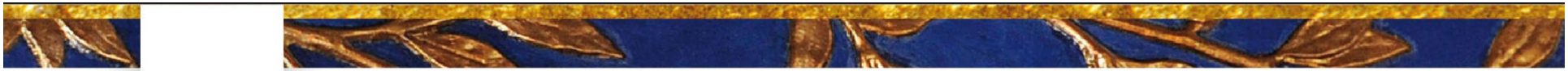
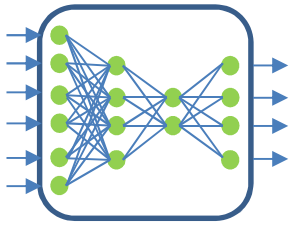


UPPSALA
UNIVERSITET

Natural Computation Methods in Machine Learning (NCML)

Lecture 5: More on Backprop, extensions and variants

UPPSALA UNIVERSITET



Challenges

From the previous lecture

- The loss function could be any differentiable function, but is very often assumed to be the squared error
  - Which part of the update equations would change, if we replaced the loss function?
- If we use pattern learning, why don't we update the weight directly, in step 4?
- What would happen if we initialized all weights to zero, instead of small random values?
- In a deep network (many layers) the chain of partial derivatives gets very long. A problem?

UPPSALA UNIVERSITET

Super Challenge

From the previous lecture

- As shown in the previous lecture and this one:

  $ab, \overline{ab}, a+b, \overline{a+b}$, are all linearly separable,

  $(a+b)\overline{c}$ (the three input solution to XOR) is too,

  but $a \oplus b = a\overline{b} + \overline{a}b = (a+b)\overline{ab}$ is not!

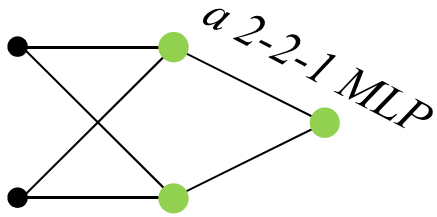
- *Where is the limit? Under which condition(s) is a Boolean expression linearly separable, when viewed as a classification problem?*

Discussion at the end of this lecture

UPPSALA UNIVERSITET

How to use Backprop

and when to stop training

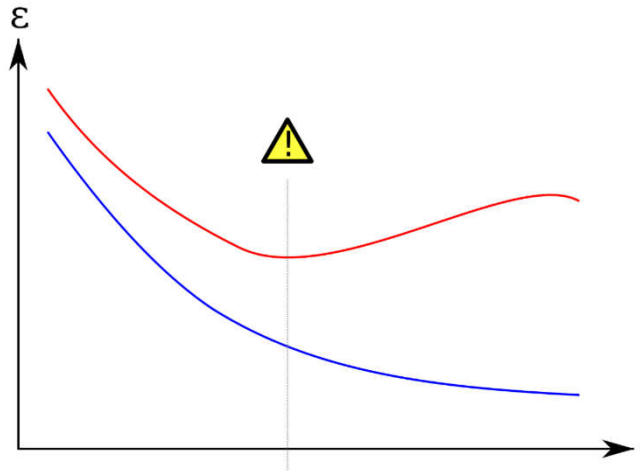


- Decide network structure/size
  - Common naming convention: N-M-K
- Collect data – a set of input-target pairs, $<\overline{x}, \overline{d}>$
- Split the data in two:
  - a training set (larger), used to update weights
  - a test set (smaller), left-out during training
- After training, measure how well the network generalizes, by computing the error of the examples in the test set

UPPSALA UNIVERSITET

Overfitting/overtraining

- Problem: The algorithm tries to minimize the error of the *training set*
- We may start to overfit, storing information instead of finding the underlying generating function



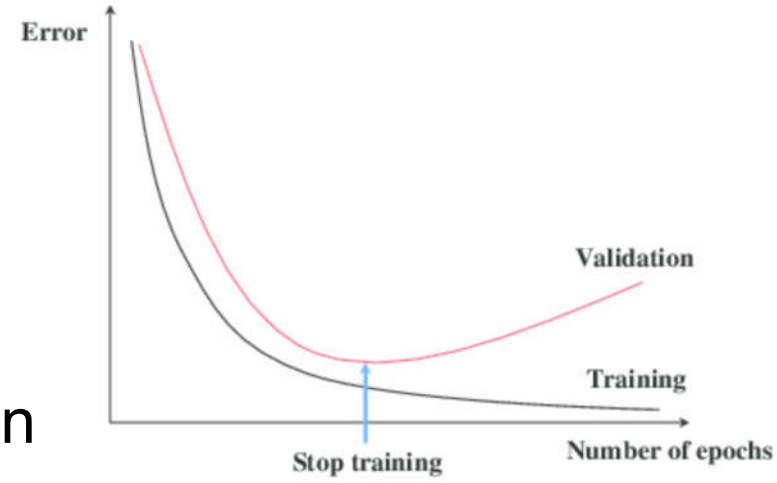
- Worst case: A network with **too many weights**, which is **trained for too long**, on **too little data**

UPPSALA UNIVERSITET

Early Stopping

A common *regularization* technique

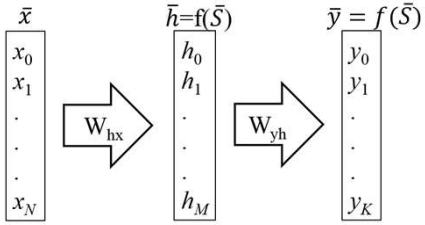
- Split data in three sets
  - a training set for training — *Largest*
  - a validation set to decide when to stop (when E is at minimum) — *Smallest*
  - a test set, as before, to evaluate the result (test for generalization)
- Requires lots of data
- Why not use the test set to decide when to stop?
  - It was used to select a solution
  - Not fair to use also to evaluate



UPPSALA UNIVERSITET

Training set size

- The number of training samples should be much larger than the number of weights (N).
  - Aim for at least N^2 *(with a very large grain of salt)*
- What to do if too little data?
  - Get more! (if possible)
  - Reduce number of weights (, nodes and layers)
  - Data augmentation
    - Noise injection. Add some noise to the input values!
    - If inputs are images, make rotation/mirror/scaling variants
  - Other regularization techniques



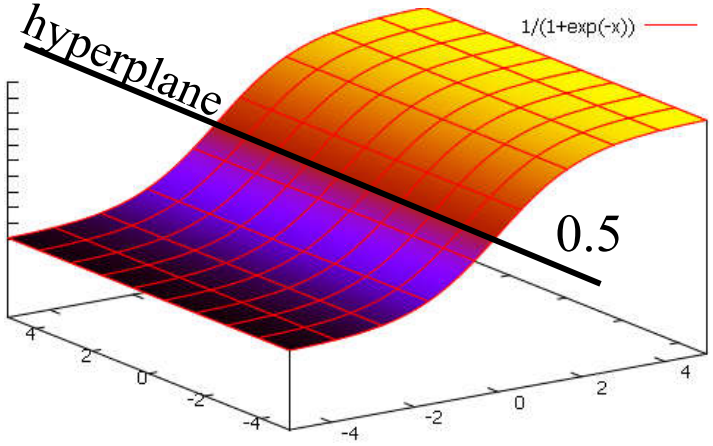
Network size

- With sufficiently many hidden nodes and layers, the MLP can approximate any function to any degree of accuracy (Hornik et al, 1989)
- In theory, we only need one hidden layer!
  - though that layer may have to be very large
- In practice, you may need two hidden layers
  - Reduces the required number of nodes in each
  - Solutions with more than two hidden layers were very rare before the Deep Learning era
  - The power of Deep Learning is not in that it can represent more complex functions.
    - It provably can't! (More on this later)
- To get a feeling for how many nodes we need we should discuss what a hidden layer really does

UPPSALA UNIVERSITET

Hidden nodes in a classifier

- MLPs consist of sigmoidal weighted summation units
- The discriminant formed by a single unit is a hyperplane

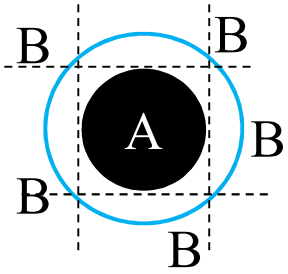


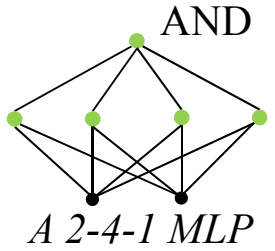
- It's the weighted sum which makes it a hyperplane, not the activation function!
  - The discriminant is linear even if the node is non-linear
- The sigmoid makes the border 'grey' (0.5), but does not (alone) change the shape

UPPSALA UNIVERSITET

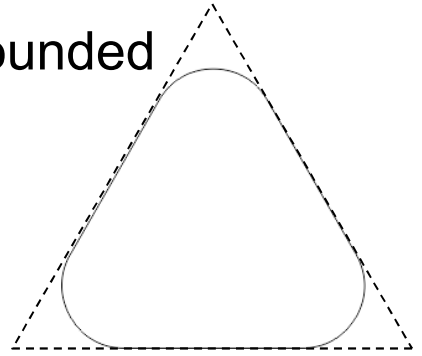
Output nodes in a classifier

- The output nodes combine the hyperplanes into regions
  - With binary hidden nodes, the result is a polygon (ish)
  - How many binary hidden nodes to separate A from B?





  - What if the hidden nodes have sigmoids?
    - With sigmoidal hidden nodes, the corners get rounded
    - Three sigmoids can be combined into a circle! (extreme case of rounded triangle)
    - So, only three hidden nodes required above

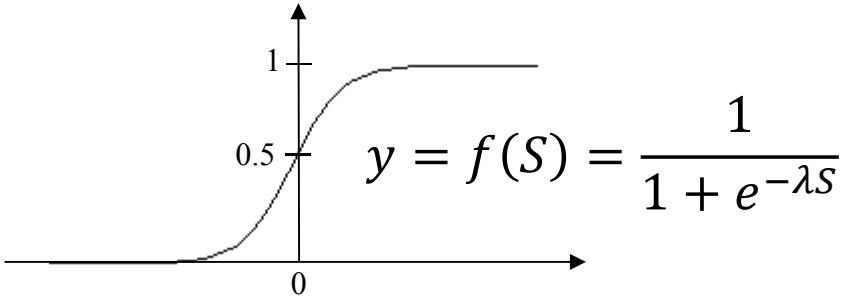


UPPSALA UNIVERSITET

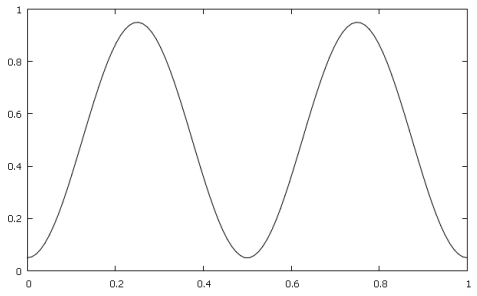
Function approximation

Regression

- Conventional MLPs consist of sigmoidal weighted summation units
  - A sigmoid is a monotonic function



- The output layer combines them
  - So each hidden node should correspond to a monotonic region of the target function
- No hyperplanes here!
  - Regression is not about border decisions



UPPSALA UNIVERSITET

Network size

- The ability to represent a function is not a guarantee that the algorithm will find it!
  - The required number of hidden nodes is greater in practice than in theory
    - More nodes → solutions easier to find (but more overfitting)
- Tip: Output nodes do not have to be non-linear!
  - The network is still a universal approximator
  - In function approximation they should be linear!
    - Or at least not bounded, to be able to output any value
  - In classification they should be sigmoidal!
    - Helps interpretation of output values as probabilities

UPPSALA UNIVERSITET

K-fold cross validation

How good is your model?

- Having decided on a model (network type/size, algorithm, parameters, etc.), how to evaluate it?
- Shuffle the data and split into K subsets *
- For all subsets, i:
  - Train on all data except subset i
  - Test on subset i

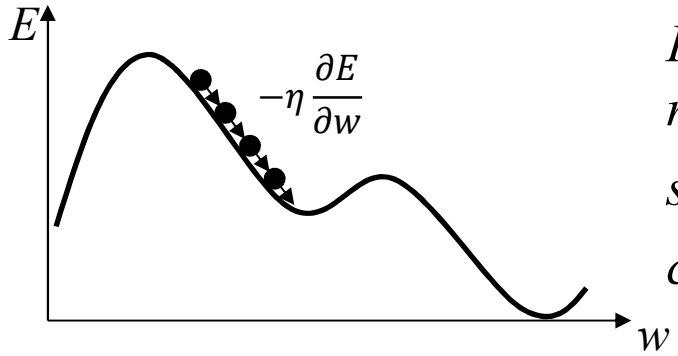
- Result: The average error of the K tests
- Note that we use all the data for both training and testing, but still get a fair generalization measure
- This is model checking, not model building
  - See https://machinelearningmastery.com/train-final-machine-learning-model/ for a good explanation

(*) The 'leave-one-out' principle is a special case, where *K* = number of examples

UPPSALA UNIVERSITET

Optimizing for speed

- Backprop implements *gradient descent*



From last lecture. The 'ball' moves in the direction of steepest slope $(\frac{\partial E}{\partial w})$ *with a step length controlled by* η.

- Idea: Increase training speed by on-line adjustments of the step length. Examples:
  - Backprop with momentum (previous lecture)
  - Start with large η and reduce over time
  - Consider gradient history. Has it changed? If so, decrease η, else increase it
    - This can be done locally for each weight

UPPSALA UNIVERSITET

RPROP: Resilient Backpropagation

Riedmiller, 1992 (MSc thesis)

- Requires epoch learning
- Adaptive step length, local for each weight
- Idea: Let $E' = \frac{\partial E}{\partial w}$ decide direction only
  - i.e. we only consider it's sign
- Step length instead decided by a new parameter, Δ_{ji} *(replaces* η*)*

$$\Delta w_{ji} = -\Delta_{ji} \text{sign}\left(\frac{\partial E}{\partial w_{ji}}\right)$$

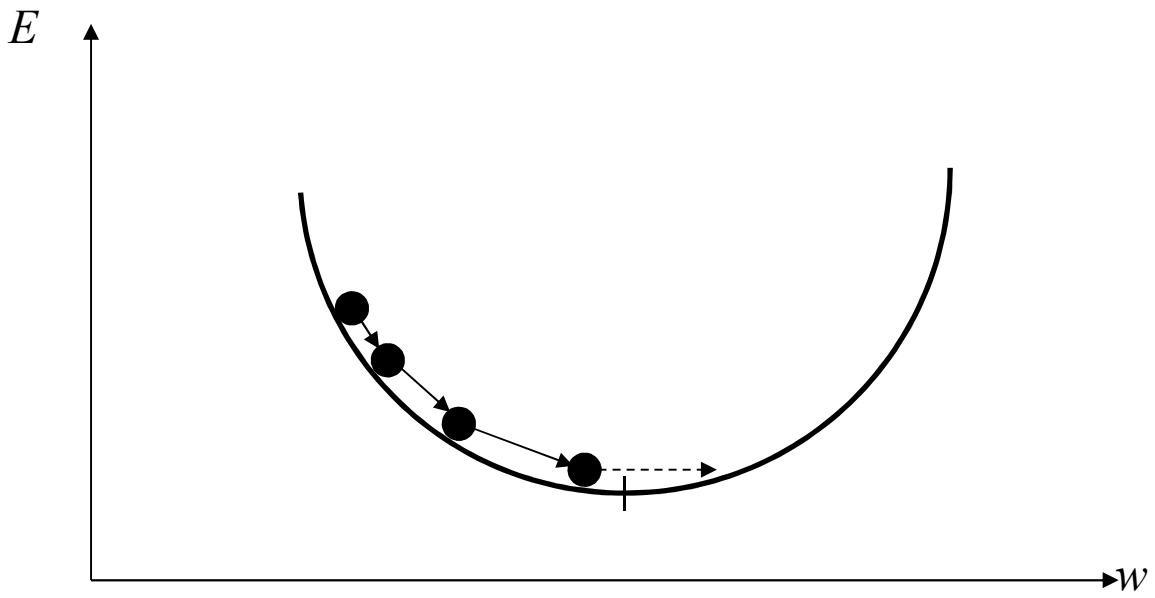
  - Δ_{ji} is updated (within a specified interval), so that:
    - If E' keeps its sign, Δ is increased by factor $\eta^+ > 1$
    - If E' changes sign, Δ is decreased by factor $\eta^- < 1$
      - and the weight change is discarded

UPPSALA UNIVERSITET

RPROP: Resilient Backpropagation

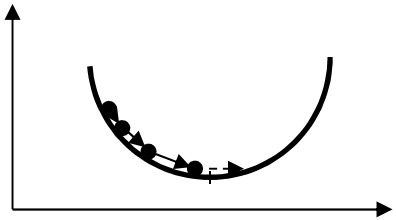
Riedmiller, 1992

- Effect: Accelerates down slopes. Breaks if we overshoot a minimum (or would have ...)



See the short-paper (2 pages) by Riedmiller for details

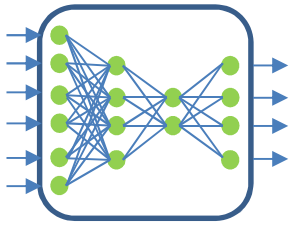
UPPSALA UNIVERSITET



Backprop v.s. RPROP

- RPROP can be very fast, but it's not because it has fewer things to compute than Backprop!
  - RPROP does not care how steep the slope is, only its direction (sign)
  - But it still has to compute and backpropagate the same δ-values as Backprop does, to extract that sign!
- You will compare them, both for classification and for function approximation, in Lab 1
  - Play around with both Backprop and RPROP!
  - Take your time! (Deadline is a week after the lab)
  - You are payed (in credits) for 20h/lab, so don't expect to finish it in the 4 hours scheduled for it

UPPSALA UNIVERSITET



Challenges

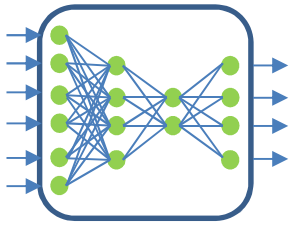
From last lecture

- The loss function could be any differentiable function, but is very often assumed to be the squared error
  - Which part of the update equations would change, if we replaced the loss function?

$$\Delta w_{ji} = \eta \delta_j x_i$$

$$\delta_j = \begin{cases} \lambda y_j (1 - y_j)(d_j - y_j), & \text{if } y_j \text{ is an output node} \\ \lambda y_j (1 - y_j) \sum_k w_{kj} \delta_k, & \text{if } y_j \text{ is a hidden node} \end{cases}$$

UPPSALA UNIVERSITET



Challenges

From last lecture

- If we use pattern learning, why don't we update the weight directly, in step 4?
  - 4. Compute weight changes iteratively, layer by layer, from the outputs to the first hidden layer:

$$\Delta w_{ji} = \eta \delta_j x_i$$

$$\delta_j = \begin{cases} \lambda y_j (1 - y_j)(d_j - y_j), & \text{if } y_j \text{ is an output node} \\ \lambda y_j (1 - y_j) \sum_k w_{kj} \delta_k, & \text{if } y_j \text{ is a hidden node} \end{cases}$$

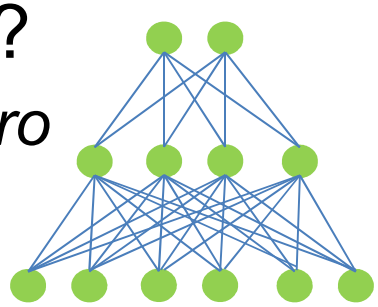
  - *Because the hidden layer δ-values depend on them! If we update the hidden-to-output weights too early, the output layer δ-values will be backpropagated through the wrong set of weights*

UPPSALA UNIVERSITET

Challenges

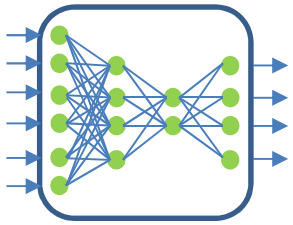
From last lecture

- What would happen if we initialized all weights to zero, instead of small random values?



  - *Hint: The problem is not that they are zero*
  - *The problem is that they are all equal*
  - *Let's assume just one hidden layer*
  - *If all weights are equal, all hidden nodes contribute to the output error by the same amount. They will therefore all be updated the same way. They will remain equal*
  - *There is no point having a committee, if all its members always agree on everything! (True also IRL)*
  - *The network will in effect only have one hidden node (which is like having no hidden layer at all)*

UPPSALA UNIVERSITET



Challenges

From last lecture

- In a deep network (many layers) the chain of partial derivatives gets very long. A problem?
  - *We multiply very many partial derivatives*
  - *If they tend to be <1 the product may become very small*
    - *'Vanishing gradients'*
    - *Very small weight changes – the network does not learn*
  - *If they tend to be >1 the product may become very large*
    - *'Exploding gradients'*
    - *Very large weight changes – the network will oscillate*

UPPSALA UNIVERSITET

Super Challenge

- As shown in the previous lecture and this one:

  $ab, \overline{ab}, a+b, \overline{a+b}$, are all linearly separable,

  $(a+b)\overline{c}$ (the three input solution to XOR) is too,

  but $a \oplus b = a\overline{b} + \overline{a}b = (a+b)\overline{ab}$ is not!
- Where is the limit? Under which condition(s) is a Boolean expression linearly separable, when viewed as a classification problem?

- *Either a conjunction where at most one term is a disjunction, or a disjunction where at most one term is a conjunction*
- *In geometrical terms: The region formed by an output node, combining the hyperplanes formed by hidden nodes, does not have to be convex, but it may not contain more than one concavity*