Deep Neural Networks - Lecture 3

Marcin Mucha

March 13, 2018

The perceptron

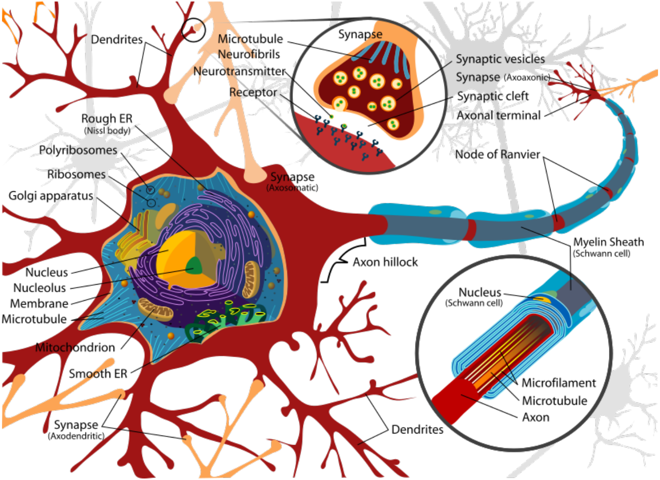
Neural Networks and Backpropagation

Trouble with neural networks

Where it all begins

- Many settings where humans outperform best algorithms.
- Idea: Emulate the human brain.
- 10^{11} neurons, 10^{14} connections in the brain. Trouble?
- Compare: a 10 TB drive can store 10^{14} bytes.

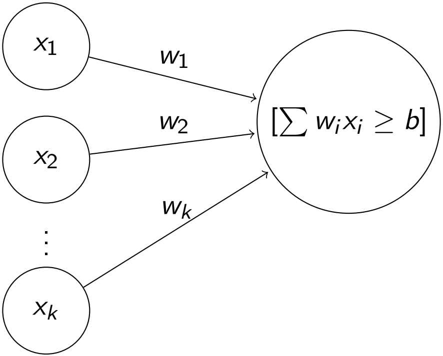
Where it all begins



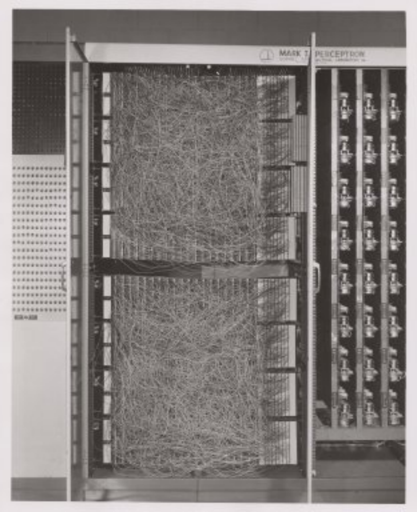
The perceptron

The perceptron is a function that, given inputs $x_1, \ldots, x_k$ outputs $[\sum_i w_i x_i + b \geq 0]$ (or a specific GD-like algo for fitting these).

Think: x_i are factors contributing to a decision, they are weighted and the perceptron makes a decision automatically!



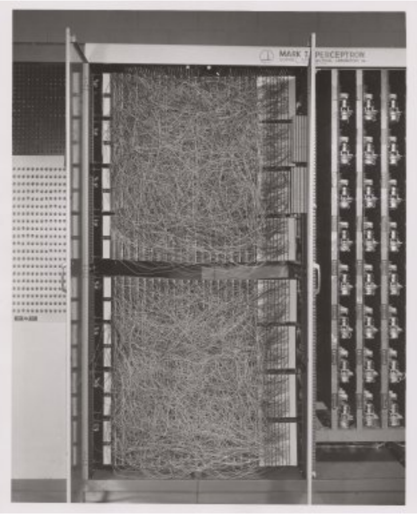
The perceptron



Mark I Perceptron (1957)

- Funded by US Navy.
- Photo recognition.
- 20x20 photo-cells.
- Weights in potentiometers.
- Trained with the perceptron algorithm - weights updated by electric motors.

The perceptron



Mark I Perceptron (1957)

- Funded by US Navy.
- Photo recognition.
- 20x20 photo-cells.
- Weights in potentiometers.
- Trained with the perceptron algorithm - weights updated by electric motors.

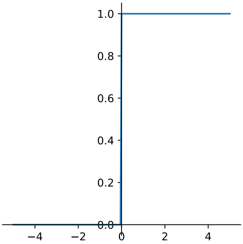
...the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence...

Training the perceptron

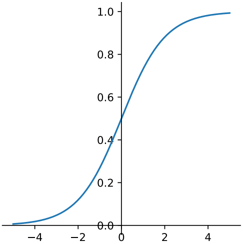
Impossible to train directly using gradient methods since gradient is 0 where it exists. Instead...

Training the perceptron

Impossible to train directly using gradient methods since gradient is 0 where it exists. Instead...



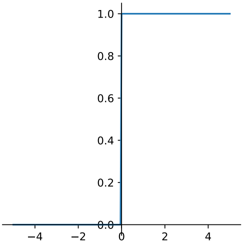
$\rightarrow$



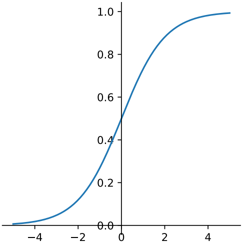
Train using a smooth approximation (not exactly what the perceptron algorithm was doing!).

Training the perceptron

Impossible to train directly using gradient methods since gradient is 0 where it exists. Instead...



$\rightarrow$



Train using a smooth approximation (not exactly what the perceptron algorithm was doing!).
Many smooth step functions possible, e.g. the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Rings a bell?

The power of perceptrons

Perceptrons can only separate half-spaces.

The power of perceptrons

Perceptrons can only separate half-spaces.

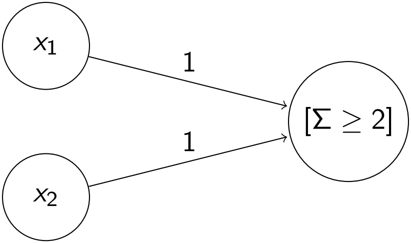
Exercise: Show that for binary inputs, it can compute AND.

The power of perceptrons

Perceptrons can only separate half-spaces.

Exercise: Show that for binary inputs, it can compute AND.

Solution:

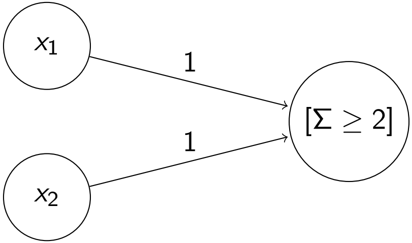


The power of perceptrons

Perceptrons can only separate half-spaces.

Exercise: Show that for binary inputs, it can compute AND.

Solution:



Exercise: Can a perceptron compute XOR?

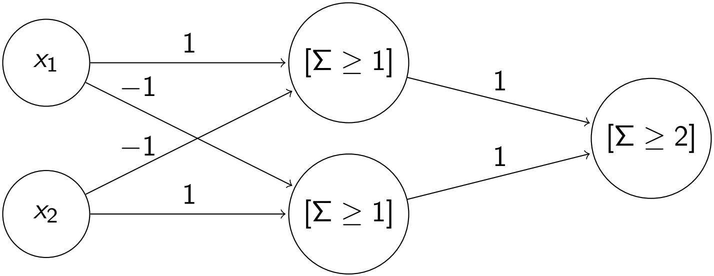
Network of perceptrons

Exercise: Show that a DAG of perceptrons can compute XOR.

Network of perceptrons

Exercise: Show that a DAG of perceptrons can compute XOR.

Solution:

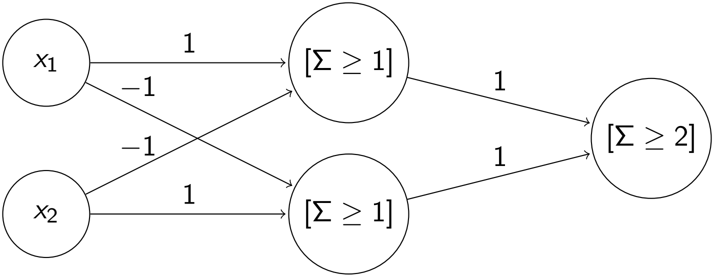


Exercise: Show that a DAG of perceptrons can compute any boolean function.

Network of perceptrons

Exercise: Show that a DAG of perceptrons can compute XOR.

Solution:



Exercise: Show that a DAG of perceptrons can compute any boolean function.

Note that this DAG can be very large (think: SAT).

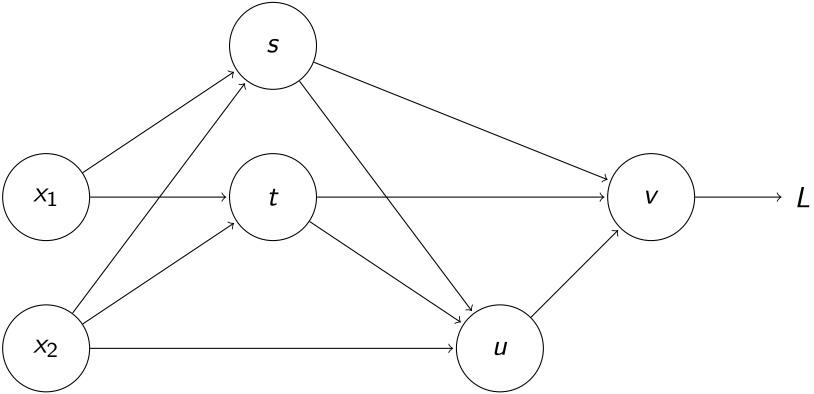
Training a DAG of perceptrons

...but can we train DAGs of perceptrons?

Training a DAG of perceptrons

...but can we train DAGs of perceptrons?

More specifically: Given a DAG of sigmoid approximations to the perceptron, can we compute gradients of the loss function?



Training a DAG of perceptrons

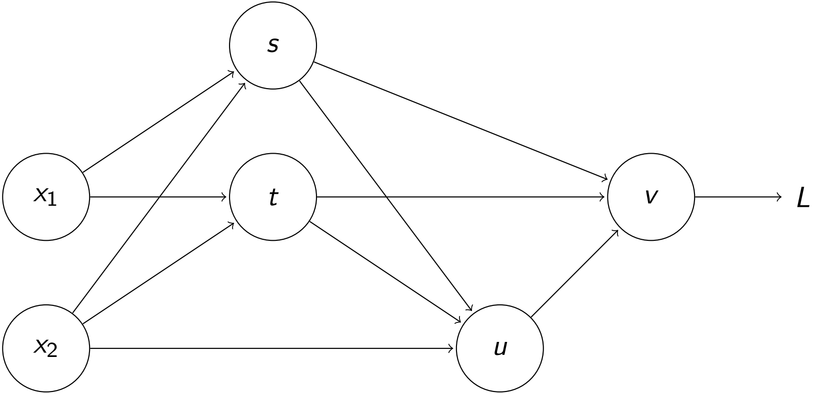
Theorem

We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.

Training a DAG of perceptrons

Theorem

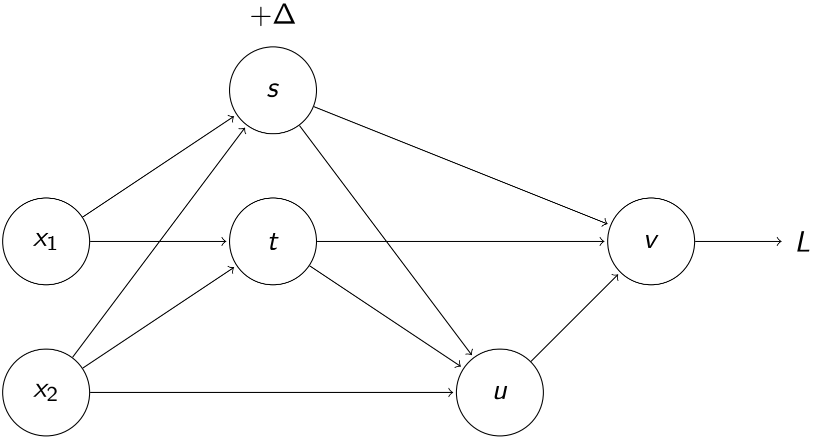
We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.



Training a DAG of perceptrons

Theorem

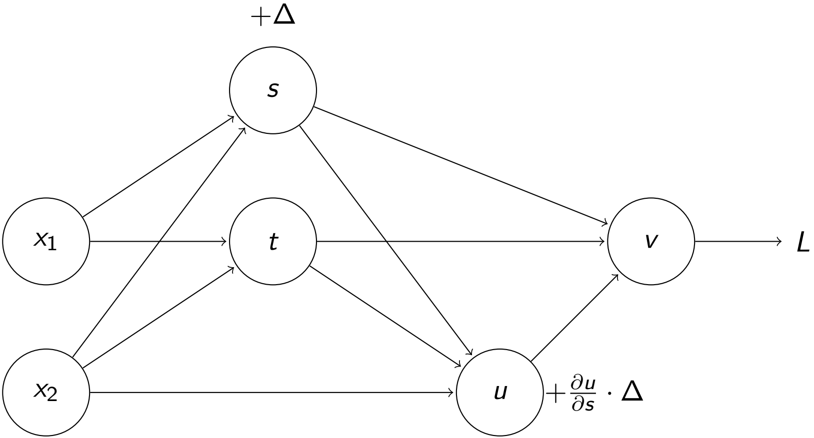
We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.



Training a DAG of perceptrons

Theorem

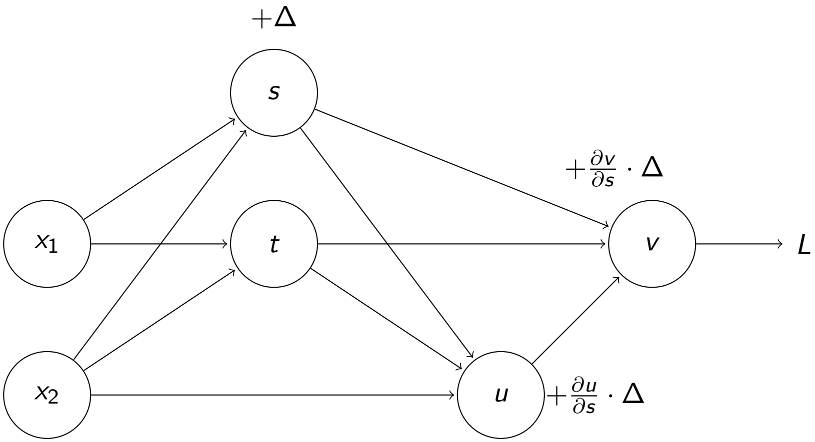
We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.



Training a DAG of perceptrons

Theorem

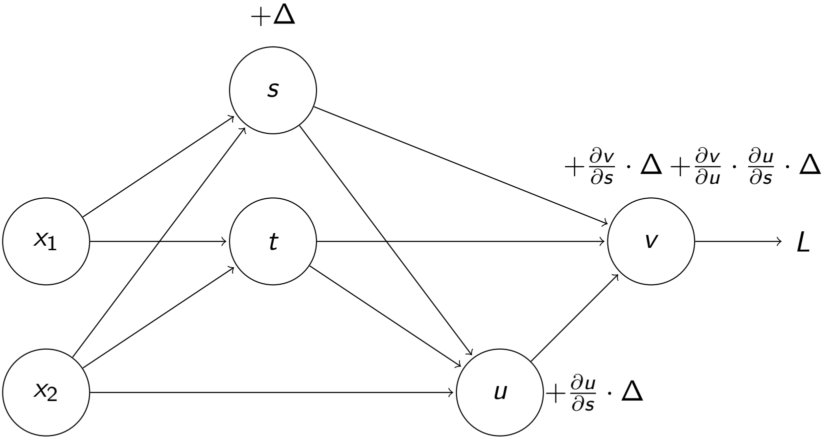
We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.



Training a DAG of perceptrons

Theorem

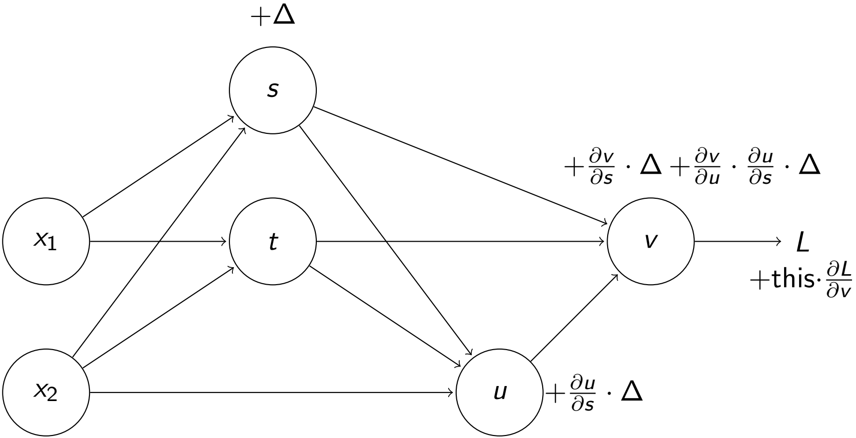
We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.



Training a DAG of perceptrons

Theorem

We can compute the derivatives of the loss function over all node values (and hence also over node weights) in linear time.



Backpropagation

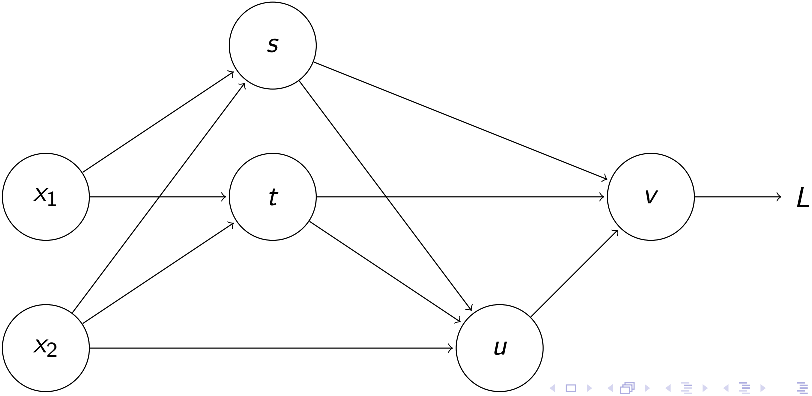
It seems that for a node z, we have

$$\frac{\partial L}{\partial z} = \sum_{z=z_0 \to \ldots \to z_l = L} \prod_{i=1}^{l-1} \frac{\partial z_{i+1}}{\partial z_i}.$$

Backpropagation

It seems that for a node z, we have

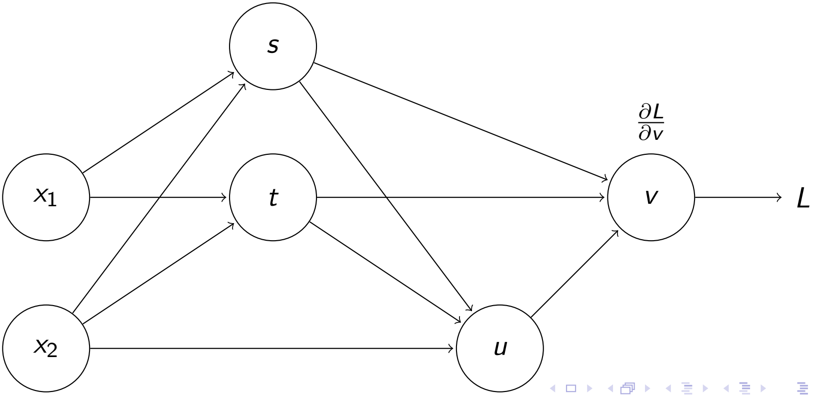
$$\frac{\partial L}{\partial z} = \sum_{z=z_0 \to \ldots \to z_l = L} \prod_{i=1}^{l-1} \frac{\partial z_{i+1}}{\partial z_i}.$$



Backpropagation

It seems that for a node z, we have

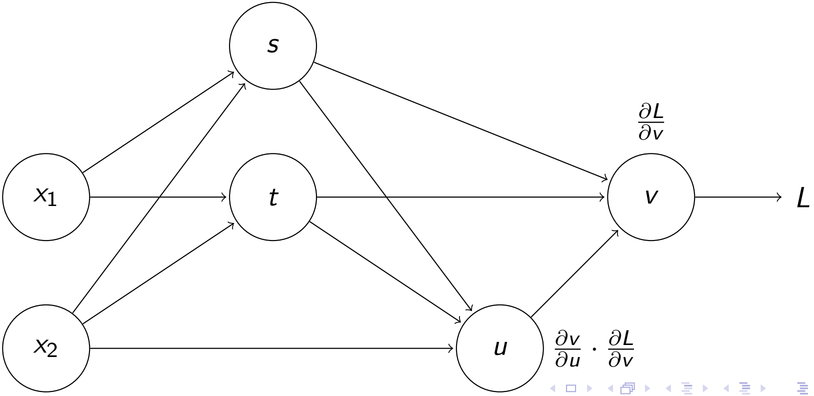
$$\frac{\partial L}{\partial z} = \sum_{z=z_0 \to \dots \to z_l = L} \prod_{i=1}^{l-1} \frac{\partial z_{i+1}}{\partial z_i}.$$



Backpropagation

It seems that for a node z, we have

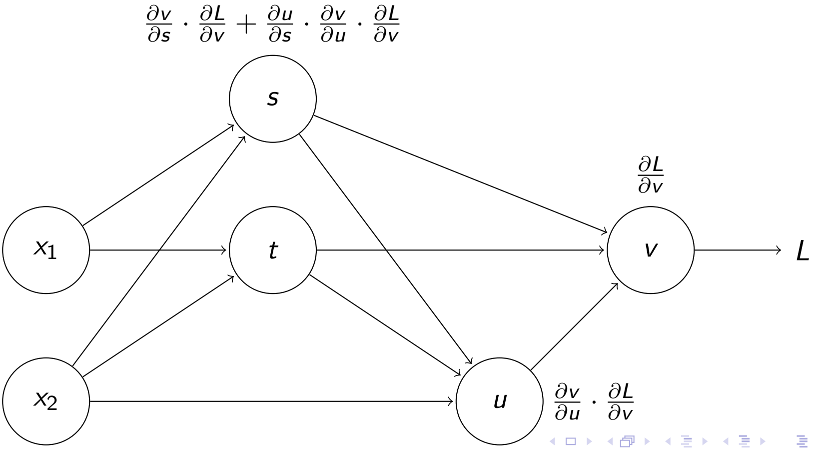
$$\frac{\partial L}{\partial z} = \sum_{z=z_0 \to \ldots \to z_l = L} \prod_{i=1}^{l-1} \frac{\partial z_{i+1}}{\partial z_i}.$$



Backpropagation

It seems that for a node z, we have

$$\frac{\partial L}{\partial z} = \sum_{z=z_0 \to \ldots \to z_l = L} \prod_{i=1}^{l-1} \frac{\partial z_{i+1}}{\partial z_i}.$$



Backpropagation

By doing topological sorting and going from the back we can compute $\frac{\partial L}{\partial z}$ for all computation nodes z.

Backpropagation

By doing topological sorting and going from the back we can compute $\frac{\partial L}{\partial z}$ for all computation nodes z.

But we want derivatives over parameters, not nodes!

Backpropagation

By doing topological sorting and going from the back we can compute $\frac{\partial L}{\partial z}$ for all computation nodes z.

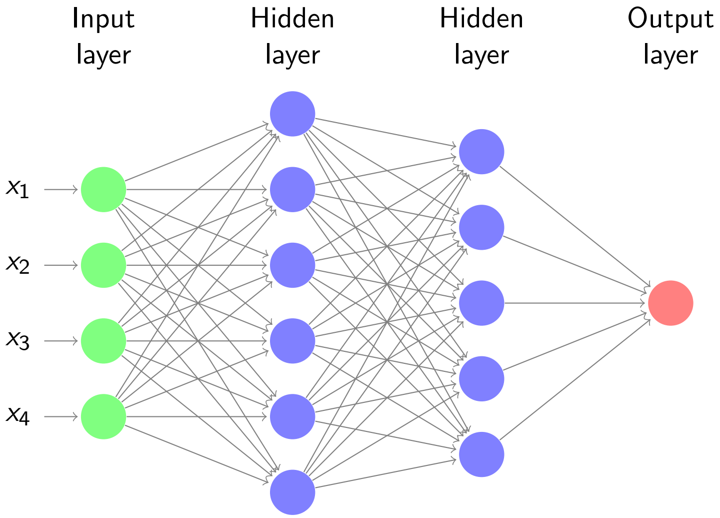
But we want derivatives over parameters, not nodes!

Computing the derivatives of nodes over their parameters is trivial.

Multilayer Perceptron (MLP)

- *Multilayer Perceptron (MLP)*, or a *Feedforward Neural Network* consists of several layers of units (sigmoid or other), each connected with previous and next.
- Number of layers (excluding inputs) is called the *depth*.
- The inner layers are called *hidden*.

Multilayer Perceptron (MLP)

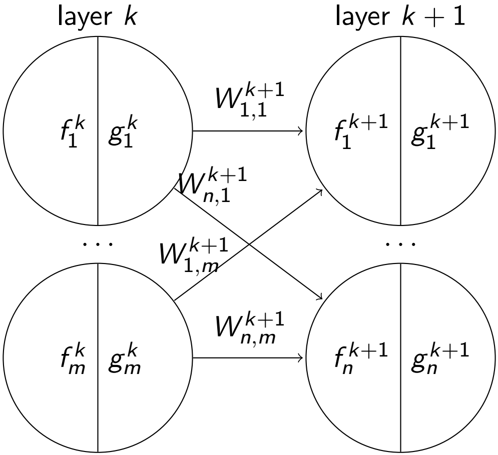


Universal approximation for MLP

Theorem (Universal approximation theorem for MLP)

Any continuous function on $[0, 1]^n$ can be approximated arbitrarily well by an MLP with a single hidden layer.

Matrix algebra - notation



Matrix algebra

Regular structure of MLPs facilitates algebraic shortcuts both in forward and backward pass.

Matrix algebra

Regular structure of MLPs facilitates algebraic shortcuts both in forward and backward pass.

$$f^k = W^k g^{k-1}.$$

Matrix algebra

Regular structure of MLPs facilitates algebraic shortcuts both in forward and backward pass.

$$f^k = W^k g^{k-1}.$$

Here f^k and g^{k-1} could be vectors if single datapoint, or matrices.

Matrix algebra

Regular structure of MLPs facilitates algebraic shortcuts both in forward and backward pass.

$$f^k = W^k g^{k-1}.$$

Here f^k and g^{k-1} could be vectors if single datapoint, or matrices.

$$\frac{\partial L}{\partial g^{k-1}} = W^T \frac{\partial L}{\partial f^k}.$$

Again, $\frac{\partial L}{\partial g^{k-1}}$ and $\frac{\partial L}{\partial f^k}$ can be vectors or matrices.

Matrix algebra

Exercise: Relate $\frac{\partial L}{\partial f^k}$ and $\frac{\partial L}{\partial g^k}$, if $g^k = \sigma(f^k)$ (coordinate-wise).

Matrix algebra

Exercise: Relate $\frac{\partial L}{\partial f^k}$ and $\frac{\partial L}{\partial g^k}$, if $g^k = \sigma(f^k)$ (coordinate-wise).

Solution:

$$\frac{\partial L}{\partial f^k} = \frac{\partial L}{\partial g^k} \circ g^k \circ \left(1 - g^k\right).$$

$A \circ B$ is the Hadamard (element-wise) product.

Matrix algebra

Exercise: Relate $\frac{\partial L}{\partial f^k}$ and $\frac{\partial L}{\partial W^k}$ (this is the matrix of partial derivatives over all elements of W^k).

Matrix algebra

Exercise: Relate $\frac{\partial L}{\partial f^k}$ and $\frac{\partial L}{\partial W^k}$ (this is the matrix of partial derivatives over all elements of W^k).

Solution:

$$\frac{\partial L}{\partial W^k} = \frac{\partial L}{\partial f^k} \left(g^{k-1}\right)^T .$$

Note that for batches summing up happens automagically!

The art of neural networks

In the non-modern setting, setting the parameters of neural networks was more art than science:

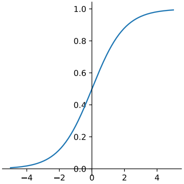
- Choosing the architecture.
- Choosing the units.
- Controlling the learning rate.
- Initializing the weights.
- And other (regularization, mini-batch size, etc.)

These issues were addressed by recent developments.

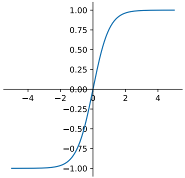
Architecture

- *If your deep neural net is not overfitting you should be using a bigger one!* – G. Hinton
- Bigger network is more expressive.
- Bigger networks take longer to train.
- Deeper networks are much harder to train.
- Modern variants of SGD are much better at training deep nets. And we have stronger hardware. And better ideas (e.g. ResNet)

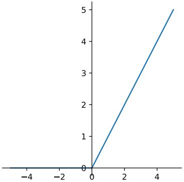
Units



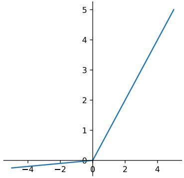
sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$



tanh: $\tanh(x) = \frac{e^{x}+e^{-x}}{e^{x}+e^{-x}}$



ReLU: $\text{ReLU}(x) = \max(0, x)$



leaky ReLU: $\text{LReLU}(x) = \max(ax, x)$

Learning rate

- If the learning rate is too small, convergence is very slow.
- Learning rate that is too high results in oscillations.
- Some modern variants of SGD can automatically adapt.

Initialization

- Very small initial weights slow down learning initially, or even completely vanish.
- Very large initial weights can lead to oscillations and random-like behaviour or even blow-up with no regularization.
- Heuristics, e.g. Glorot initialization: draw a weight from normal distribution with variance $\frac{2}{n_{in}+n_{out}}$.
- Modern techniques like batch normalization make initialization less of an issue.

Recap

- Perceptron and its biological and computational motivation.
- Multilayer perceptron and how to compute its gradients fast.
- Tricky decisions when tuning MLPs.

Recap

- Perceptron and its biological and computational motivation.
- Multilayer perceptron and how to compute its gradients fast.
- Tricky decisions when tuning MLPs.

But do not worry:

- Sometimes MLPs work well even without excessive tuning.
- Next lecture, we will make things much more stable.