Informatics 1 Cognitive Science

Lecture 6: Multilayer Perceptrons and Backpropagation

Frank Keller

26 January 2024

School of Informatics
University of Edinburgh
keller@inf.ed.ac.uk

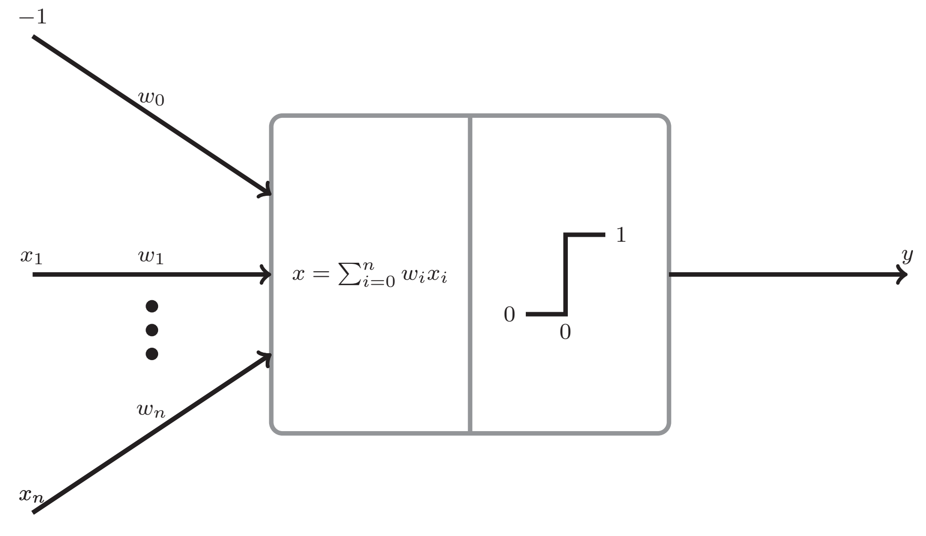
Slide credits: Frank Mollica, Chris Lucas, Mirella Lapata

Overview

Recap: Perceptrons

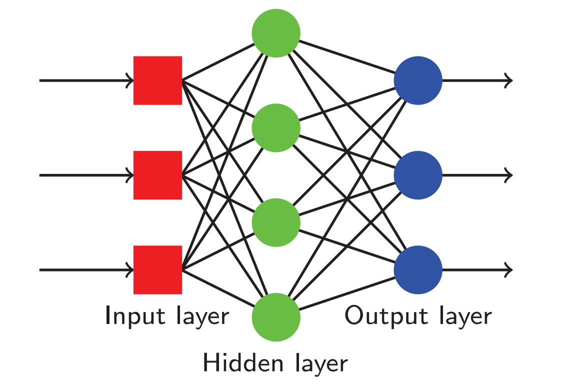
- Neural networks (aka deep learning) is a computer modeling approach inspired by networks of biological neurons.
- A neural net consits of units and connections.
- The perceptron is the simplest neural network model; it is a linear classifier.
- A learning algorithm for perceptrons exists.
- **Key limitation:** only works for linearly separable data.

Recap: Perceptrons



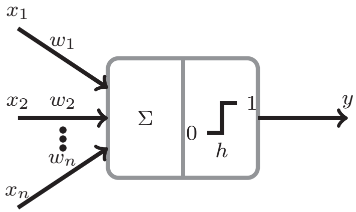
Multilayer Perceptrons

Multilayer Perceptrons (MLPs)

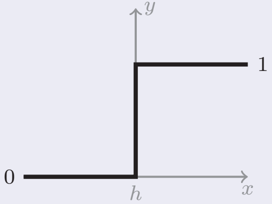


- MLPs are feed-forward neural networks, organized in layers.
- One input layer, one or more hidden layers, one output layer.
- Each node in a layer connected to all other nodes in next layer.
- Each connection has a weight (can be zero).

Activation Functions

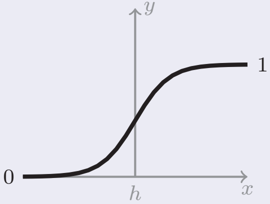


Step function



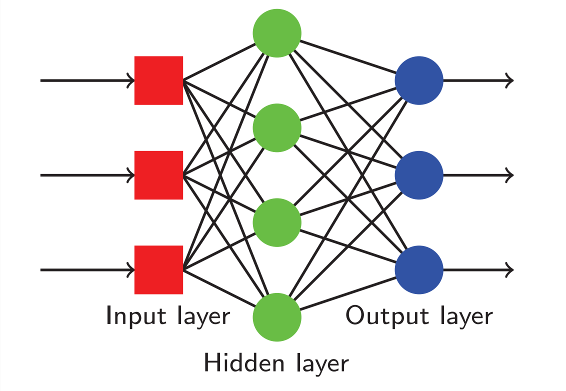
Outputs 0 or 1.

Sigmoid function



Outputs a real value between 0 and 1.

Learning with MLPs



- As with perceptrons, finding the right weights is very hard!
- Solution: learning alorithm.
- Learning: adjusting the weights based on training examples.

Supervised Learning

General Idea

1. Send the MLP an input pattern x from the **training set.**
2. Get the output y from the MLP.
3. Compare y with the "right answer", or target t, to get the **error quantity.**
4. Use the error quantity to modify the weights, so next time y will be closer to t.
5. Repeat with another x from the training set.

When updating weights after seeing x, the network doesn't just change the way it deals with x, but other inputs too . . .

Even inputs it has not seen yet!

Generalization is the ability to deal accurately with unseen inputs.

Learning and Error Minimization

Recall: Perceptron Learning Rule

Minimize the difference between the output o and the target t:

$$w_i \leftarrow w_i + \eta(t - o)x_i$$

Error Function: Mean Squared Error (MSE)

An **error function** represents such a difference over a set of inputs:

$$E(\vec{w}) = \frac{1}{2N} \sum_{p=1}^{N} (t^p - o^p)^2$$

- N is the number of patterns
- t^p is the target output for pattern p
- o^p is the output obtained for pattern p
- Actually MSE/2; the 2 makes little difference, but makes life easier later on!

Learning and Error Minimization

Time for a short quiz on Wooclap!

https://app.wooclap.com/TDNYYK

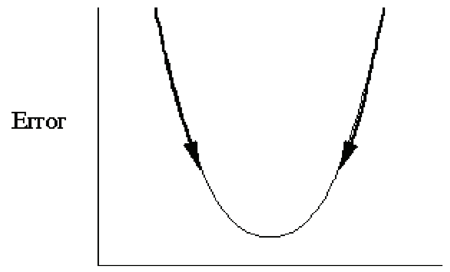
Gradient Descent

Gradient Descent

- We would like a learning rule that tells us how to update weights, like this:

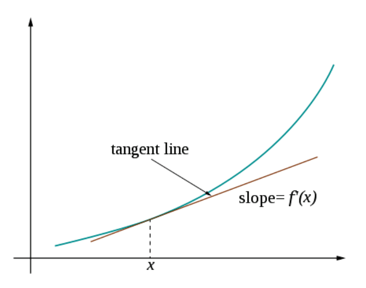
$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

- But what should Δw_{ij} be?
- Idea: Pick Δw_{ij} so that it minimizes the error function E.
- Gradient descent is a technique for minimizing a function.

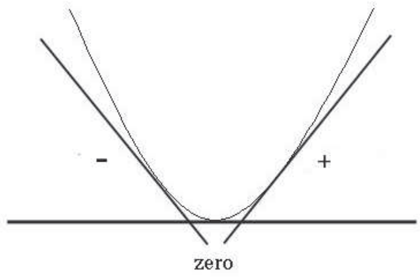


Gradient and Derivatives: The Idea

- The derivative is a measure of the rate of change of a function, as its input changes.
- For function $y = f(x)$, the derivative $\frac{dy}{dx}$ indicates how much y changes in response to changes in x.
- If x and y are real numbers, and if the graph of y is plotted against x, the derivative measures the slope or gradient of the line at each point, i.e., it describes the steepness or incline.

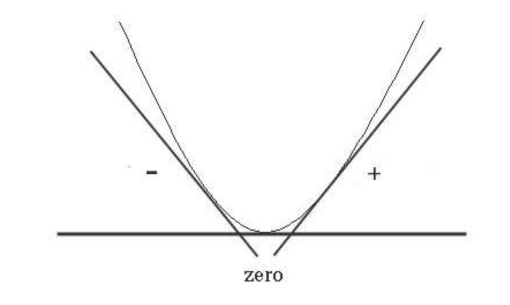


Gradient and Derivatives: The Idea



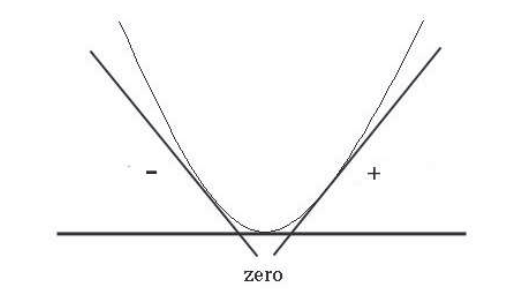
- $\frac{dy}{dx} > 0$ implies that y increases as x increases. If we want to find the minimum y, we should reduce x.

Gradient and Derivatives: The Idea



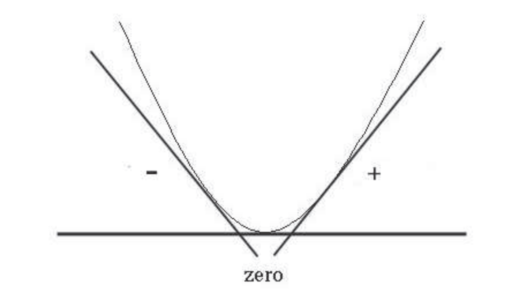
- $\frac{dy}{dx} > 0$ implies that y increases as x increases. If we want to find the minimum y, we should reduce x.
- $\frac{dy}{dx} < 0$ implies that y decreases as x increases. If we want to find the minimum y, we should increase x.

Gradient and Derivatives: The Idea



- $\frac{dy}{dx} > 0$ implies that y increases as x increases. If we want to find the minimum y, we should reduce x.
- $\frac{dy}{dx} < 0$ implies that y decreases as x increases. If we want to find the minimum y, we should increase x.
- $\frac{dy}{dx} = 0$ implies that we are at a minimum or maximum or a plateau.

Gradient and Derivatives: The Idea



- $\frac{dy}{dx} > 0$ implies that y increases as x increases. If we want to find the minimum y, we should reduce x.
- $\frac{dy}{dx} < 0$ implies that y decreases as x increases. If we want to find the minimum y, we should increase x.
- $\frac{dy}{dx} = 0$ implies that we are at a minimum or maximum or a plateau.

To get closer to the minimum: $$x_{new} = x_{old} - \eta \frac{dy}{dx}$$

Gradient and Derivatives: The Idea

- So, we know how to use derivatives to adjust one input value.
- But we have several weights to adjust!
- We need to use partial derivatives.
- A partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant.

Example

If $y = f(x_1, x_2)$, then we can have $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$.
In our learning rule case, if we can work out the partial derivatives, we can use this rule to update the weights:

$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

where $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$.

Summary So Far

- We learned what a multilayer perceptron is.
- We know a learning rule for updating weights in order to minimise the error:

$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

$$\text{where } \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

- Δw_{ij} tells us in which direction and how much we should change each weight to roll down the slope (descend the gradient) of the error function E.

Summary So Far

- We learned what a multilayer perceptron is.
- We know a learning rule for updating weights in order to minimise the error:

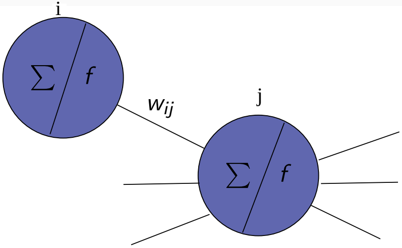
$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

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

- Δw_{ij} tells us in which direction and how much we should change each weight to roll down the slope (descend the gradient) of the error function E.
- So, how do we calculate $\frac{\partial E}{\partial w_{ij}}$?

The Update Rule

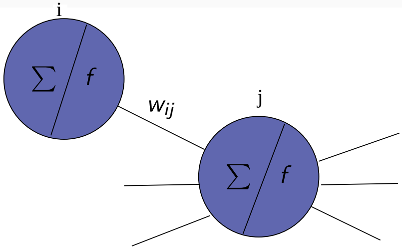
Using Gradient Descent to Minimize the Error



The mean squared error function E, which we want to minimize:

$$E(\vec{w}) = \frac{1}{2N} \sum_{p=1}^{N} (t^p - o^p)^2$$

Using Gradient Descent to Minimize the Error



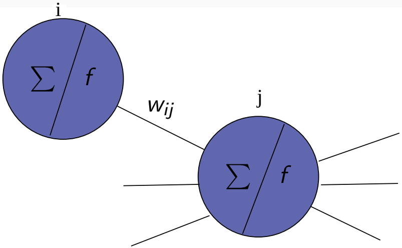
If we use a sigmoid activation function f, then the output of neuron i for pattern p is:

$$o_i^p = f(u_i) = \frac{1}{1 + e^{-au_i}}$$

where a is a pre-defined constant and u_i is the result of the input function in neuron i:

$$u_i = \sum_j w_{ij} x_{ij}$$

Using Gradient Descent to Minimize the Error

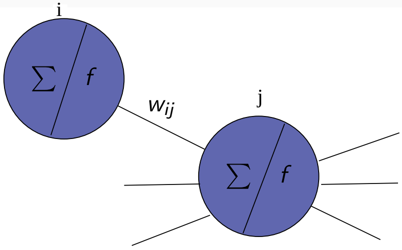


For the pth pattern and the ith neuron, we use gradient descent on the error function:

$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} = \eta (t_i^p - o_i^p) f'(u_i) x_{ij}$$

where $f'(u_i) = \frac{df}{du_i}$ is the derivative of f with respect to u_i.
If f is the sigmoid function, $f'(u_i) = af(u_i)(1 - f(u_i))$.

Using Gradient Descent to Minimize the Error

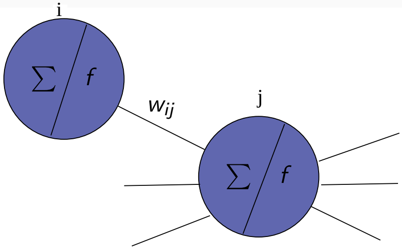


We can update weights after processing each pattern, using rule:

$$\Delta w_{ij} = \eta \left(t_i^p - o_i^p\right) f'(u_i)\, x_{ij}$$

$$\Delta w_{ij} = \eta\, \delta_i^p\, x_{ij}$$

Using Gradient Descent to Minimize the Error



We can update weights after processing each pattern, using rule:

$$\Delta w_{ij} = \eta \left(t_i^p - o_i^p \right) f'(u_i) \, x_{ij}$$

$$\Delta w_{ij} = \eta \, \delta_i^p \, x_{ij}$$

This is known as the generalized delta rule.

Updating Output vs Hidden Neurons

We can update output neurons using the generalized delta rule:

$$\Delta w_{ij} = \eta \, \delta_i^p \, x_{ij}$$

$$\delta_i^p = (t_i^p - o_i^p) f'(u_i)$$

This δ_i^p is only good for the output neurons, since it relies on target outputs. But we don't have target output for the hidden nodes! What can we use instead?

$$\delta_i^p = \sum_k w_{ki} \, \delta_k \, f'(u_i)$$

This rule propagates error back from output nodes to hidden nodes. If effect, it blames hidden nodes according to how much influence they had. So, now we have rules for updating both output and hidden neurons!

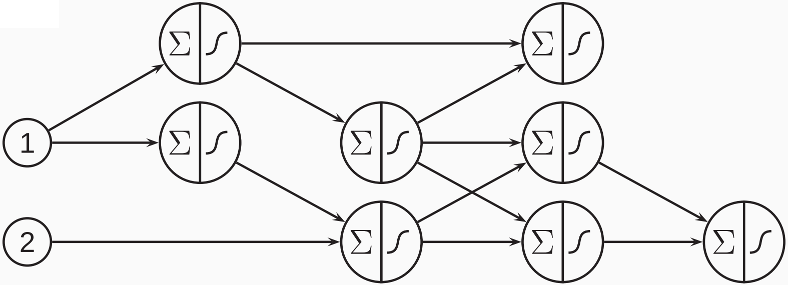
Updating Output vs Hidden Neurons

Time for a short quiz on Wooclap!

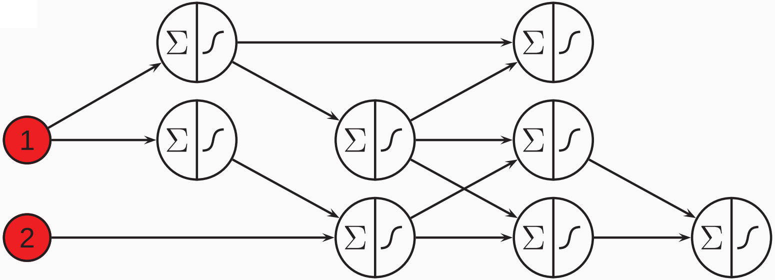
https://app.wooclap.com/TDNYYK

Backpropagation

Backpropagation

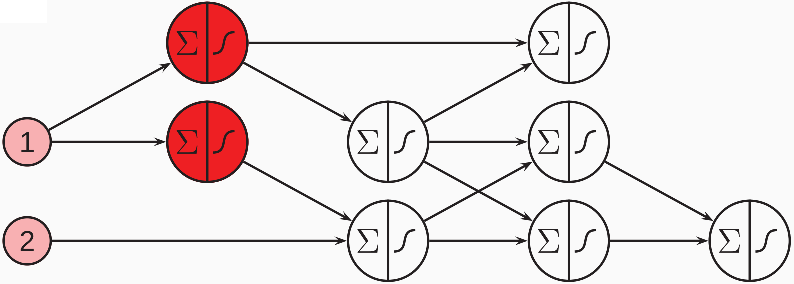


Backpropagation



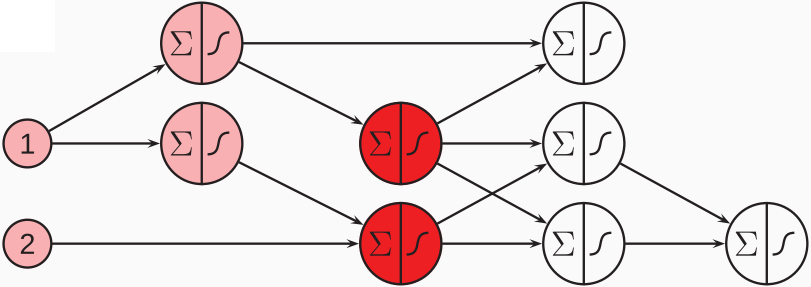
1. Present the pattern at the input layer.

Backpropagation



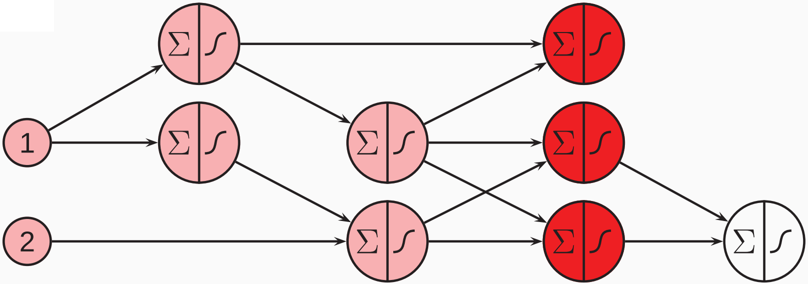
1. Present the pattern at the input layer.
2. Propagate forward activations.

Backpropagation



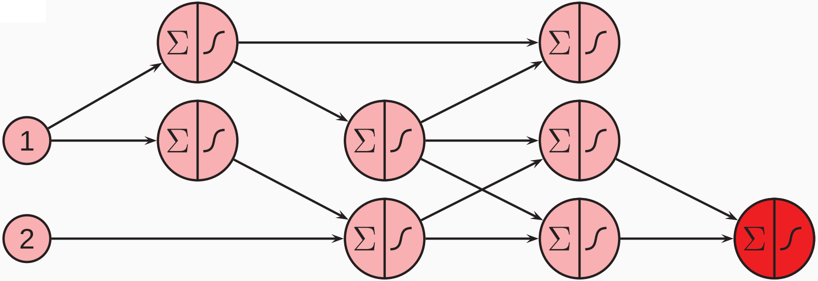
1. Present the pattern at the input layer.
2. Propagate forward activations.

Backpropagation



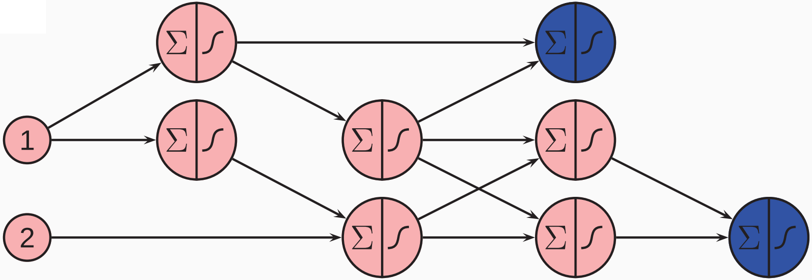
1. Present the pattern at the input layer.
2. Propagate forward activations.

Backpropagation



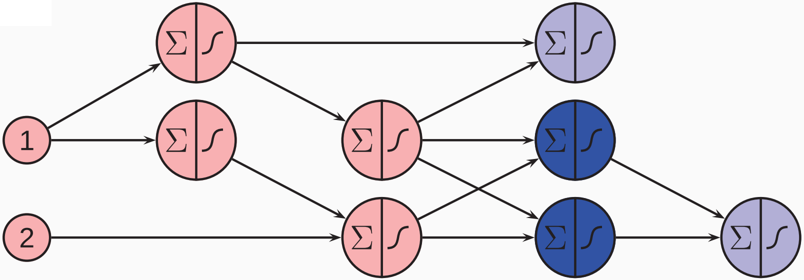
1. Present the pattern at the input layer.
2. Propagate forward activations.

Backpropagation



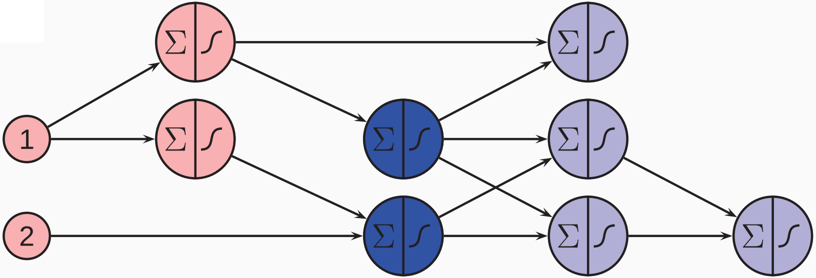
1. Present the pattern at the input layer.
2. Propagate forward activations.
3. Calculate error for the output neurons.

Backpropagation



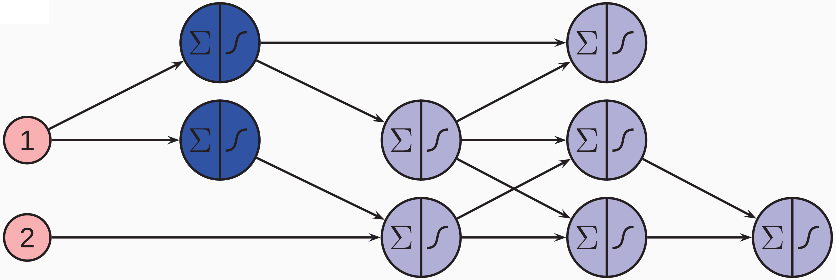
1. Present the pattern at the input layer.
2. Propagate forward activations.
3. Calculate error for the output neurons.
4. Propagate backward error.

Backpropagation



1. Present the pattern at the input layer.
2. Propagate forward activations.
3. Calculate error for the output neurons.
4. Propagate backward error.

Backpropagation



1. Present the pattern at the input layer.
2. Propagate forward activations.
3. Calculate error for the output neurons.
4. Propagate backward error.

Online Backpropagation

Initialize all weights to small random values.
repeat
  for each training example **do**
    Forward propagate the input features of the example to determine the MLP's outputs.
    Back propagate error to generate Δw_{ij} for all weights w_{ij}.
    Update the weights using Δw_{ij}.
  end for
until stopping criteria reached.

Time for a short quiz on Wooclap!

https://app.wooclap.com/TDNYYK

Summary

- We learned what a multilayer perceptron is.
- We have some intuition about using gradient descent on an error function.
- We know a learning rule for updating weights in order to minimize the error: $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$
- If we use the squared error, we get the generalized delta rule: $\Delta w_{ij} = \eta \delta_i^p x_{ij}$.
- We know how to calculate δ_i^p for output and hidden layers.
- We can use this rule to learn an MLP's weights using the backpropagation algorithm.

Next lecture: a neural network model of the past tense.