Lecture 10

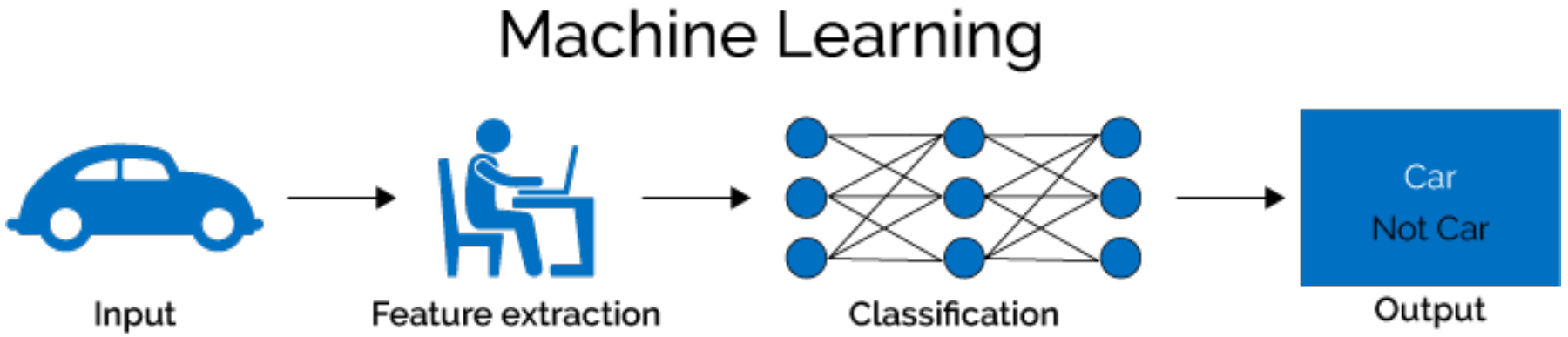
Deep Learning

Ioanna Miliou, PhD

Senior Lecturer, Stockholm University

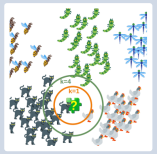
Stockholms universitet

Classification

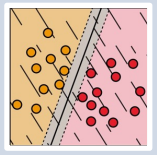


Stockholms universitet

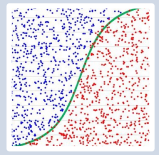
Decision Tree



K-Nearest Neighbor

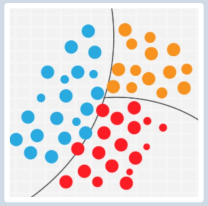


Support Vector Machine

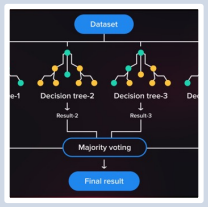


Logistic Regression

Stockholms universitet



Naive Bayes Classifier



Random Forest (bagging)

AdaBoost (boosting)

Stockholms universitet

Stacking

Algorithm Stacking

1: Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
2: Ouput: ensemble classifier H
3: *Step 1: learn base-level classifiers*
4: **for** $t = 1$ to T **do**
5: learn h_t based on D
6: **end for**
7: *Step 2: construct new data set of predictions*
8: **for** $i = 1$ to m **do**
9: $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
10: **end for**
11: *Step 3: learn a meta-classifier*
12: learn H based on D_h
13: return H

new dataset for the meta model

Stacking

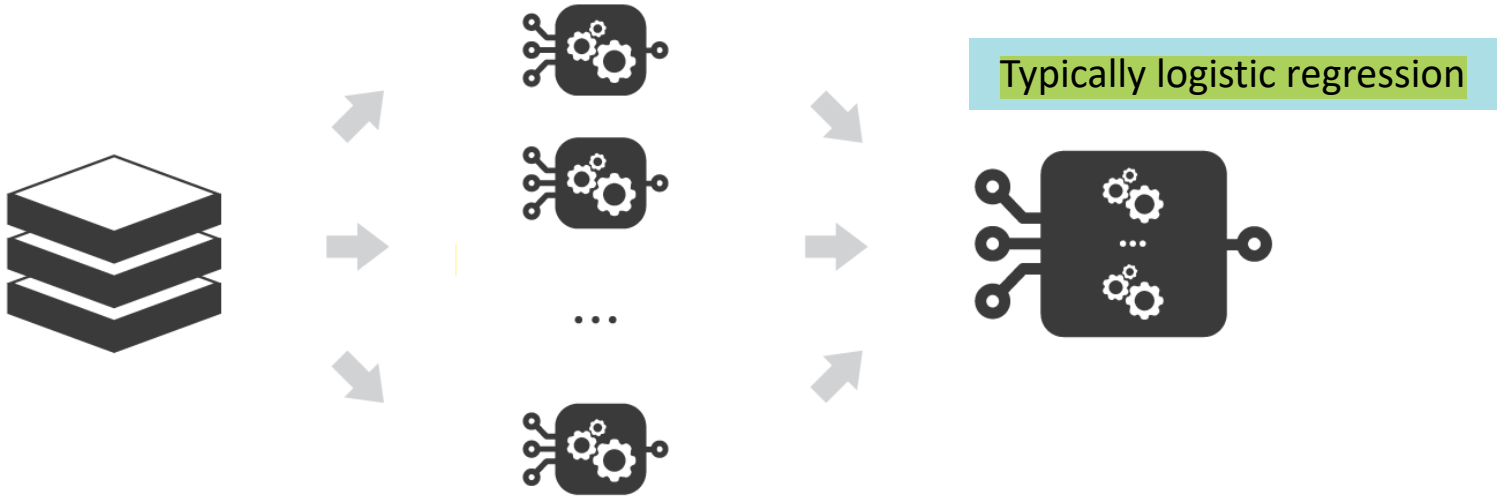
Algorithm Stacking

1: Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
2: Ouput: ensemble classifier H
3: *Step 1: learn base-level classifiers*

- Base models are trained on a complete training set
- Then, a meta-model is trained on the outputs of the base-level model as features

10: **end for**
11: *Step 3: learn a meta-classifier*
12: learn H based on D_h
13: return H

Stacking - overview

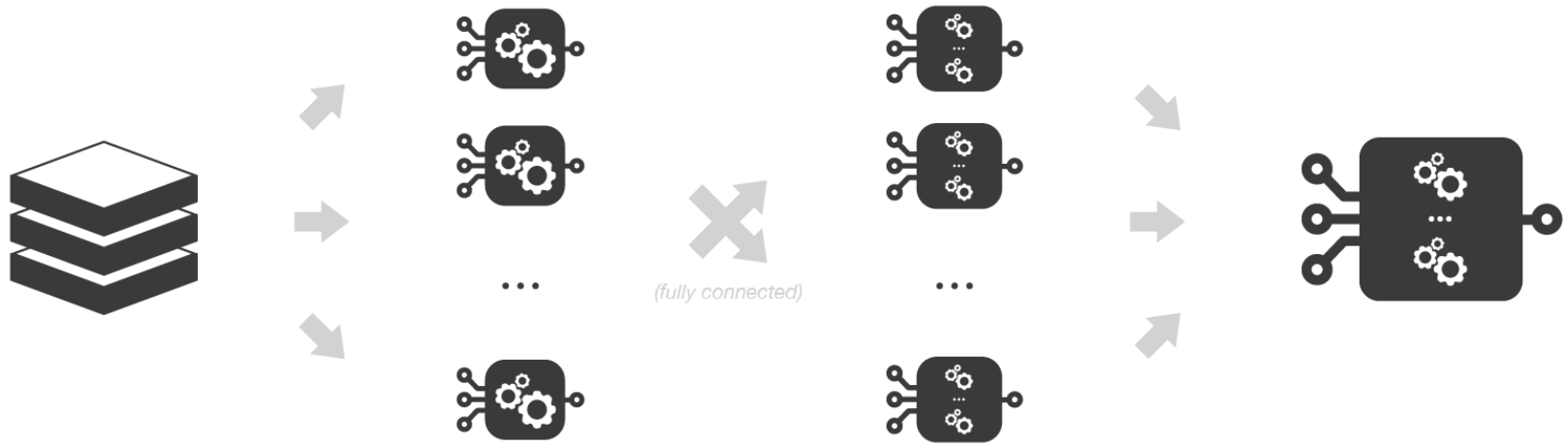


Stacking consists in training a meta-model to produce outputs based on the outputs returned by some lower layer weak learners.

Stockholms universitet

Multi-level stacking

Fit M meta-models instead of one, and then combine them into a final meta-model



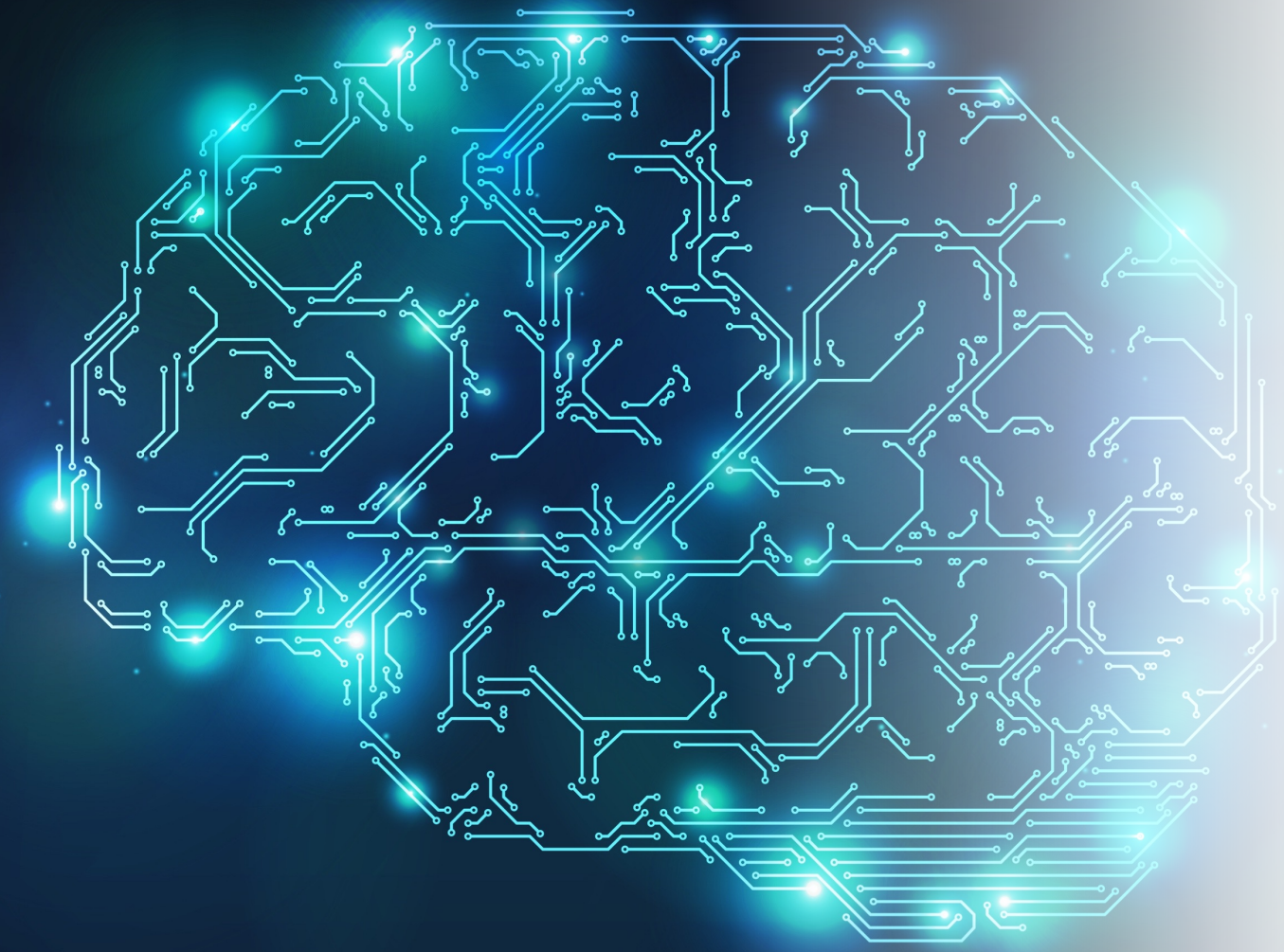
initial dataset

L weak learners
(that can be non-homogeneous)

M meta-models
(**trained** to output predictions based
on previous layer predictions)

final meta-model
(**trained** to output predictions based
on previous layer predictions)

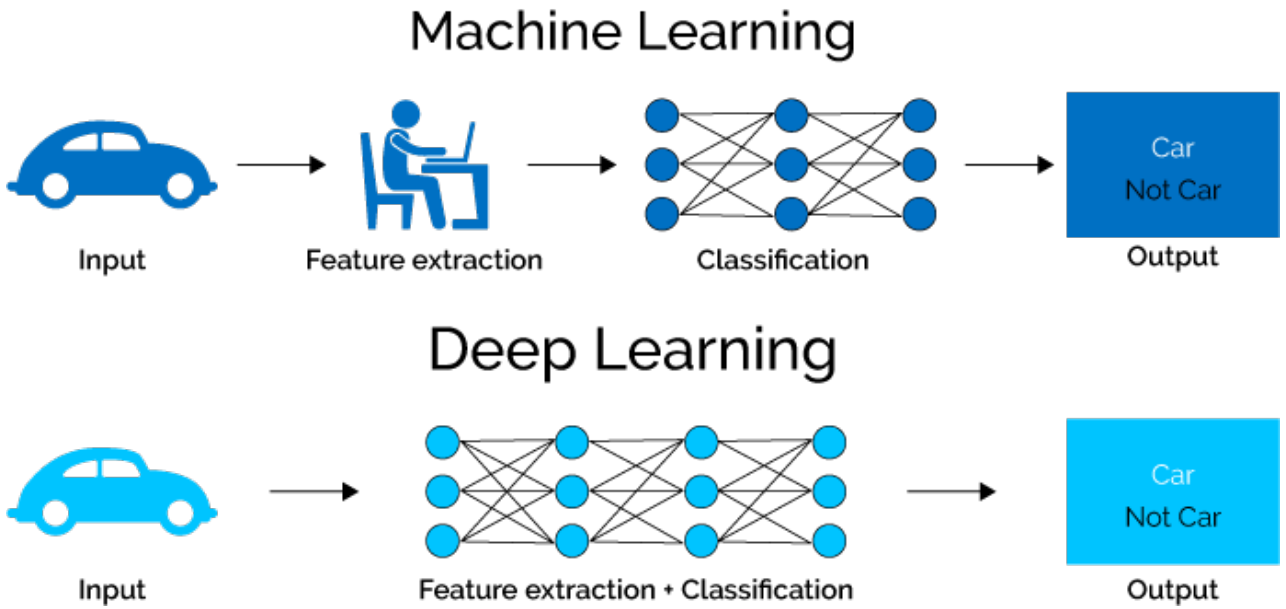
Stockholms universitet



Deep Learning

Deep Learning

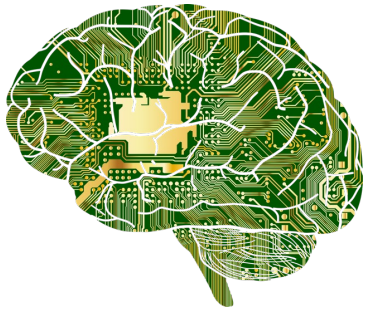
- A subfield of machine learning using Artificial Neural Networks for learning representations of data
- Exceptionally effective at learning hidden patterns



Stockholms universitet

Deep Learning

- Deep learning algorithms attempt to learn (multiple levels of) feature representations by using a hierarchy of multiple layers
- If you provide "tons" of information as input, it begins to understand it and respond in useful ways

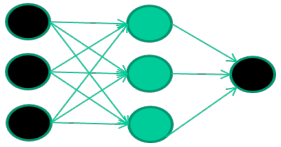


The series of layers between input & output do **feature identification and processing in a series of stages**, just as our brains do!

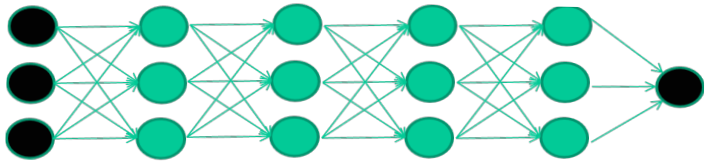
Why now?

- Multi-layer neural networks have been around for 60 years
- What is actually new?

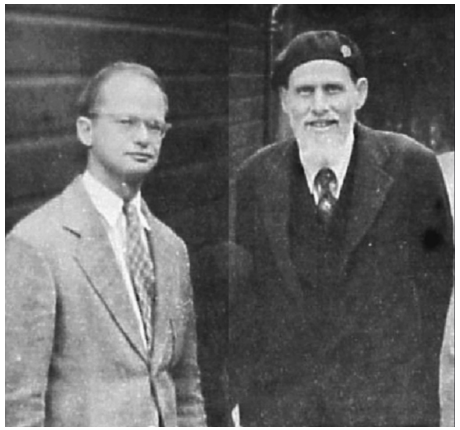
We have always had good algorithms for learning the weights in networks with 1 hidden layer



but these algorithms are not good at learning the weights for networks with more hidden layers



What is new: algorithms for training multi-layer networks, the availability of massive amounts of data, computing power



McCulloch & Pitts, 1943

"A logical calculus of the ideas immanent in nervous activity"

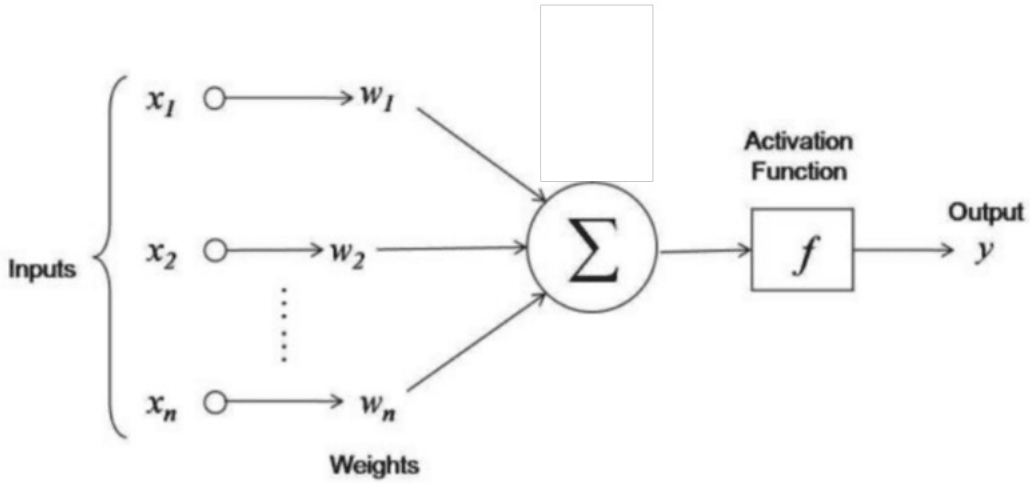
tried to understand how the brain could produce highly complex patterns by using many basic cells that are connected

The simplest ANN

- The perceptron
  - An artificial neuron
  - n inputs: **X = <x_1, x_2, …, x_n>**
  - n weights: **W = <w_1, w_2, w_3, …, w_n>**
  - Summation function (**dot product**):

    $x_1w_1 + x_2w_2 + x_3w_3 + \ldots$
  - Activation function:

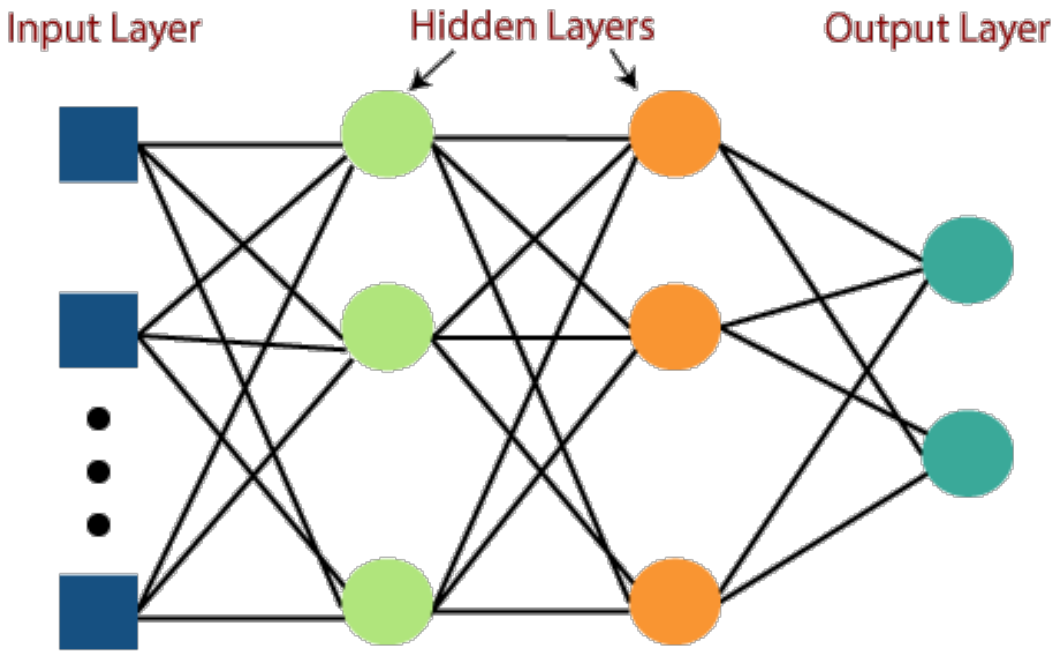
    f (X*W) > 0
  - Output**:** a class label **y**



Stockholms universitet

A multi-layer perceptron

- More complex problems may require intermediate layers



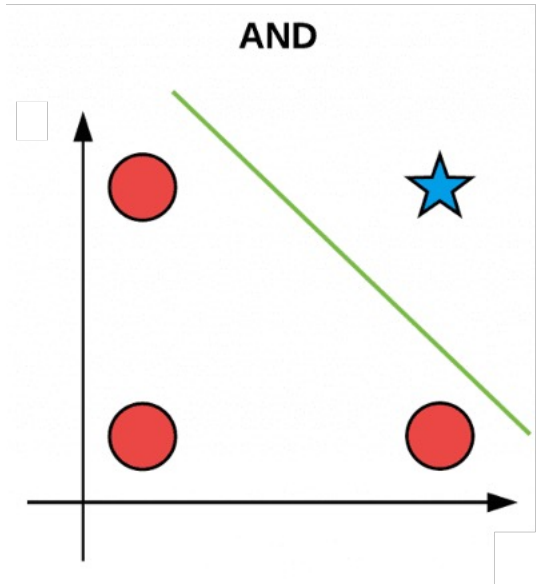
Stockholms universitet

Perceptron Learning Theorem

- *Recap*: A perceptron (artificial neuron) can learn anything that it can represent (i.e., anything separable with a hyperplane)

AND Function

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



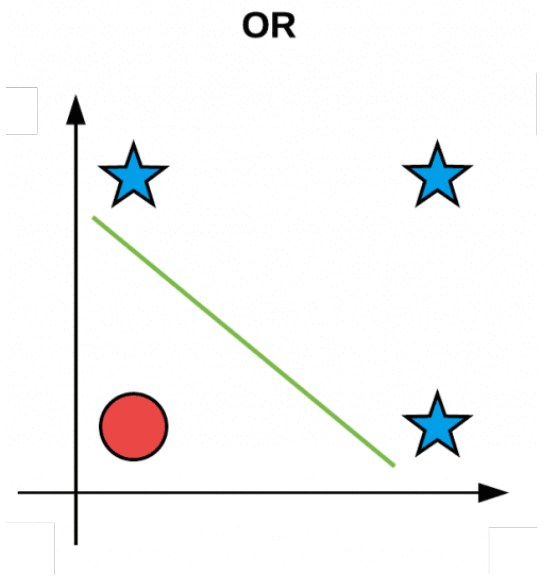
Stockholms universitet

Perceptron Learning Theorem

- *Recap*: A perceptron (artificial neuron) can learn anything that it can represent (i.e., anything separable with a hyperplane)

OR Function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



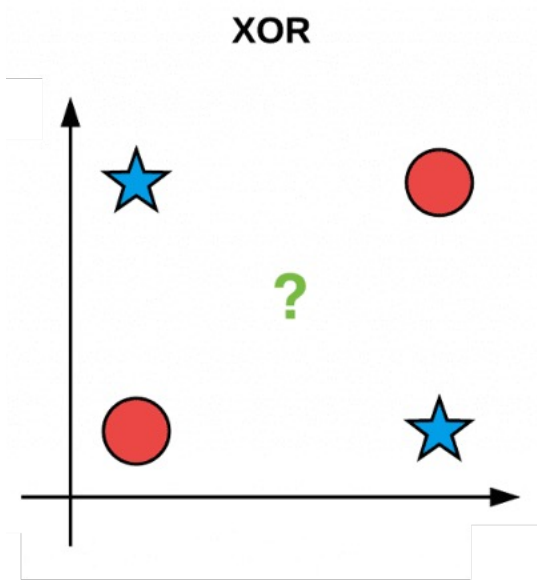
Stockholms universitet

The Exclusive OR (XOR) problem

- A Perceptron cannot represent XOR since it is not linearly separable

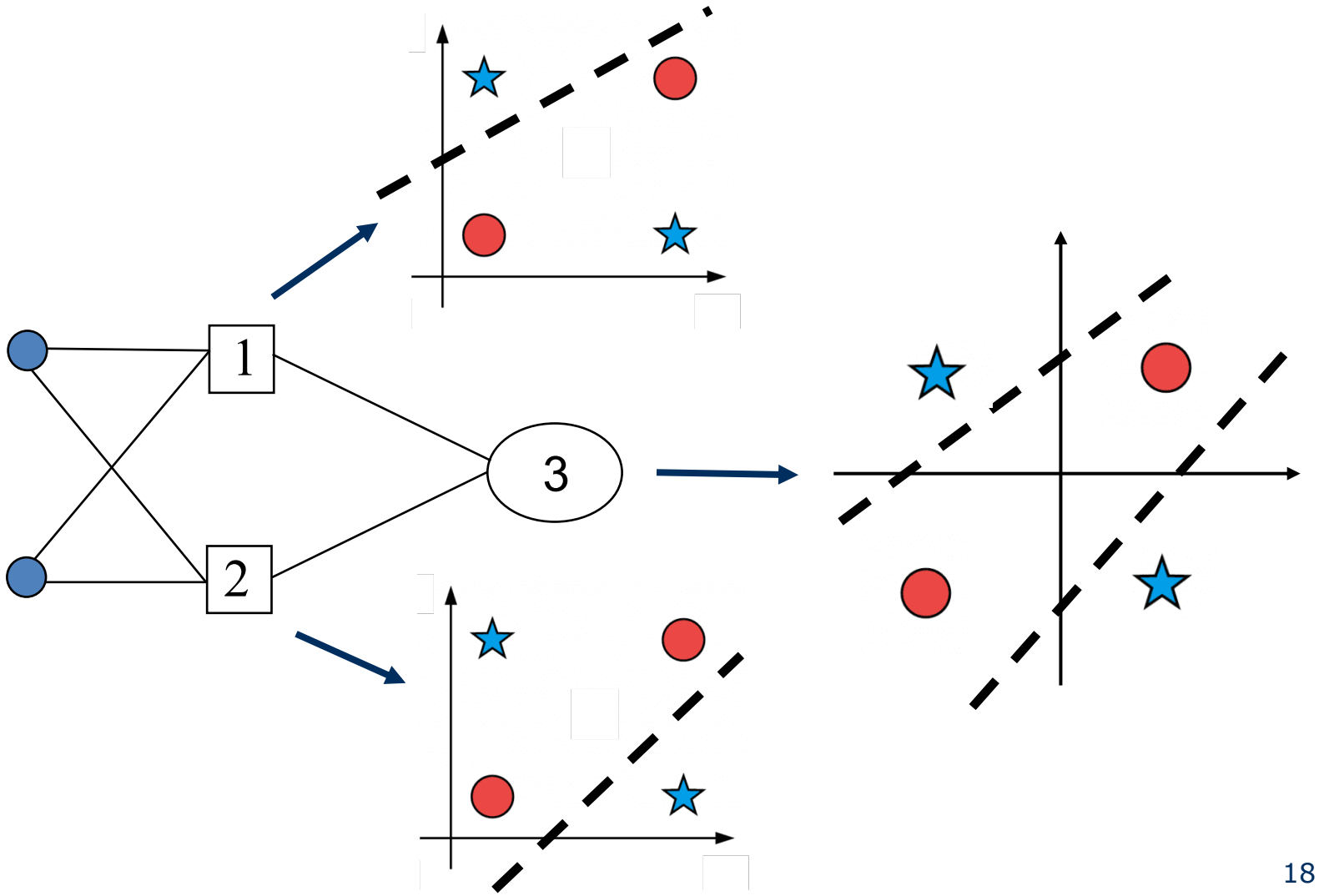
XOR Function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



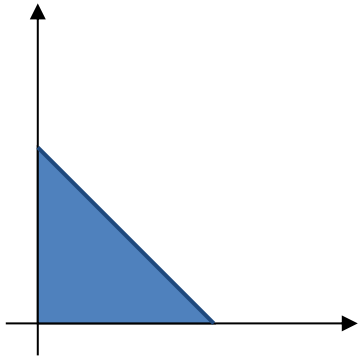
Stockholms universitet

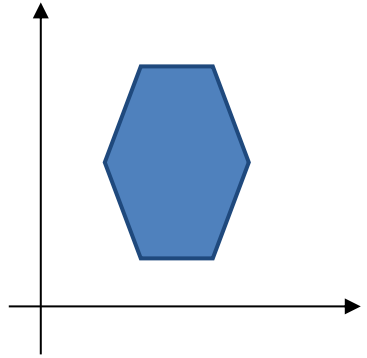
Minsky & Papert (1969) offered a solution to the XOR problem by combining perceptron unit responses using a second layer of units!

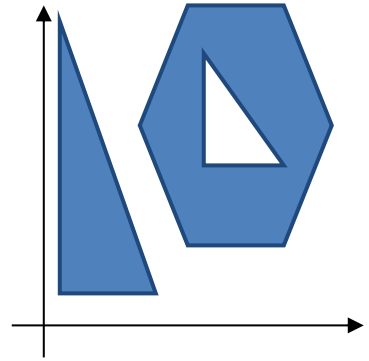


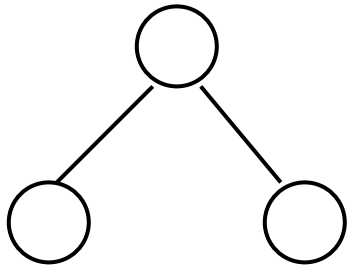
Stockholms universitet

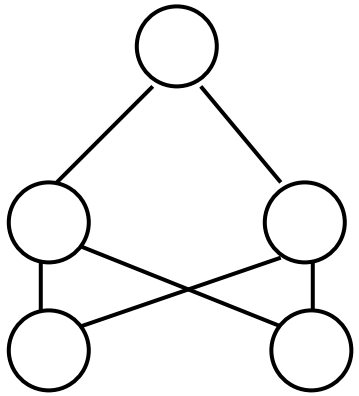
Multiple layers: their role

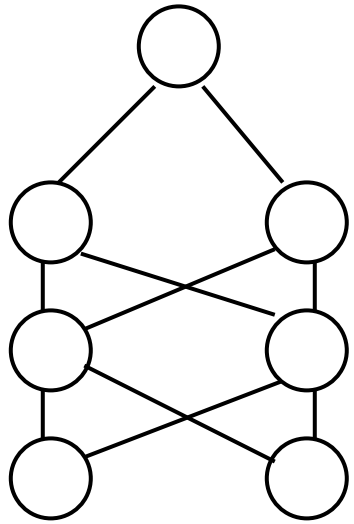












1st layer draws linear boundaries

2nd layer combines the boundaries

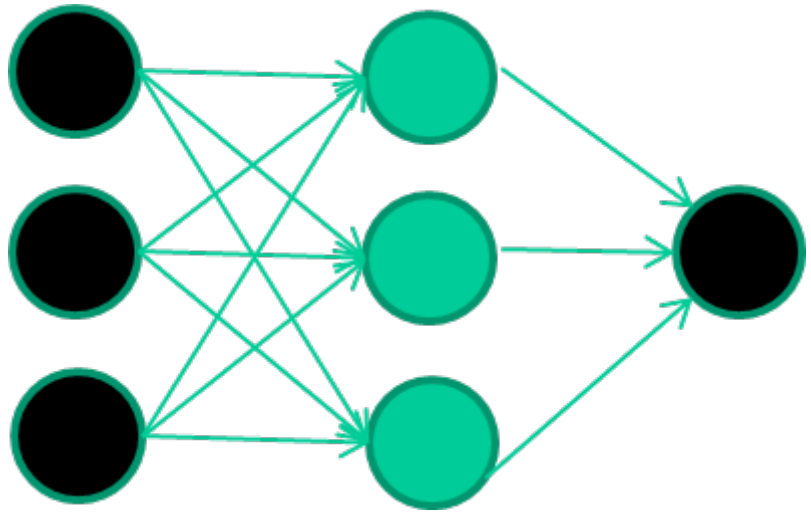
3rd layer can generate arbitrarily complex boundaries

An example

A dataset

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

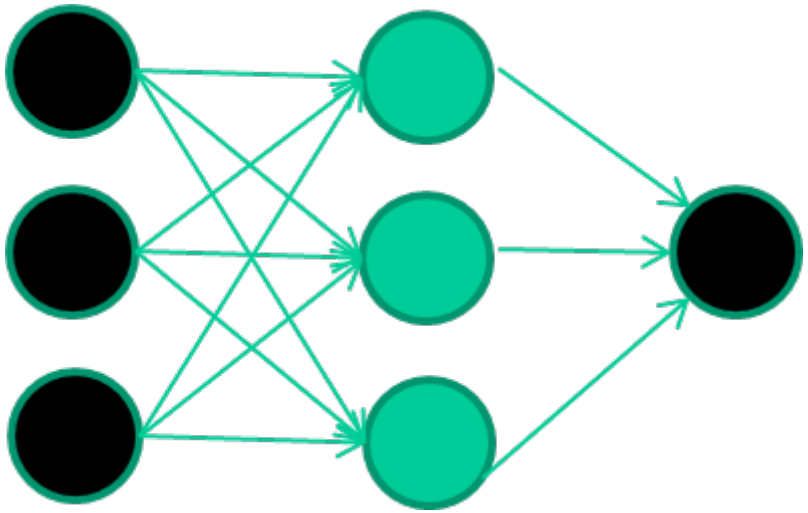


An example

Training the neural network

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



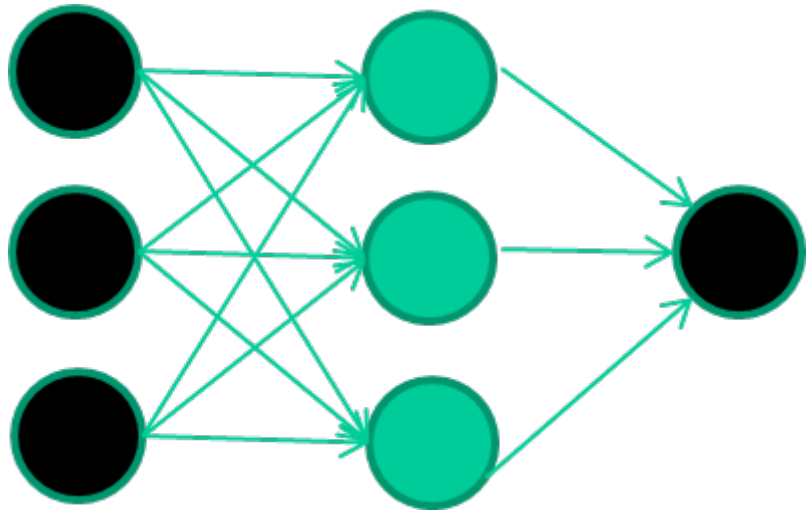
An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Initialise with random weights

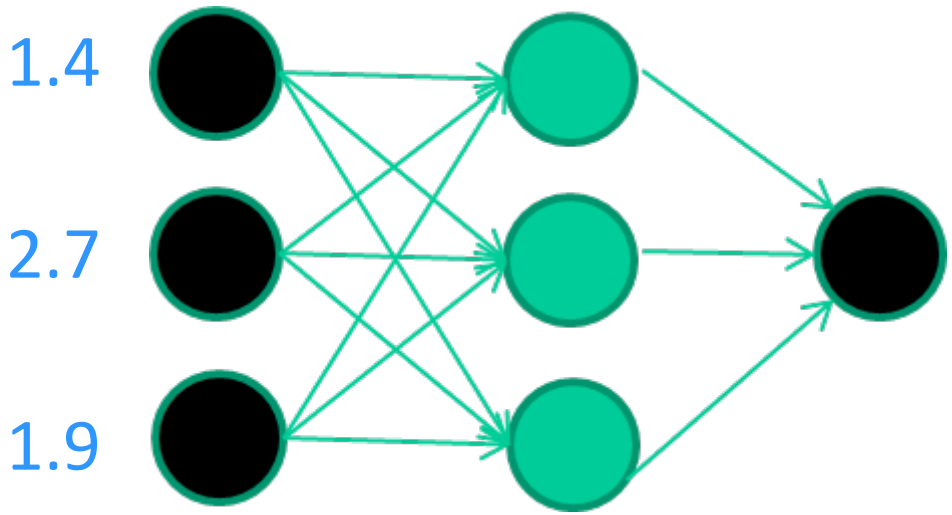


An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern



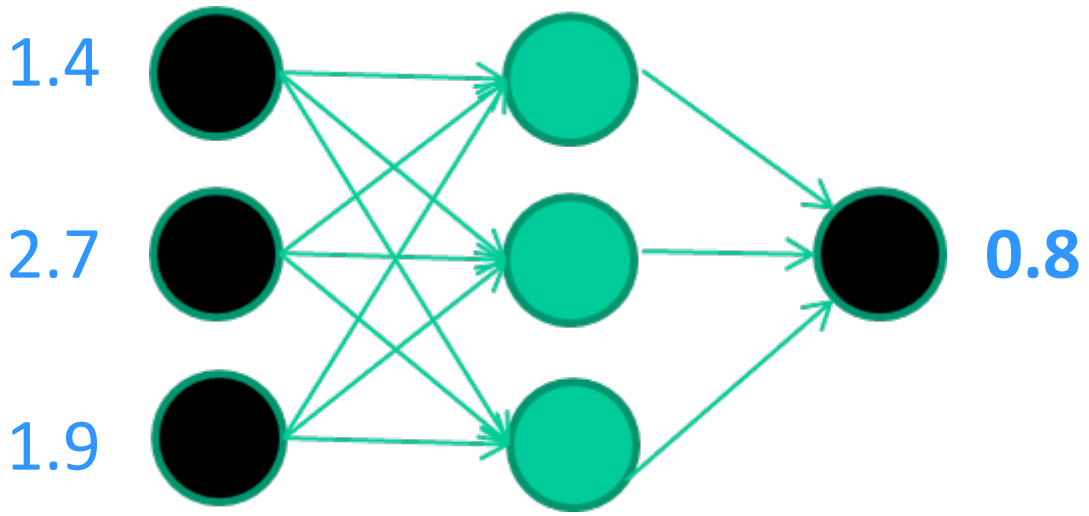
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get the output



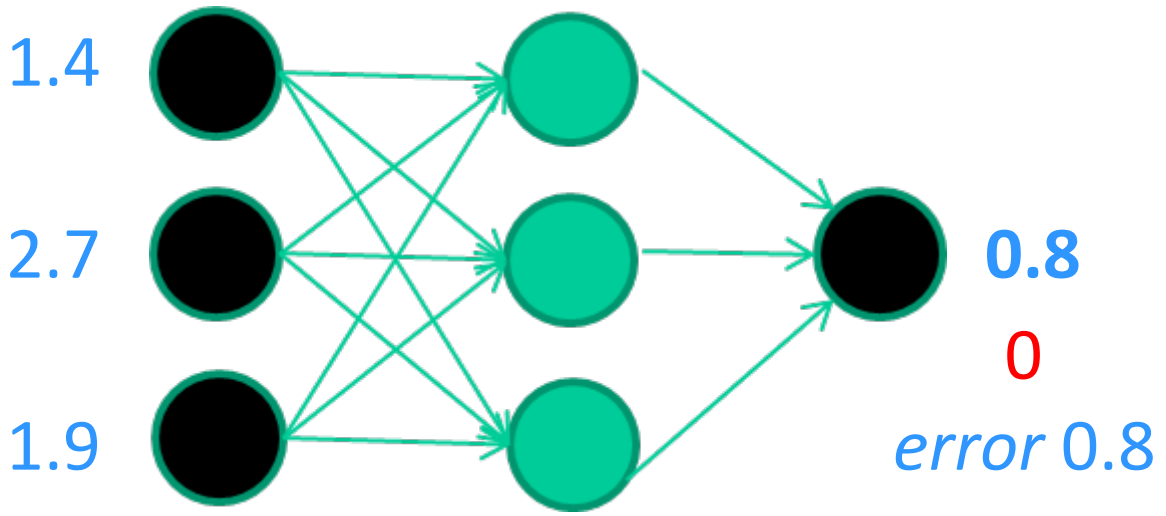
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with the target output



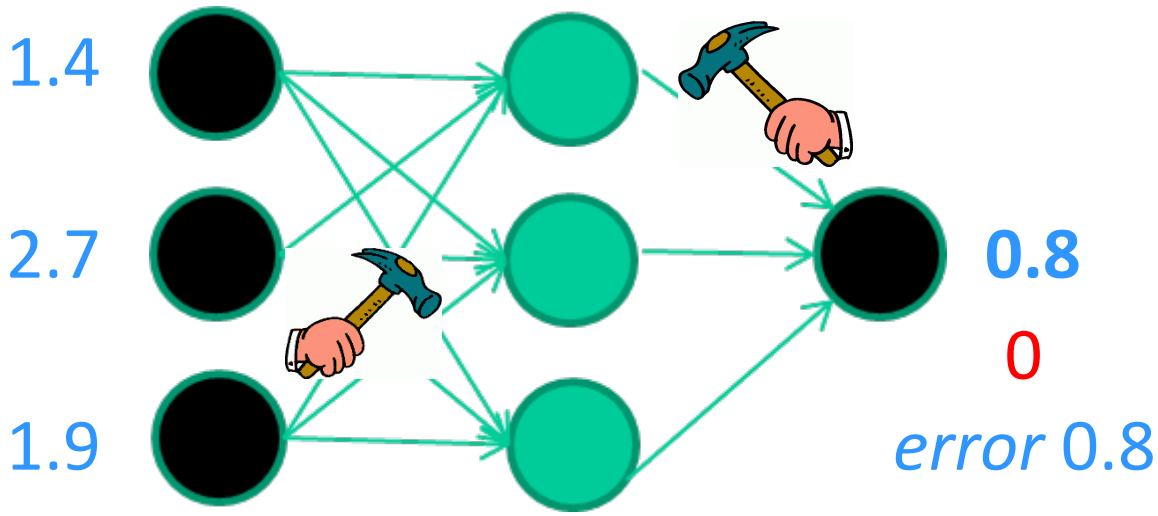
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error



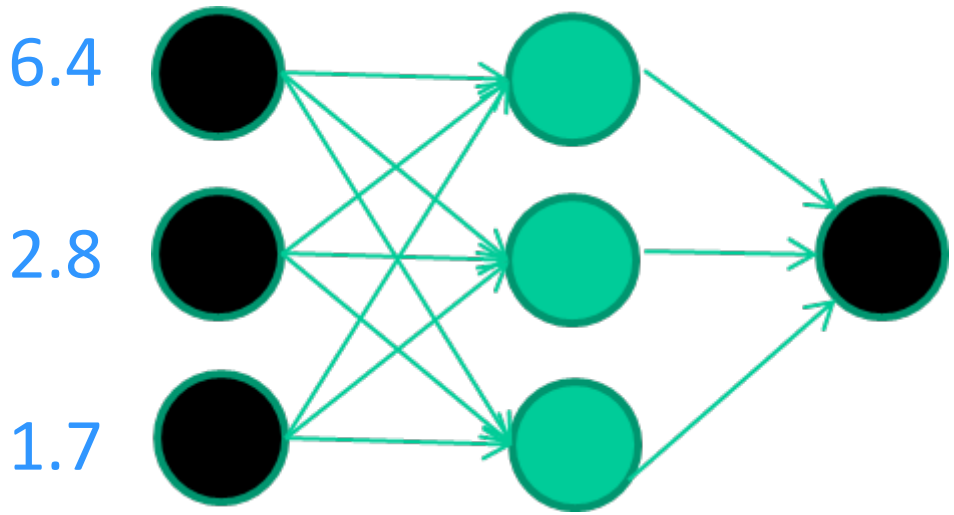
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern



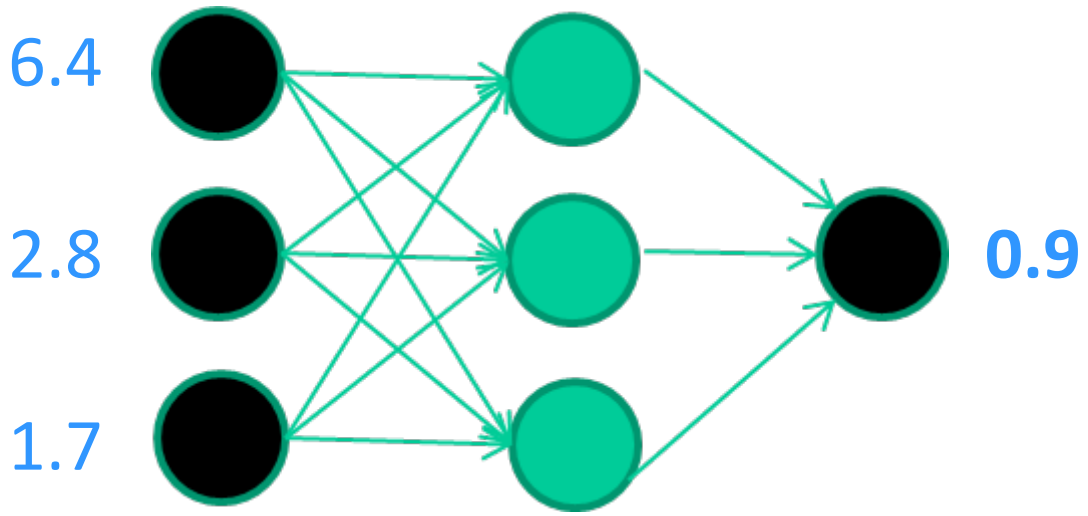
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get the output



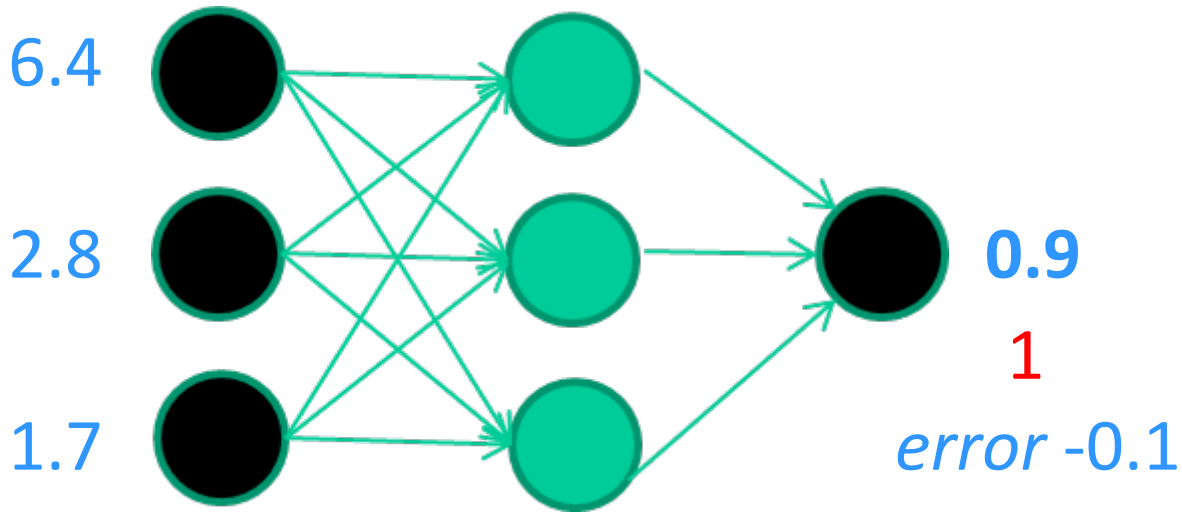
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with the target output



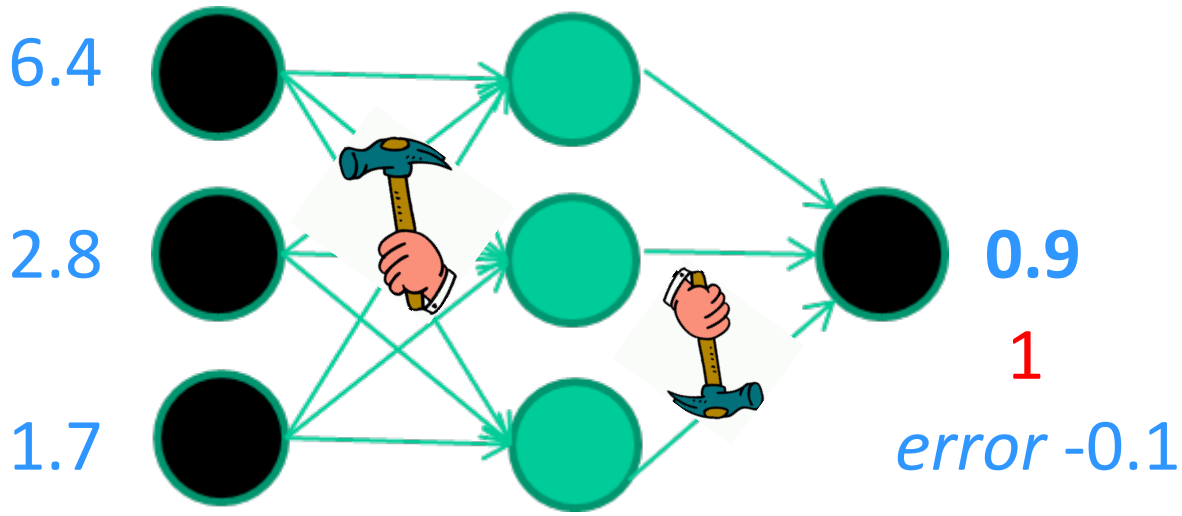
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error



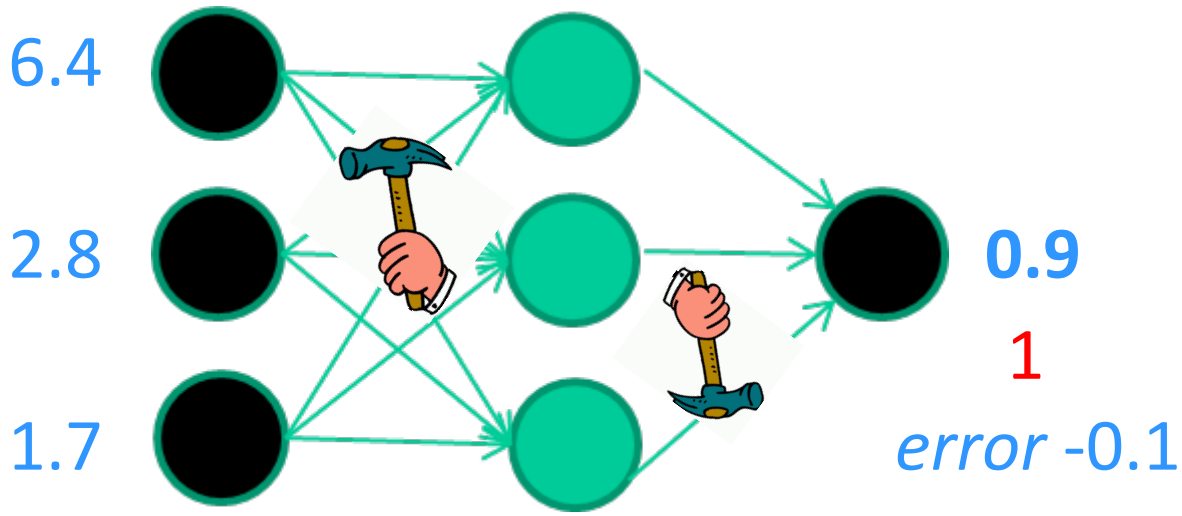
Stockholms universitet

An example

Training data

Fields			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

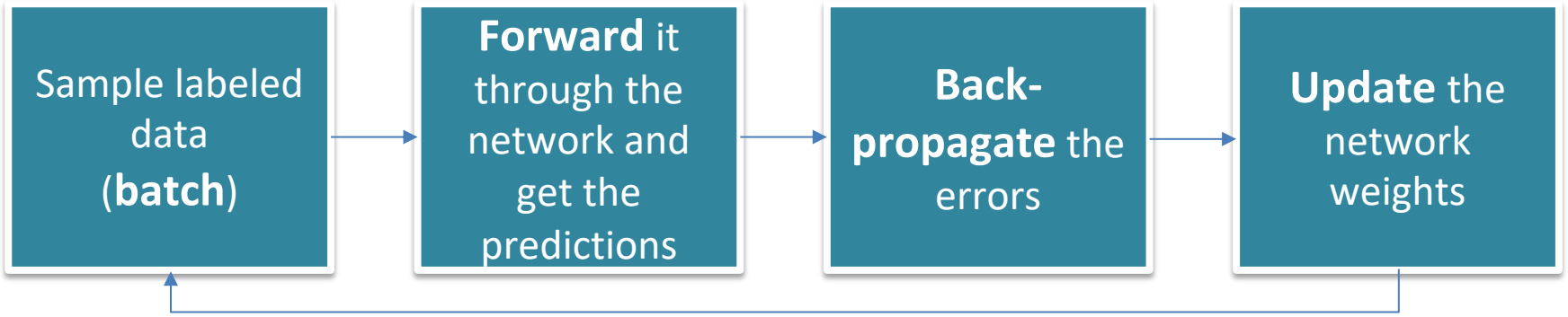
And so on ….



- Repeat this, thousands, maybe millions of times – each time taking a random training instance and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

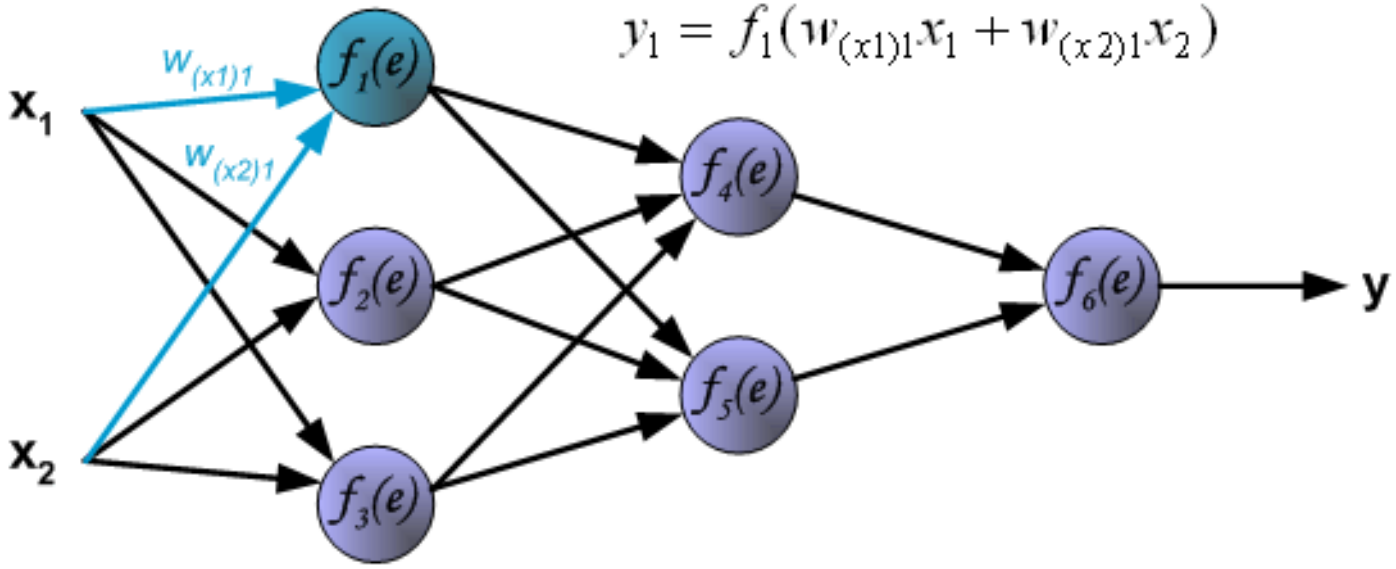
Training Neural Networks



- Optimize (min or max) objective/cost function
- Generate an error signal that measures the difference between the predictions and the target values
- Use error signal to change the weights and get more accurate predictions
- **Epoch:** one pass over the training set

Learning Algorithm: forward propagation

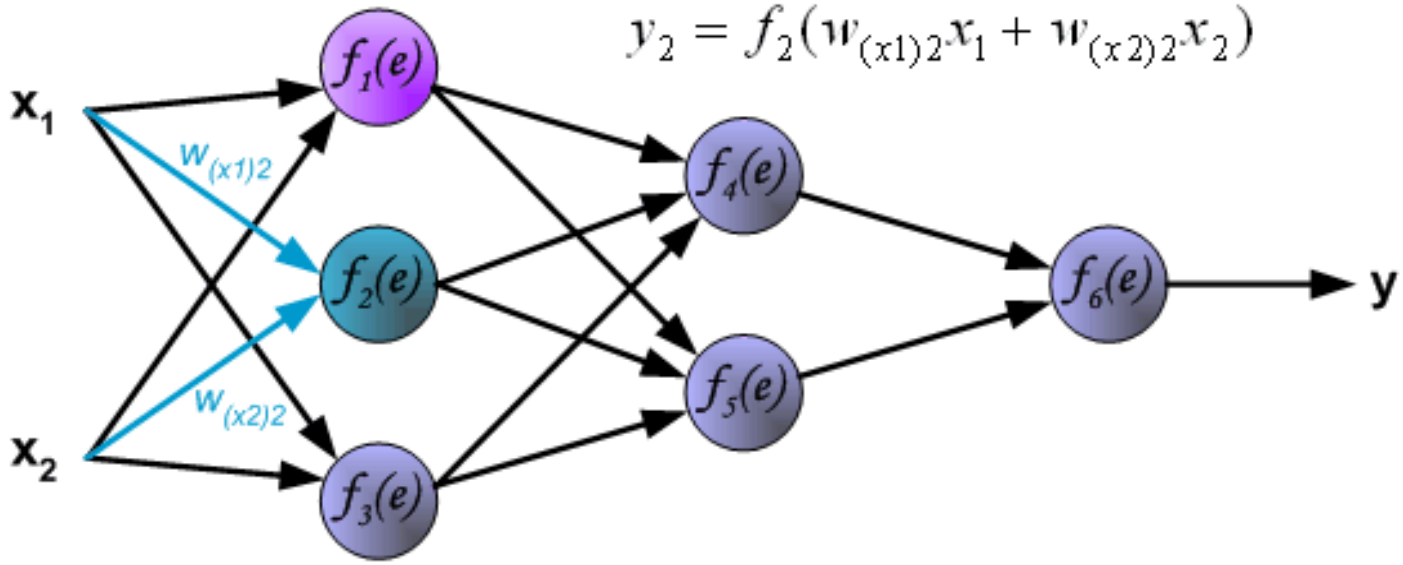
- The “signal” is propagating through the network
- $w_{(xm)n}$: weights of connections between network input x_m and neuron n in the input layer
- y_n: represents the output signal of neuron n



Stockholms universitet

Learning Algorithm: forward propagation

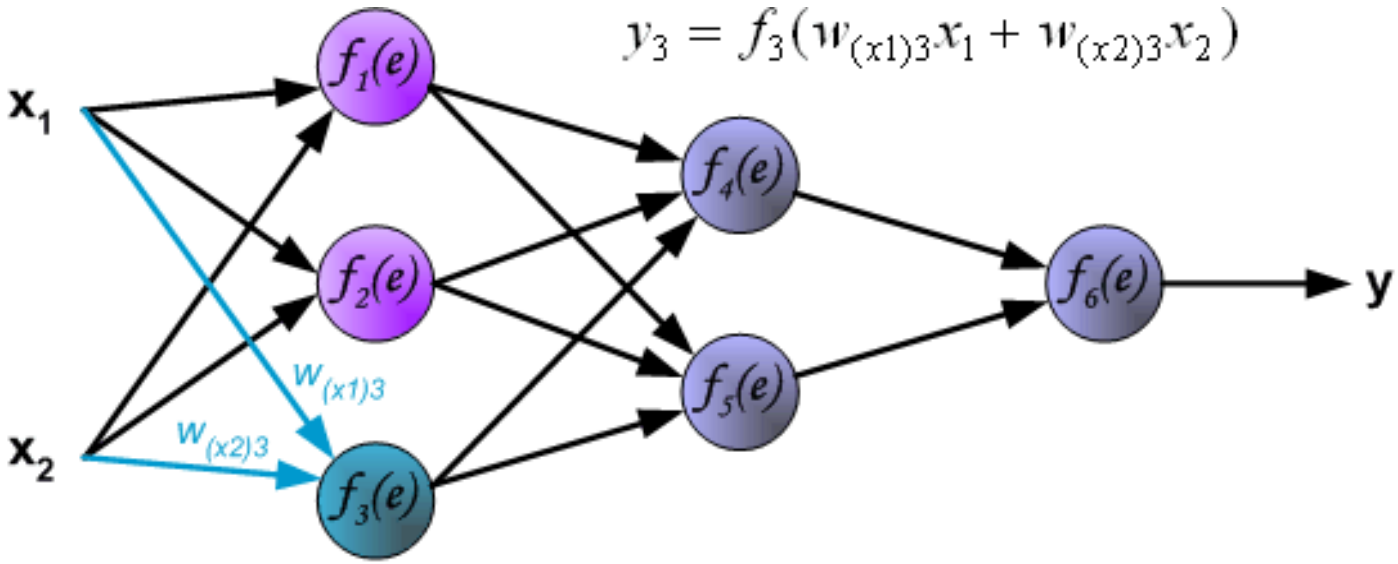
- The “signal” is propagating through the network
- $w_{(xm)n}$: weights of connections between network input x_m and neuron n in the input layer
- y_n: represents the output signal of neuron n



Stockholms universitet

Learning Algorithm: forward propagation

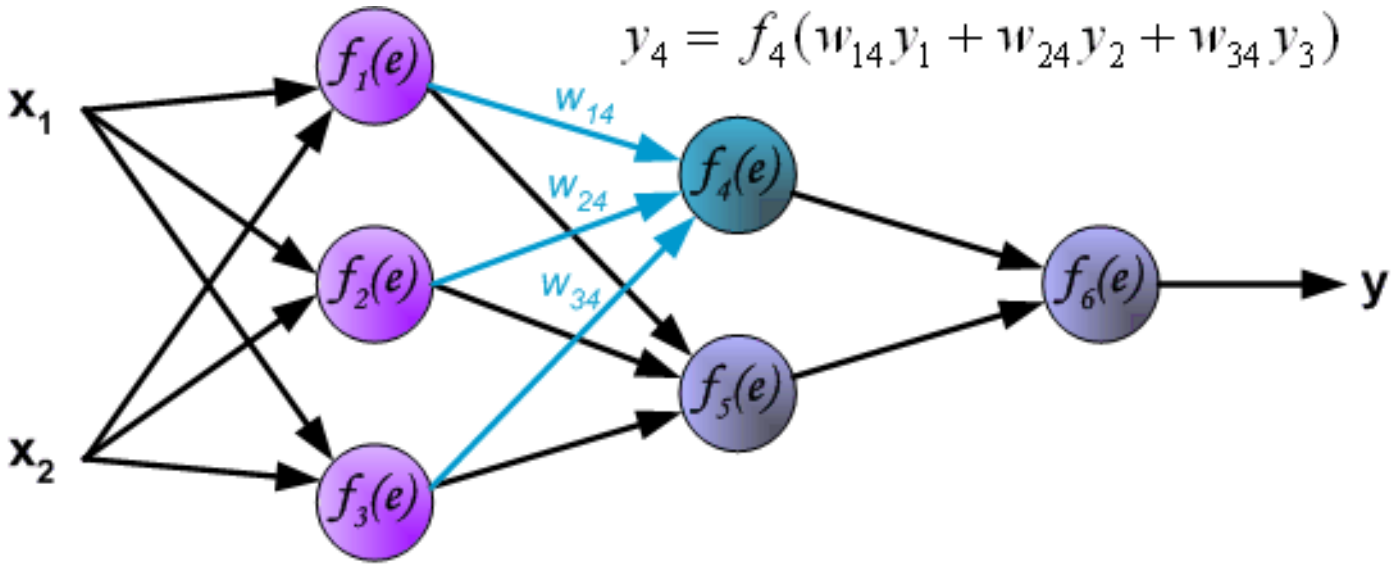
- The “signal” is propagating through the network
- $w_{(xm)n}$: weights of connections between network input x_m and neuron n in the input layer
- y_n: represents the output signal of neuron n



Stockholms universitet

Learning Algorithm: forward propagation

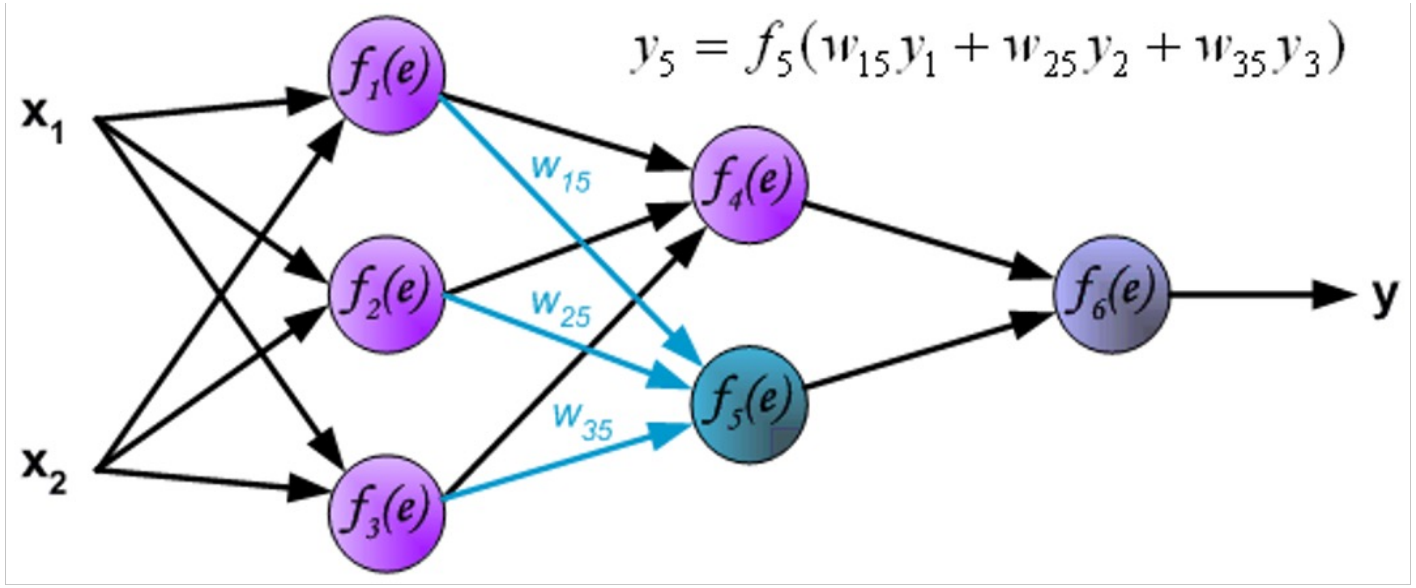
- Propagation of signals through the hidden layer
- w_{mn}: weights of connections between the output of neuron ***m*** and input of neuron ***n*** in the next layer



Stockholms universitet

Learning Algorithm: forward propagation

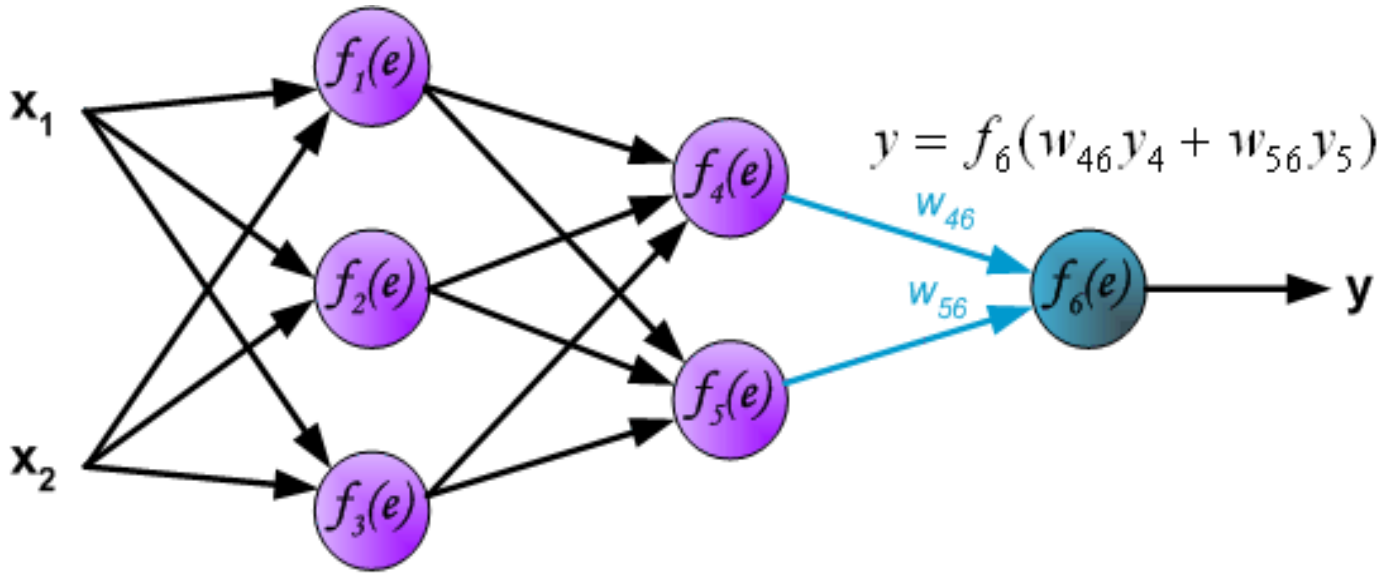
- Propagation of signals through the hidden layer
- w_{mn}: weights of connections between the output of neuron ***m*** and input of neuron ***n*** in the next layer



Stockholms universitet

Learning Algorithm: forward propagation

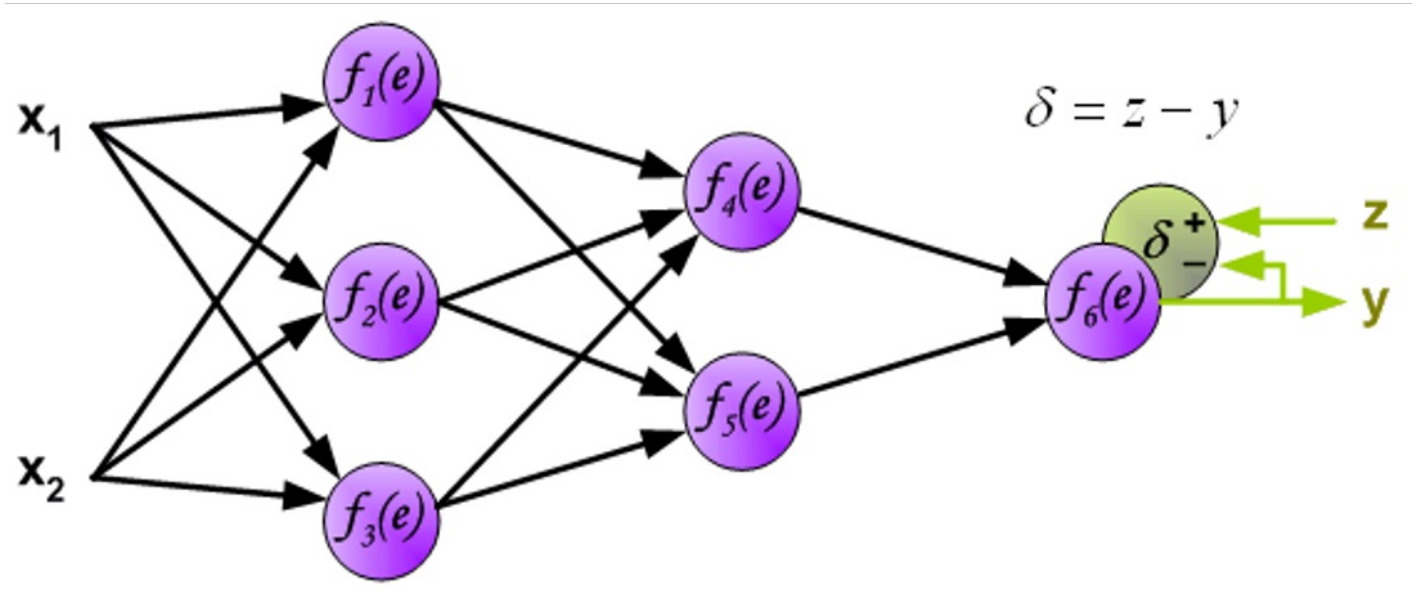
- Propagation of signals through the hidden layer
- w_{mn}: weights of connections between the output of neuron ***m*** and input of neuron ***n*** in the next layer



Stockholms universitet

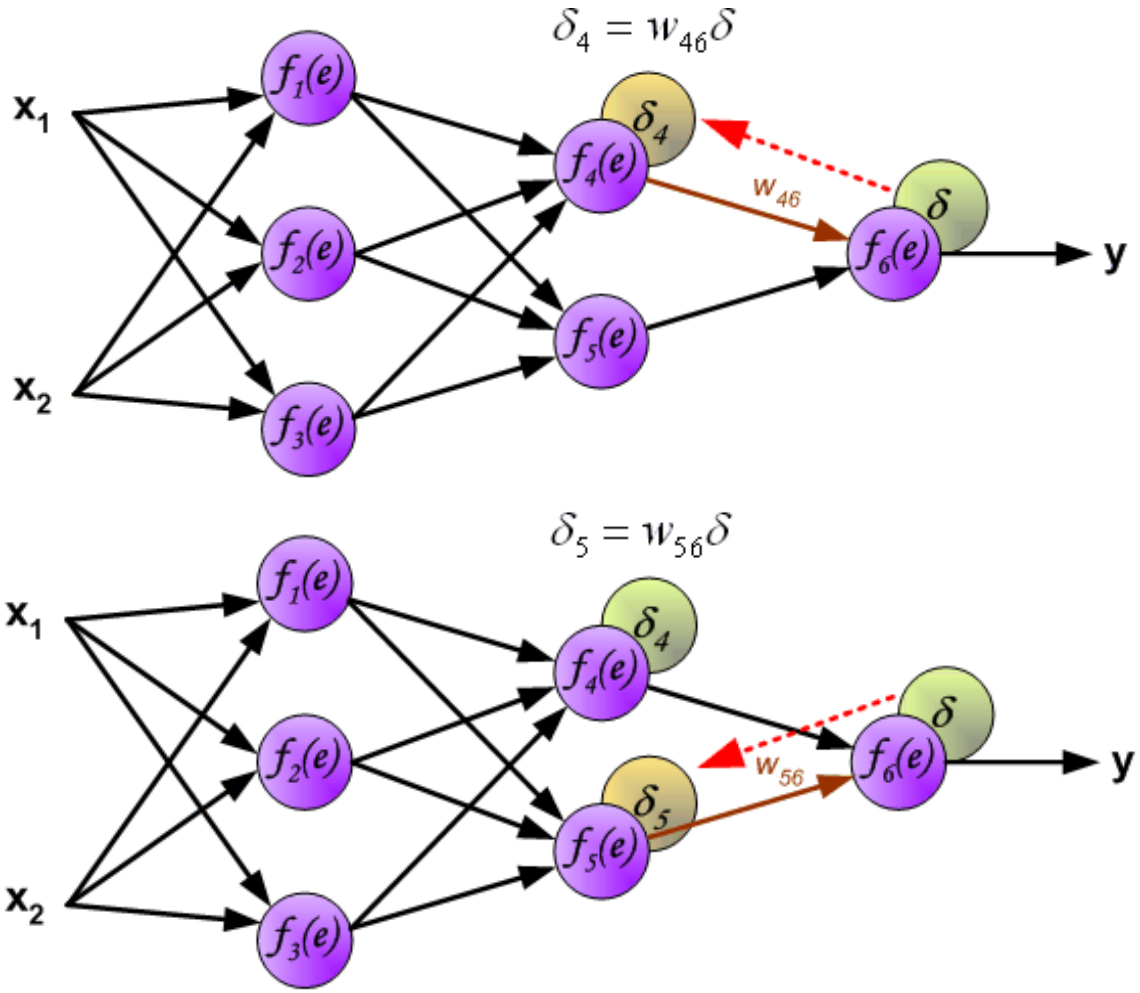
Learning Algorithm: forward propagation

- The output signal of the network ***y*** is compared with the desired output value ***z*** (the target), which is found in the training data set
- The difference is called error signal δ of the output layer neuron



Stockholms universitet

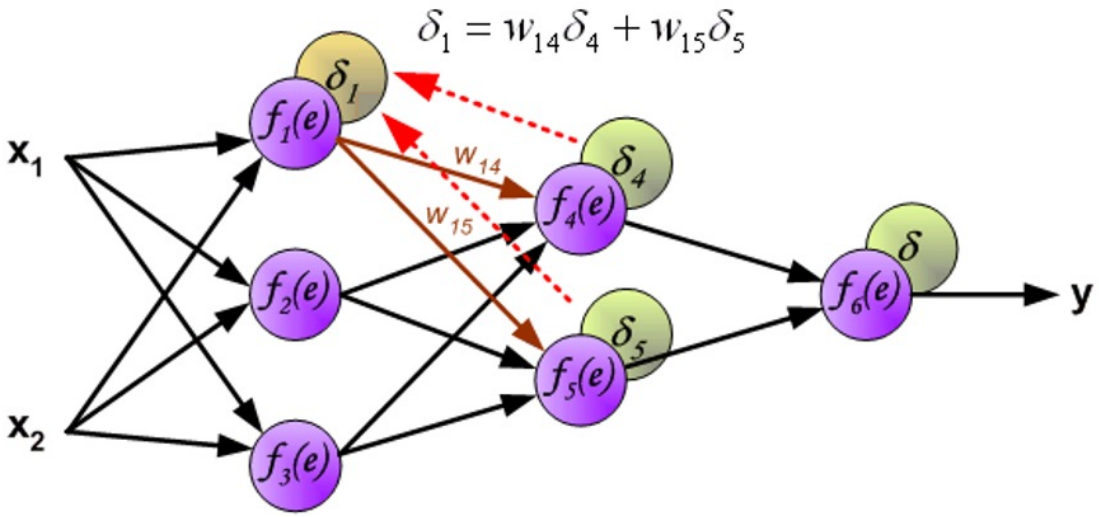
Learning Algorithm: backpropagation



The idea is to propagate the error δ (computed in a single teaching step) back to all neurons

Stockholms universitet

Learning Algorithm: backpropagation

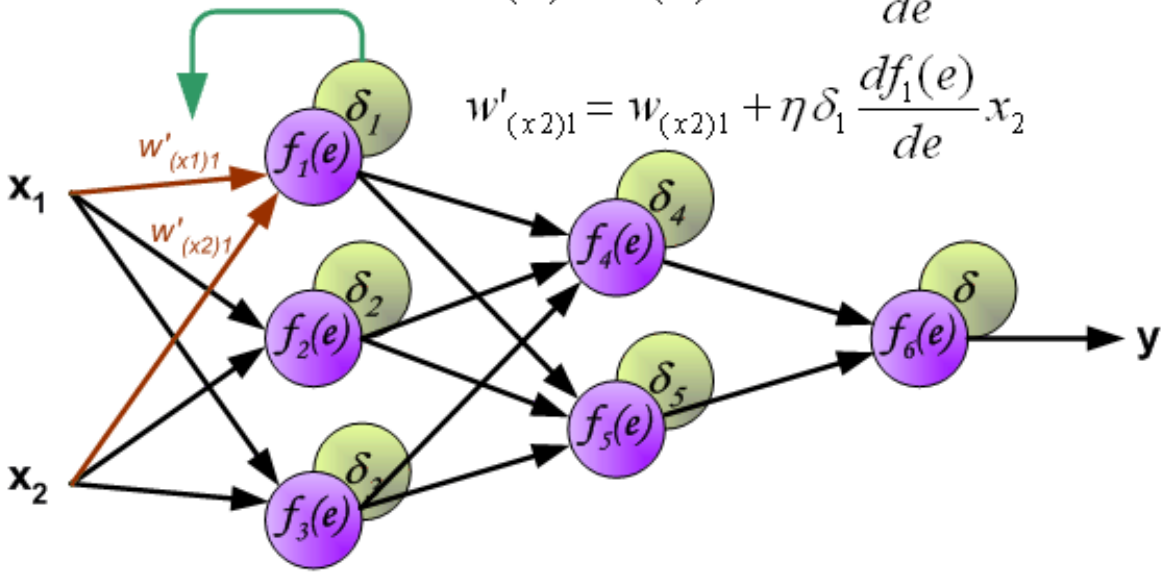


- The weight coefficients w_{mn} used to propagate errors back are equal to the ones used during the computation of the output value
- Only the direction of data flow is changed (signals are propagated from output to inputs one after the other)
- This technique is used for all network layers
- If propagated errors come from many neurons, they are added

Learning Algorithm: backpropagation

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



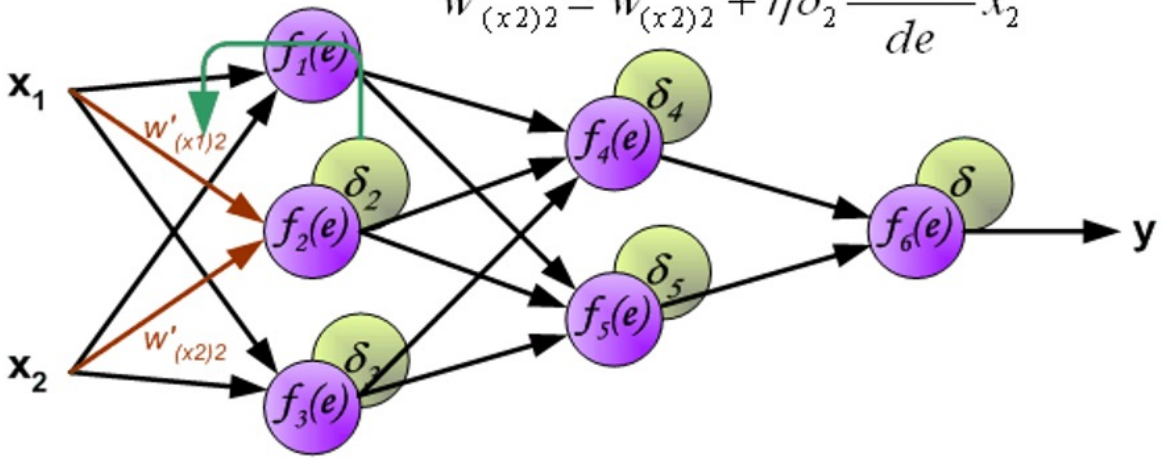
- When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified
- *df(e)/de* is the derivative of the neuron activation function (weights are modified)
- ***η*** is the learning rate (default 0.01): smaller ***η*** requires more epochs

Stockholms universitet

Learning Algorithm: backpropagation

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

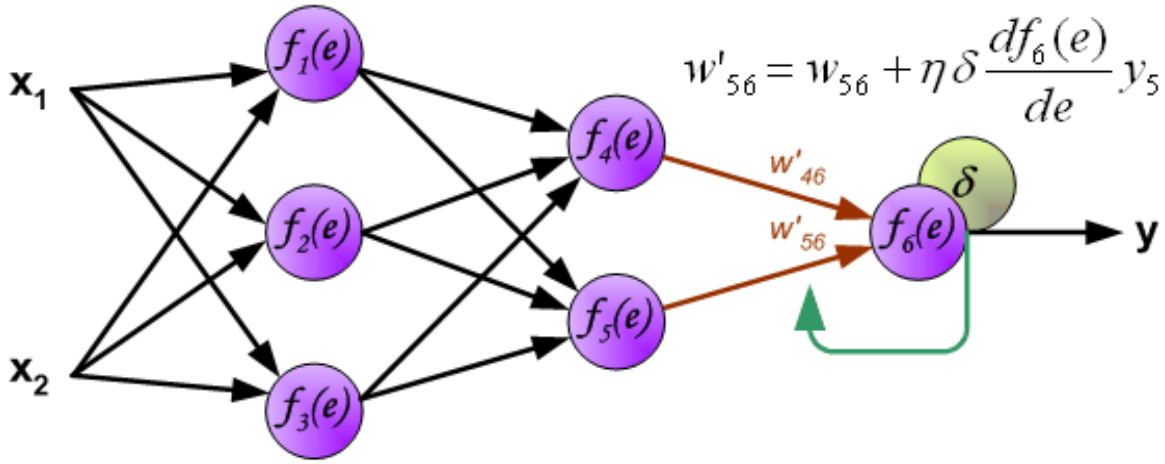


- When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified
- *df(e)/de* is the derivative of the neuron activation function (weights are modified)
- η is the learning rate (default 0.01): smaller η requires more epochs

Learning Algorithm: backpropagation

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

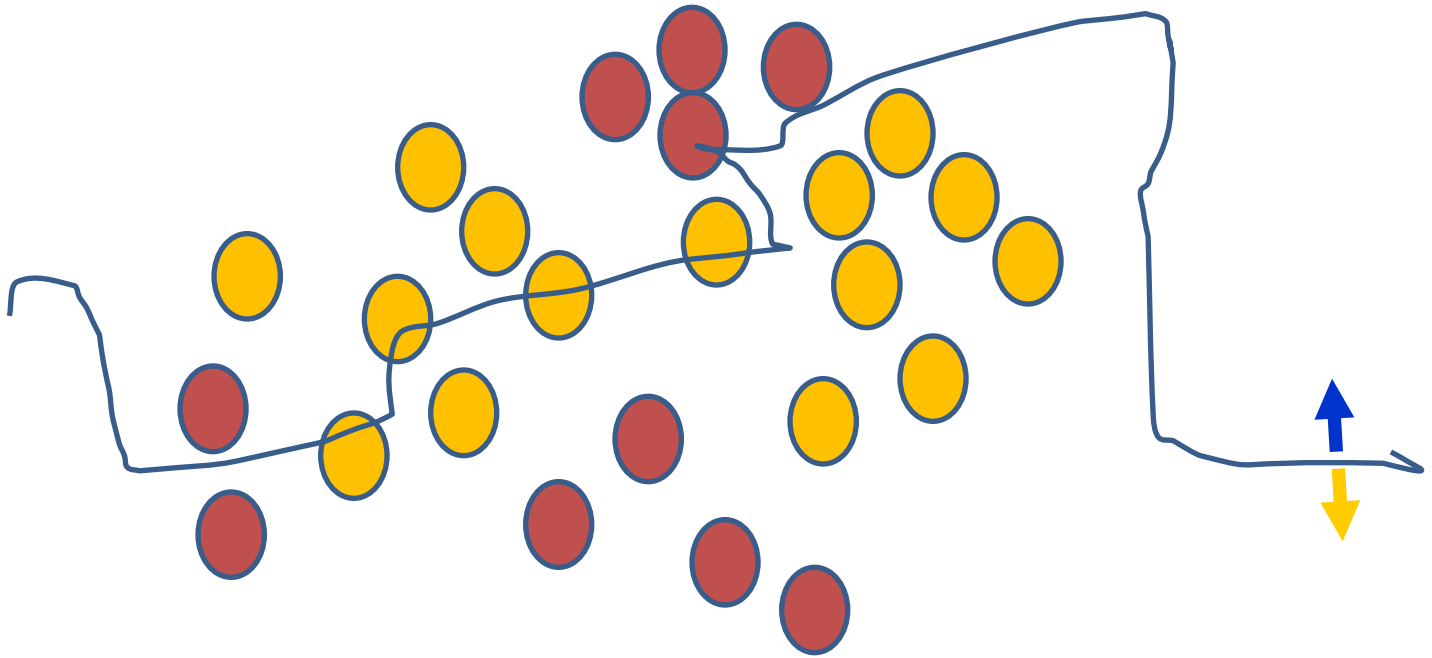
$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



- When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified
- *df(e)/de* is the derivative of the neuron activation function (weights are modified)
- ***η*** is the learning rate (default 0.01): smaller ***η*** requires more epochs

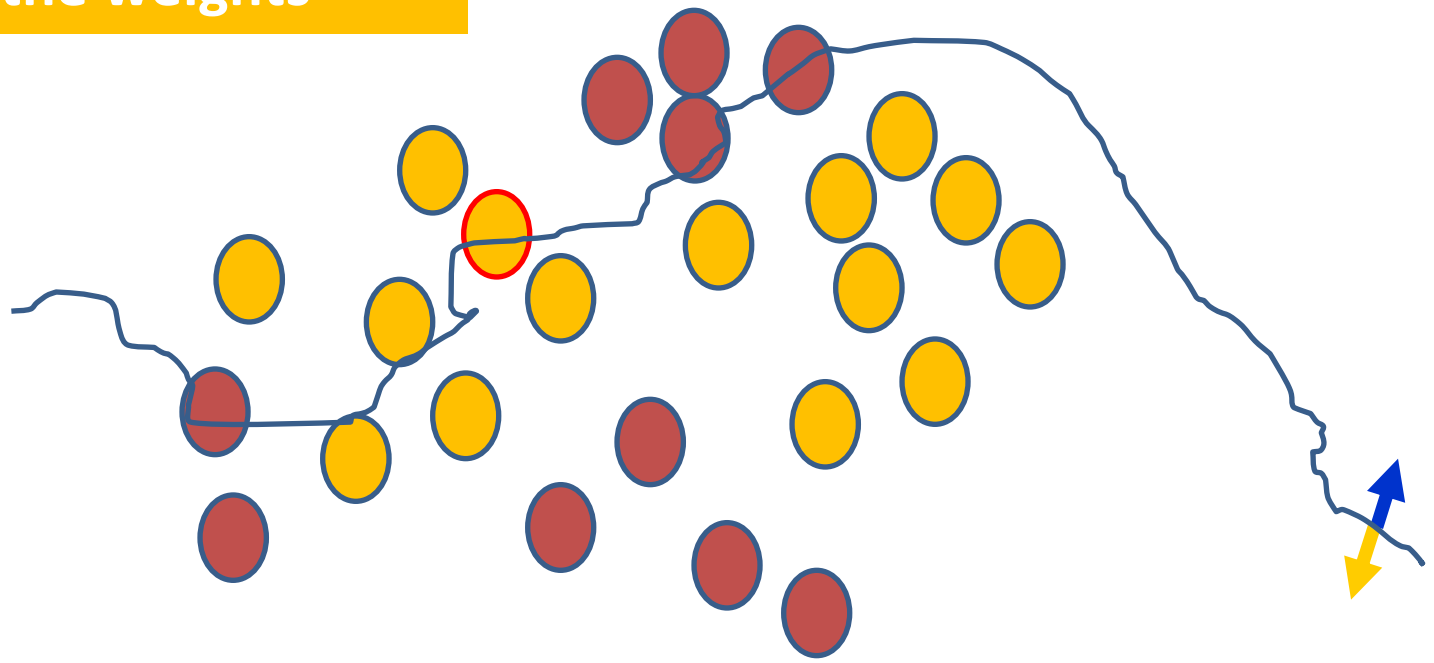
The decision boundary perspective…

Initial random weights



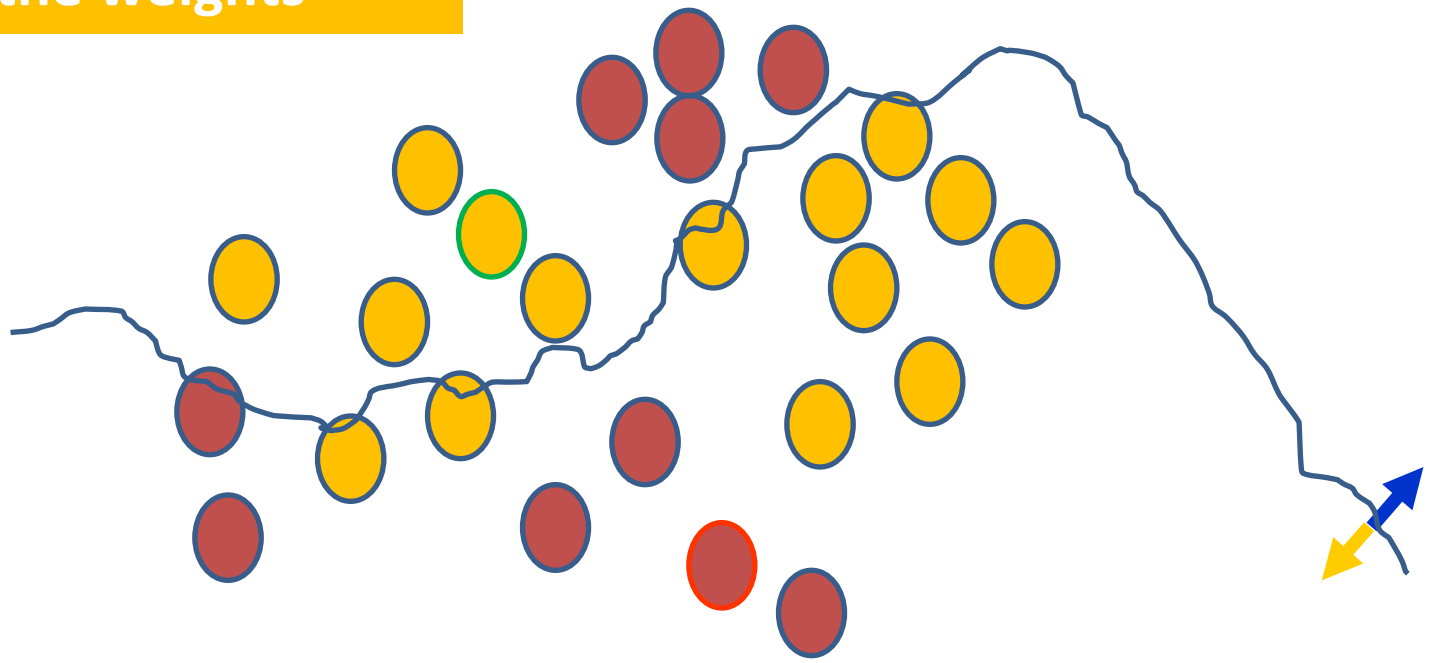
The decision boundary perspective…

Present a training instance / adjust the weights



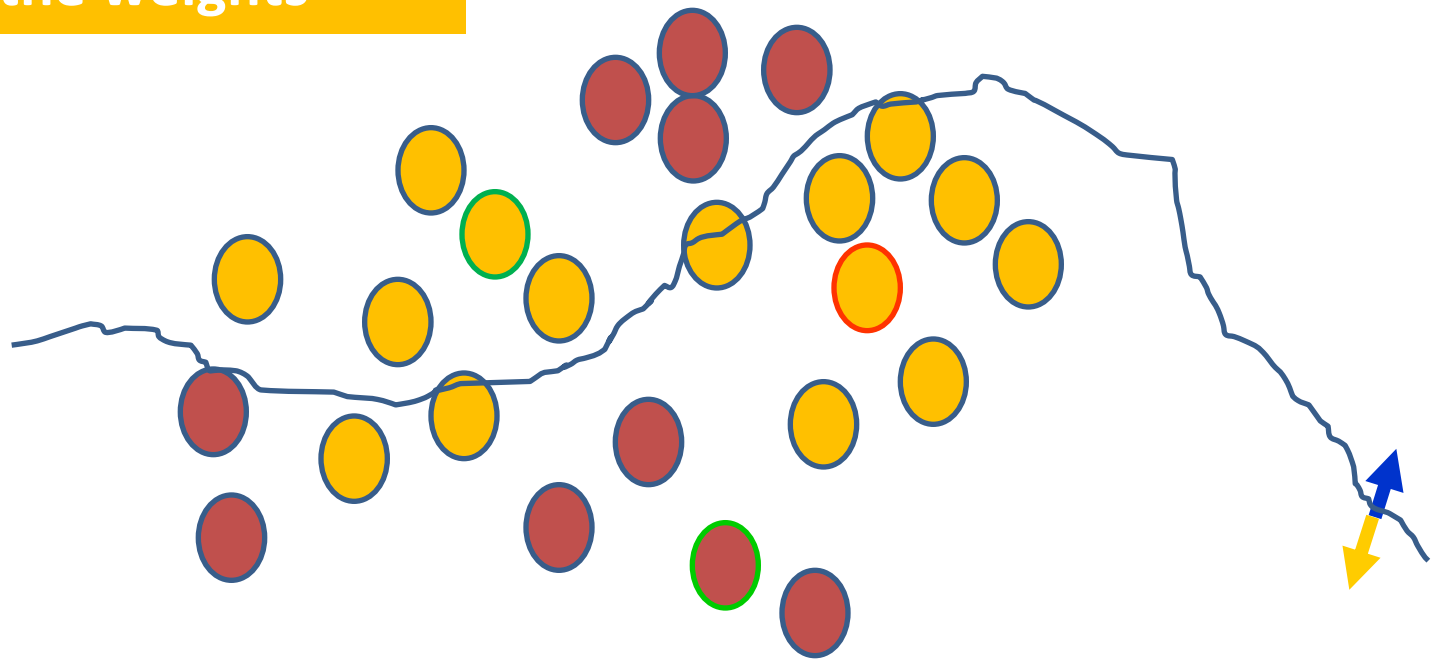
The decision boundary perspective…

Present a training instance / adjust the weights



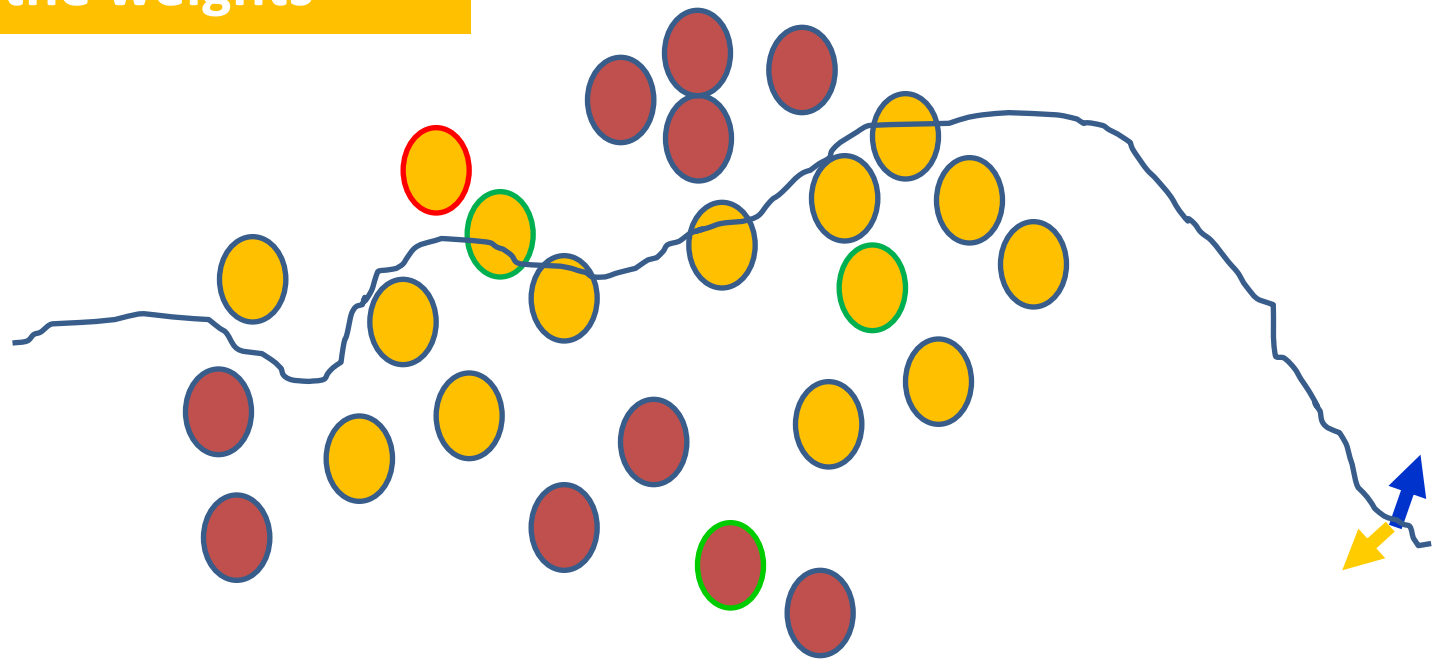
The decision boundary perspective…

Present a training instance / adjust the weights



The decision boundary perspective…

Present a training instance / adjust the weights

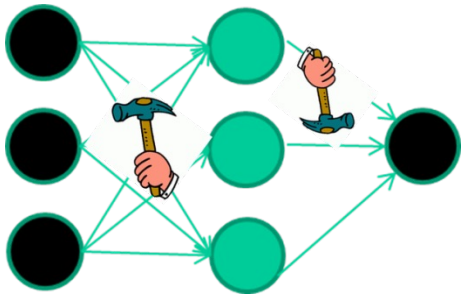


The decision boundary perspective…

Eventually ….

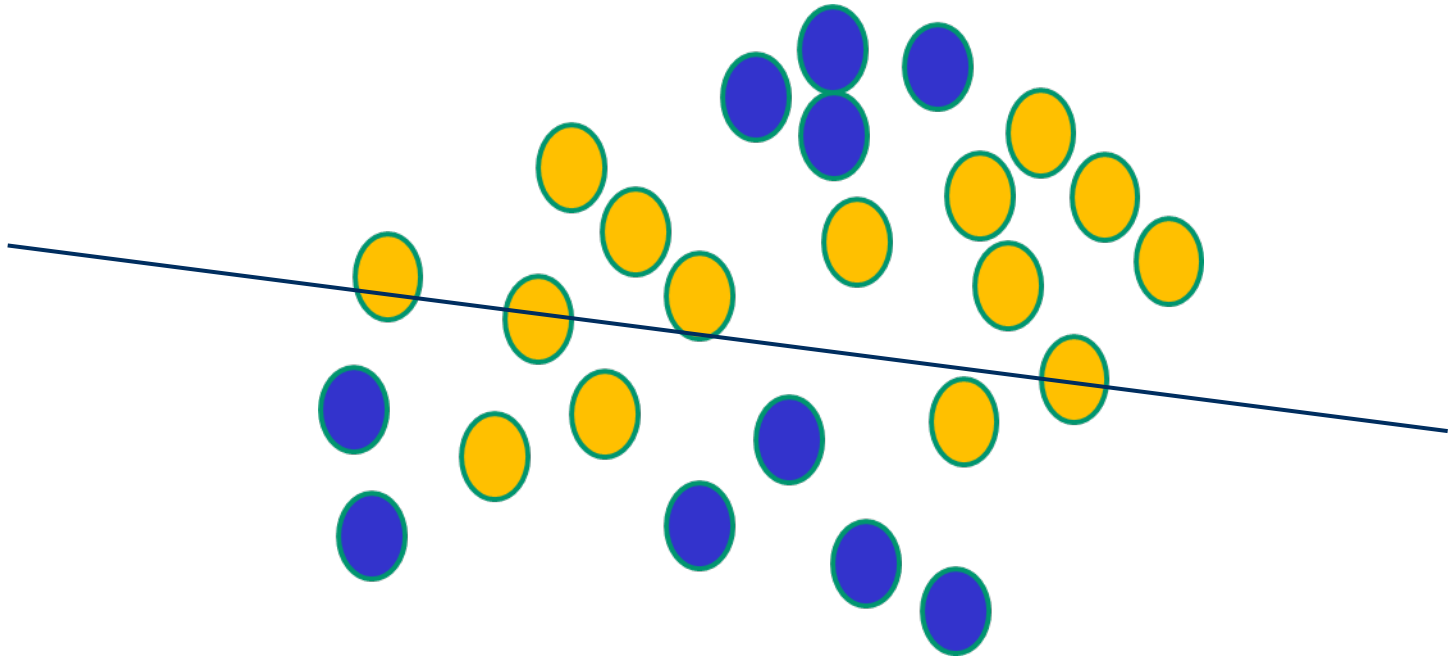
Intuition: very simplistic!

- Weight-learning algorithms for ANNs are simple
- They work by making *thousands* and *thousands* of tiny adjustments, each making the network do better at the most recent pattern but perhaps a little worse on many others
- But, by "dumb luck", eventually this tends to be good enough to learn effective classifiers for many real applications



Activation function f

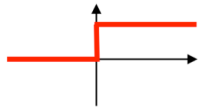
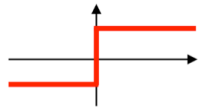
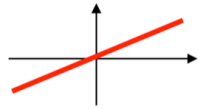
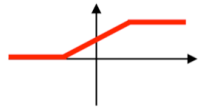
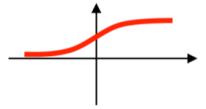
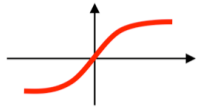
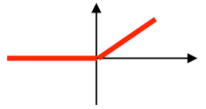
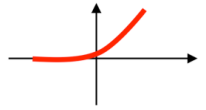
- If ***f*(*x*)** is linear, the ANN can **only** draw straight decision boundaries (even if there are many layers of units)



Activation function f

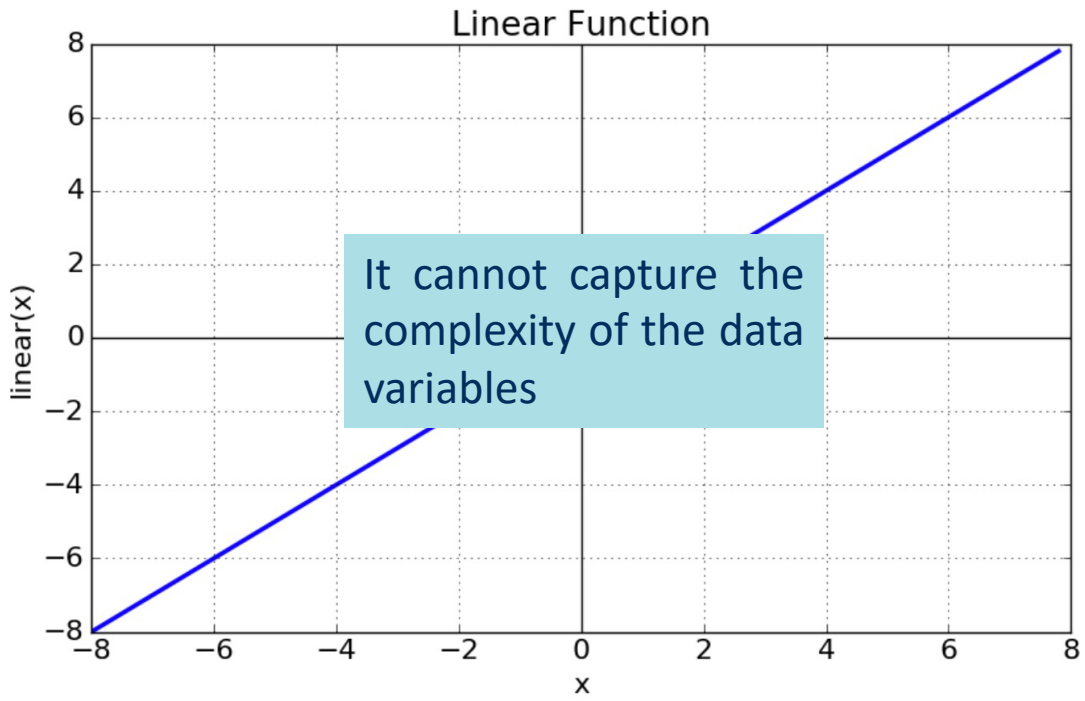
- If $f(x)$ is non-linear, a network with 1 hidden layer can theoretically learn any classification problem!
- A set of weights exists that can produce the targets from the inputs
- The problem is finding them!

Many possible activation functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Linear activation function

- **Equation :** f(x) = x
- **Range :** (-infinity, infinity)

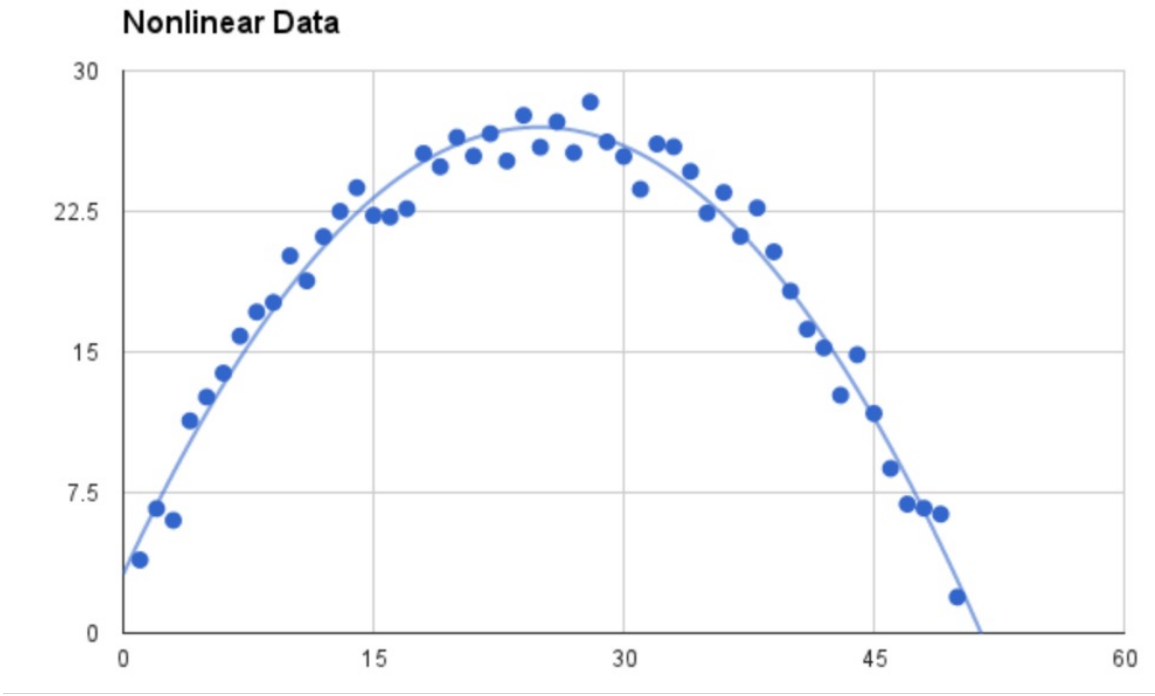


It cannot capture the complexity of the data variables

Stockholms universitet

Non-linear activation functions

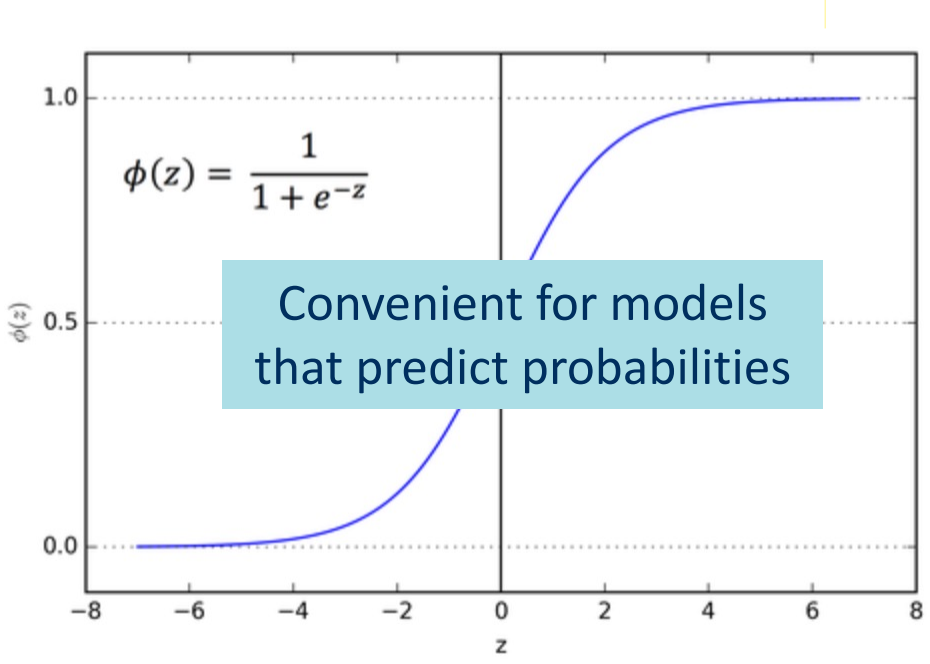
- It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output



Stockholms universitet

Sigmoid activation function

- **Equation :** f(x) = 1 / (1 + exp(-x))
- **Range :** (0, 1)



The function is:

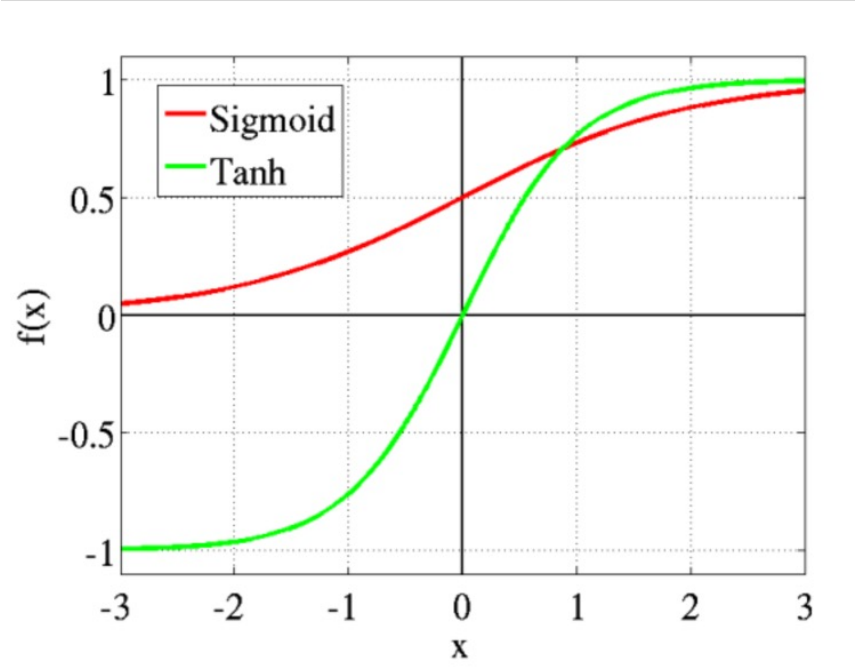
- **differentiable:** we can find the slope of the sigmoid curve at any two points
- **monotonic** but its derivative is not

It can cause a neural network to get stuck at the training time

The **softmax function** is a more generalized logistic activation function that is used for **multiclass classification**

Tanh activation function

- **Range :** (-1, 1)
- Takes negative values



The function is:

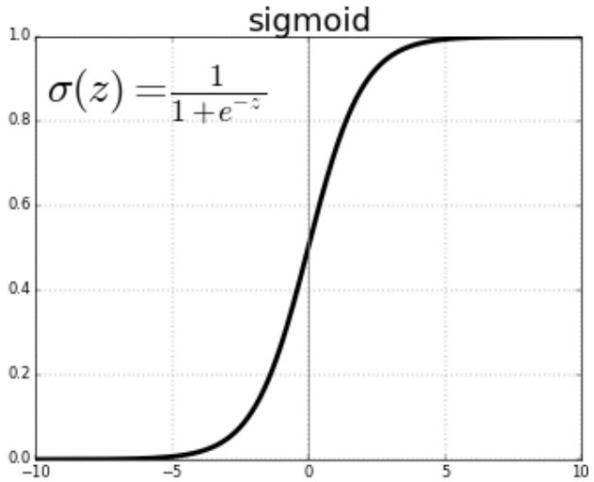
- **differentiable:** we can find the slope of the curve at any two points
- **monotonic** but its derivative is not

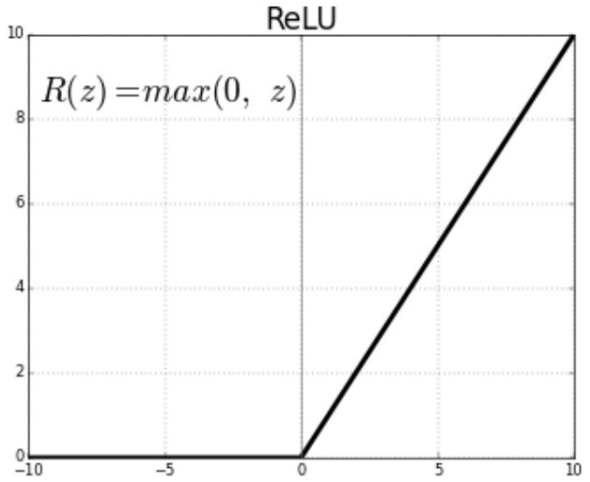
Negative inputs are mapped to strongly negative values, and zero inputs are mapped to near zero

Stockholms universitet

ReLU activation function

- **Range :** (0, infinity)





The function and its derivative are both **monotonic**

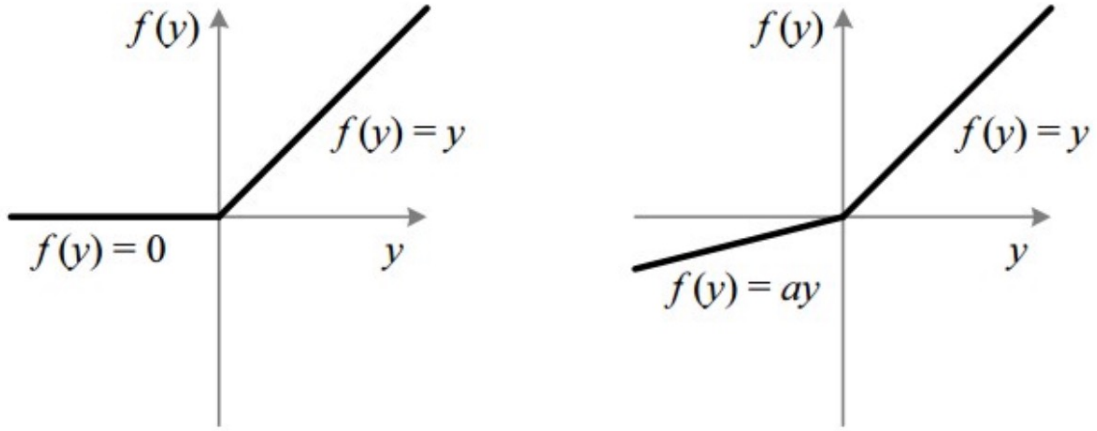
Problem: negative values become zero immediately

- decreases the model's ability to fit or train from the data properly
- negative input given to the ReLU activation function turns the value into zero immediately, hence not mapping the negative values appropriately

Stockholms universitet

Leaky ReLU activation function

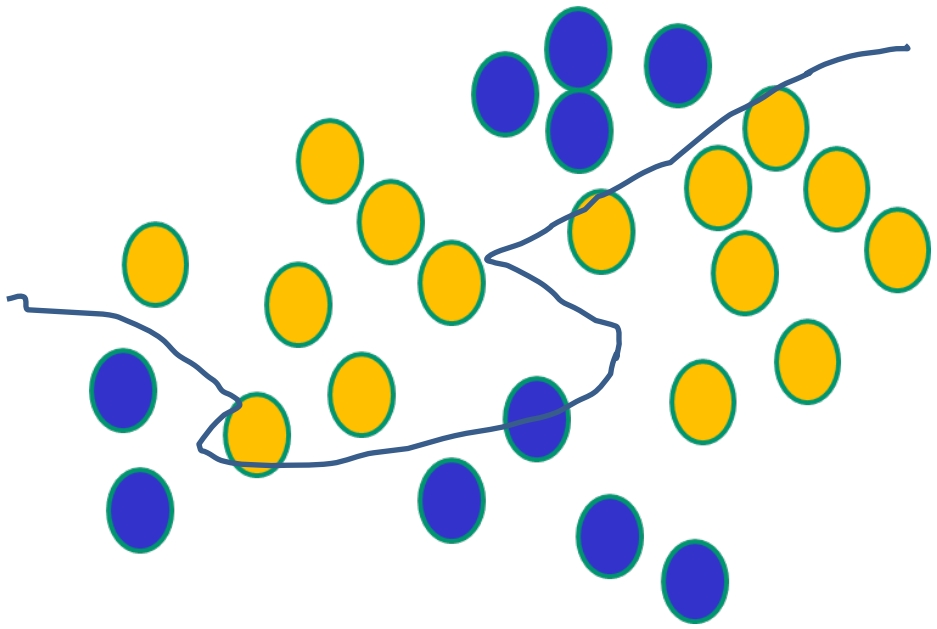
- **Range :** (-infinity, infinity)



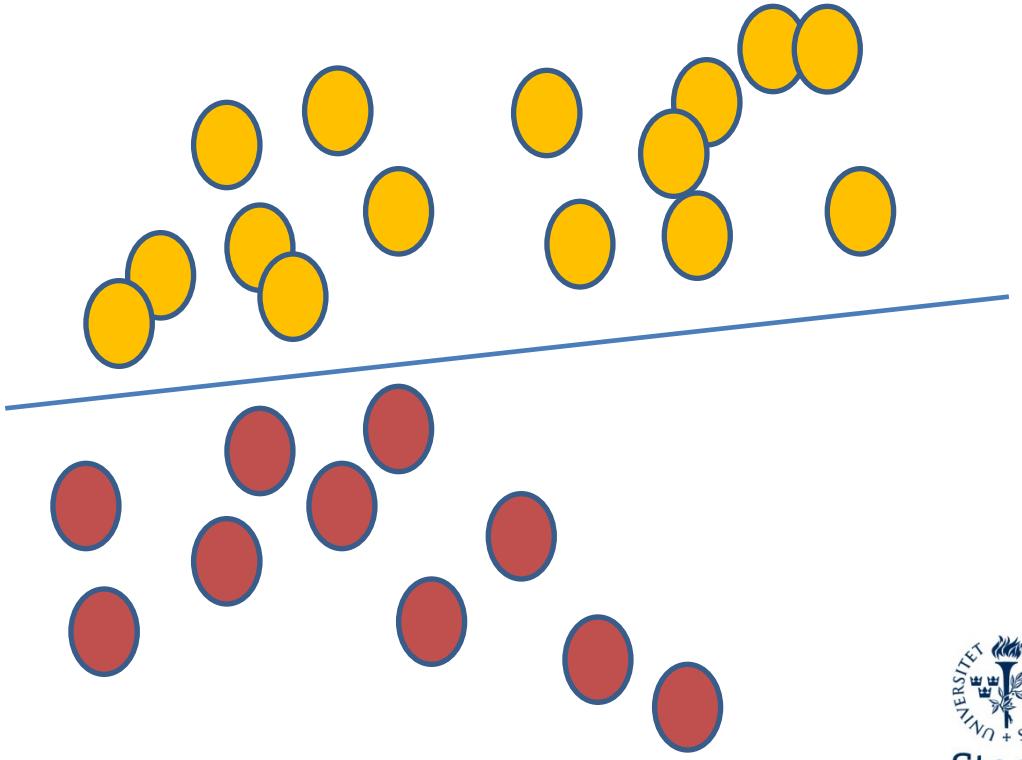
- The leak helps to increase the range of the ReLU function
- Usually, **a** is 0.01 (leaky), or it may have any other value (randomized)
- Both Leaky and Randomized ReLU functions, and their derivatives, are **monotonic** in nature

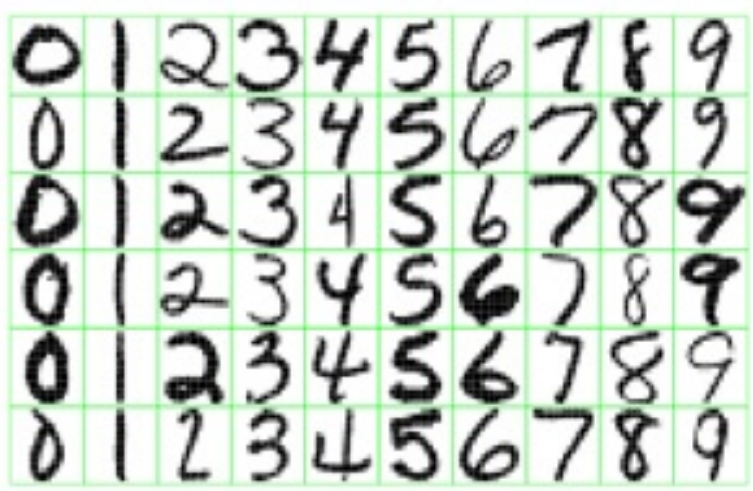
ANN vs SVM

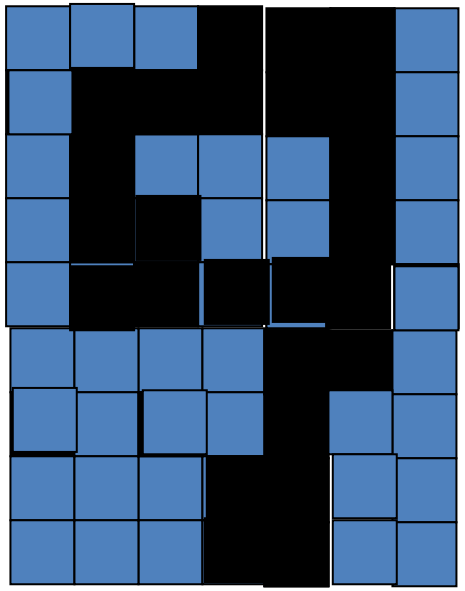
ANNs use non-linear $f(x)$, so they can draw complex boundaries but keep the data unchanged



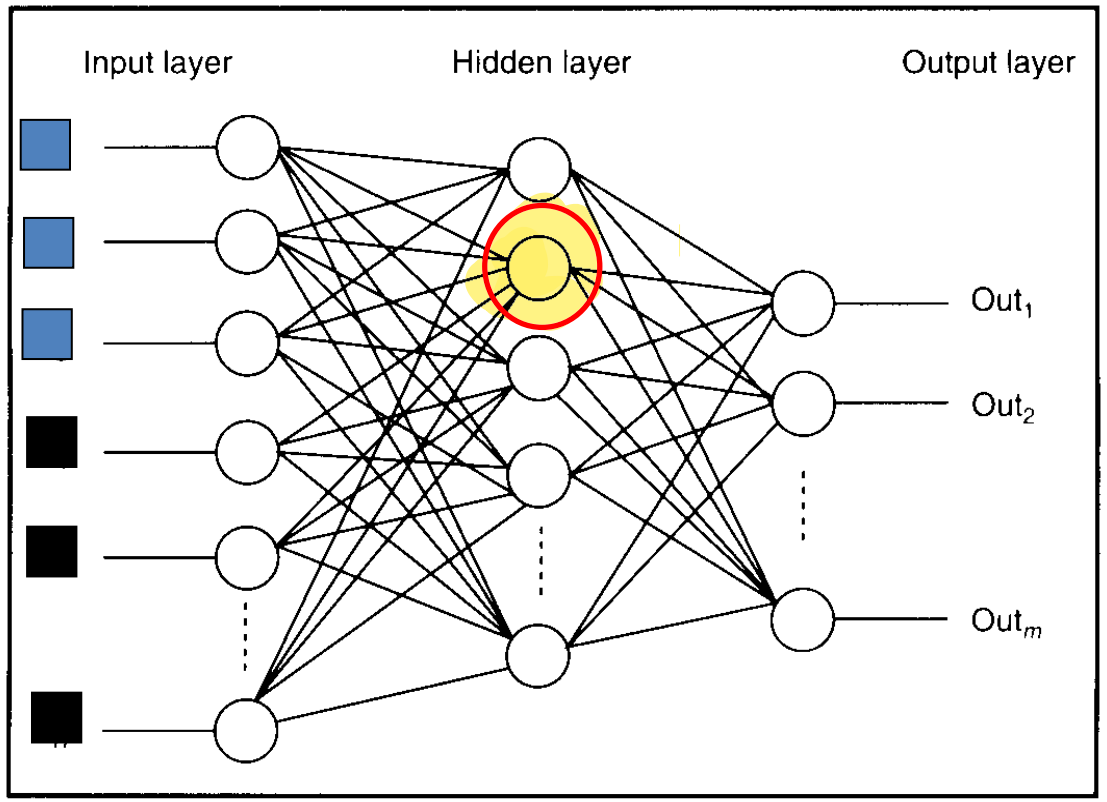
SVMs only draw straight lines, but they transform the data first in a way that makes that OK





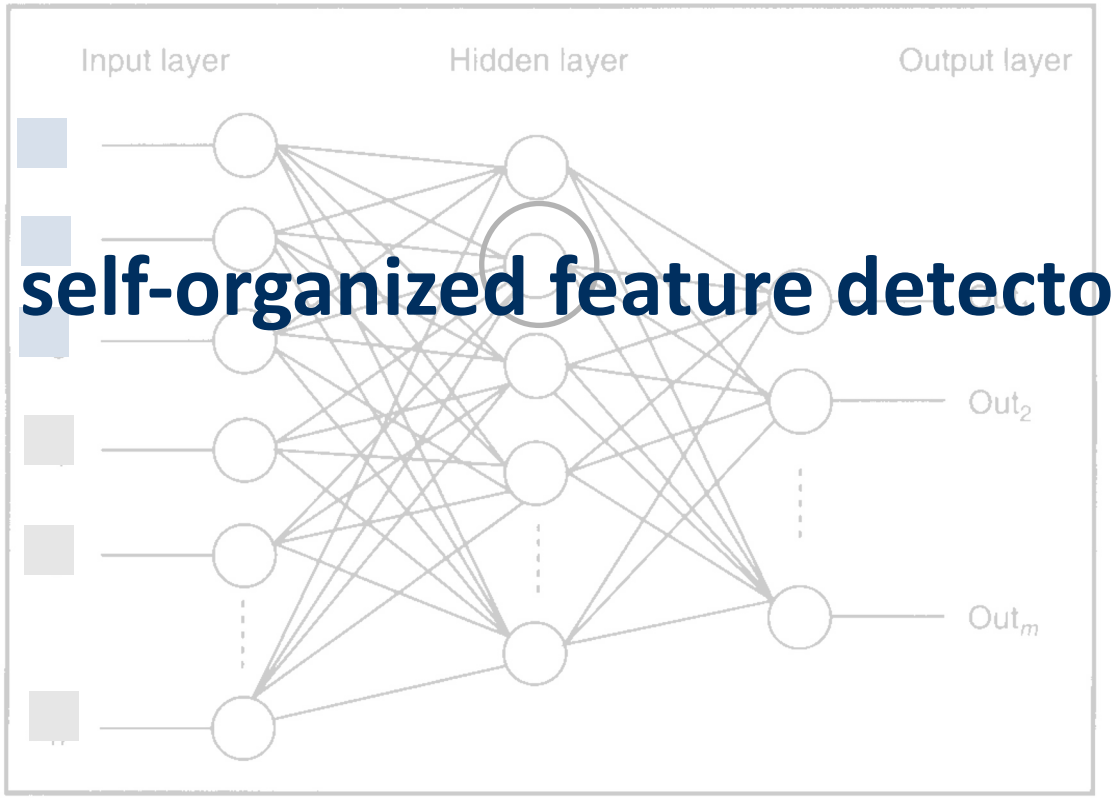


What is this unit doing?



Stockholms universitet

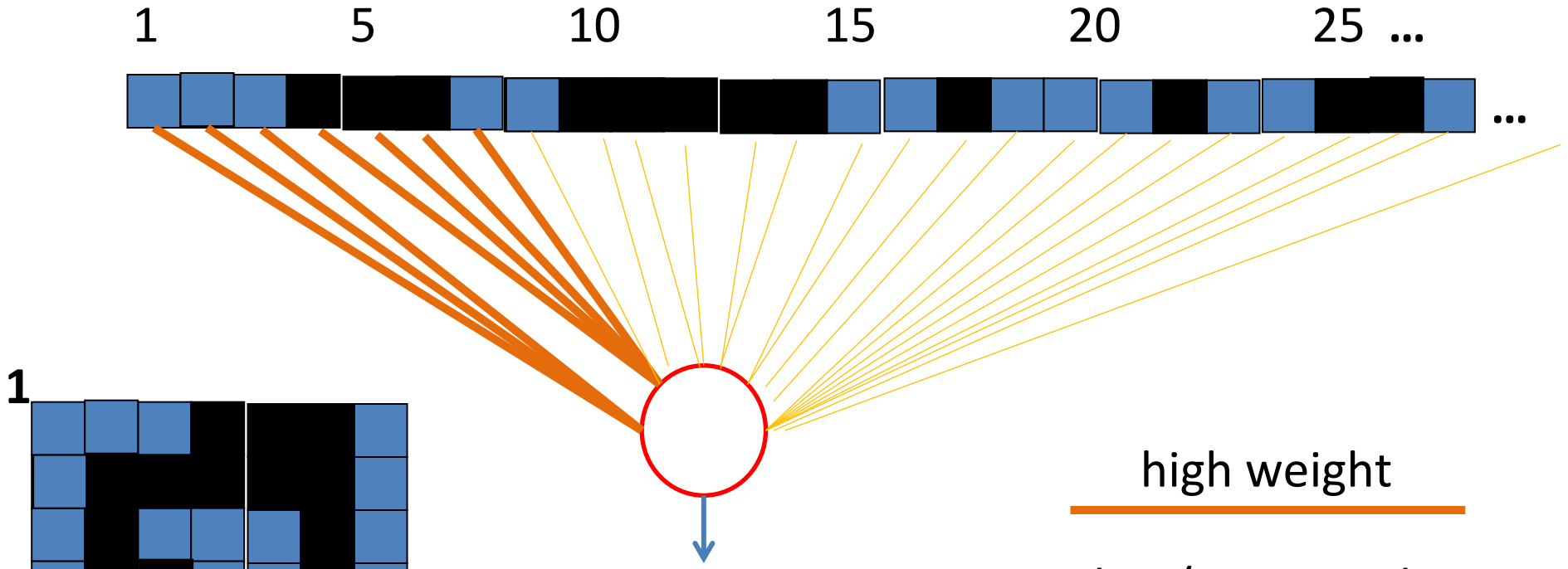
What is this unit doing?

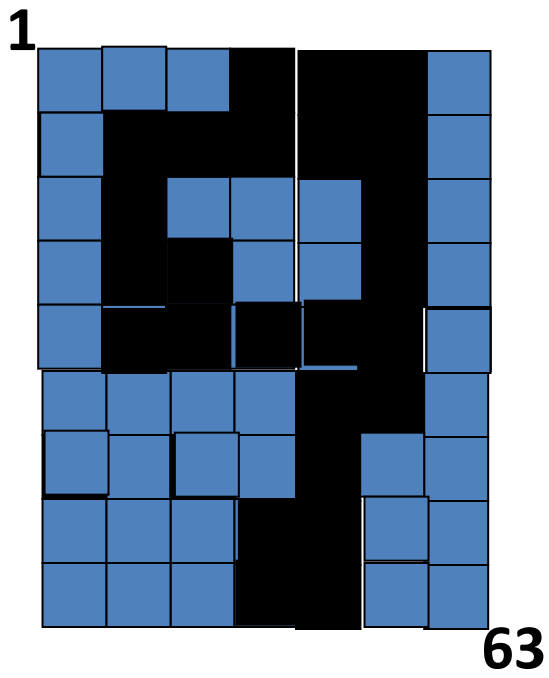


Hidden layer units become self-organized feature detectors

Stockholms universitet

What does this unit detect?



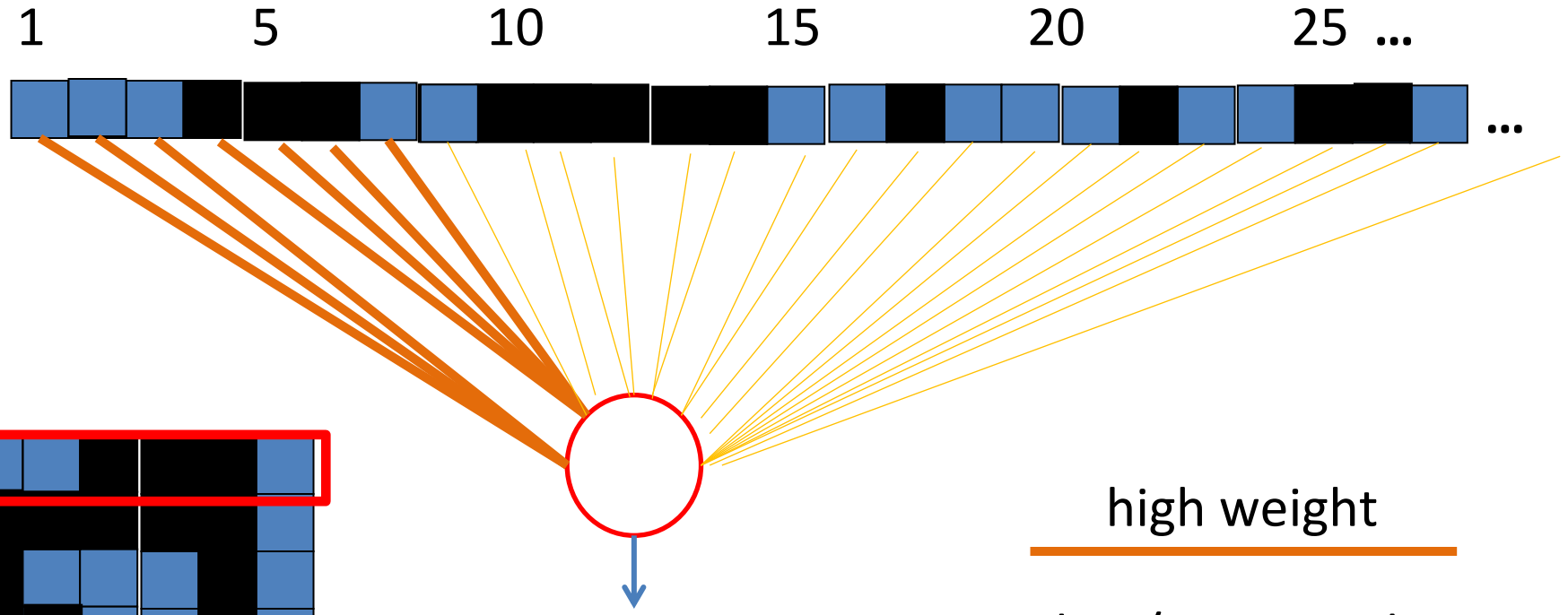


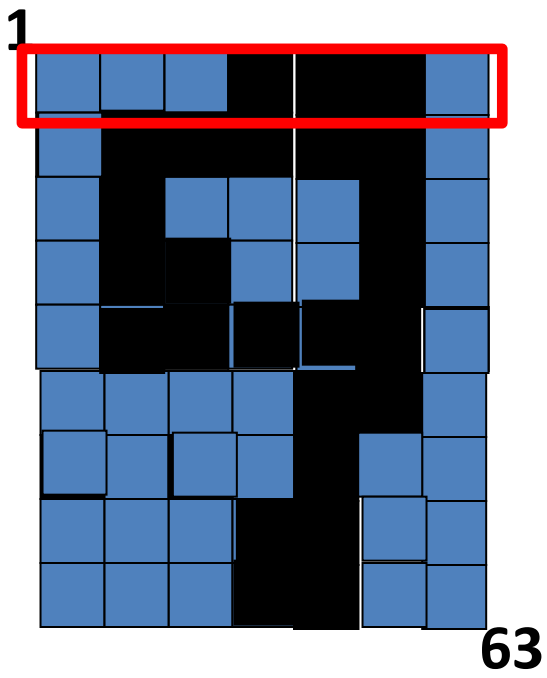
high weight

low/zero weight

Stockholms universitet

What does this unit detect?





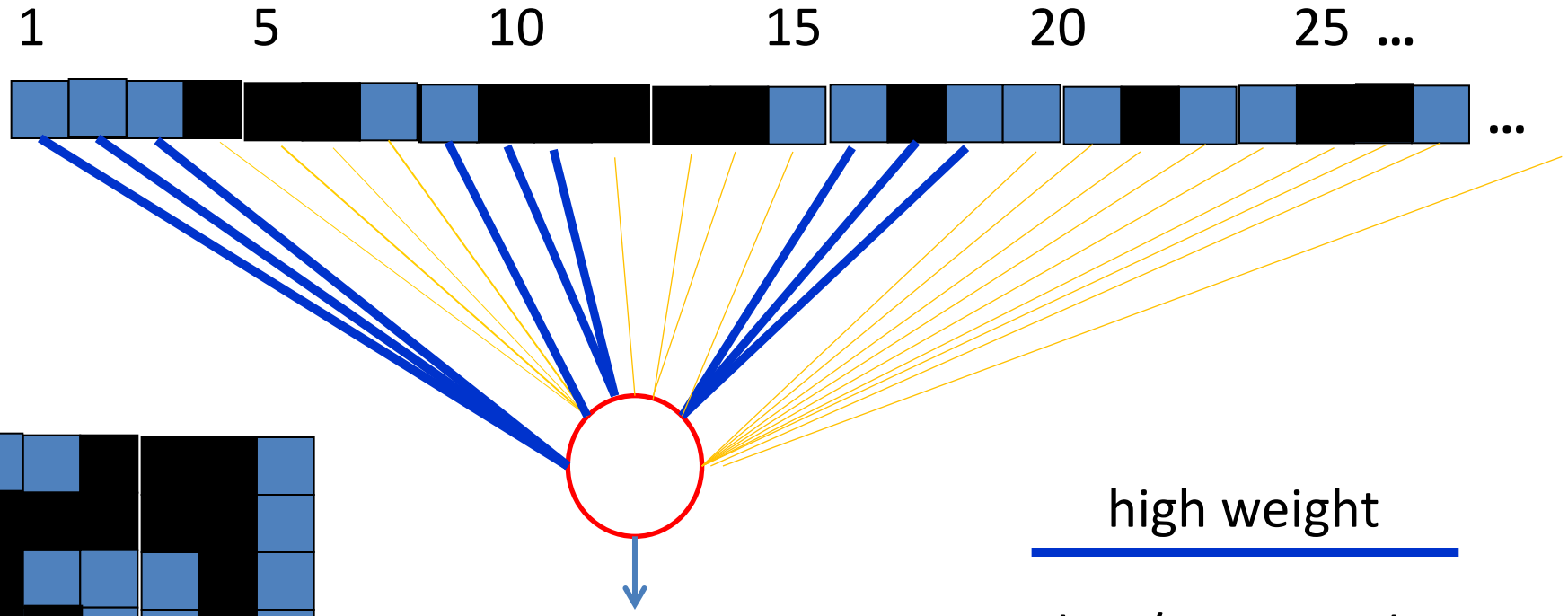
high weight

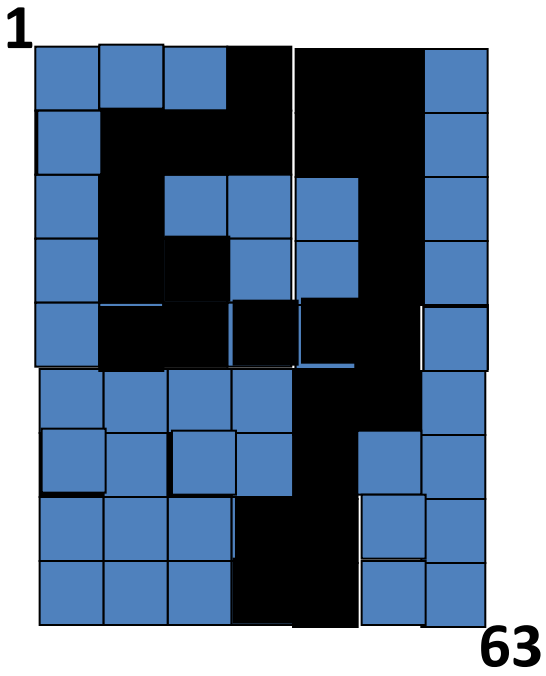
low/zero weight

It will send a strong signal for a horizontal line in the top row, ignoring everywhere else

Stockholms universitet

What does this unit detect?



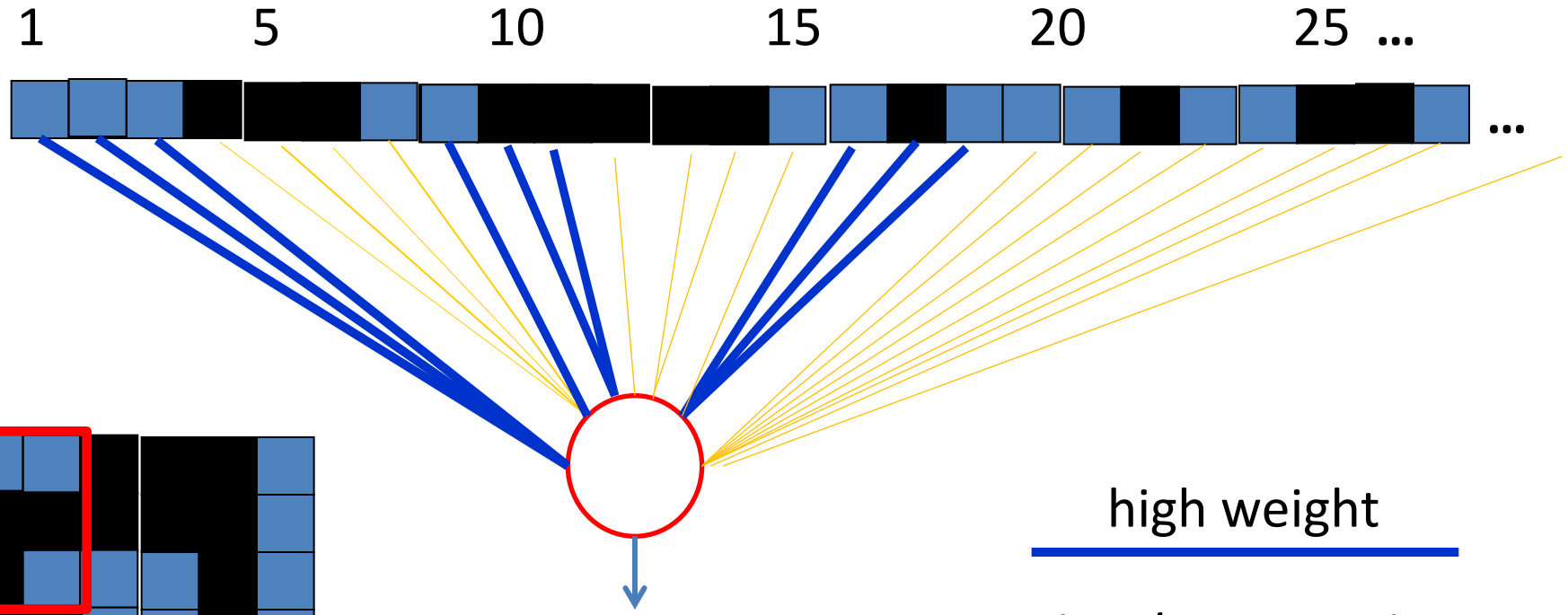


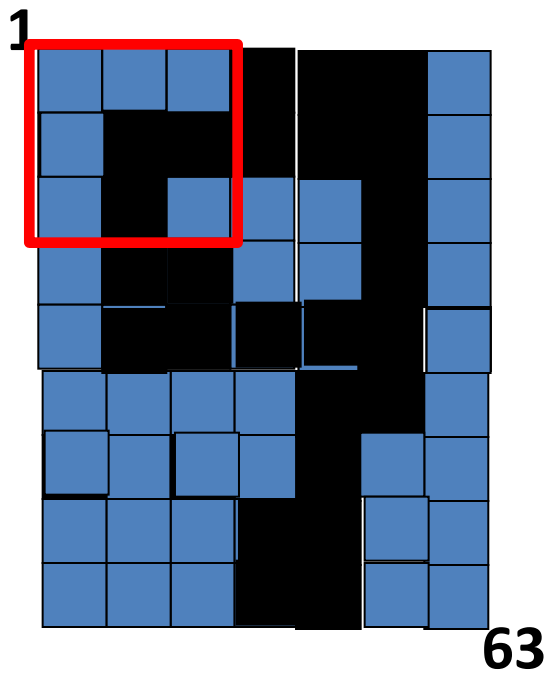
high weight

low/zero weight

Stockholms universitet

What does this unit detect?





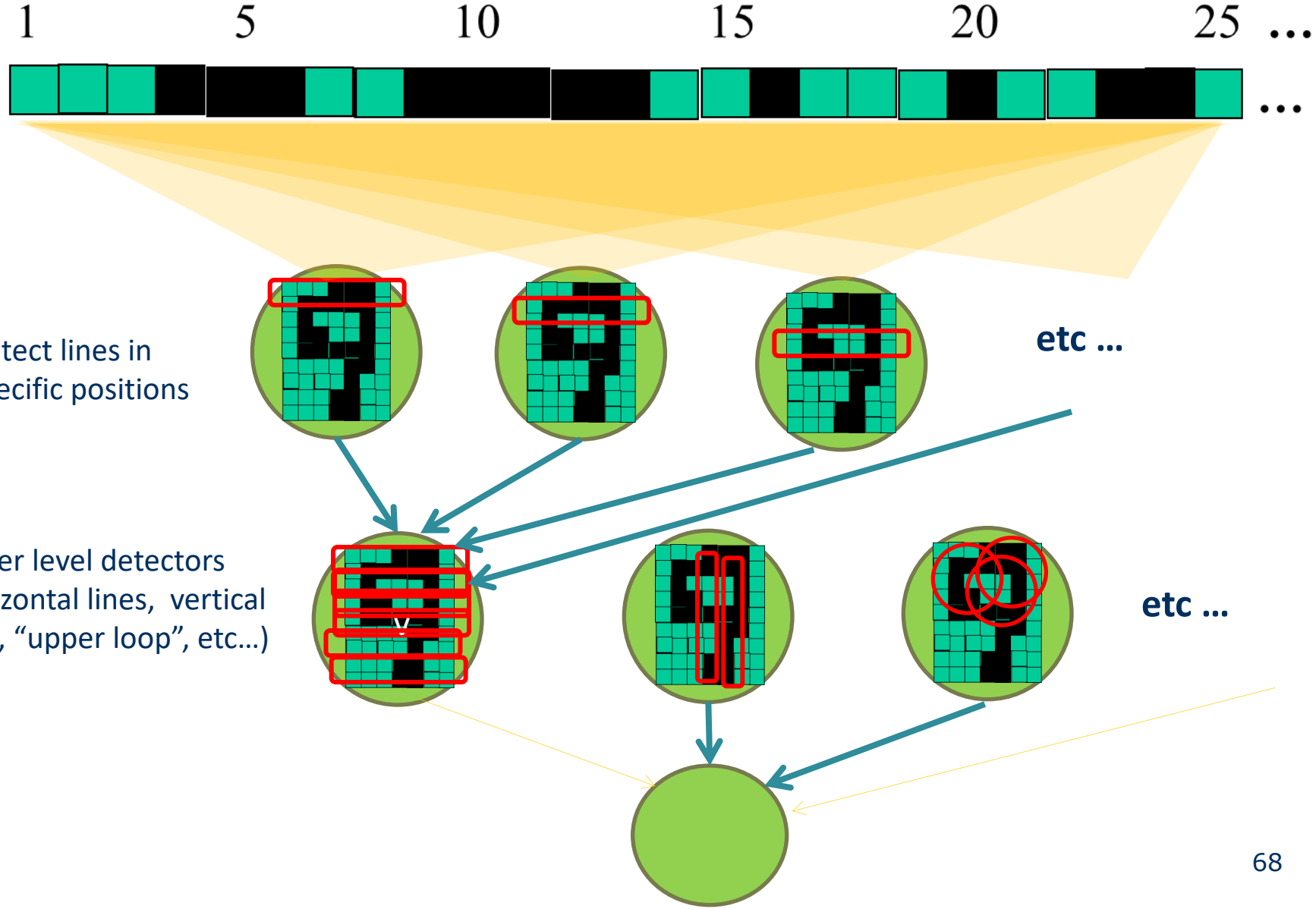
high weight

low/zero weight

Strong signal for an area in the top left corner

Stockholms universitet

Successive layers can learn higher-level features

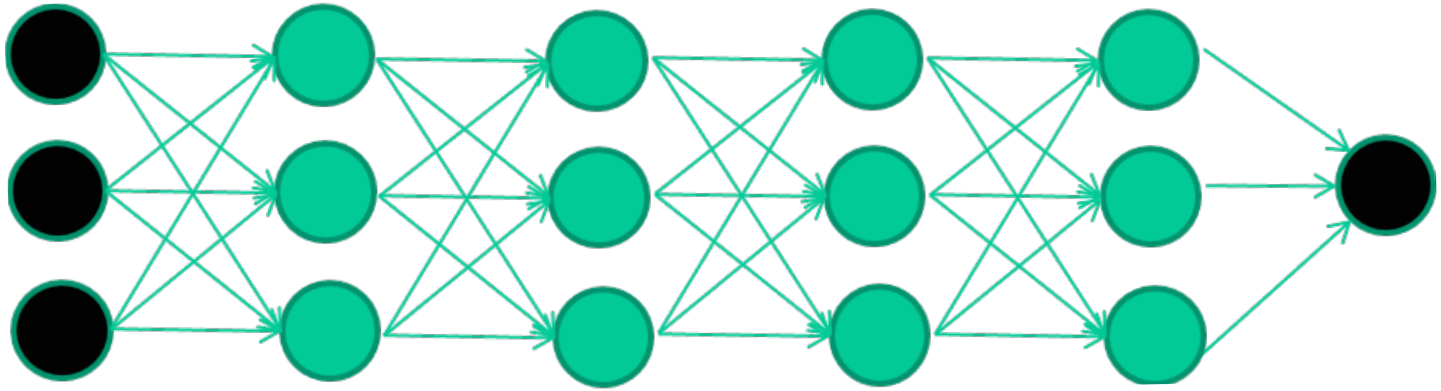


Stockholms universitet

So…multiple layers make sense!

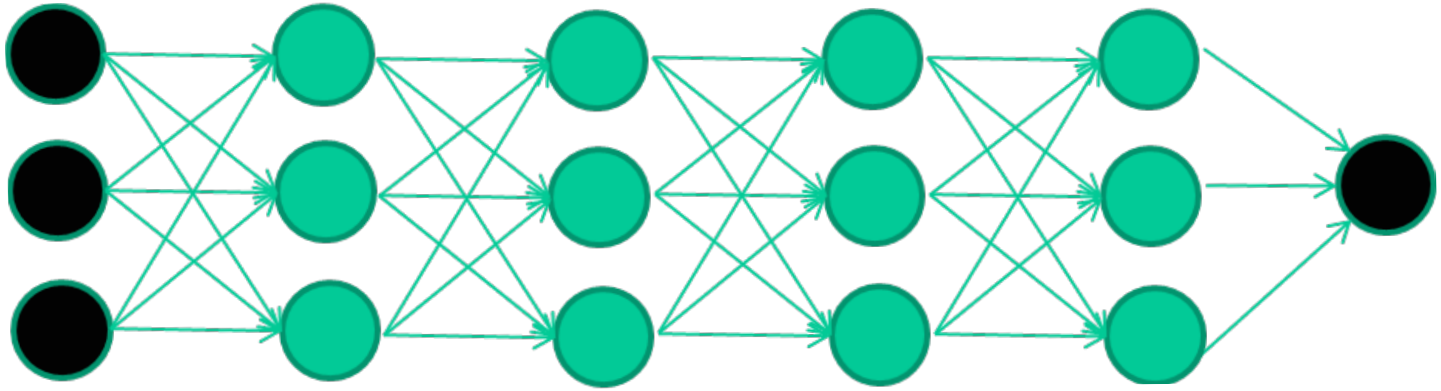
Our brains work that way too ☺

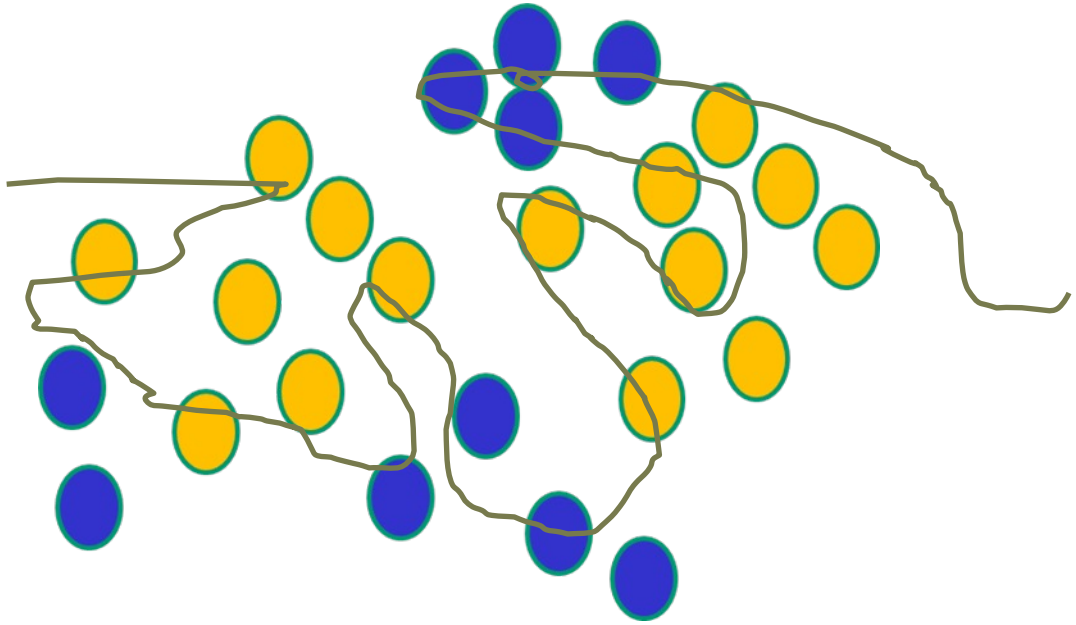
Multi-layer neural network architectures should be capable of learning the true underlying features and "feature logic", and therefore generalize very well …



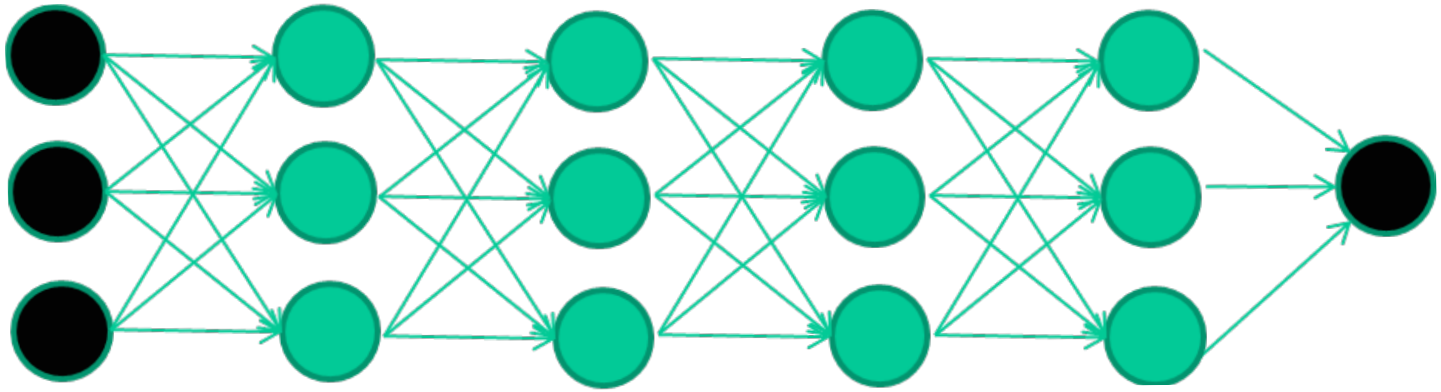
Stockholms universitet

But, until very recently, our weight-learning algorithms simply did not work on multi-layer architectures

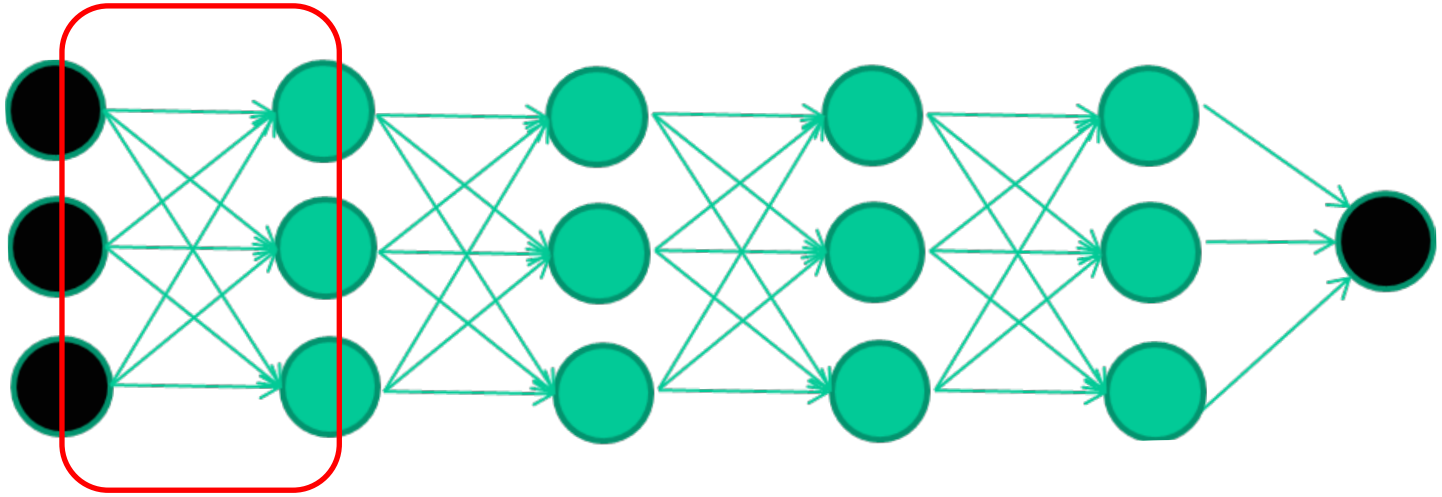




The new way to train multi-layer ANNs…

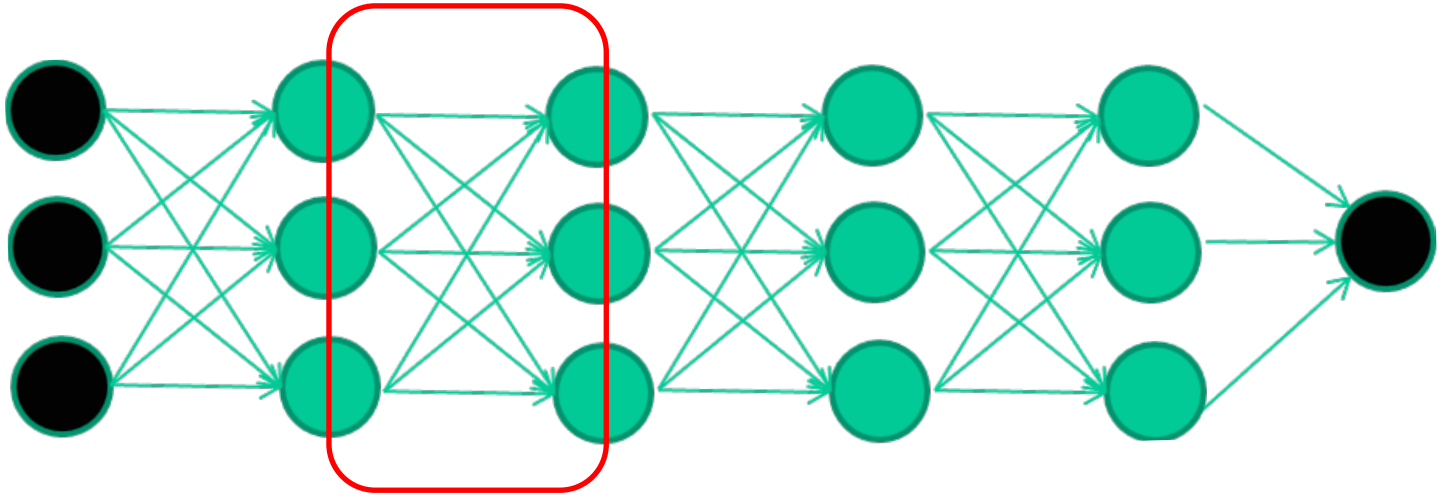


The new way to train multi-layer ANNs…



Train **this** layer first

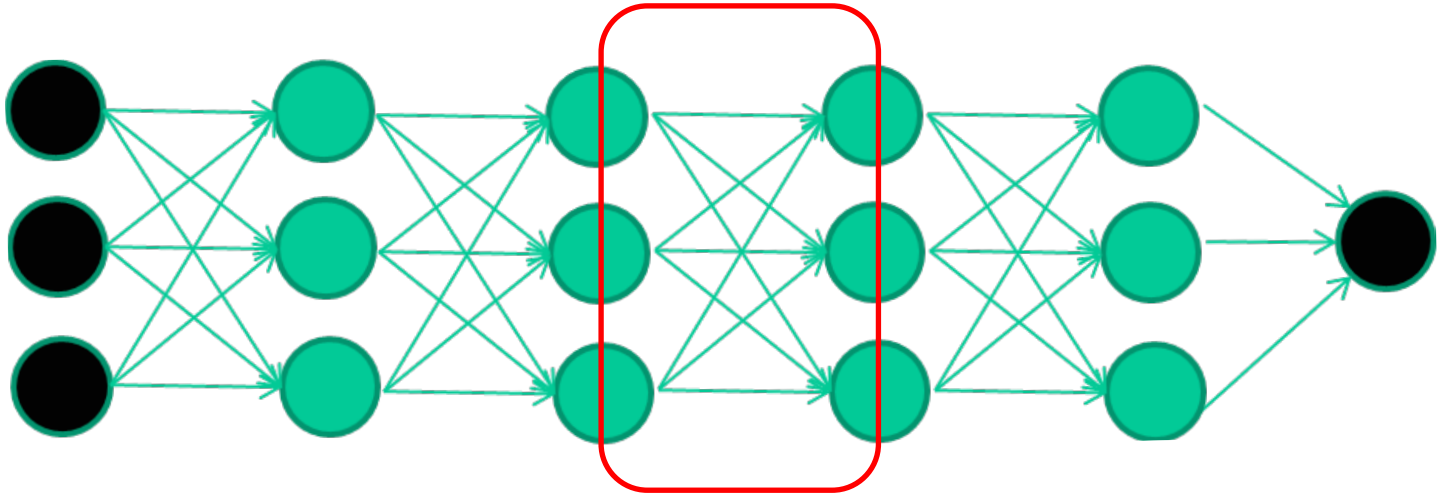
The new way to train multi-layer ANNs…



Train **this** layer first

then **this** layer

The new way to train multi-layer ANNs…

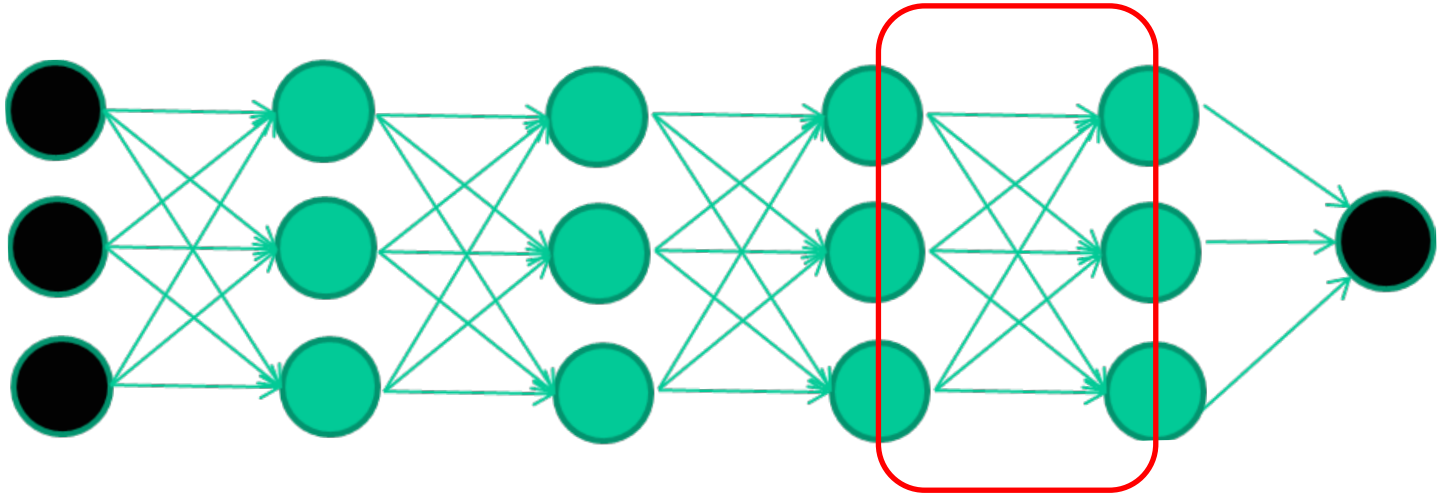


Train **this** layer first

then **this** layer

then **this** layer

The new way to train multi-layer ANNs…



Train **this** layer first

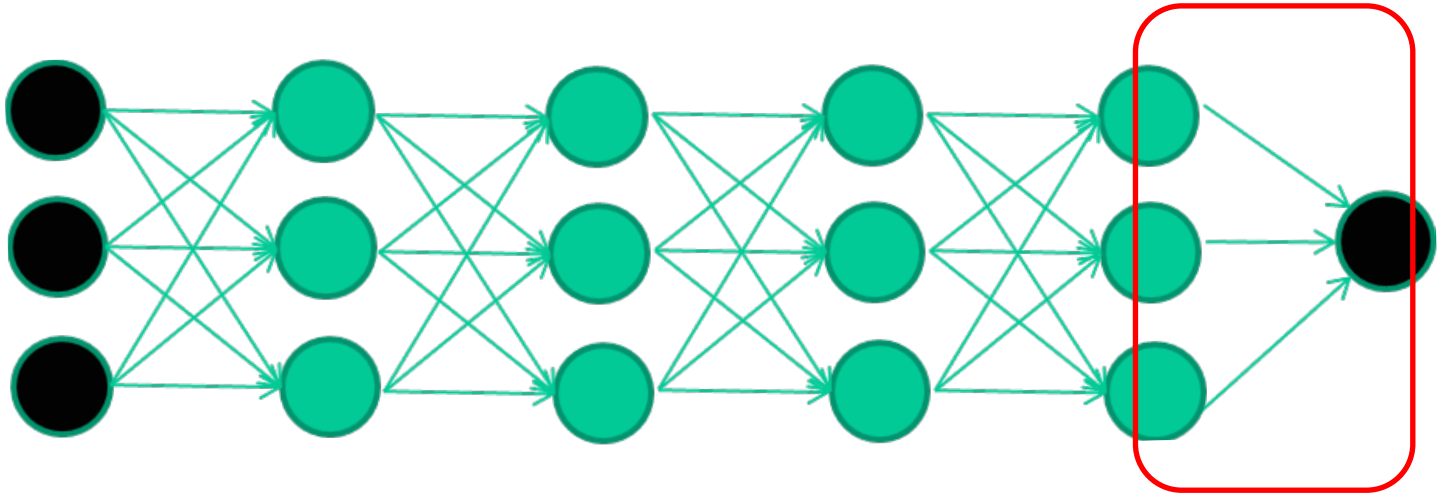
then **this** layer

then **this** layer

then **this** layer

Stockholms universitet

The new way to train multi-layer ANNs…



Train **this** layer first

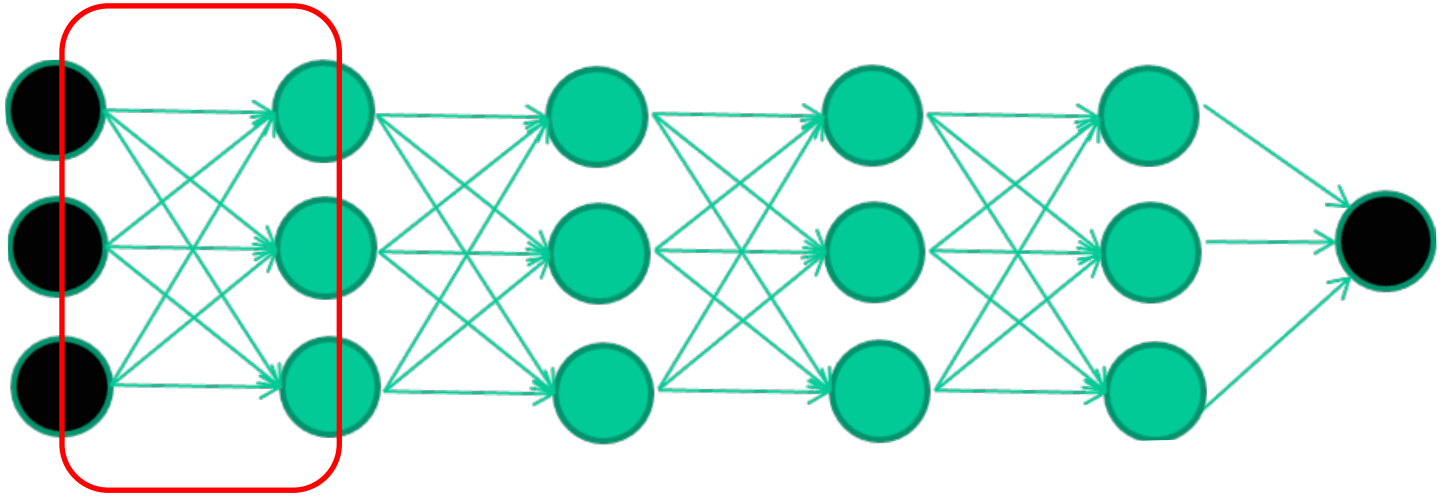
then **this** layer

then **this** layer

then **this** layer

finally, **this** layer

The new way to train multi-layer ANNs…

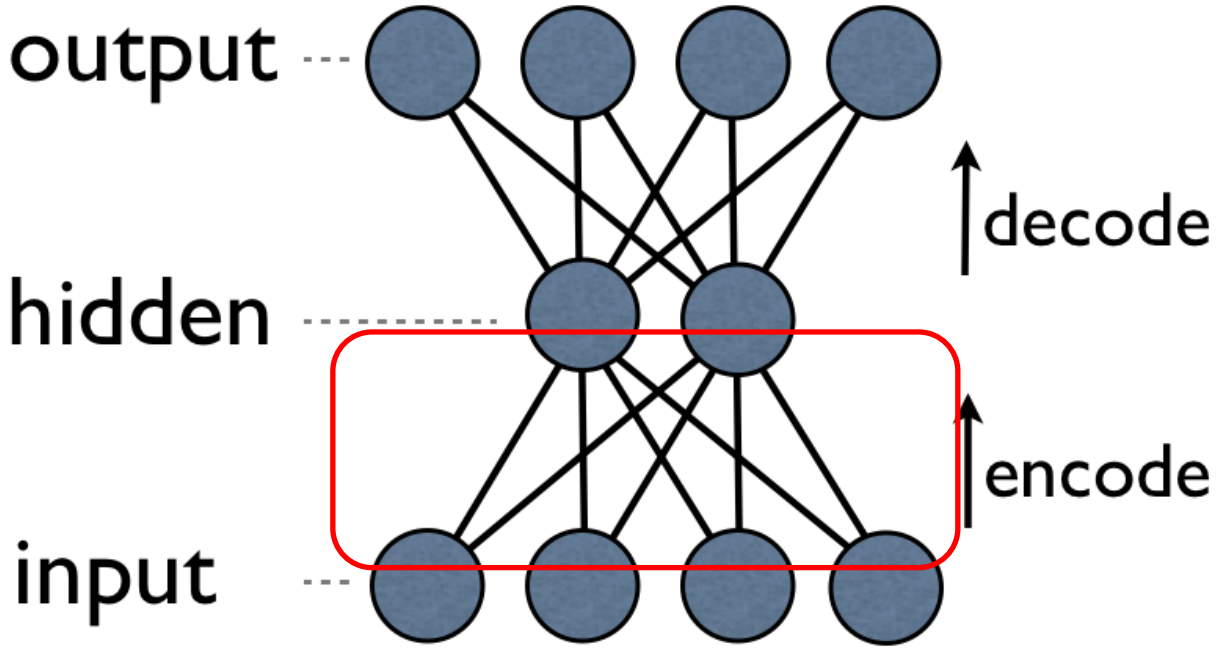


*EACH of the (non-output) layers is trained to be an **autoencoder***

Basically, it is forced to learn good features that describe what comes from the previous layer

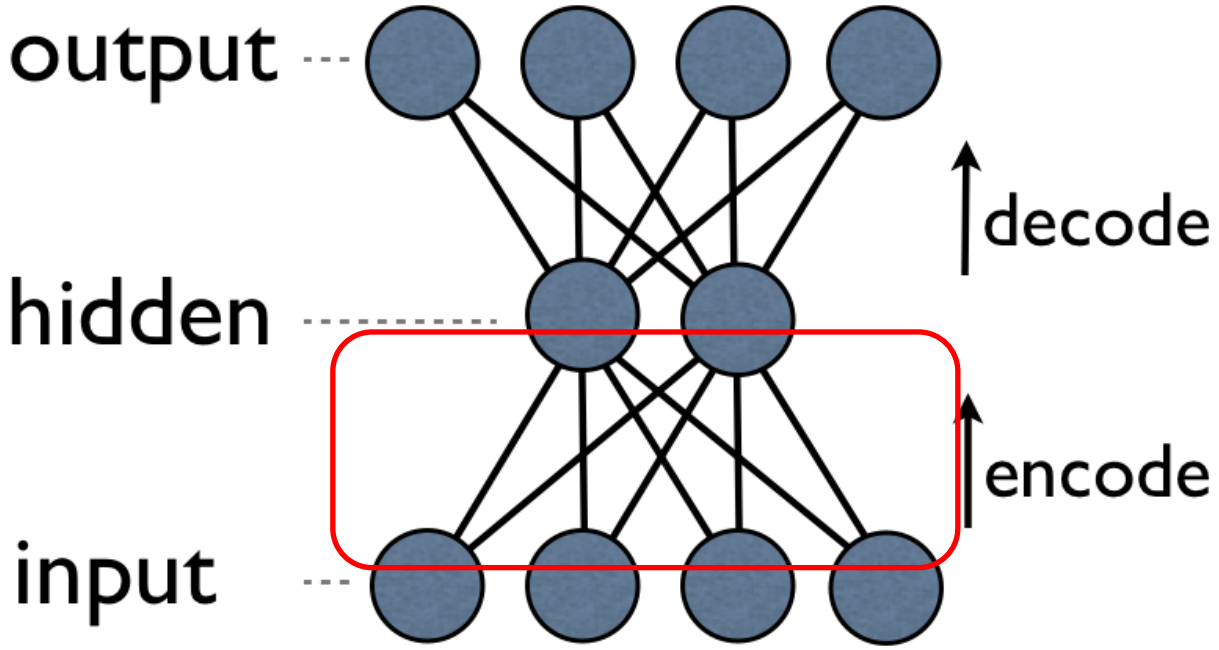
Autoencoders

An auto-encoder is trained with an absolutely standard weight-adjustment algorithm to reproduce the input



Autoencoders

An auto-encoder is trained with an absolutely standard weight-adjustment algorithm to reproduce the input

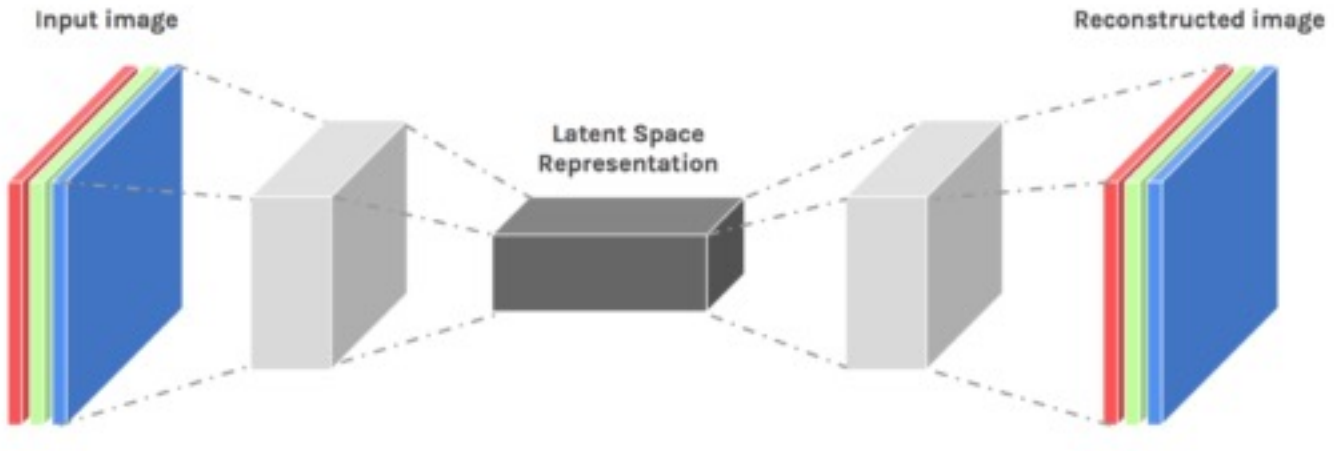


By making this happen with (many) fewer units than the inputs, this forces the "hidden layer" units to become good feature detectors

Autoencoders

Autoencoders are designed to reproduce their input, especially for images:

- The key point is to reproduce the input from a learned encoding

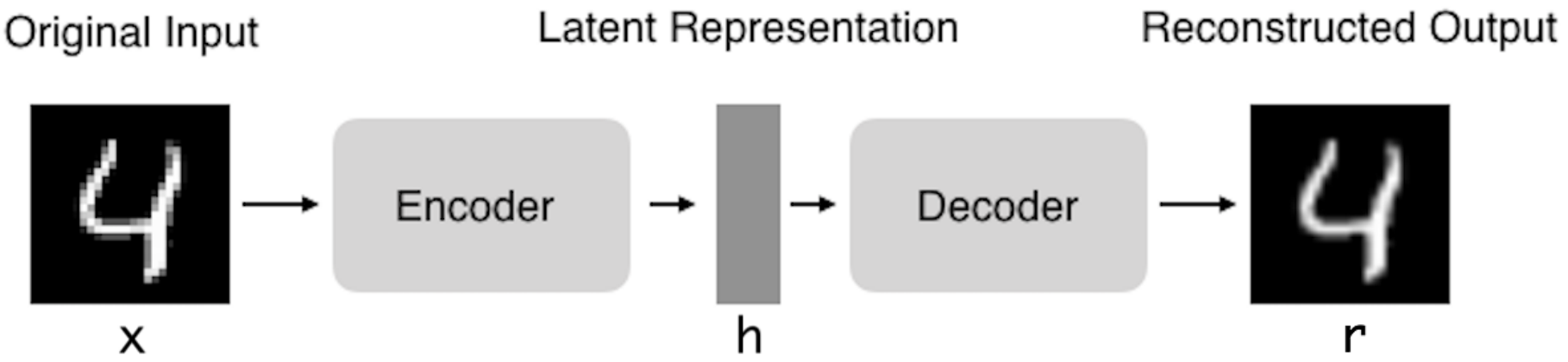


https://www.edureka.co/blog/autoencoders-tutorial/

Stockholms universitet

Autoencoders: structure

- Encoder: compress input into a latent-space of usually smaller dimension; h = f(x)
- Decoder: reconstruct the input from the latent space; r = g(f(x)) with r as close to x as possible



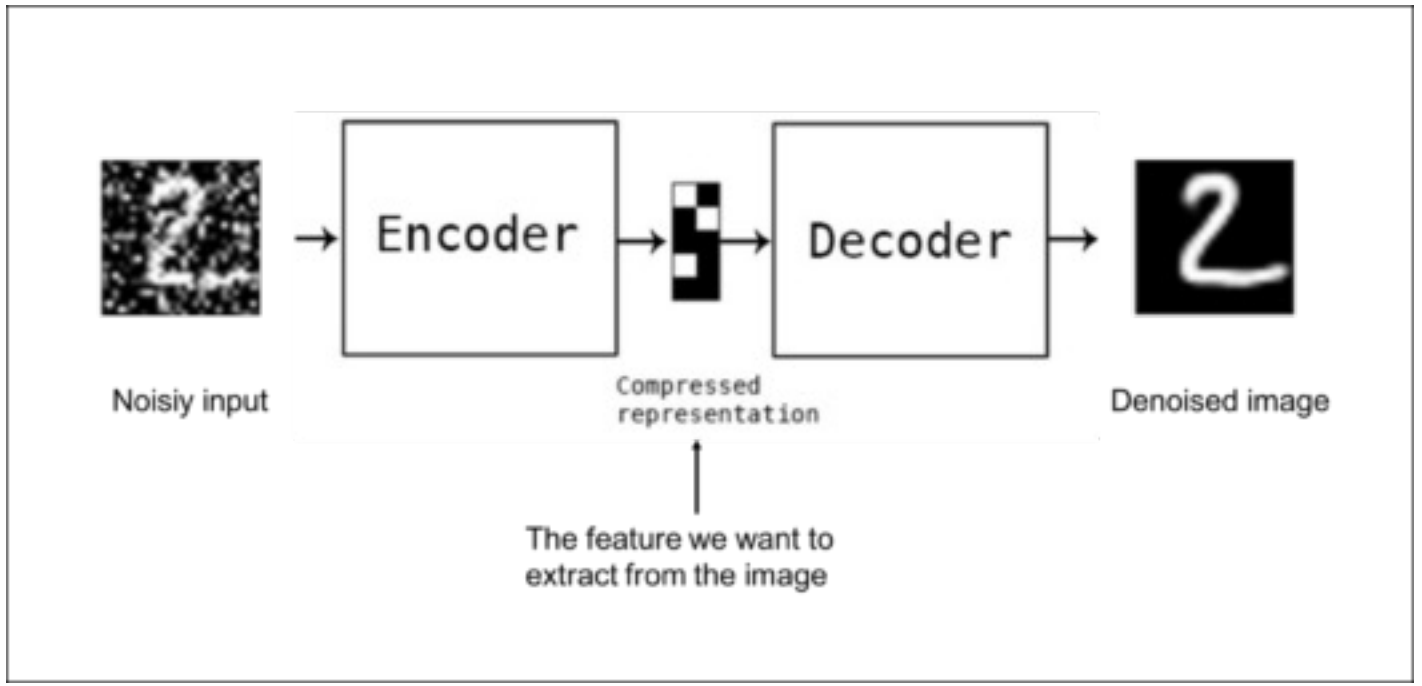
Stockholms universitet

Autoencoders

- Compare PCA/SVD
  - PCA takes a collection of vectors (images) and produces a usually smaller set of vectors that can be used to approximate the input vectors via a linear combination
  - Very efficient for certain applications
- Neural network autoencoders
  - Can learn non-linear dependencies
  - Can use convolutional layers
  - Can use transfer learning

Autoencoders: applications

- Denoising: input clean image + noise and train to reproduce the clean image



Stockholms universitet

Autoencoders: applications

- Image coloring: input black and white and train to produce color images

Stockholms universitet

Properties of Autoencoders

- **Data-specific**: Autoencoders are only able to compress data similar to what they have been trained on
- **Lossy:** The decompressed outputs will be degraded compared to the original inputs
- **Learned automatically from examples:** It is easy to train specialized instances of the algorithm that will perform well on a specific type of input

Size of Training Data

- Rule of thumb: the number **N** of training examples should be at least five to ten times the number of weights of the network

- Another rule:

$$N > \frac{|W|}{(1-a)}$$

where **|W|** is the number of weights and **a** is the expected accuracy on the test set

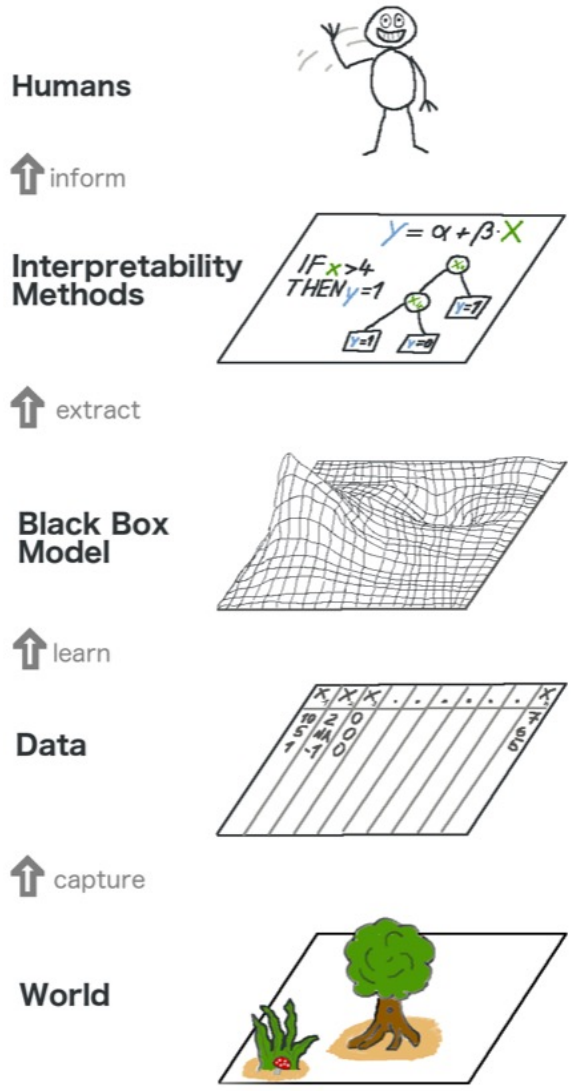
Neural Networks in Use

- Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs, including:
  - Sales forecasting
  - Recognition of speakers in communications
  - Medical diagnosis
  - Undersea mine detection
  - 3D object recognition
  - Hand-written word recognition
  - Face recognition
  - Procedural content generation

Disadvantages of Neural Networks

- The individual relations between the input variables and the output variables are not developed by engineering judgment, so the model tends to be a black-box or input/output table without an analytical basis
- The sample size has to be large
- Require lots of trial and error, so training can be time-consuming!

Interpretable Machine Learning



Statisticians:

- Data
  - i.e., clinical trials, surveys
- Black Box ML Model
- Interpretability Methods

Machine Learning specialists:

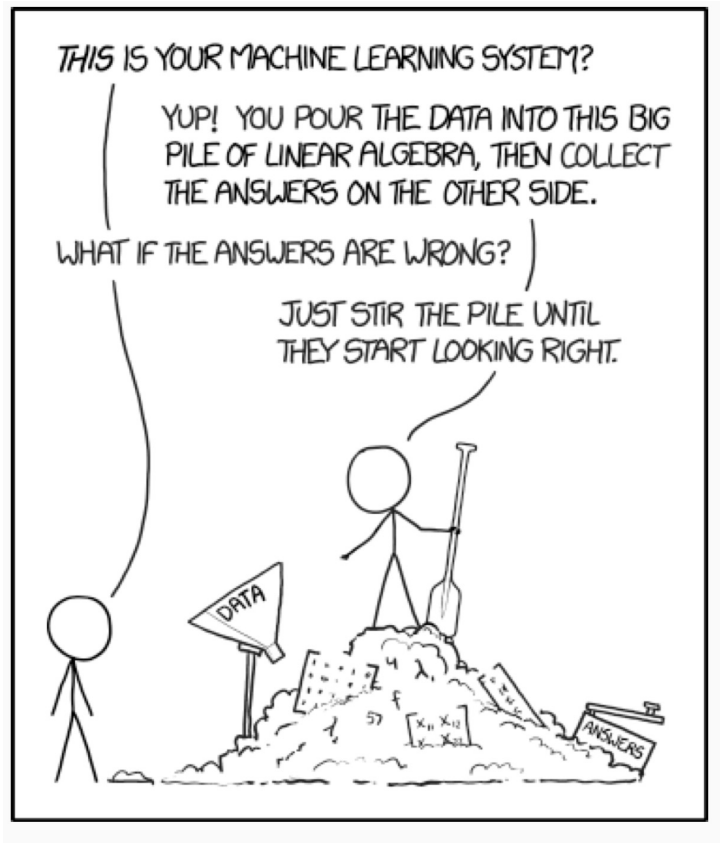
- Data
  - i.e., labeled samples of skin cancer images, crawling Wikipedia
- Black Box ML Model
- Interpretability Methods

Not covered in class

Stockholms universitet

Three levels of model interpretability

- Model level: what does a pattern or rule of our model look like?
- Data level: what is the most important dimension in the data?
- Prediction level: why did the model come up with a given prediction?



Not covered in class

Stockholms universitet

Importance

Knowing **WHAT** the prediction is and **WHY** it came to it enables:

- **Human curiosity:** Update existing mental models about the world
- **Uncover new knowledge from black-box models:** Where the model becomes the source of knowledge and not the data
  - *Useful in scientific disciplines like healthcare and genomics*
- **Safety:** Detect **edge cases** to take safety measures
  - *Understanding what makes a self-driving car detect bicycles could prevent false negatives*
- **Debugging tool:** For detecting **bias** in ML models such as racism, discrimination, etc.

Not covered in class

Scope of Interpretability

- **Algorithm Transparency**: "How does the algorithm create the model?"
- **Global, Holistic Model Interpretability**: *"How does the trained model make predictions?"*
- **Global Model Interpretability on a Modular Level**: *"How do parts of the model affect predictions?"*
- **Local Interpretability for a Single Prediction**: *"Why did the model make a certain prediction for an instance?"*
- **Local Interpretability for a Group of Predictions**: *"Why did the model make specific predictions for a group of instances?"*

Stockholms universitet

Not covered in class

Taxonomy of Interpretability Methods

Intrinsic or **Post hoc**

- Intrinsic: low complexity of the machine learning model
  - i.e., decision tree
- Post hoc: an interpretation method that analyzes the model after training

Model-specific or **Model-agnostic**

- Model-specific: an interpretation method that is limited to a specific model
- Model-agnostic: an interpretation method that can be used on any machine learning model

Local or **Global**

- Local: an interpretation method that explains an individual prediction
- Global: an interpretation method that explains the entire model behaviour

Not covered in class

LIME: Local Interpretable Model-Agnostic Explanations

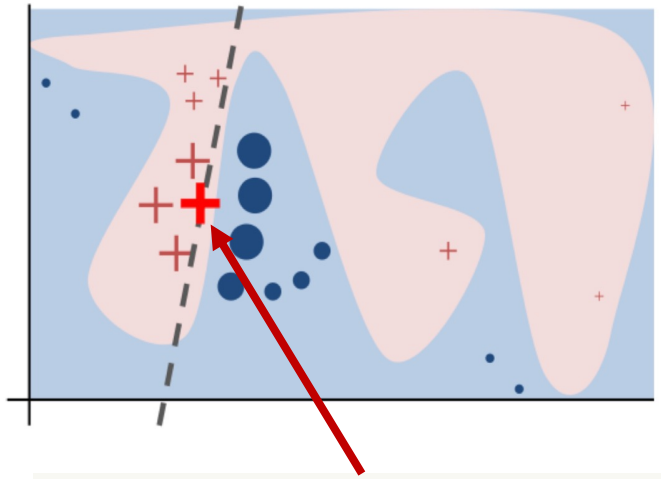
- **Goal:** we want to be model-agnostic
  - Learn the behavior of the underlying model without being dependent on the model itself
- How?
  - Perturb the input and see how the predictions change
  - Benefit in terms of interpretability!
  - We can perturb the input by changing components that make sense to humans, e.g., words or parts of an image
  - Even if the model uses much more complicated components as features, such as word embeddings

Not covered in class

LIME: Local Interpretable Model-Agnostic Explanations

Key intuition:

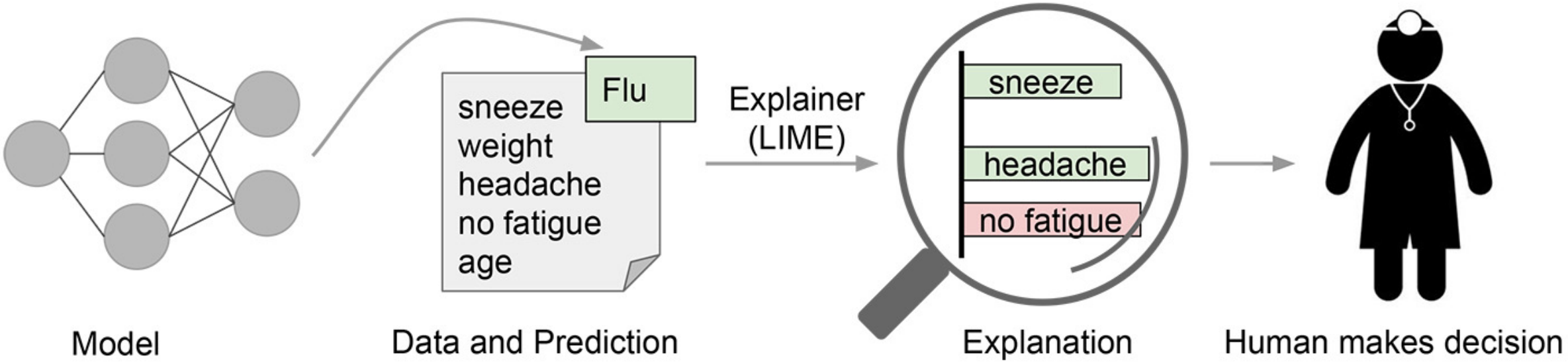
- It is much easier to approximate a black-box model by a simple model (surrogate) *locally* (in the neighborhood of the prediction we want to explain)
- As opposed to trying to approximate a model globally!



example that needs to be explained

- The **global** classification border is way too complex to understand!
- The **local** examples from both classes (closer to the test example) can be used to explain it
- Sample them and build simple/interpretable classifier components (e.g., linear SVMs)

Not covered in class



Not covered in class

Stockholms universitet

Example:
Google's pre-trained Inception neural network

(a) Original Image

Top 3 classes:

- "Electric Guitar"

  (p = 0.32)
- "Acoustic guitar"

  (p = 0.24)
- "Labrador"

  (p = 0.21)

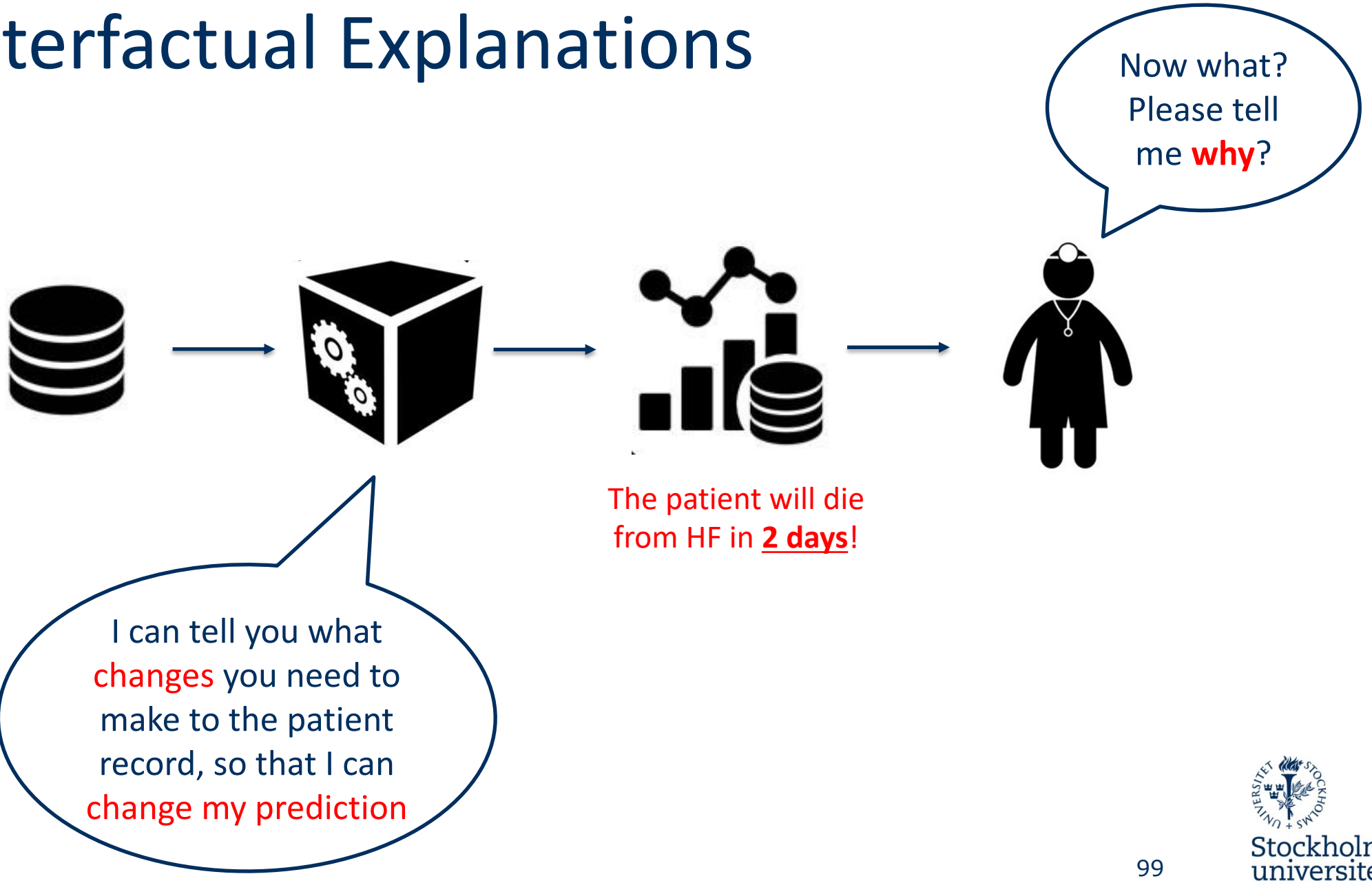
Not covered in class

Example:
Google's pre-trained Inception neural network

(b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Not covered in class

Counterfactual Explanations



Not covered in class

Stockholms universitet

Counterfactual Explanations

- **Idea:** imagine an **alternative reality** to establish causal relationships

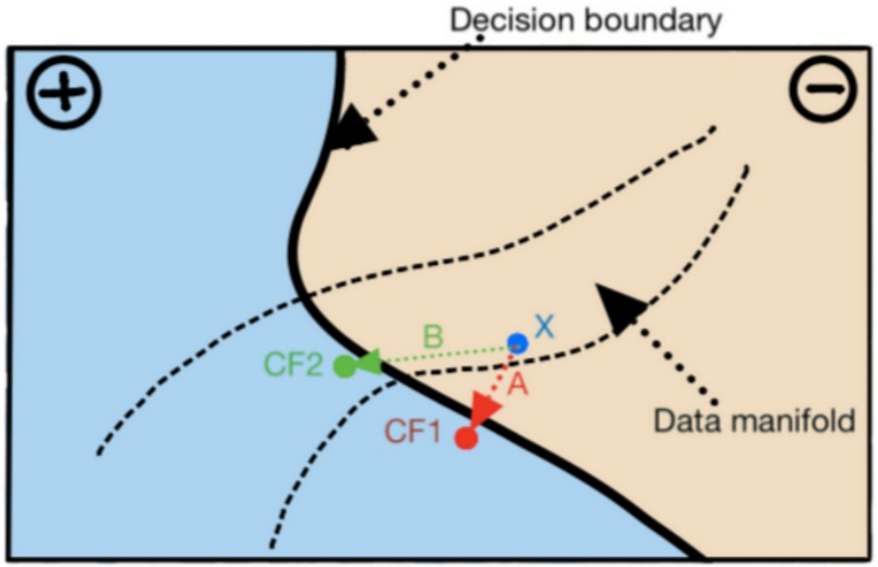
- Counterfactuals explain the **cause** of the prediction, not the actual phenomena captured in the data

- “A counterfactual explanation of a prediction describes the **smallest change** to the feature values that **changes the prediction** to a predefined output.”

Not covered in class

Counterfactual Explanations

- Given a data point **X** and its prediction **N** from a model, a counterfactual **CF** is a data point close to **X**, such that the model predicts it to be in a different class **P** (**N ≠ P**)
- Several counterfactuals can be generated for a data point (**CF1** and **CF2**), which differ in **closeness** to the original data point and other **desirable properties**
  - **actionability**: a feature that can actually change
  - **sparsity**: change only a few features
  - **proximity**: smallest possible change
  - **data manifold**: similar to other data points
  - **…**



Not covered in class

Stockholms universitet

Today…

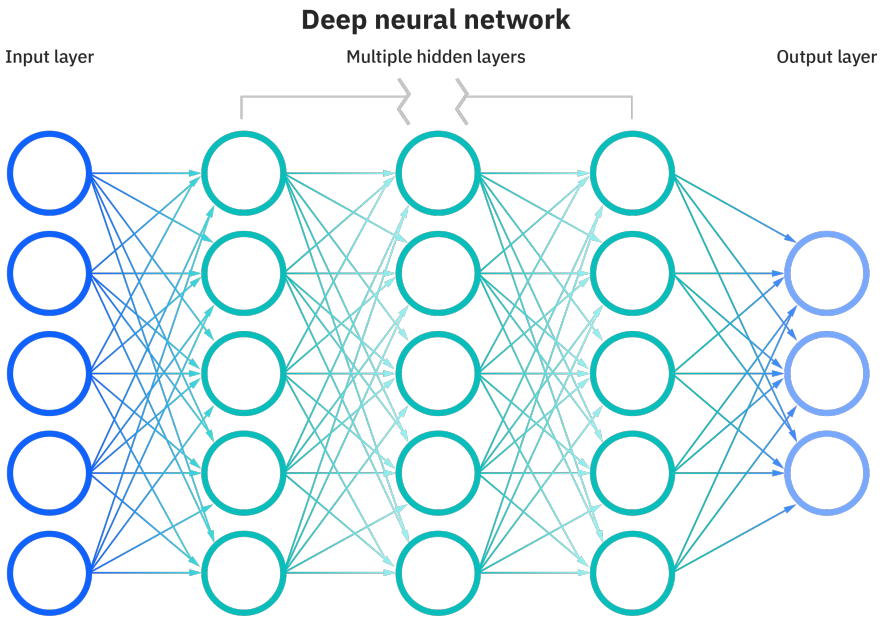
What is **Stacking**?

What is **Deep Learning**?

Why do we need **Intermediate Layers**?

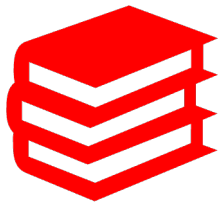
What is **Backpropagation**?

What are the ways to **explain** DL models?
(not covered in class)



Stockholms universitet

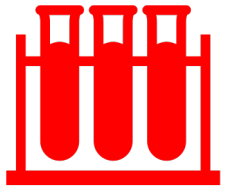
TODOs



Reading:

Main course book:

Chapter 12: 12.1, 12.2

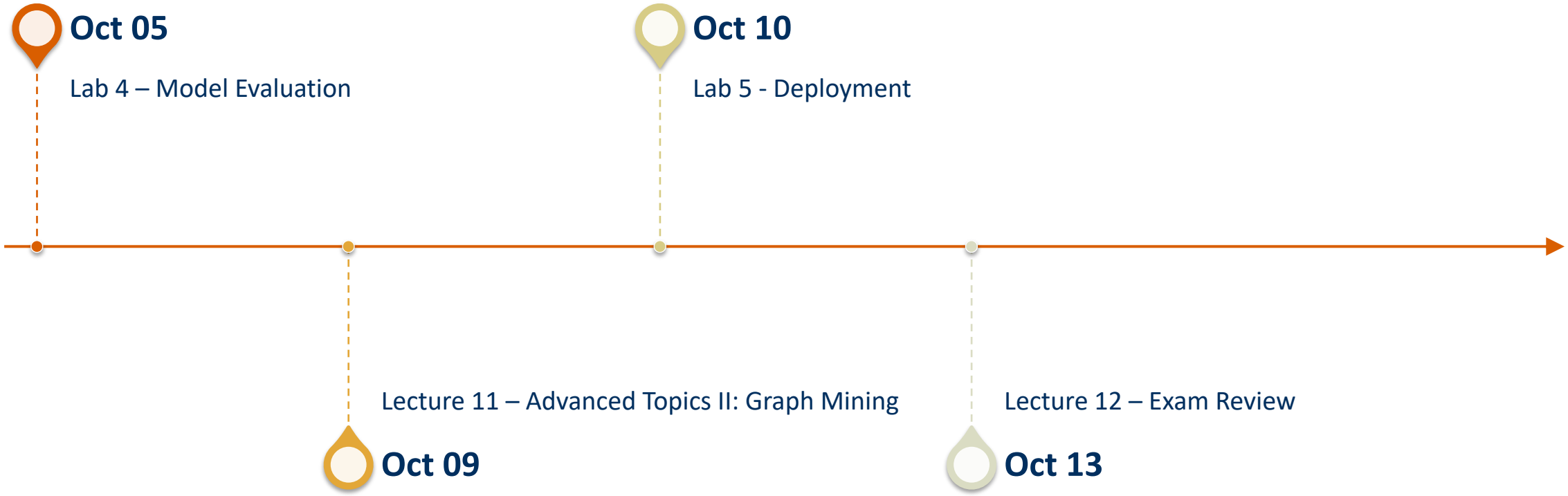


Lab 4

Recommended to complete the lab before the end of the week

Quiz 4

Coming up next



Stockholms universitet