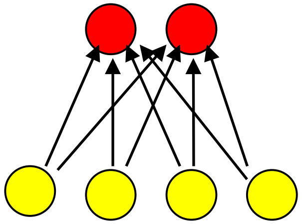
Neural Networks

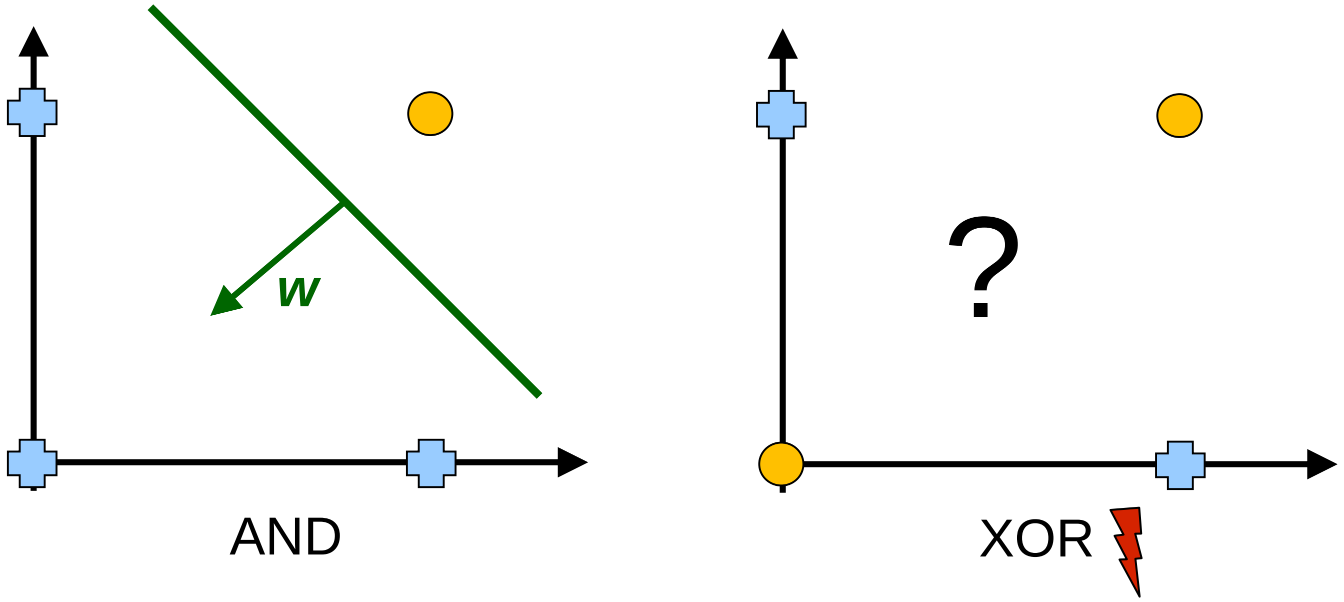
Lecture 3: Learning in Multilayer Networks

http://www.informatik.uni-hamburg.de/WTM/

Revision: The Perceptron

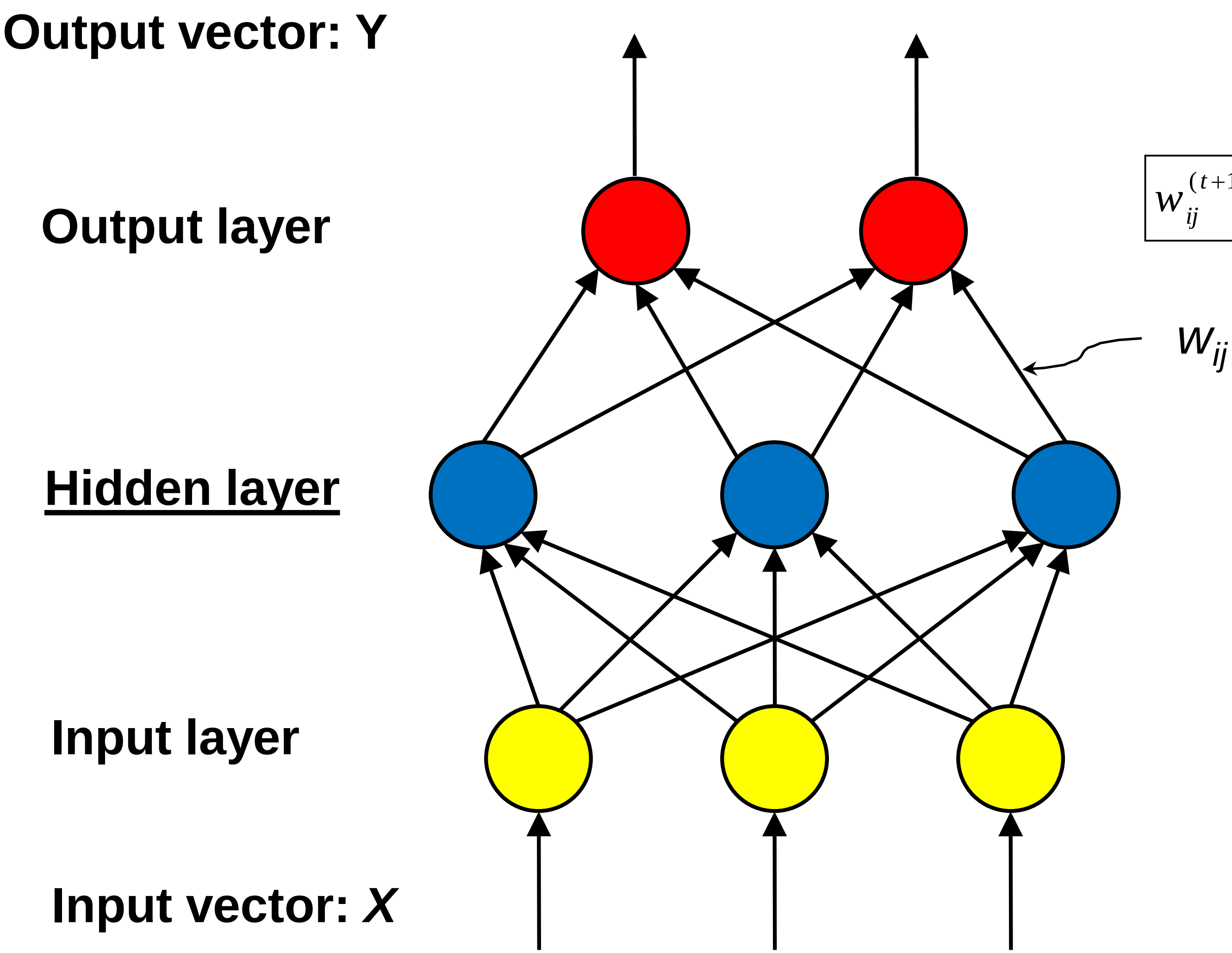


- The Perceptron is a linear classifier
- The update algorithm learns a *straight line* (decision boundary) that can separate the classes. It cannot learn the classes if the data is not linearly separable: (Minsky/Papert)



- How can this fundamental limitation be overcome?

The importance of the Neural Architecture - Solution: Multilayer Perceptron (MLP)

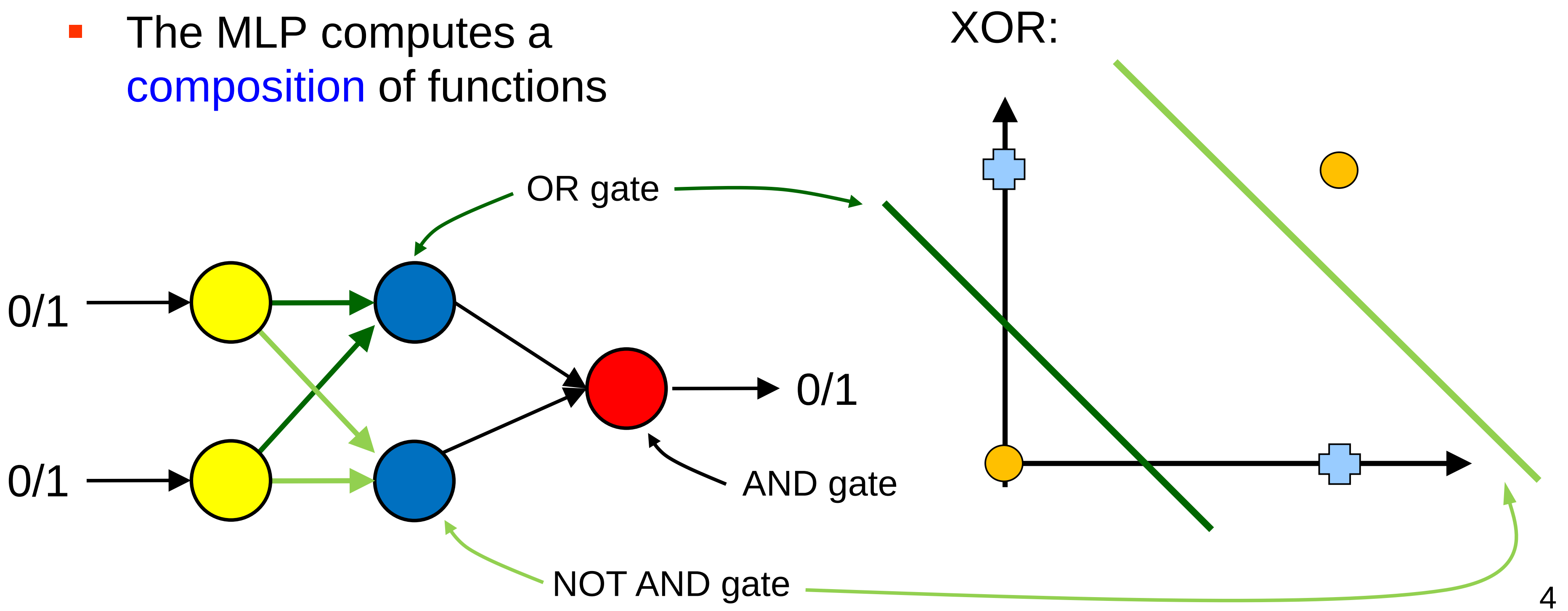


$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta (t_j - y_j) x_i$$

Learning XOR

- Need to use two lines to separate the classes
- MLP below can solve this task.

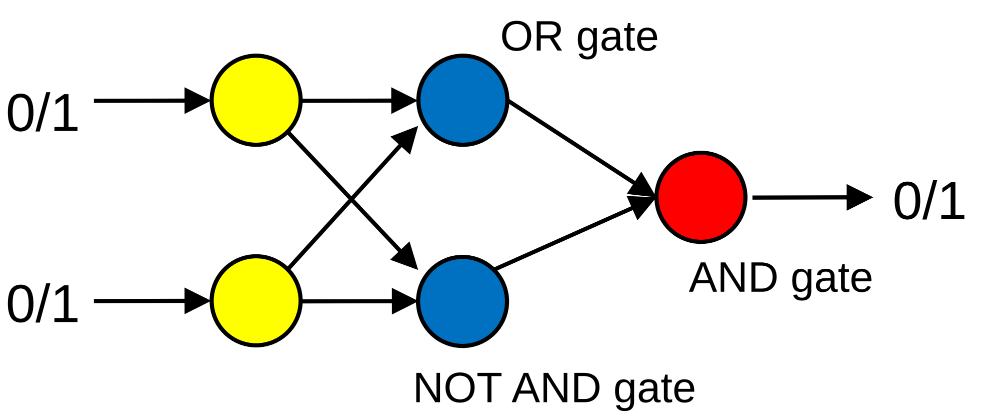
- Each hidden unit describes one of the two lines:
- The MLP computes a composition of functions



AND(OR(a,b), NAND(a,b))=XOR(a,b)

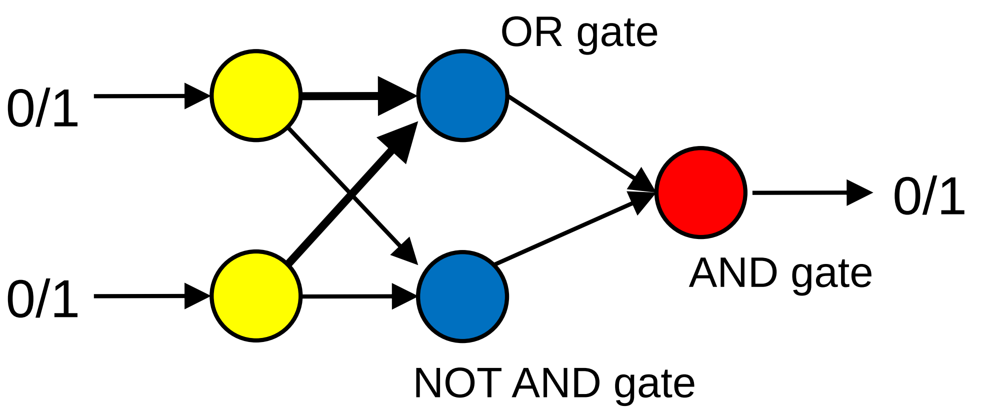
Learning XOR

x1	x2				
0	0				
0	1				
1	0				
1	1				



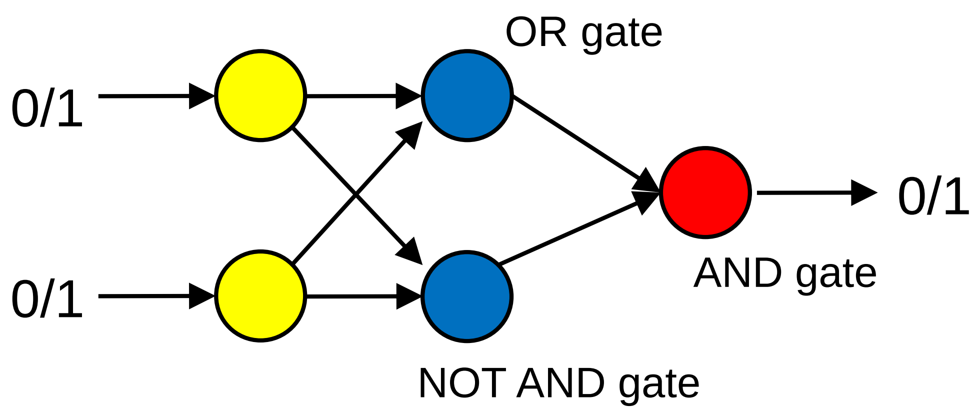
Learning XOR

x1	x2	OR			
0	0	0			
0	1	1			
1	0	1			
1	1	1			



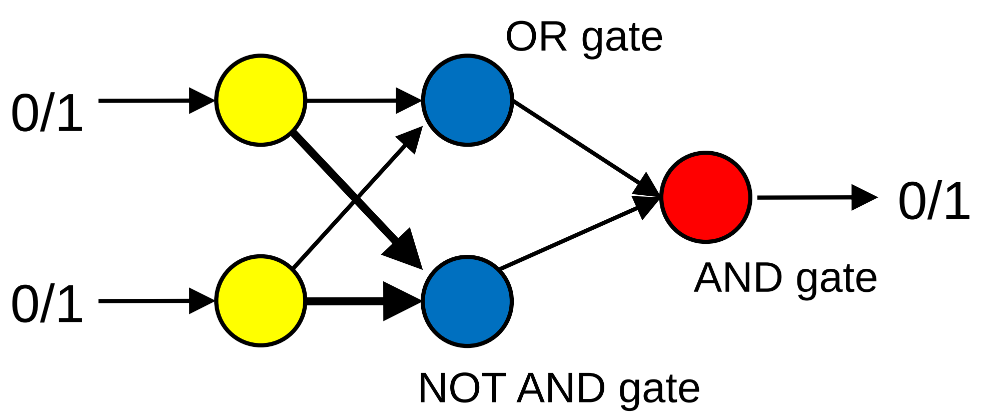
Learning XOR

x1	**x2**	**OR**	**AND**		
0	**0**	0	**0**		
0	**1**	1	**0**		
1	**0**	1	**0**		
1	**1**	1	**1**		



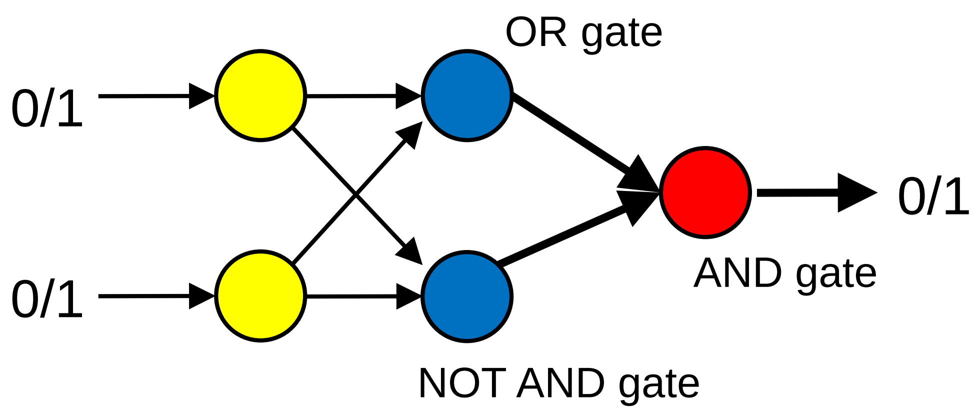
Learning XOR

x1	**x2**	**OR**	**AND**	**NOT AND**	
0	**0**	0	0	**1**	
0	**1**	1	0	**1**	
1	**0**	1	0	**1**	
1	**1**	1	1	**0**	



Learning XOR

x1	x2	OR	AND	NOT AND	XOR
0	0	**0**	0	**1**	**0**
0	1	**1**	0	**1**	**1**
1	0	**1**	0	**1**	**1**
1	1	**1**	1	**0**	**0**



Learning with hidden units

- Networks *without hidden* units very *limited* in input-output mappings
  - More layers of *linear* units do not help since still linear
- Need *multiple* layers of *adaptive non-linear* hidden units ($\rightarrow$ *universal approximator*)
- How can we learn?
  - Need an efficient way of adapting *all* the weights, not just the last layer
  - No target labels in the hidden layers
  - Learning the weights going into hidden units is equivalent to learning features

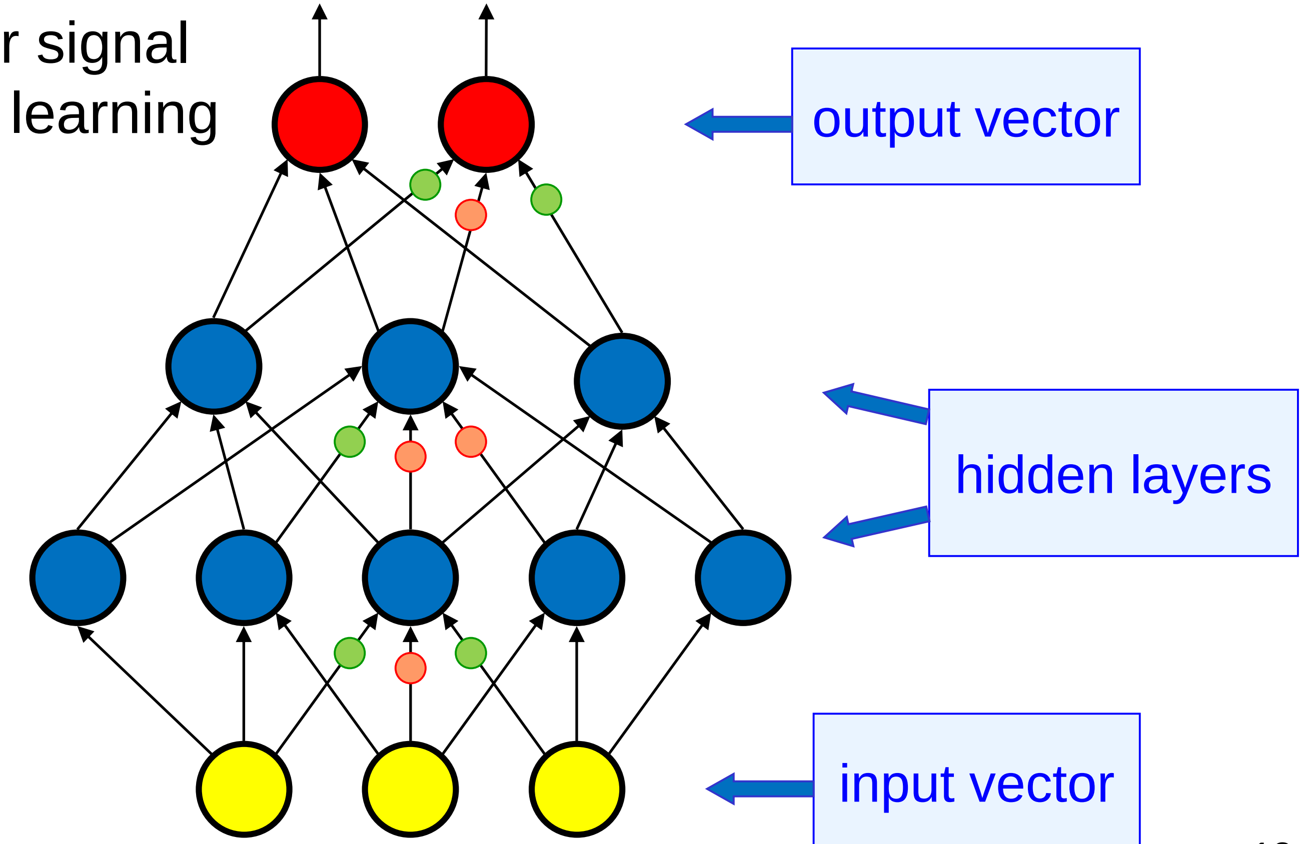
The concept behind backpropagation

- Instead of using desired activities to train the hidden units, use error derivatives for hidden activities
- Each hidden unit can affect many other “higher” hidden or output units
- From error derivatives for hidden activities, we can get error derivatives for the weights going into a hidden unit
- This learning rule is called **backpropagation**.
  It is a generalization of the delta rule (see last week)

Learning by back-propagating error derivatives

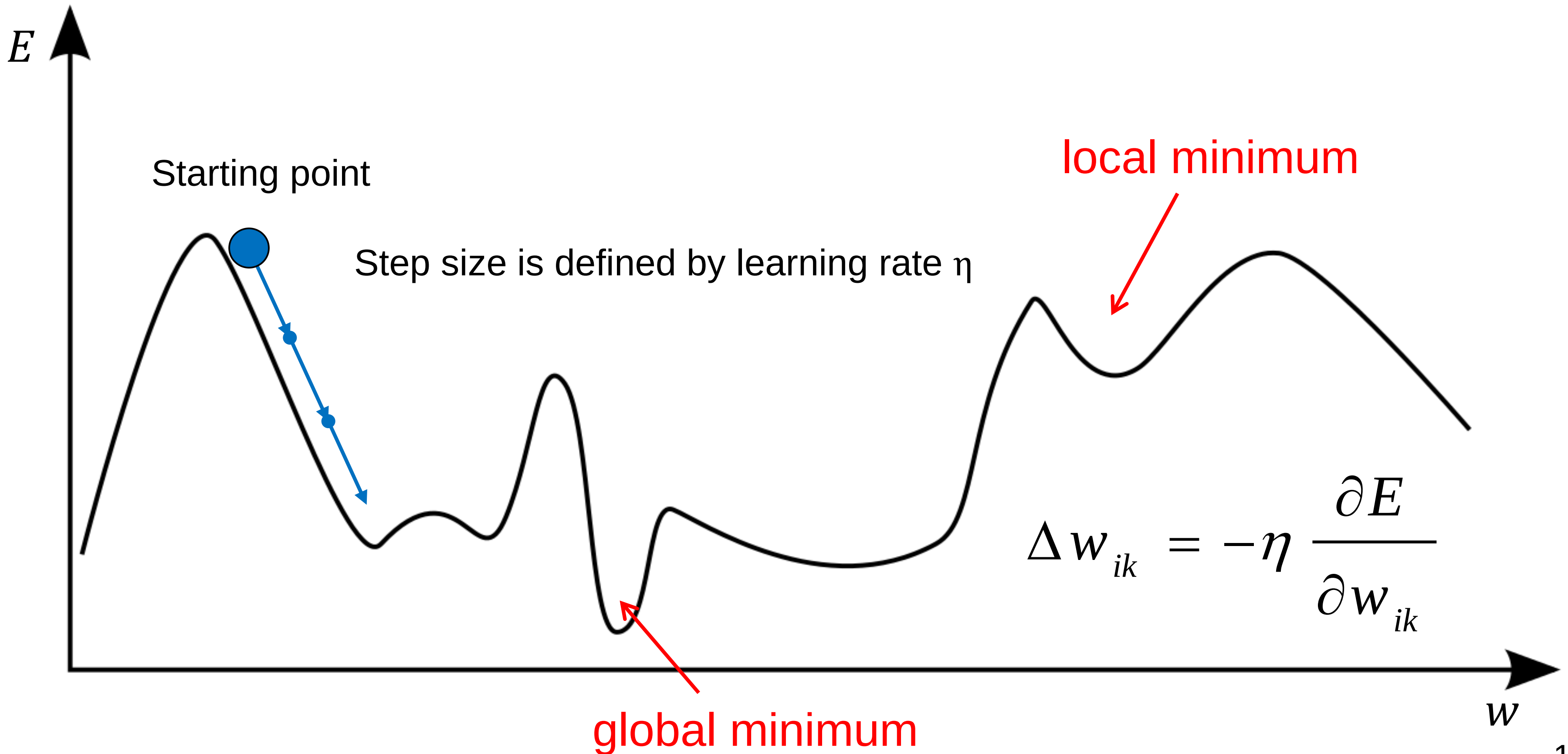
Compare outputs with correct answer
to get error signal

Back-propagate error signal
to get derivatives for learning



Gradient descent

- Backpropagation is a *gradient descent* method to find the minimum error on the error surface:



$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}}$$

Error Terms

- Need to differentiate the sigmoid function
- Gives us the following *error terms* (deltas)
  - For the outputs

$$\delta_k = (t_k - y_k) y_k (1 - y_k)$$

  - For the hidden nodes from the input activities a

$$\delta_j = a_j (1 - a_j) \sum_k w_{jk} \delta_k$$

Update Rules

- This gives us the necessary update rules
  - For the weights connected to the outputs:

$$w_{jk} \leftarrow w_{jk} + \eta \delta_k a_j^{\text{hidden}}$$

  - For the weights connected to the hidden nodes:

$$v_{ij} \leftarrow v_{ij} + \eta \delta_j x_i$$

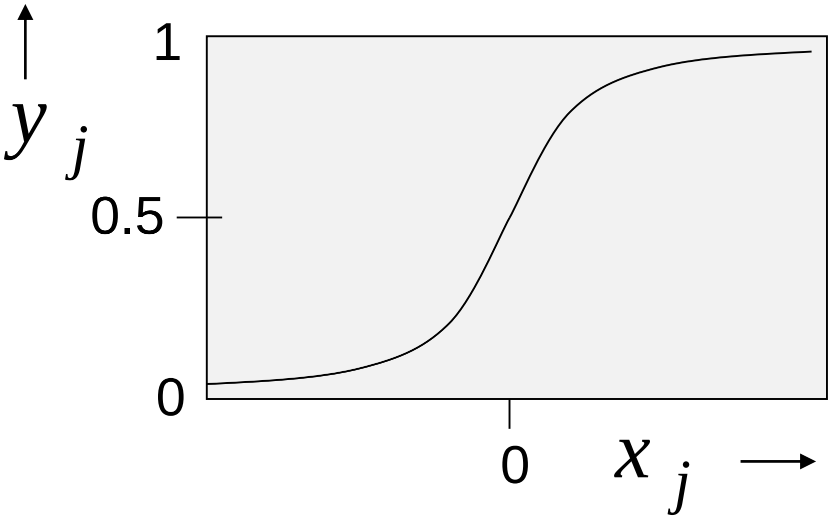
Non-linear neurons with smooth derivatives

- For backpropagation, we need neurons that have well-behaved derivatives.
  - Typical: the logistic/sigmoid function
  - The output is a smooth function of the summed input

$$x_j = b_j + \sum_i y_i w_{ij}$$

$$\boxed{y_j = \frac{1}{1 + e^{-x_j}}}$$

$$\frac{dy_j}{dx_j} = y_j (1 - y_j)$$



Sketch of the backpropagation algorithm on a single training case

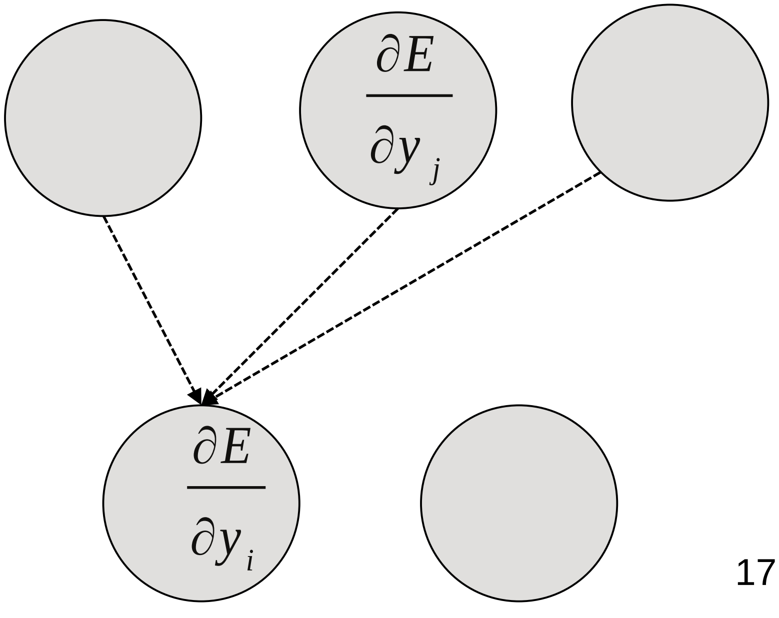
- 1. convert difference between each output and its target value into an error derivative

$$E = \tfrac{1}{2}\sum_{j} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = t_j - y_j$$

- 2. compute error derivatives in each hidden layer from error derivatives in the layer above

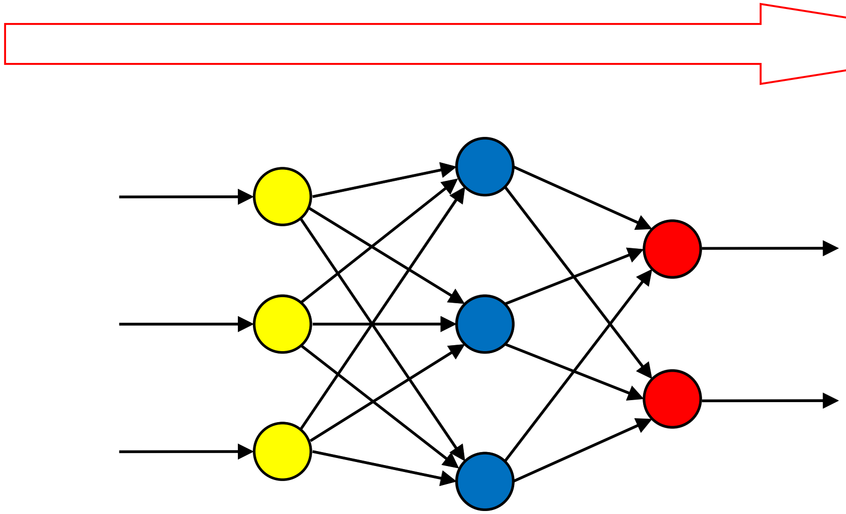
- 3. use error derivatives for activities to get error derivatives for the weights.



Training MLP

(1) Forward Pass

- Put the input values in the input layer
- Calculate the activations of the hidden nodes
- Calculate the activations of the output nodes
- Calculate the errors using the targets



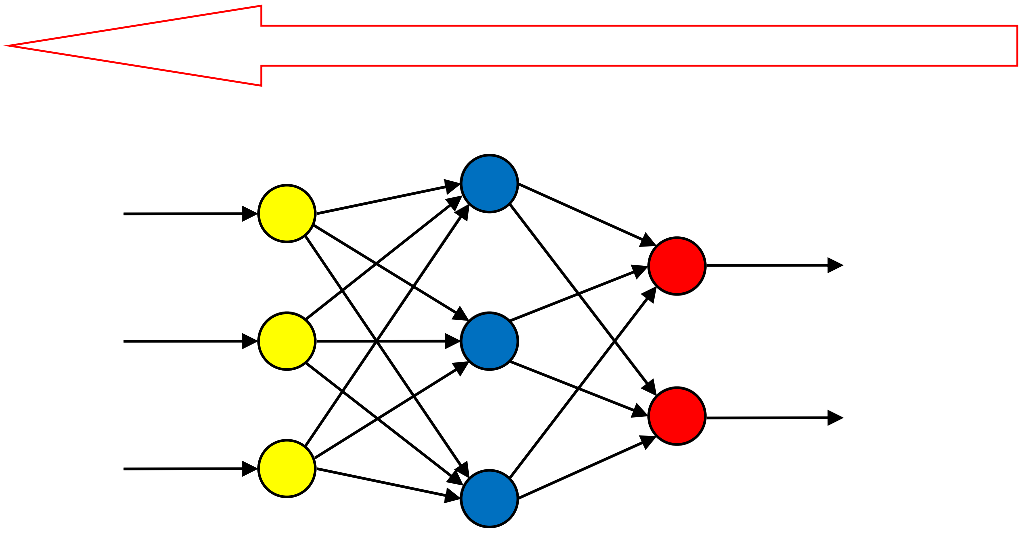
For example

$$t_j - y_j$$

Training MLPs

(2) Backward Pass

- From output errors, update last layer of weights
- From these errors, update next layer
- Work backwards through the network
- Error is backpropagated through the network



Training issues

- How to update the weights?
  - After each training sample: Incremental/Online training
  - At the end of each epoch:
    - Batch training: Epoch ends when all training samples seen
    - Mini-batch training: Epoch ends when subset of training data seen

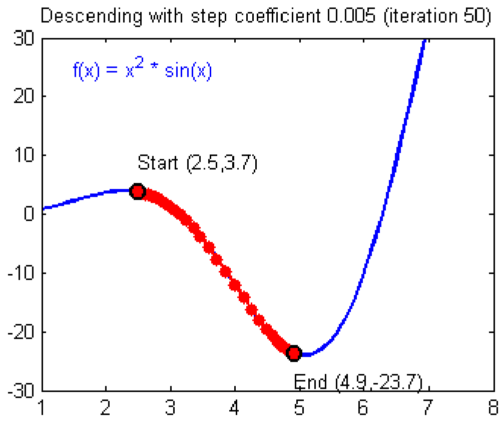
- How to choose the learning parameters?
  - Use a fixed learning rate
  - Adapt the learning rate?
  - Use steepest descent or not?
  - Add momentum?

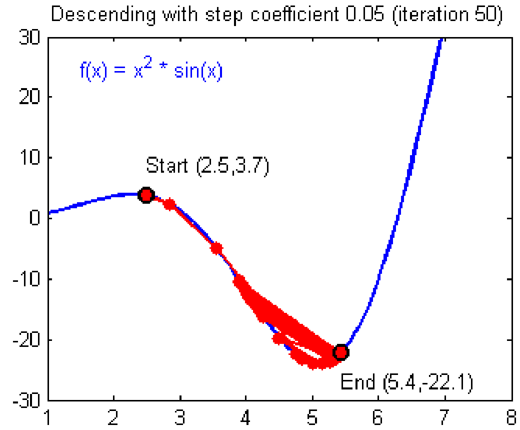
Learning rate and momentum

- Tuning the learning rate
  - Too small: cannot escape local minima.
  - Too large: overshooting narrow valleys

- Momentum term $$\Delta w_{jk}(t) = \eta \delta_k y_j + \mu w_{jk}(t-1)$$
  - Keep fraction of previous gradient
  - Gradient keeps pointing in same direction: Increase step size
  - Gradient keeps oscillating in valley: Slow down

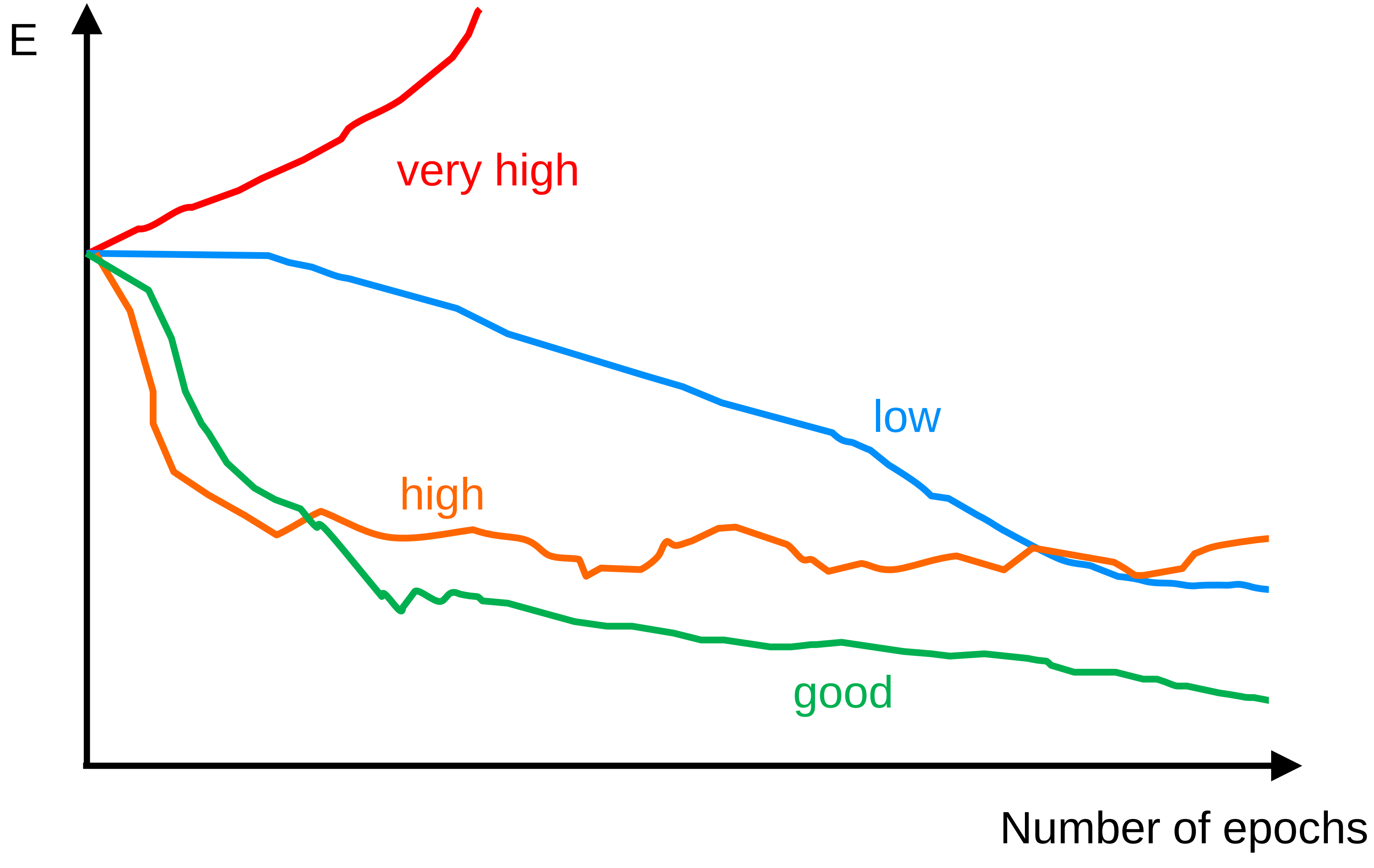
small





large

Learning rate

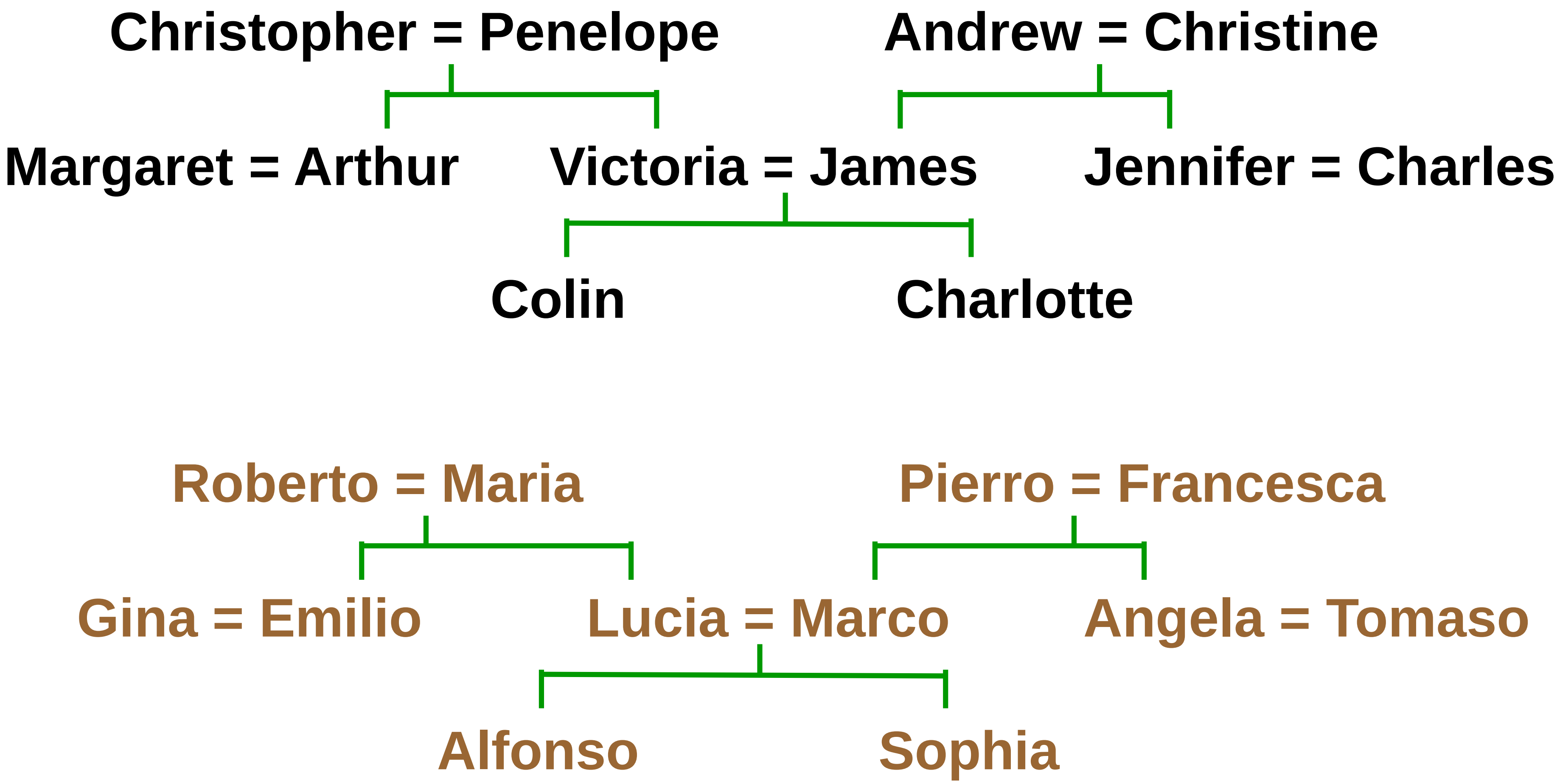


What can be learned with MLP and how to evaluate

- Supervised learning problems with input and output classes

- What problems:
- Classification problems (obvious supervised learning problem)
- Regression problems (obvious supervised learning problem)
- Prediction problems (obvious supervised learning problem)
- Symbolic knowledge relationships? (harder? not so obvious?)

Example Problems: relational information



(Hinton 1986)

Another way to express the same information

- Using the 12 relationships:
  - son, daughter, nephew, niece
  - father, mother, uncle, aunt
  - brother, sister, husband, wife

- Make a set of propositions
  - (colin has-father james)
  - (colin has-mother victoria)
  - (james has-wife victoria) this follows from the two above
  - (charlotte has-brother colin)
  - (victoria has-brother arthur)
  - (charlotte has-uncle arthur) this follows from the above

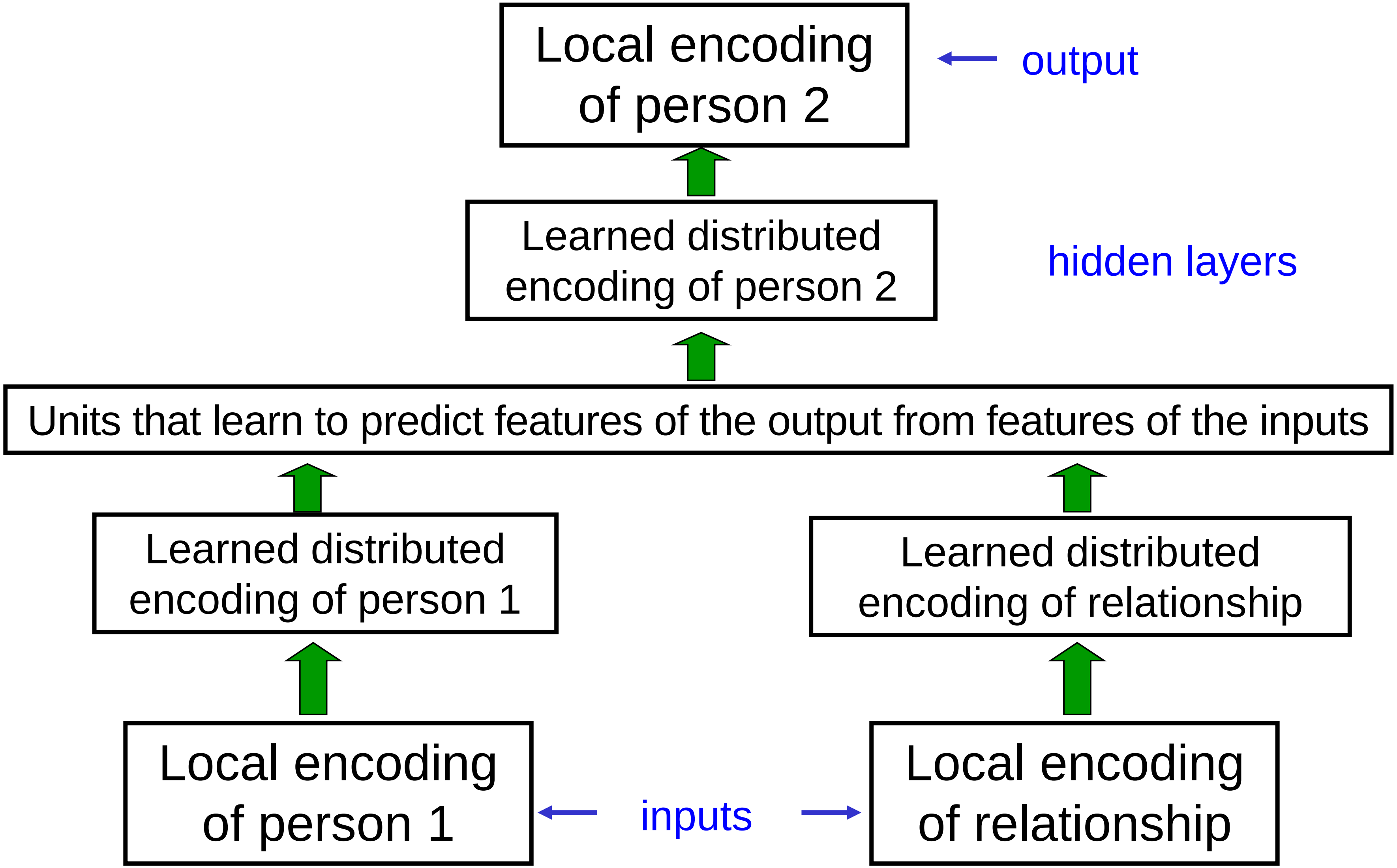
A relational learning task

- Given a large set of triples that come from some family trees, figure out the regularities.
  - The obvious way to express the regularities is as symbolic rules

`(x has-mother y) & (y has-husband z) => (x has-father z)`

- Finding symbolic rules involves a difficult search through a very large discrete space of possibilities
- Can a neural network capture the same symbolic knowledge by searching through a continuous space of weights?

One possible structure of a neural net



One way to see that it works

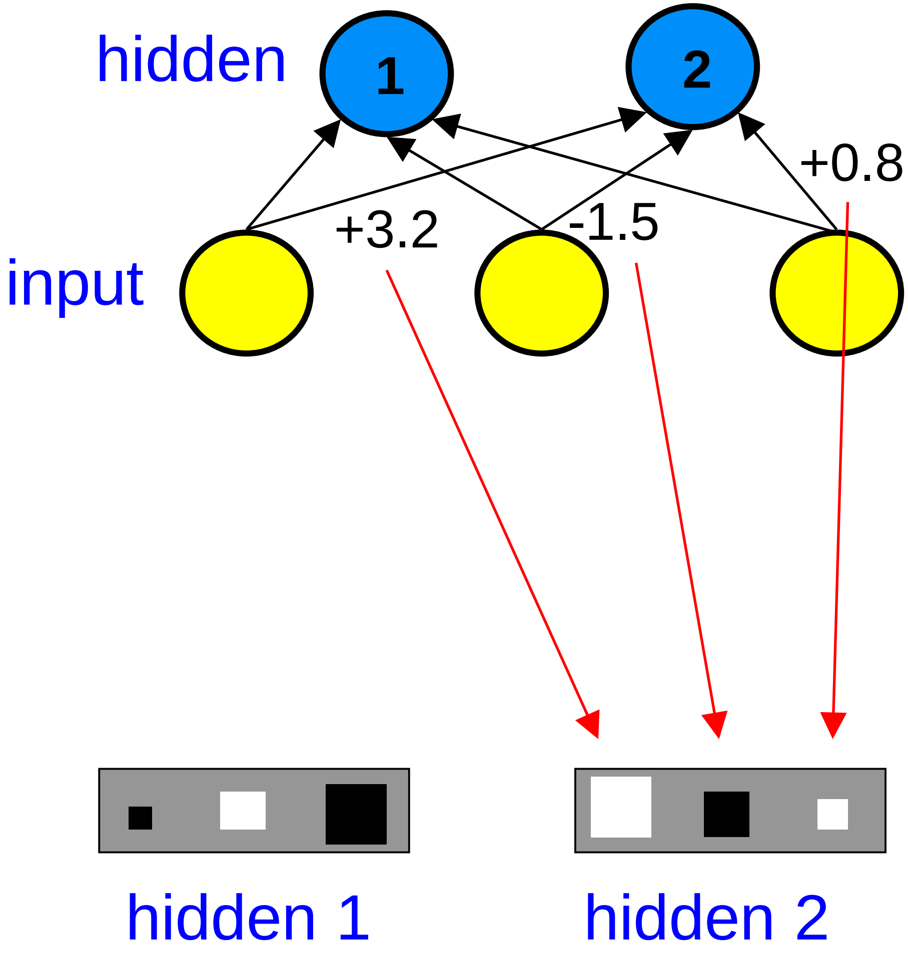
- Train the network on all but some of the triples that can be made using the 12 relationships
  - It needs to sweep through the training set many times adjusting the weights slightly each time.

- Test the network

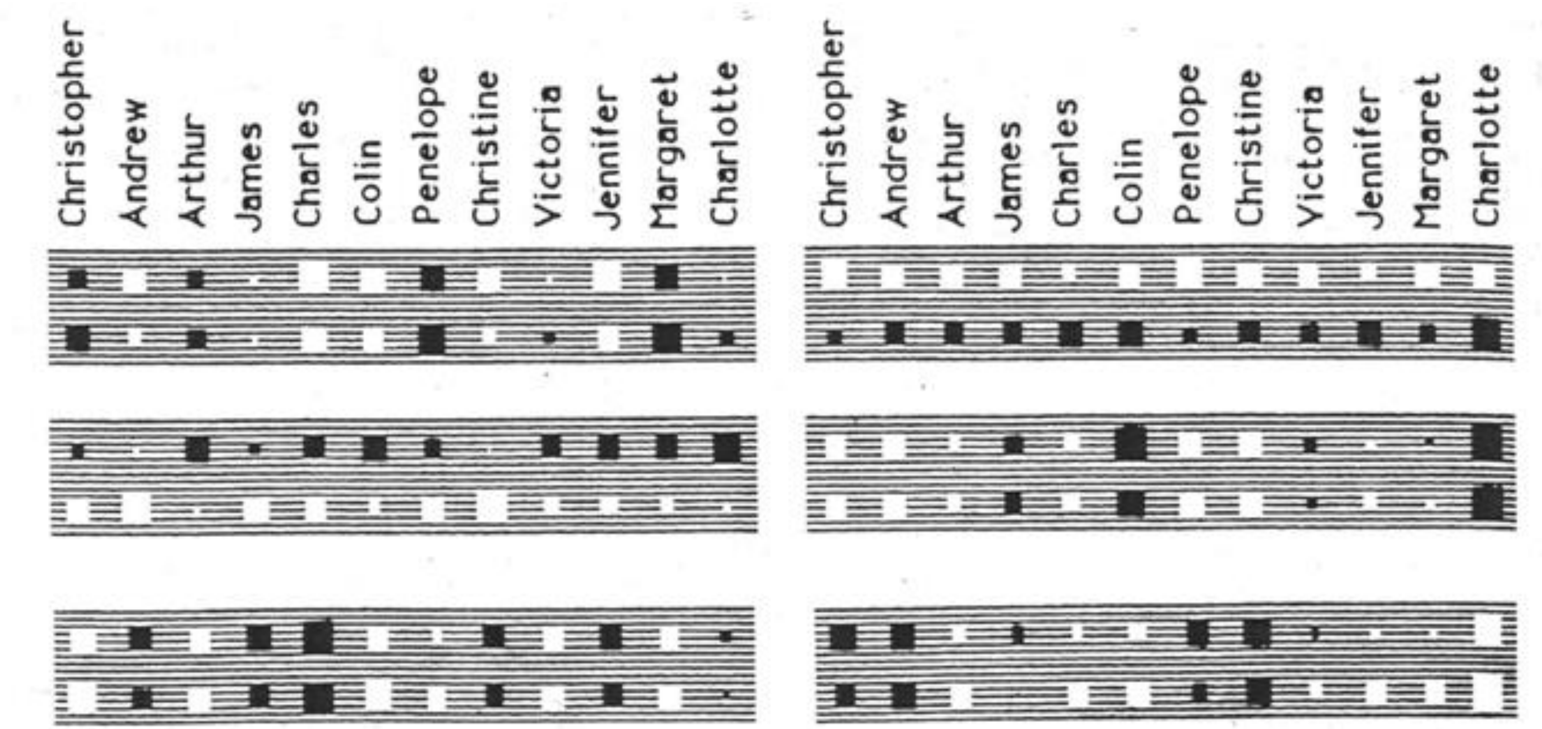
- But what is the knowledge in a network ?

How to interpret the weights of hidden units

- Obvious method is to show numerical weights for connections:
  - Try showing 25,000 weights this way…
- Its better to show weights as black or white boxes in the locations of the neurons that they come from
  - Better use of pixels
  - Easier to see patterns



The weights of 24 input units (person) to 6 hidden units in second layer



Christopher = Penelope **Andrew = Christine**

Margaret = Arthur **Victoria = James** **Jennifer = Charles**

Colin **Charlotte**

What the network learns

- Six hidden units connected to the input representation of person 1 learn to represent useful features for predicting answer.
  - Nationality, generation, branch of the family tree.
- Higher central layer learns how features predict other features.
  - For example:

    Input person is of generation 3

    and

    relationship requires answer to be one generation up

    implies

    Output person is of generation 2

Why this is interesting

Debate in cognitive science between two rival theories about concept representation:

- *Feature theory*: Concept is a set of *semantic features*.
  - good for explaining similarities between concepts

- *Structuralist theory*: Concept meaning is in *relationships to other concepts*
  - conceptual knowledge best expressed as a relational graph

A neural net can use *semantic features to implement the relational graph*

- it learns the intuitively obvious consequences of the facts
- no explicit inference required (-> word2vec)

Other prediction and classification examples

- We cannot identify phonemes perfectly in noisy speech
  - The acoustic input is often ambiguous: there are several different words that fit the acoustic signal equally well.

- People use their understanding of the meaning of the utterance to hear the right word.
  - We do this unconsciously and well

- Speech recognizers have to know which words are likely to come next
  - Can this be done without full understanding?

The traditional standard “trigram” method

- Count frequencies of all triples of words of text. Then use these frequencies to make bets on the next word in a b ?

$$\frac{p(w_3 = c \mid w_2 = b, w_1 = a)}{p(w_3 = d \mid w_2 = b, w_1 = a)} = \frac{count(abc)}{count(abd)}$$

- Until recently this was state-of-the-art.
  - We cannot use a bigger context because there are too many quadgrams
  - We have to “back-off” to bigrams when the count for a trigram is zero.
    - The probability is not zero just because we did not see one.

Why the trigram model is inefficient

- Example sentence

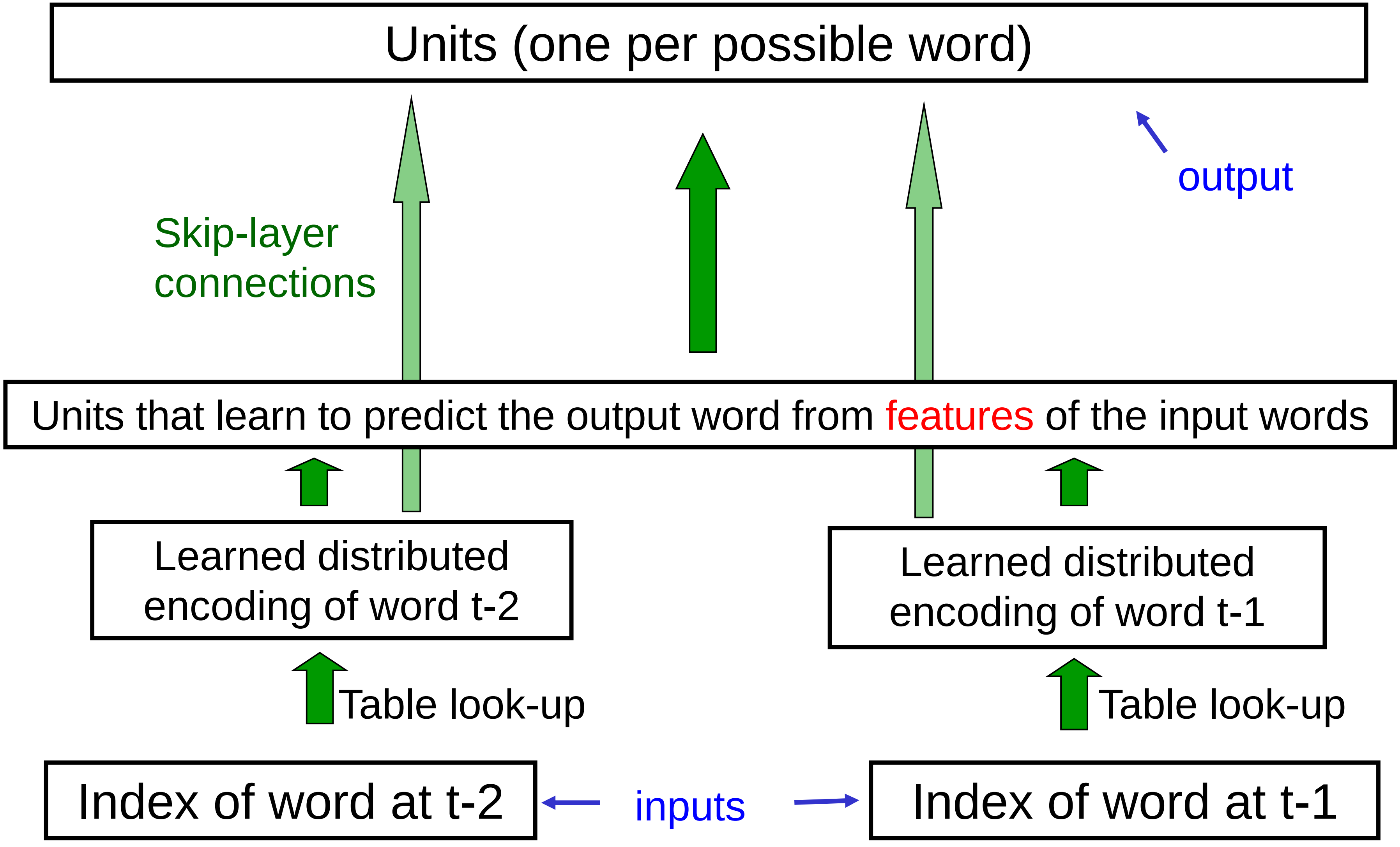
  "the cat got squashed in the garden on friday"

- should to predict words in

  "the dog got flattened in the yard on monday"

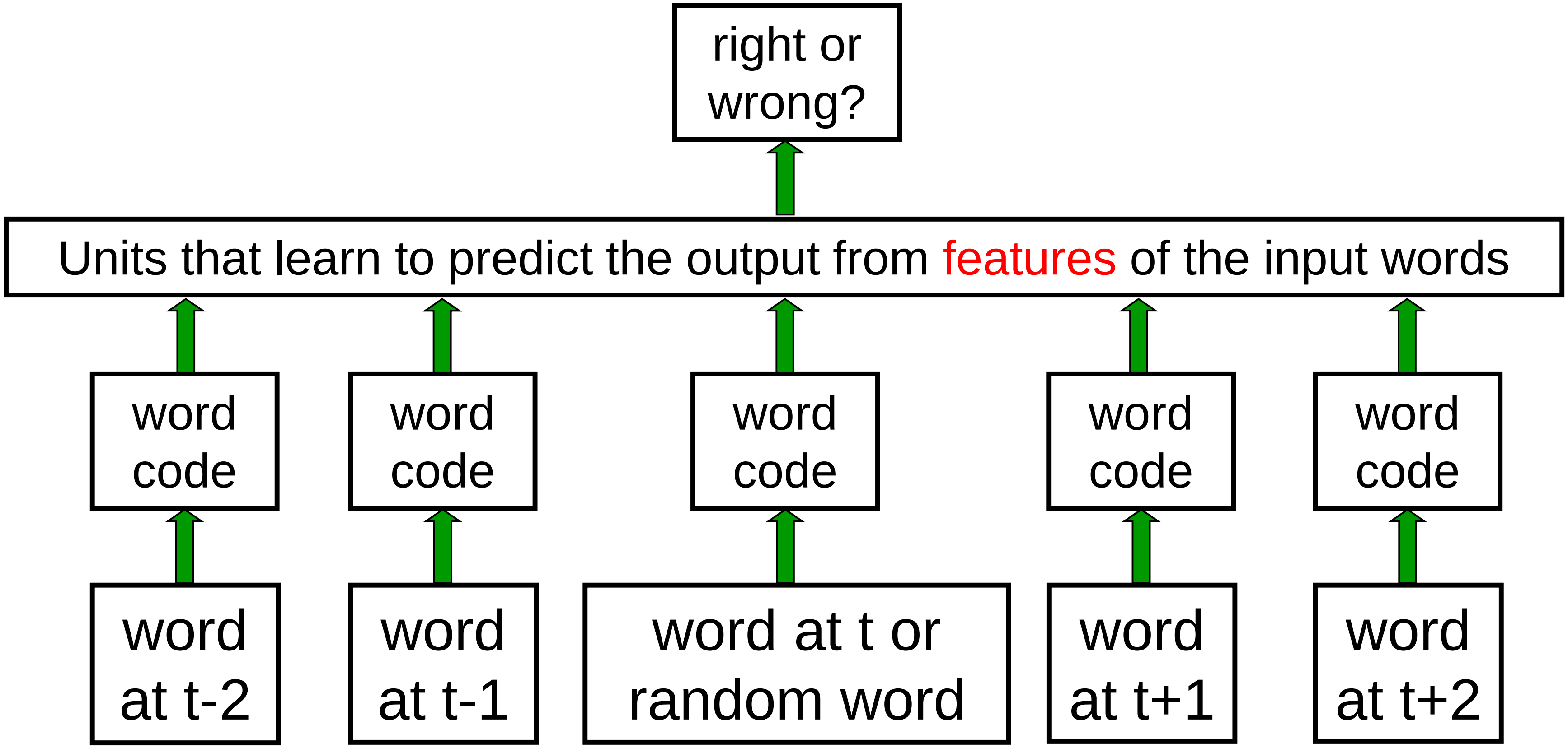
- A trigram model does not understand the similarities between
  - cat/dog   squashed/flattened   garden/yard   friday/monday
- Need to use *features* of previous words to predict features of next word
  - Using a feature representation and a learned model of how past features predict future ones, we can use many more words from the past history.

Neural prediction: Bengio's neural net for predicting the next word



Neural classification (Collobert and Weston network)

- Learn to judge if a word fits the 5-word context on either side of it. Train on ~600 million words.

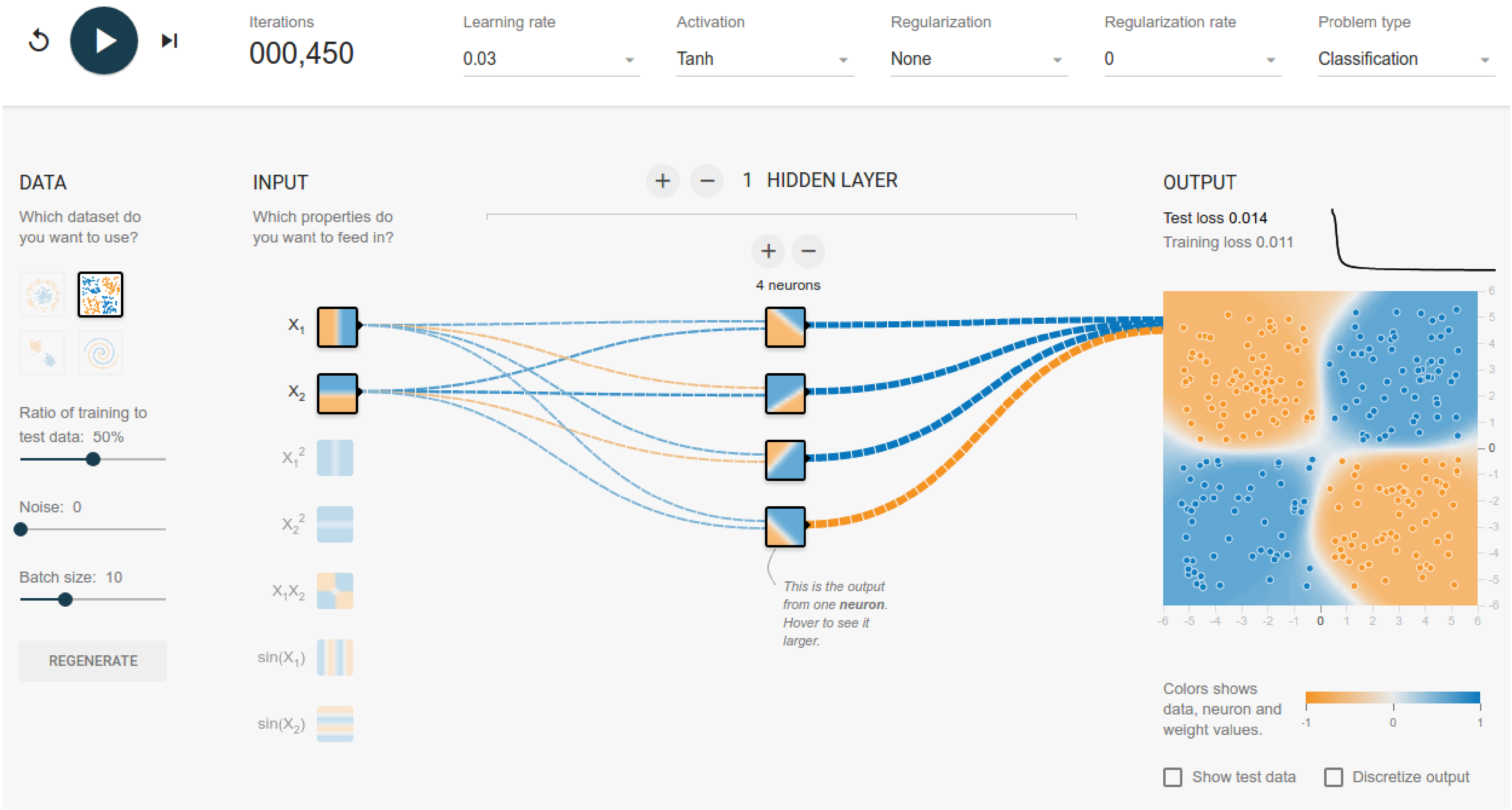


Further success stories

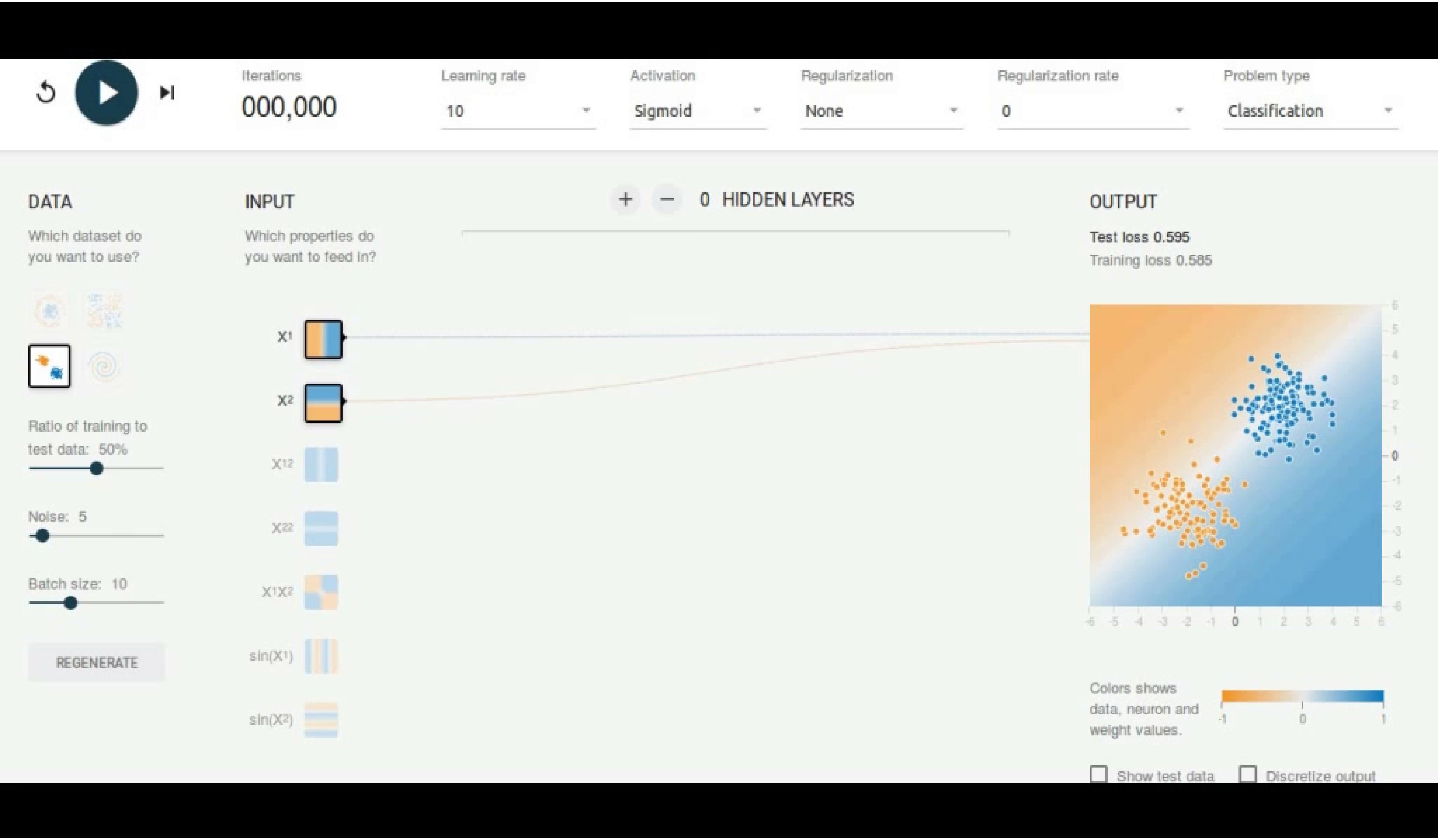
- Backpropagation has been used for a large number of practical applications

  - Recognizing hand-written characters
  - Predicting the future price of stocks
  - Detecting credit card fraud
  - Recognizing speech
  - Predicting the next word in a sentence from the previous words
  - Understanding the effects of brain damage

Try out your first neural network at a high level: playground.tensorflow.org



Demonstration



Universal Approximation Theorem

- The MLP is a universal approximator:
  - For *every* continuous, measurable function there is a Multilayer Perceptron (MLP) with *1 hidden layer* and a finite number of units which approximates this function

- But careful:
  - We cannot guarantee an *exact* solution
  - We cannot guarantee that this theoretically existing network can in fact be *learned* with gradient descent

[White89: Multilayer feedforward networks are universal approximators]

Summary

- Multilayer Perceptrons are very effective in various *classification, prediction, regression* tasks.

- MLPs can be trained with *gradient descent* methods, e.g. the backpropagation algorithm
  - Suitable and *derivable transfer functions* are necessary.
  - Reasonable stopping criteria help to avoid *overfitting*.
  - Divergence during training can be avoided using *momentum*.

- MLPs also can be used to *predict features* of future events. Extension: Recurrent connections...

Further Reading

- Recap:
  - Marsland (Chapter 3),
  - Goodfellow (Chapter 6.1 and 6.5)
  - Rojas (Chapters 6.1-7.3),
- Visual Proof of universal approximation theorem: http://neuralnetworksanddeeplearning.com/chap4.html
- http://playground.tensorflow.org/
- More in CommSy (Material > *Literature and Background*)