the distance of the k-th nearest neighbor.

Denote by $N_k(A)$ the set of all objects in the distance from A up to k-distance(A).

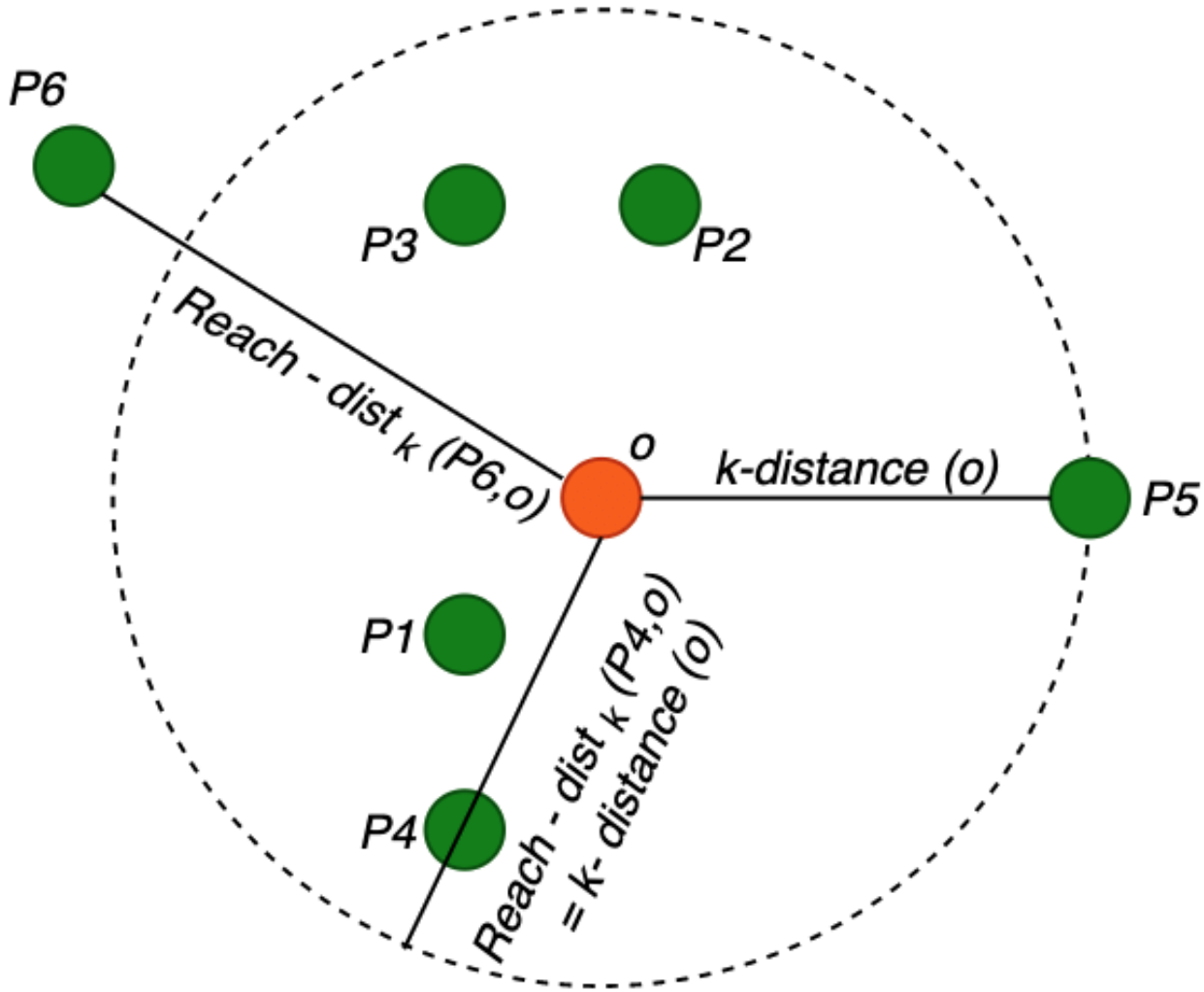
Note that if there are objects in the same distance from A, we may have more than k elements in $N_k(A)$.

Define

$$\text{reachability-distance}_k(A, B) = \max\{k\text{-distance}(B), d(A, B)\}$$

The reachability distance of A from B is the distance between A and B but at least the distance from B to the k-nearest neighbor.

Reachability Distance



LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of A from its k-nearest neighbors.

LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of A from its k-nearest neighbors.

We define *local outlier factor* of an object A by

$$\mathrm{LOF}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \mathrm{lrd}_k(B)}{|N_k(A)|} \right) / \mathrm{lrd}_k(A)$$

LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of A from its k-nearest neighbors.

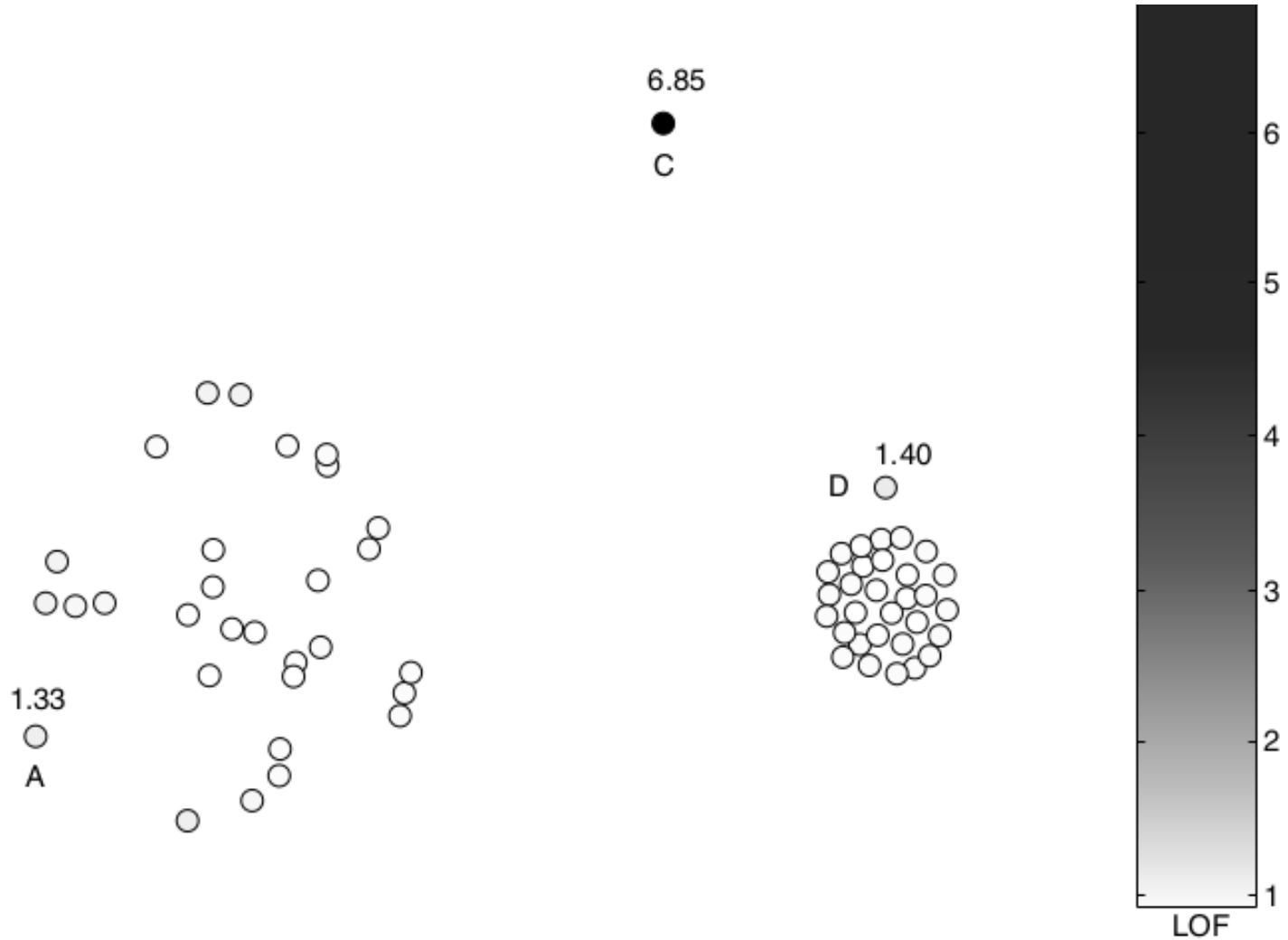
We define *local outlier factor* of an object A by

$$\mathrm{LOF}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \mathrm{lrd}_k(B)}{|N_k(A)|} \right) / \mathrm{lrd}_k(A)$$

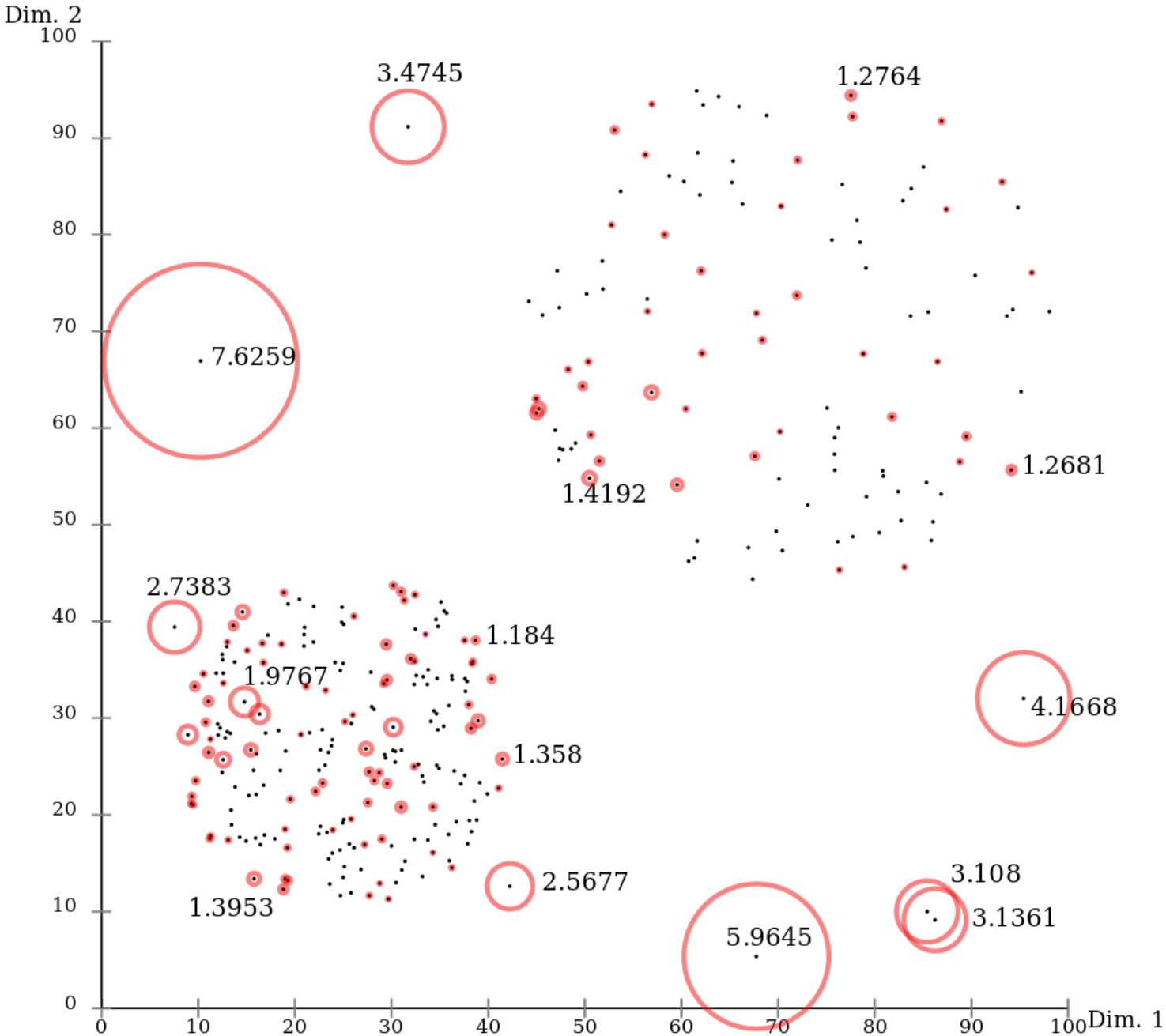
Now

- $\mathrm{LOF}_k(A) \approx 1$ - similar density as the neighbors have
- $\mathrm{LOF}_k(A) < 1$ - higher density of neighbors than the neighbors have (inlier?)
- $\mathrm{LOF}_k(A) > 1$ - smaller density of neighbors than the neighbors have (outlier?)

Example



Example


Dim. 2
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Dim. 1
3.4745
1.2764
7.6259
1.2681
1.4192
2.7383
1.184
1.9767
4.1668
1.358
2.5677
3.108
3.1361
1.3953
5.9645

Comments on Density Based Methods

- As opposed to the distance-based methods, quantifies the distance of the neighborhood of the (potential) outliers.
- Time complexity still $\mathcal{O}(p^2)$ but may be reduced for low dimensional data (to $\mathcal{O}(p \log p)$) using special data structures.
- Still need to determine the parameter k.

Clustering-Based Methods

Clustering Approach

Clustering is supposed to group similar objects.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.

This approach is problematic as it strongly depends on the size/number of clusters.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.

This approach is problematic as it strongly depends on the size/number of clusters.

A better approach would be to asses how strongly objects belong to clusters.

Definition 4

An object is a cluster-based outlier if the object does not strongly belong to any cluster.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.

This approach is problematic as it strongly depends on the size/number of clusters.

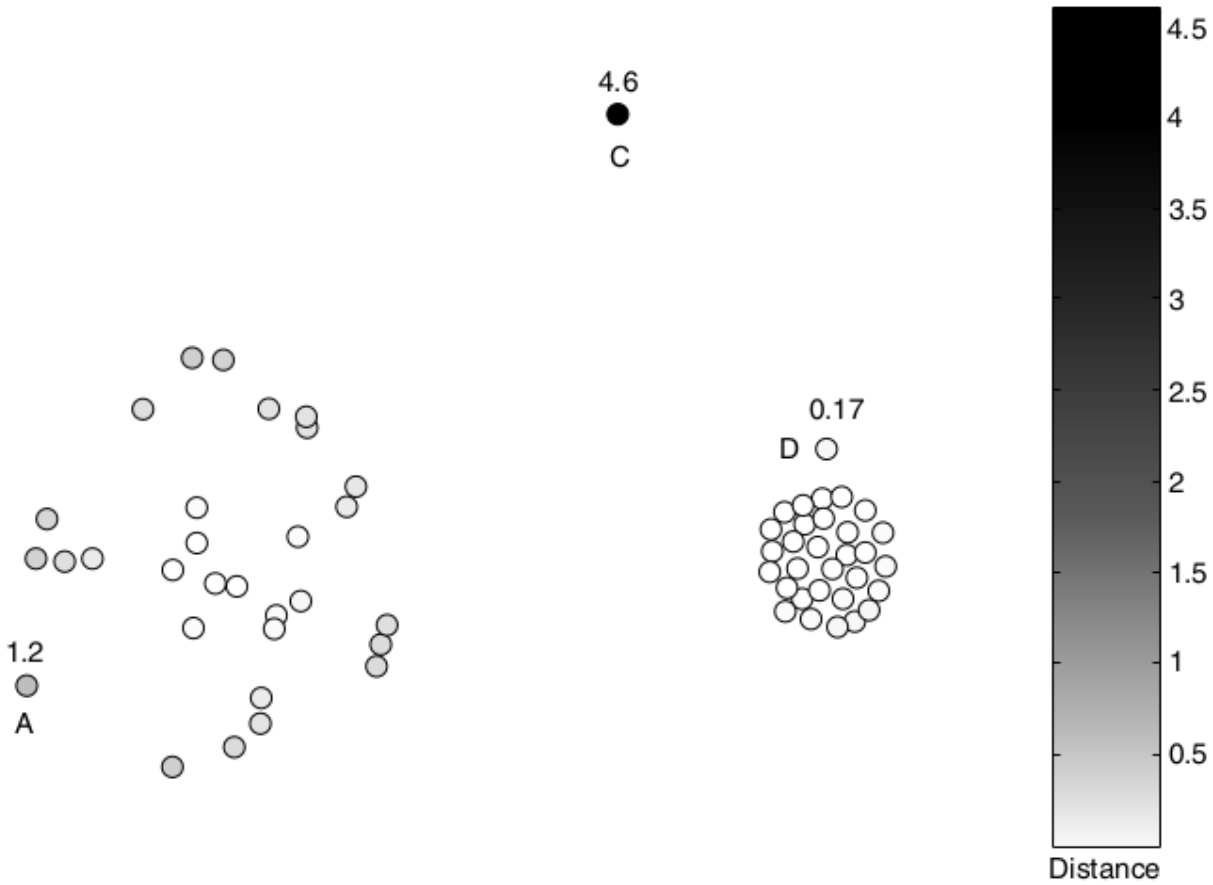
A better approach would be to asses how strongly objects belong to clusters.

Definition 4

An object is a cluster-based outlier if the object does not strongly belong to any cluster.

Depending on the particular clustering algorithm, we have (at least some) information about the cohesion of objects.

Anomaly Detection Using k-Means Clustering

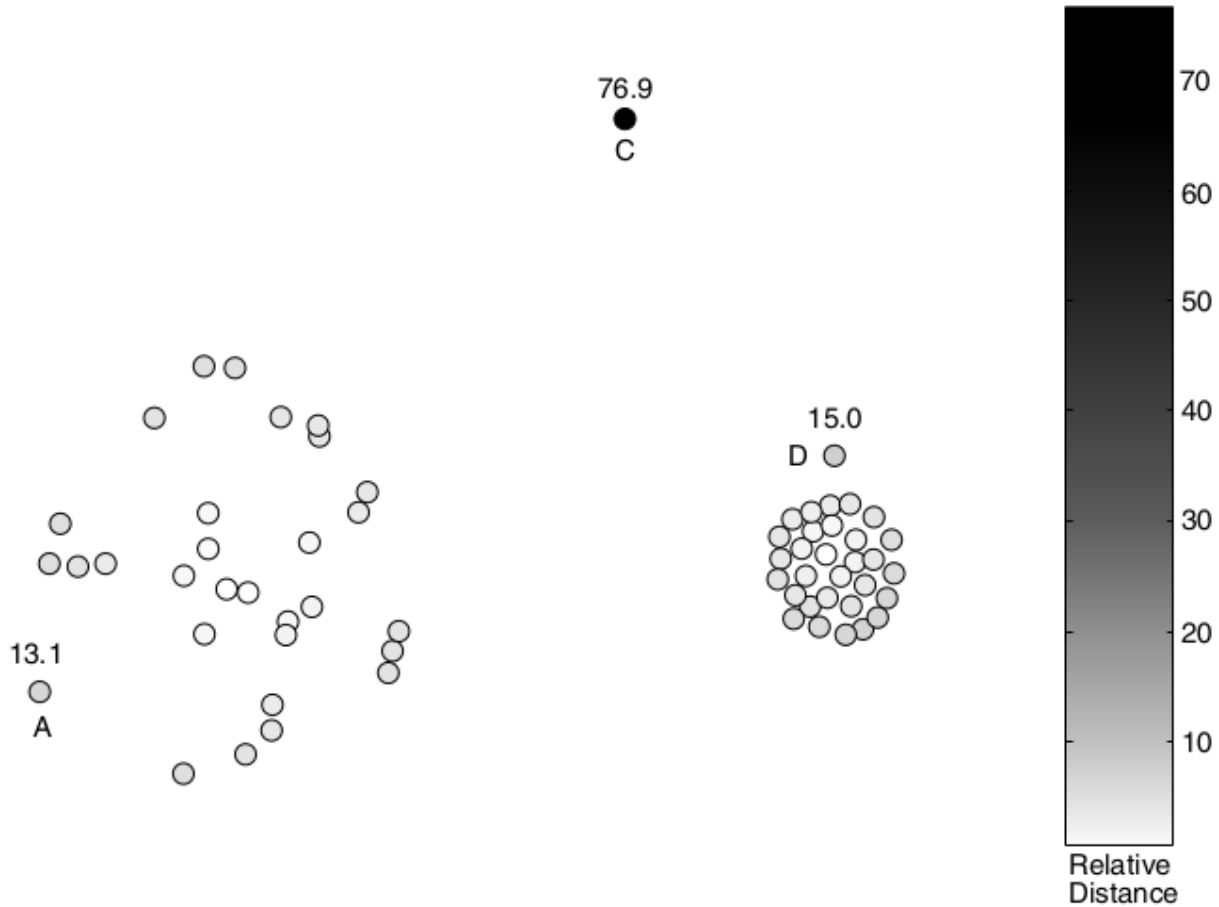


In k-means, the clusters are represented by centroids.

We may measure the anomaly of a given object using the distance to its cluster centroid.

This approach does not take into account the density of clusters.

Anomaly Detection Using k-Means Clustering



Here, we compute the relative distance to the cluster center, that is, the ratio of the object's distance to the centroid and the median distance of all objects of the same cluster to the centroid.

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the k-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the k-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Setting the proper number of clusters is an issue as well.

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the k-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Setting the proper number of clusters is an issue as well.

For anomaly detection, a larger number of smaller clusters may be beneficial as they may be more cohesive. If an object seems to be an outlier with small clusters, it is probably an outlier.

Comments on Clustering Based Methods

Some clustering methods have a sub-quadratic complexity.

Conceptually, clustering complements anomaly detection, so it is natural to compute both together.

On the other hand, the results strongly depend on the number of clusters, and anomalies may distort the clustering.

Autoencoders as Anomaly Detectors

... just a short comment

Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

However, what should we do with data with high or even variable dimensions?

- Images
- Text
- Video
- Semantic graphs
- ...

One way is to transform them into lower-dimensional data.

Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

However, what should we do with data with high or even variable dimensions?

- Images
- Text
- Video
- Semantic graphs
- ...

One way is to transform them into lower-dimensional data.

Train a neural network that takes the large input and returns a smaller representation, which (hopefully) preserves crucial features of the input.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

Training: Assume

$$\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

where $\vec{x}_i \in \mathbb{R}^n$ for all $i \in \{1, \ldots, n\}$.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

Training: Assume

$$\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

where $\vec{x}_i \in \mathbb{R}^n$ for all $i \in \{1, \ldots, n\}$.

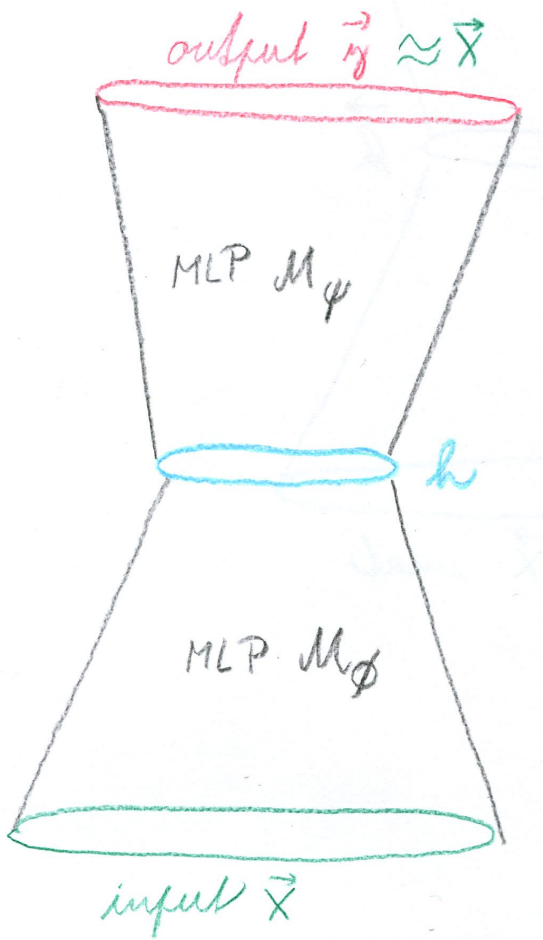
Minimize the **reconstruction** error

$$E = \sum_{i=1}^{p} (\vec{x}_i - \psi(\phi(\vec{x}_i)))^2$$

Autoencoders – neural networks

Both ϕ and ψ can be represented using MLP $\mathcal{M}_\phi$ and $\mathcal{M}_\psi$, respectively.

$\mathcal{M}_\phi$ and $\mathcal{M}_\psi$ can be connected into a single network.



Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.

Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.

Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.
- Generative versions – (roughly) generate $\vec{h}$ from a known distribution, let $\mathcal{M}_\psi$ generate realistic inputs/outputs $\vec{x}$

Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.
- Generative versions – (roughly) generate $\vec{h}$ from a known distribution, let $\mathcal{M}_\psi$ generate realistic inputs/outputs $\vec{x}$
- **Anomaly detection** (see the next slide)

Anomaly Detection with Autoencoders

Straightforward approach:

- Train the autoencoder on the normal data.
- Detect an anomaly using the large reconstruction error.

  The idea is that an anomaly will not be properly reconstructed.

Anomaly Detection with Autoencoders

Straightforward approach:

- Train the autoencoder on the normal data.
- Detect an anomaly using the large reconstruction error.

  The idea is that an anomaly will not be properly reconstructed.

More general approach:

- Train the autoencoder and transform the normal data to their low dimensional latent representations.
- Use one of the previous approaches to anomaly detection to detect anomalies in the latent representations.

  The assumption is that the latent representation of an anomaly will substantially differ from the latent representations of normal instances.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders
- Semi-supervised anomaly detection is usually concerned with data where the normal class is known.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders
- Semi-supervised anomaly detection is usually concerned with data where the normal class is known.
- There are many more methods: One class SVM, isolation forests, etc.