LUND UNIVERSITY

FMAN-45: Machine Learning

Lecture 7: Neural Networks

Henning Petzka

Contents

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Metaview

What is a neural network? Very broadly, it is just a function inside a black box from a well-defined input space to another well-defined output space for regression or classification.

The way to model the function inside the black box, use a neural network structure.

The model

Given a labeled dataset $(x_n, y_n)_n$

$$\sigma_{out}\left(W_3 \cdot \sigma\left(W_2 \cdot \sigma(W_1 \cdot x_n)\right)\right) = \hat{y}_n$$

x_n is n-th input vector
W_i are matrices
σ is a differentiable version of the step function $\bar{\sigma}$ of the perceptron. The function is applied componentwise.
The output is a prediction $\hat{y}_n$

This model can be seen as multiple perceptrons building a network.

The objective

Given a labeled dataset $(x_n, y_n)_n$ and the model

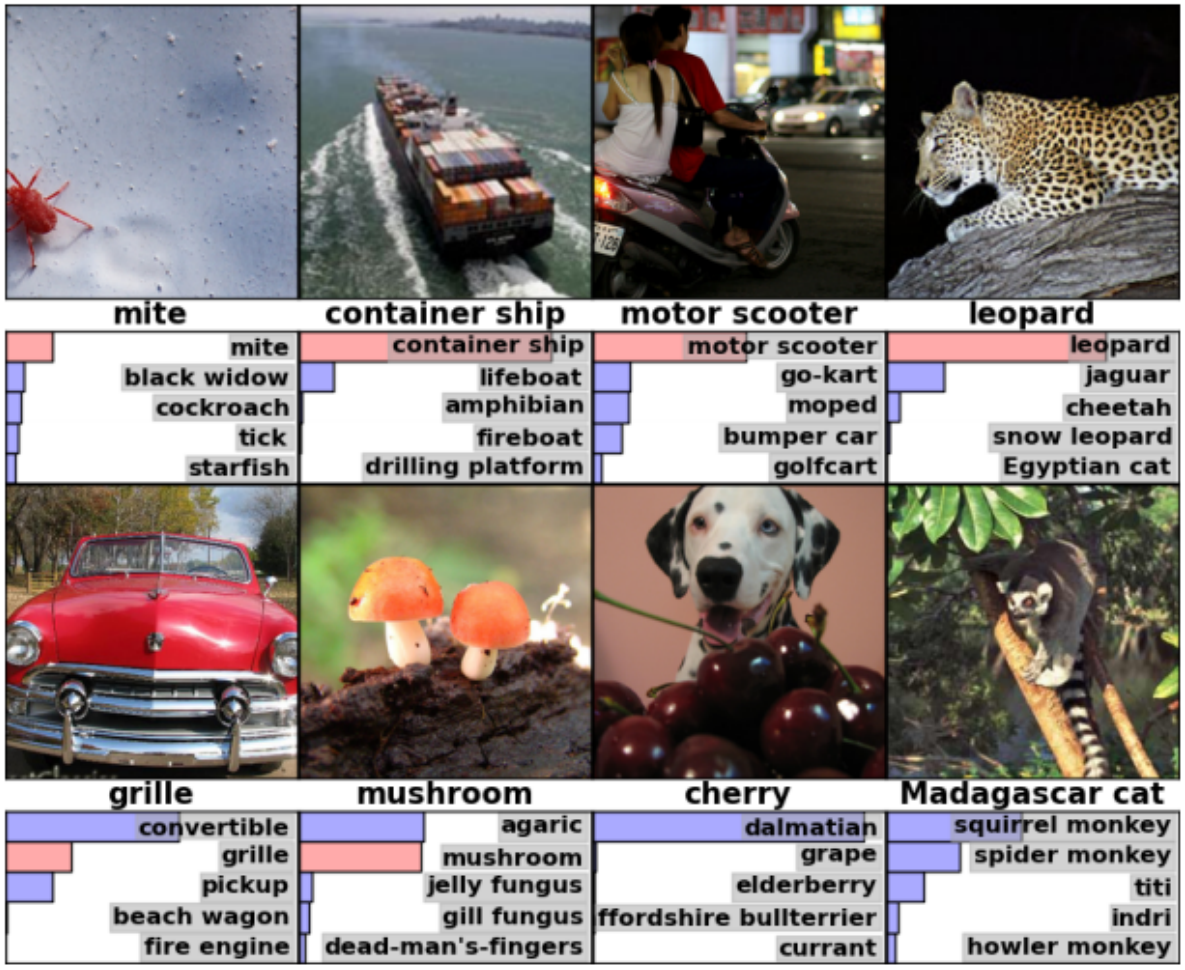
$$\sigma_{out}\left(W_3 \cdot \sigma\left(W_2 \cdot \sigma(W_1 \cdot x_n)\right)\right) = \hat{y}_n,$$

like in any other supervised learning setting, we are trying to minimize the empirical loss given by a loss function, for example

$$L = \frac{1}{2N} \sum_{n=1}^{N} |\hat{y}_n - y_n|^2.$$

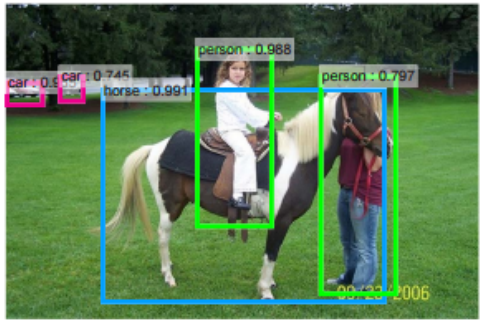
We will see later, how to do this here efficiently.

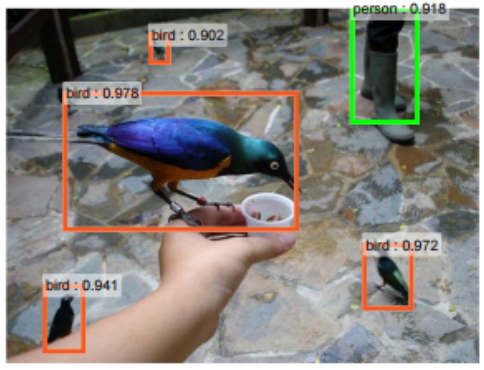
Image Classification

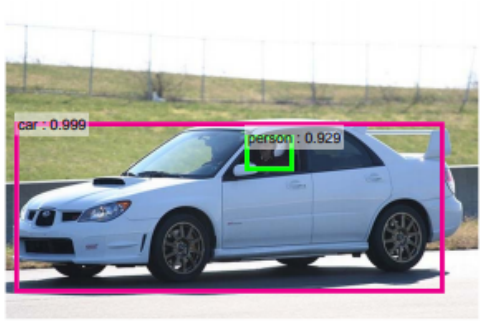


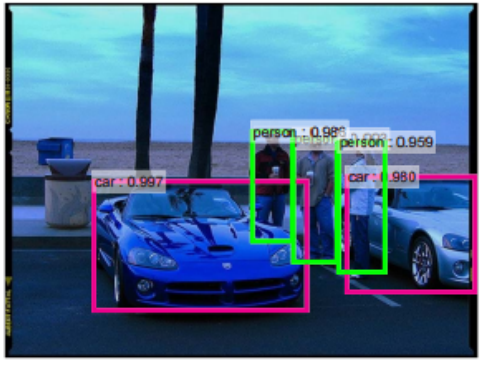
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Object Detection





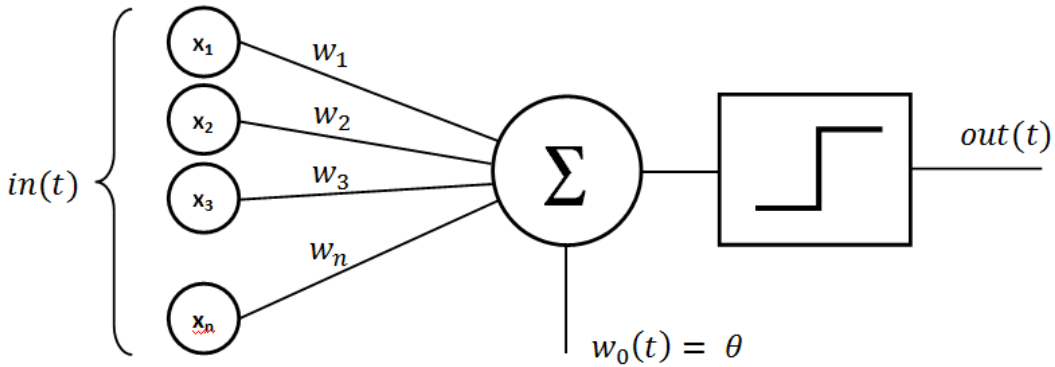


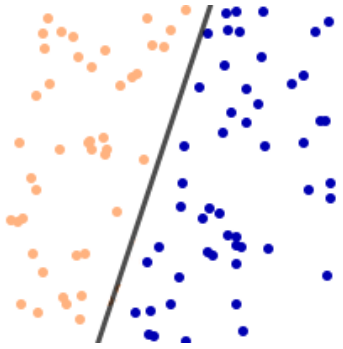


Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.

Perceptron

The perceptron is a model that can linearly separate two classes, where output $t = -1$ means one class and output $t = 1$ the other (binary classification).





$$\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \rightarrow \bar{\sigma} \left(\begin{bmatrix} w_1, w_2, \dots, w_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} + \theta \right)$$

$$\text{where } \sigma(x) = \begin{cases} 1 & , \ x > 0 \\ -1 & , \ x \leq 0 \end{cases}$$

- The perceptron as a the first neural network was invented by Frank Rosenblatt in 1957.

Perceptron

The perceptron is a model that can linearly separate two classes, where output $t = -1$ means one class and output $t = 1$ the other (binary classification).

$$f(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + \theta)$$

$$\text{where } \sigma(x) = \begin{cases} 1 & , \ x > 0 \\ -1 & , \ x \leq 0 \end{cases}$$

- The perceptron's can be motivated by error function minimization.
- A natural error would be number of misclassified patterns.
- However, this does not lead to a simple learning algorithm, because the error is piecewise constant (no information how to improve)

Perceptron

$$f(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + \theta)$$

- Given a training set $\{\mathbf{x}_i\}$, we would like all samples to satisfy $(\mathbf{w} \cdot \mathbf{x}_i + \theta)t_i > 0$.
- The perceptron's objective is given by

$$E(\mathbf{w}) = -\sum_{i \in M} (\mathbf{w} \cdot \mathbf{x}_i + \theta)t_i$$

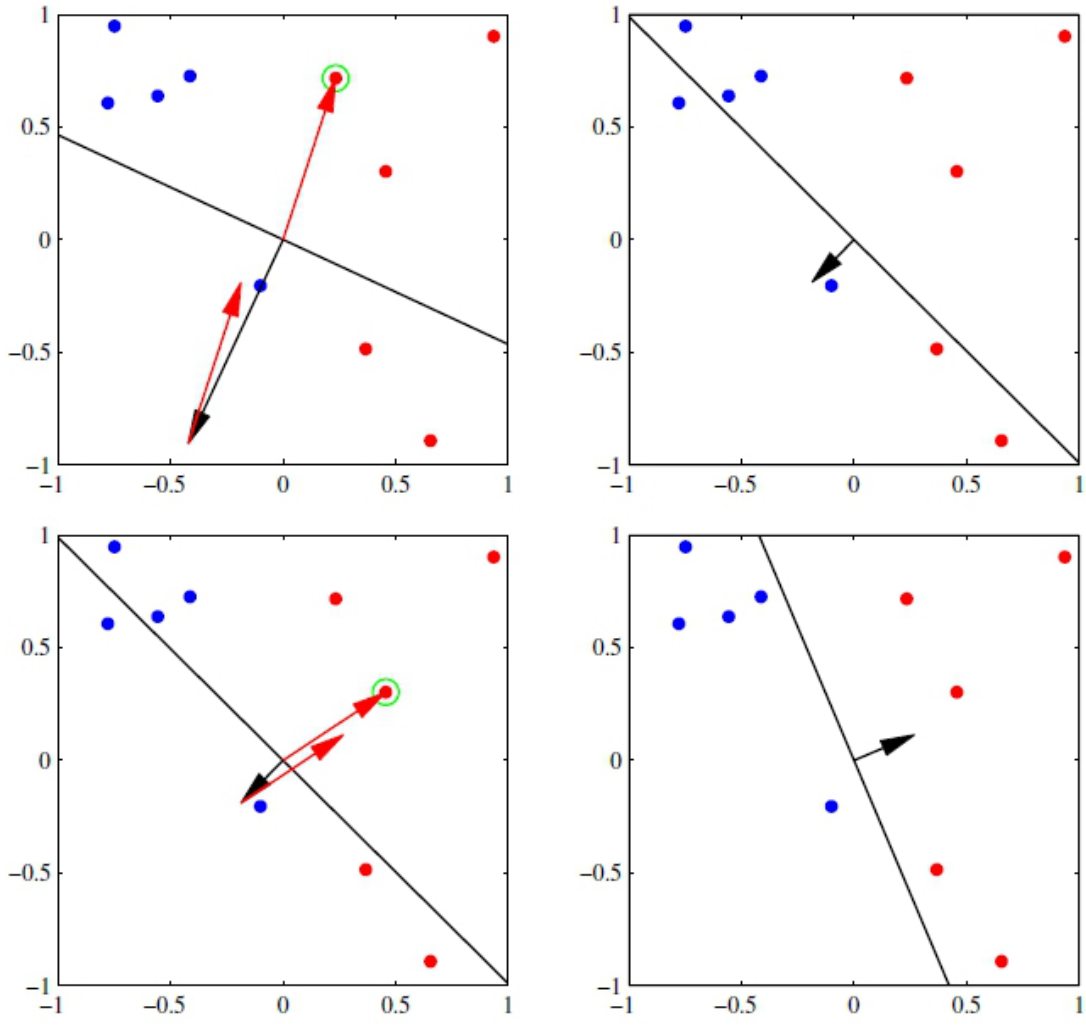
  where M is the set of misclassified examples.
- For a misclassified sample, we can improve its classification by reducing the objective function

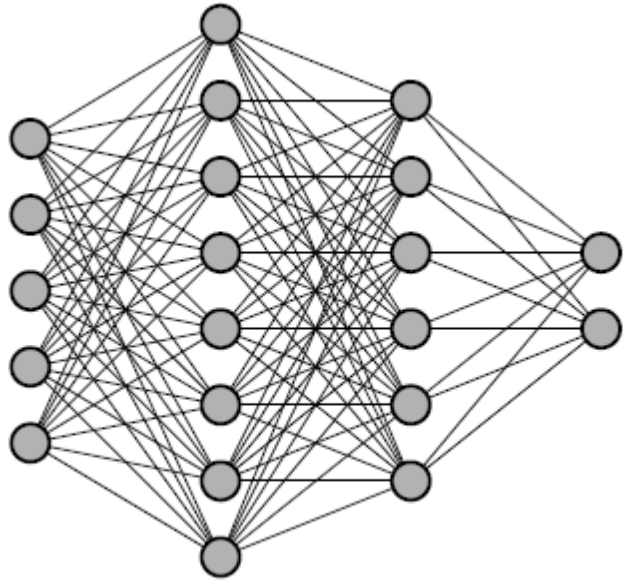
$$\mathbf{w}_{new} = \mathbf{w}_{old} - \nabla_{\mathbf{w}} E(\mathbf{w}, \mathbf{x}_i) = \mathbf{w}_{old} + \mathbf{x}_i t_i$$

Perceptron Algorithm

$$f(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + \theta)$$

- Initialize the parameters $\mathbf{w}$ randomly
- Cycle through the training samples
- For each sample, evaluate $\sigma\left(\mathbf{w} \cdot \mathbf{x}_i + \theta\right)$
- If the sample is classified correctly, leave $\mathbf{w}$ unchanged.
- If the sample is not classified correctly, add $\mathbf{x}_i t_i$ to $\mathbf{w}$


1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

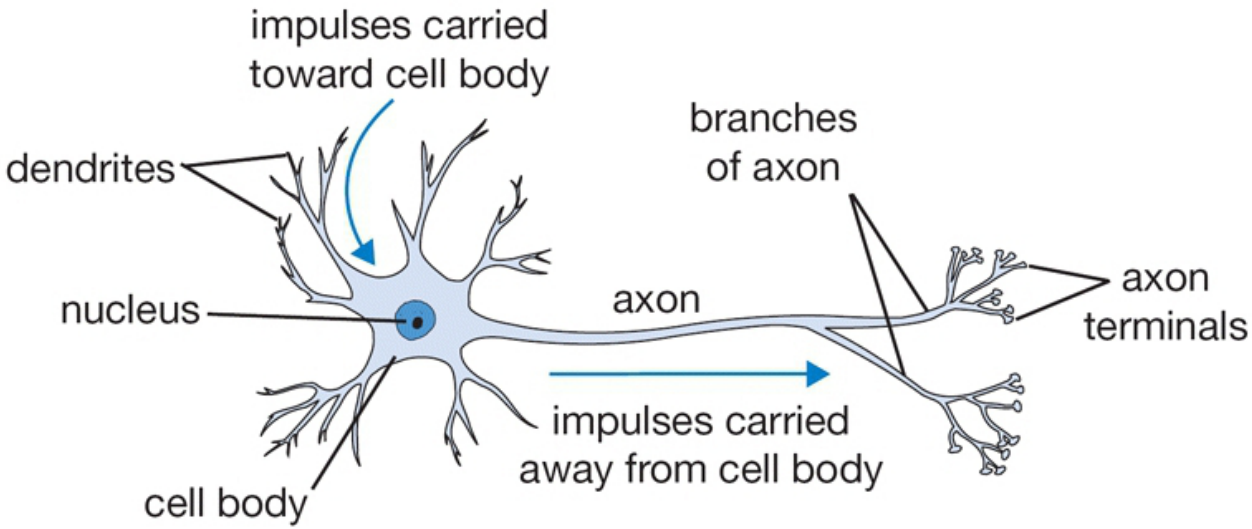


- A neural network can be considered as a multiperceptron.
- First resurgence of NNs in 1986, when Rumelhart, Hinton and Williams showed that neural networks can learn automatically interesting representations
- New area of big data and more computing power led to second resurgence around 2010.
- Recently, neural networks gained lots of attention under the name of "Deep Learning"

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Neural Networks

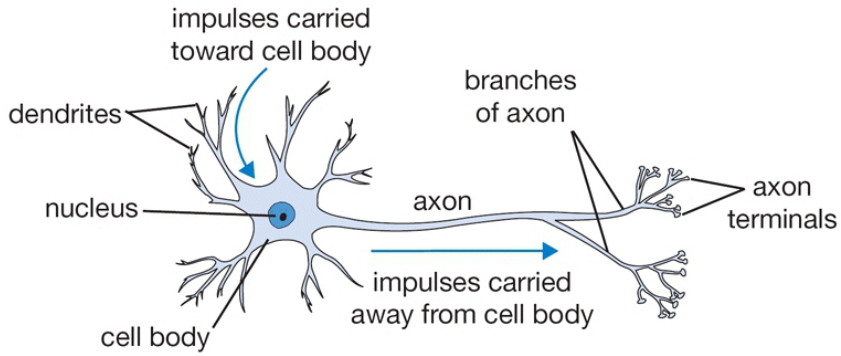
As their name suggests, neural networks are inspired by biological neural networks, in particular by the human brain.



- There are many different types of neurons and they are quite complicated
- Neurons are connected to each other with synapses forming a network.
- We consider a greatly simplified model.

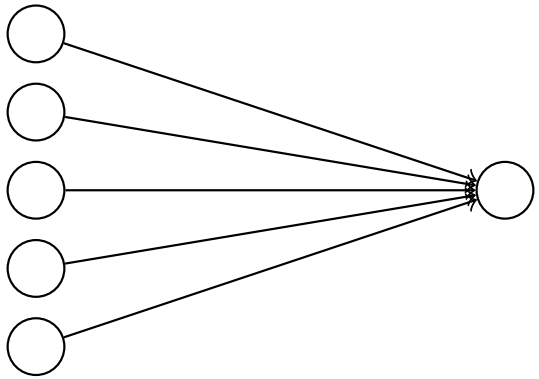
http://cs231n.github.io/neural-networks-1/

A simplified model



- A neuron receives input from a number of other neurons.
- The input comes as electrical signals over the connections.
- There are different kind of connections: Connections that strengthen the signal, and connections that reduce the signal.
- The incoming signals are summed up and passed through an activation function, which passes the signal on to other neurons or not.

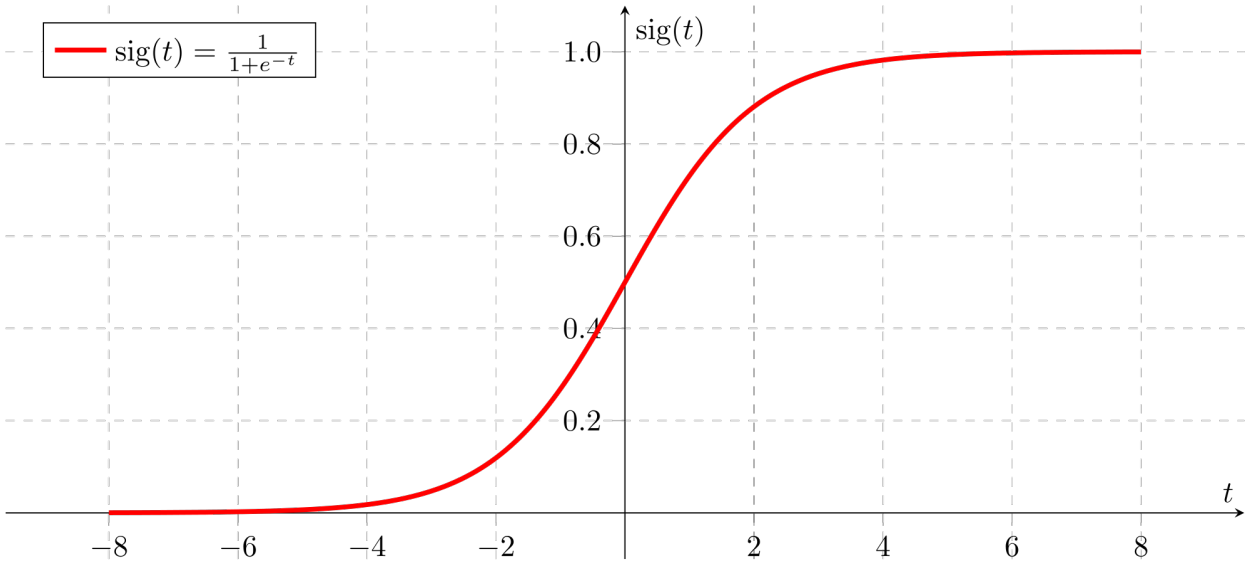
A simplified model



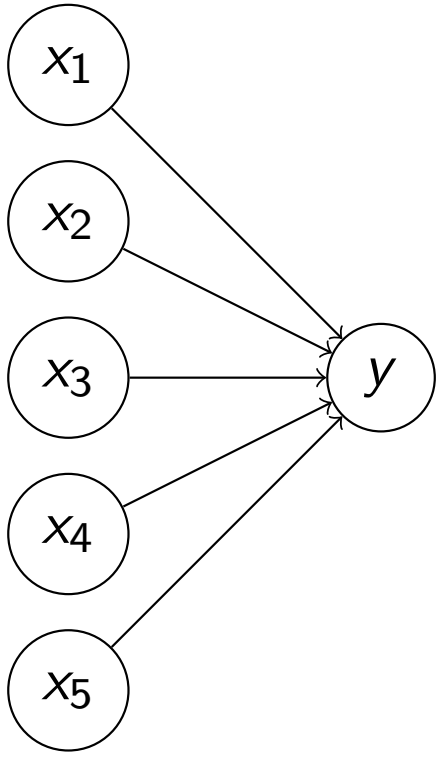
- A neuron receives input from a number of other neurons. For us, each neuron is given as a variable.
- The input comes as electrical signals over the connections, for us the signals are modeled by parameters w_i.
- There are different kind of connections: Connections that strengthen the signal ($w_i > 0$), and connections that reduce the signal ($w_i < 0$).
- The incoming signals are summed up and passed through an activation function. For us, the activation function is often given by the sigmoid function $sig(t) = \frac{1}{1+e^{-t}}$.

The sigmoid function

Our activation function, the sigmoid function $sig(t) = \frac{1}{1+e^{-t}}$, is a differentiable approximation of the step function $\hat{\sigma}$ of the perceptron.



Artificial Neurons



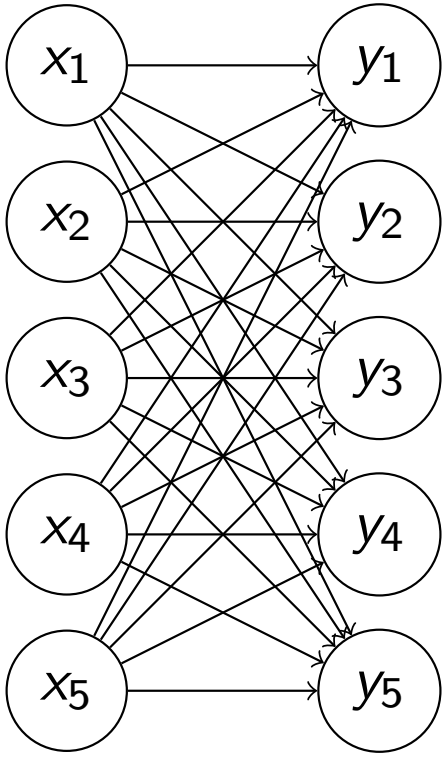
$$y = \sigma\left(\sum_{j=1} w_j x_j + b\right)$$

$$= \sigma\left(W \cdot x + b\right)$$

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

- Each neuron sends an activation x_i that is scaled with a weight w_i.
- The sum is passed through an activation (sigmoid) function.
- The value y is between 0 (not activated) and 1 (activated).

Fully connected Two-layer Neural network

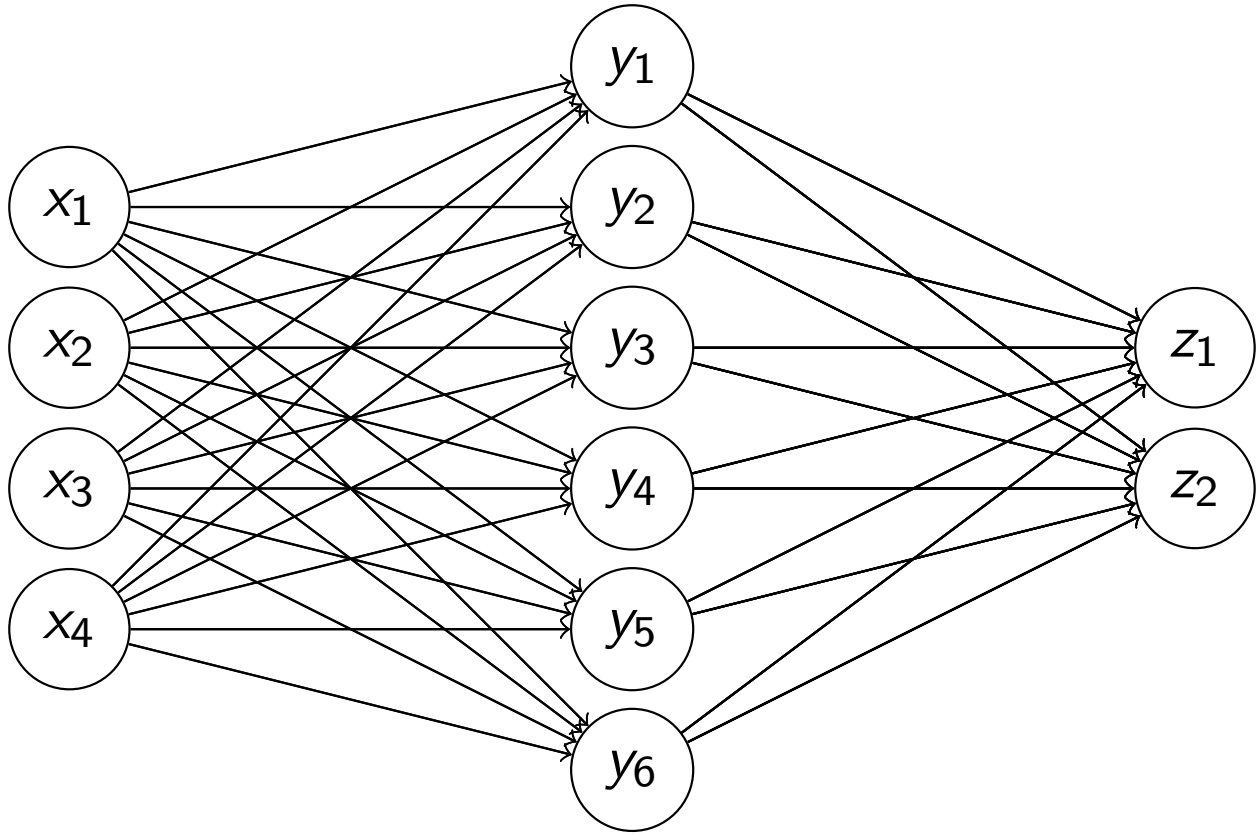


$$y_i = \sigma\left(\sum_{j=1} w_{ij}x_j + b_i\right)$$

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

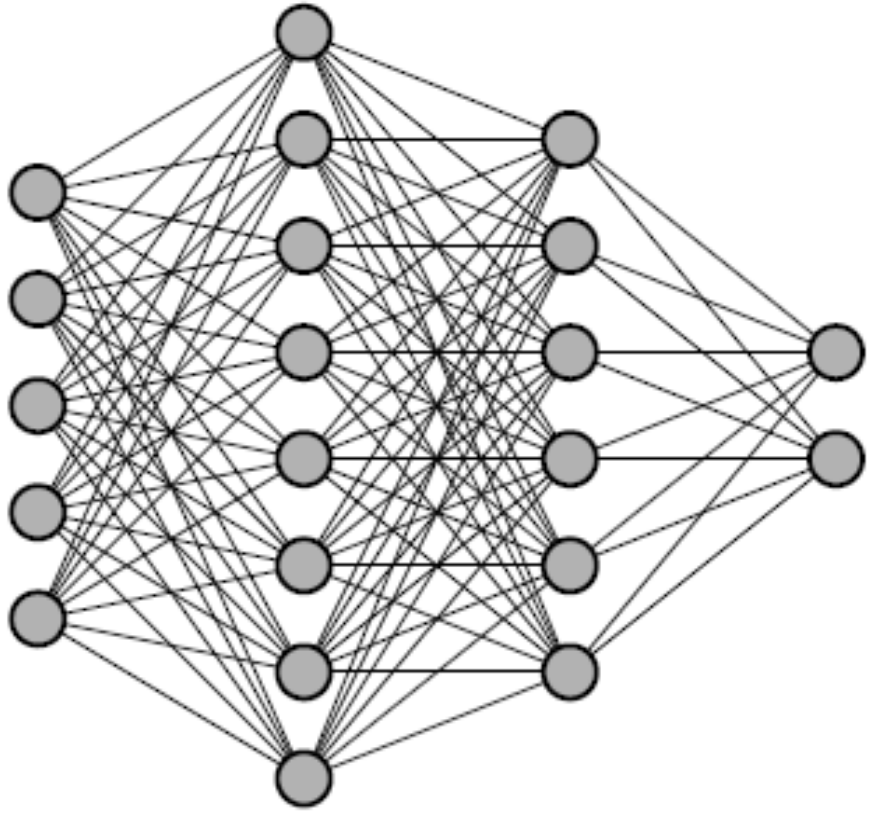
- Two layer neural network

Fully Connected Neural Network with one hidden layer



Input layer Hidden layer Output layer

- Can be used for binary classification.
- Input dimension is determined by data, output dimension by the task, the number of hidden layers and number of neurons may be chosen arbitrarily.



An example of a (feedforward) neural network with two hidden layers.
Feedforward neural networks are also called multi-layer perceptrons.

Back to our equation

Suppose we are given a training set (x_n, y_n). Let W_i denote the matrix of connection weights from layer $i-1$ to layer i. Then we compute our prediction

$$\hat{y}_n = \sigma_{out}\left(W_n \cdot \sigma\left(W_{n-1} \cdot \sigma\left(\ldots \cdot \sigma\left(W_2 \cdot \sigma(W_1 \cdot x_n)\right)\ldots\right)\right)\right)$$

$$= (\sigma_{out} \circ W_n \circ \sigma \circ W_{n-1} \circ \circ \ldots \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x_n)$$

Back to our equation

Suppose we are given a training set (x_n, y_n). Let W_i denote the matrix of connection weights from layer $i-1$ to layer i. Then we compute our prediction

$$\hat{y}_n = \sigma_{out}\left(W_n \cdot \sigma\left(W_{n-1} \cdot \sigma\left(\ldots \cdot \sigma\left(W_2 \cdot \sigma(W_1 \cdot x_n)\right)\ldots\right)\right)\right)$$

$$= (\sigma_{out} \circ W_n \circ \sigma \circ W_{n-1} \circ \circ \ldots \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x_n)$$

- The activation function σ should be non-linear and may be chosen as the sigmoid function (taken in each component of the vector).

Back to our equation

Suppose we are given a training set (x_n, y_n). Let W_i denote the matrix of connection weights from layer $i-1$ to layer i. Then we compute our prediction

$$\hat{y}_n = \sigma_{out}\left(W_n \cdot \sigma\left(W_{n-1} \cdot \sigma\left(\ldots \cdot \sigma\left(W_2 \cdot \sigma(W_1 \cdot x_n)\right)\ldots\right)\right)\right)$$

$$= (\sigma_{out} \circ W_n \circ \sigma \circ W_{n-1} \circ \circ \ldots \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x_n)$$

- The activation function σ should be non-linear and may be chosen as the sigmoid function (taken in each component of the vector).
- σ_{out} depends on the task.
  - For regression tasks, σ_{out} is not needed and may be chosen as the identity function.
  - For classification tasks, σ_{out} should map into $[0, 1]$ to obtain a confidence values for each class. Hence, $\sigma_{out} = \sigma$ may be used.

Back to our equation

Suppose we are given a training set (x_n, y_n). Let W_i denote the matrix of connection weights from layer $i-1$ to layer i. Then we compute our prediction

$$\hat{y}_n = \sigma_{out}\left(W_n \cdot \sigma\left(W_{n-1} \cdot \sigma\left(\ldots \cdot \sigma\left(W_2 \cdot \sigma(W_1 \cdot x_n)\right)\ldots\right)\right)\right)$$

$$= (\sigma_{out} \circ W_n \circ \sigma \circ W_{n-1} \circ \circ \ldots \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x_n)$$

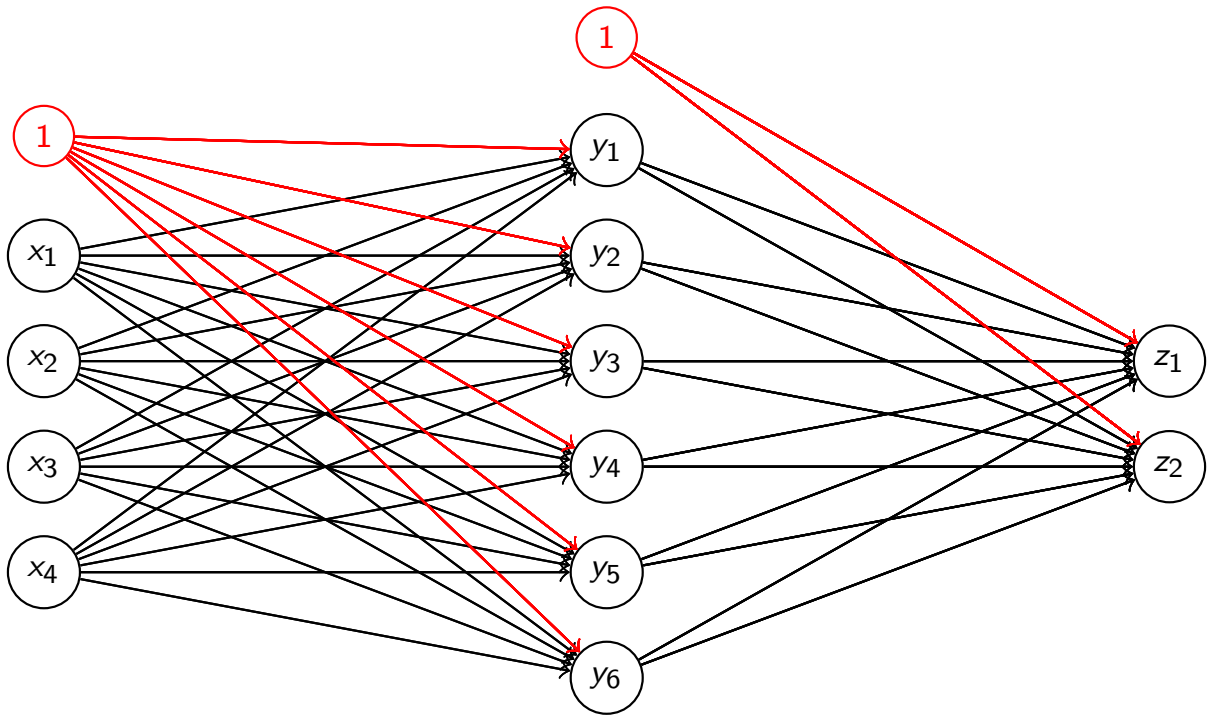
- The activation function σ should be non-linear and may be chosen as the sigmoid function (taken in each component of the vector).
- σ_{out} depends on the task.
  - For regression tasks, σ_{out} is not needed and may be chosen as the identity function.
  - For classification tasks, σ_{out} should map into $[0, 1]$ to obtain a confidence values for each class. Hence, $\sigma_{out} = \sigma$ may be used.
- Our goal is to find the weights in W_i such that $||\hat{y}_n - y_n||$ is minimal.

The final equation of neural networks

Usually, the linear function $W_i \cdot z_i$ at each layer i (determined by the weights matrix W_i) gets extended by adding a so-called bias vector b_i:

$$W_i \cdot z_i + b_i.$$

This corresponds to an additional neuron with a constant activation of 1.



The final equation of neural networks

This results in the model
(for regression with $\sigma_{out}(x) = x$ for all x)

$$\hat{y} = (b_n + W_n \cdot \sigma\left(b_{n-1} + W_{n-1} \cdot \sigma\left(\ldots \cdot \sigma\left(b_2 + W_2 \cdot \sigma(b_1 + W_1 \cdot x_n)\right)\ldots\right)\right))$$

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Training a Neural Net

We define a loss function on top of the final layer. Let f_W denote the mapping represented by the network for particular weights W, where W contains the collection $\{W_1, \ldots, W_n\}$ of weights from all layers.
For given example input x, let

$$f_W(x) = \hat{y}$$

denote the output vector of the neural network and y the desired output vector.

Training a Neural Net

We define a loss function on top of the final layer. Let f_W denote the mapping represented by the network for particular weights W, where W contains the collection $\{W_1, \ldots, W_n\}$ of weights from all layers.
For given example input x, let

$$f_W(x) = \hat{y}$$

denote the output vector of the neural network and y the desired output vector.

A loss function ℓ assigns value $\ell(x, y; W) = \ell(\hat{y}, y)$ that quantifies how far away the prediction $\hat{y}$ is from y. So ideally, it should be minimal if and only if $\hat{y} = y$. For example, for regression tasks, $\ell(\hat{y}, y) = ||\hat{y} - y||^2$ is a common choice (where $|| \cdot ||$ denotes the Euclidean norm).

Losses

- The loss should model whatever we want the neural network to do.
- Minimizing such a function will direct the network to its intended functionality.

Losses - Softmax for classification

Let $\mathbf{z} = \begin{bmatrix} z_1^{(L)} & z_2^{(L)} & \dots & z_n^{(L)} \end{bmatrix}$ denote the values at the last layer. Then let

$$\hat{y}_i = \sigma_{out}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

- This is called a softmax transformation. If the z-values are not too close $\hat{y}_i \approx 1$ for the largest z_i and $\hat{y}_i \approx 0$ for the other.

Losses - Softmax for classification

Let $\mathbf{z} = \begin{bmatrix} z_1^{(L)} & z_2^{(L)} & \dots & z_n^{(L)} \end{bmatrix}$ denote the values at the last layer. Then let

$$\hat{y}_i = \sigma_{out}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

- This is called a softmax transformation. If the z-values are not too close $\hat{y}_i \approx 1$ for the largest z_i and $\hat{y}_i \approx 0$ for the other.
- Note that $0 < \hat{y}_i < 1$ and $\sum_{i=1}^{n} \hat{y}_i = 1$, so it models probabilities.

Losses - Softmax for classification

Let $\mathbf{z} = \begin{bmatrix} z_1^{(L)} & z_2^{(L)} & \dots & z_n^{(L)} \end{bmatrix}$ denote the values at the last layer. Then let

$$\hat{y}_i = \sigma_{out}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

- This is called a softmax transformation. If the z-values are not too close $\hat{y}_i \approx 1$ for the largest z_i and $\hat{y}_i \approx 0$ for the other.
- Note that $0 < \hat{y}_i < 1$ and $\sum_{i=1}^{n} \hat{y}_i = 1$, so it models probabilities.
- Negative log likelihood of class c

$$\ell(\hat{y}, y = 1_c) = -\log(\hat{y}_c) = -\log\left(\frac{e^{z_c}}{\sum_{j=1}^{n} e^{z_j}}\right) = -z_c + \log\left(\sum_{j=1}^{n} e^{z_j}\right)$$

- A low value corresponds to a good and confident prediction.

Losses - Margin loss for classification

Another choice is the margin loss.
Let σ_{out} be the softmax function as above.
Let $0 < \rho < 1$ denote a margin value.
Then

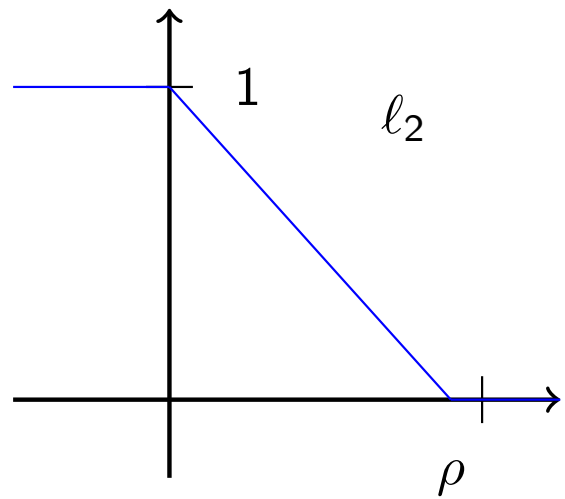
$$\ell(\hat{y}, y = 1_c) = \ell_2(\ell_1(\hat{y}, 1_c))$$

with

$$\ell_1(\hat{y}, 1_c) = \hat{y}_c - \max_{d \neq c} \hat{y}_d$$

and

$$\ell_2(x) = \begin{cases} 0 & , \ x \leq \rho \\ 1 - \frac{x}{\rho} & , \ 0 \leq x \leq \rho \\ 1 & , \ z \leq 0 \end{cases}$$



Training a Neural Net

Network output $f_W(x) = \hat{y}$
Loss function ℓ

- We wish to find good weight parameters W_i to complete our model.
- Given training data $(x_i, y_i)_{i=1}^N$. We wish to minimize the average loss over the training data (the empirical loss)

$$\mathcal{L}(W) = \mathcal{L}(W_1, W_2, \ldots, W_n) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i; W)$$

- $\mathcal{L}(W)$ is a nonlinear function in parameters $W_1, \ldots, W_n$. The minimization of a multivariate function is a well-known learning problem.

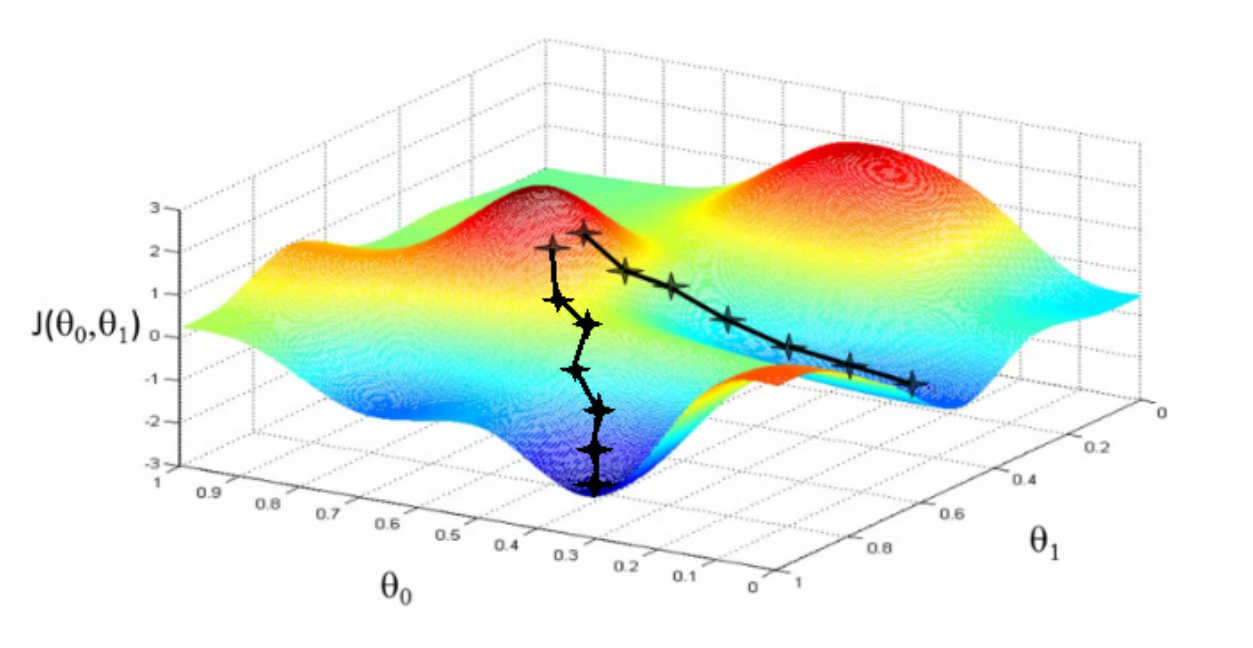
- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Gradient Descent

So we have a function dependent on the weight parameters that we want to minimize.

$$\mathcal{L}(W) = \mathcal{L}(W_1, W_2, \ldots, W_n) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i; W)$$

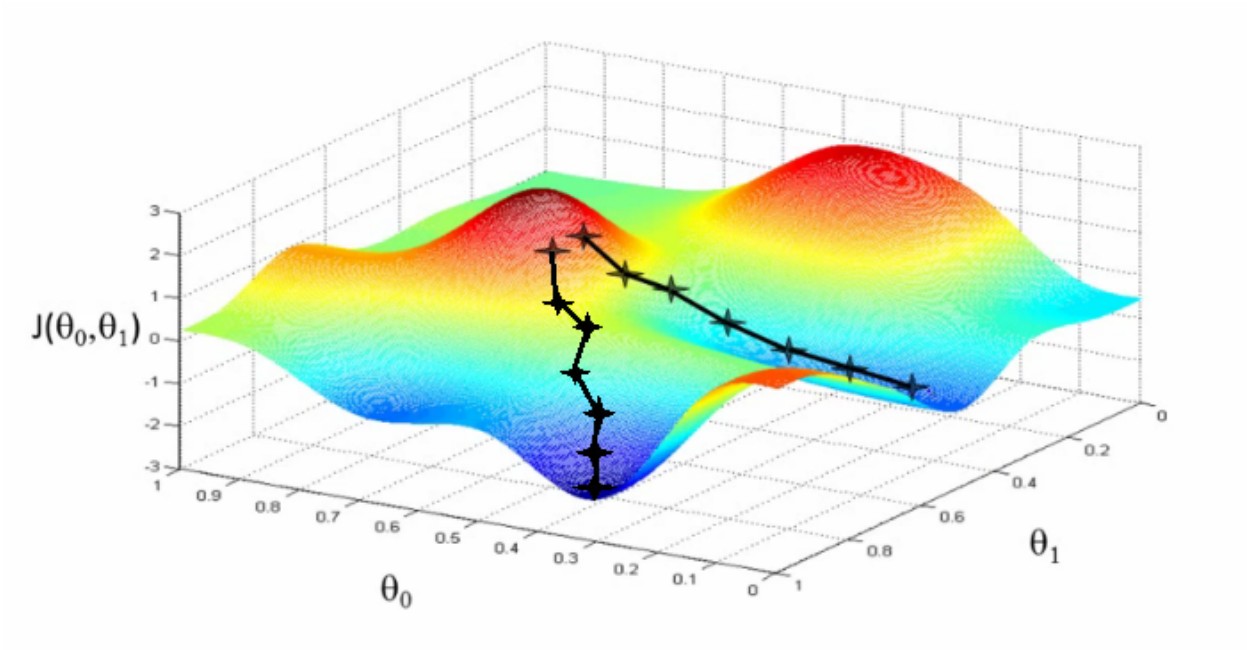
Gradient Descent



https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/03/06100746/grad.png

- Goal: Minimize function $\mathcal{L} = \mathcal{L}(W)$
- Start at a random point $W^0 = (W_1^0, W_2^0, \ldots, W_n^0)$
- Iteratively, at each point go into direction of steepest descent
- Direction of steepest descent is given by the negative of the gradient.
- $\Rightarrow W^{n+1} = W^n - \nabla_\theta \mathcal{L}$,

Gradient Descent



https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/03/06100746/grad.png

- Goal: Minimize function $\mathcal{L} = \mathcal{L}(W)$
- Start at a random point $W^0 = (W_1^0, W_2^0, \ldots, W_n^0)$
- Iteratively, at each point go into direction of steepest descent
- Direction of steepest descent is given by the negative of the gradient.
- $\Rightarrow W^{n+1} = W^n - \alpha \cdot \nabla_\theta \mathcal{L}$,
  α is called step size and is a hyperparameter to be chosen

Stochastic gradient descent

The total loss over the training data, given by

$$\mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i; W),$$

has gradient

$$\nabla \mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W \ell(x_i, y_i; W).$$

So at each step of gradient descent we need to compute N many values of the derivative.

Stochastic gradient descent

The total loss over the training data, given by

$$\mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i; W),$$

has gradient

$$\nabla \mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W \ell(x_i, y_i; W).$$

So at each step of gradient descent we need to compute N many values of the derivative.

- The number of training data points N is usually very large.
- (Full) Gradient descent is too expensive.

Stochastic gradient descent

The total loss over the training data, given by

$$\mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i; W),$$

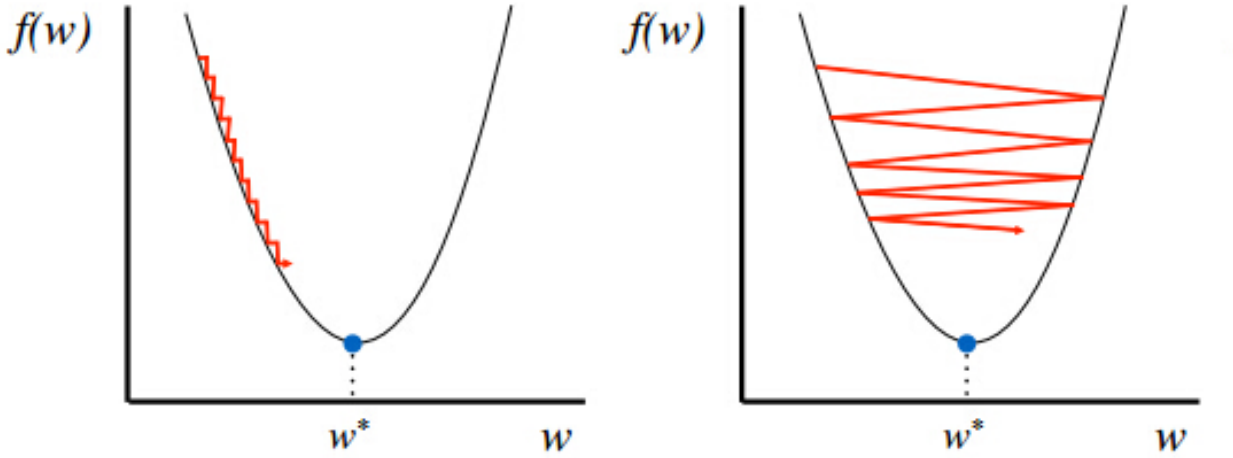
has gradient

$$\nabla \mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W \ell(x_i, y_i; W).$$

So at each step of gradient descent we need to compute N many values of the derivative.

- The number of training data points N is usually very large.
- (Full) Gradient descent is too expensive.
- We divide the training data into (random) small batches
- We use gradient descent with small batches, called **stochastic gradient descent (SGD)**.

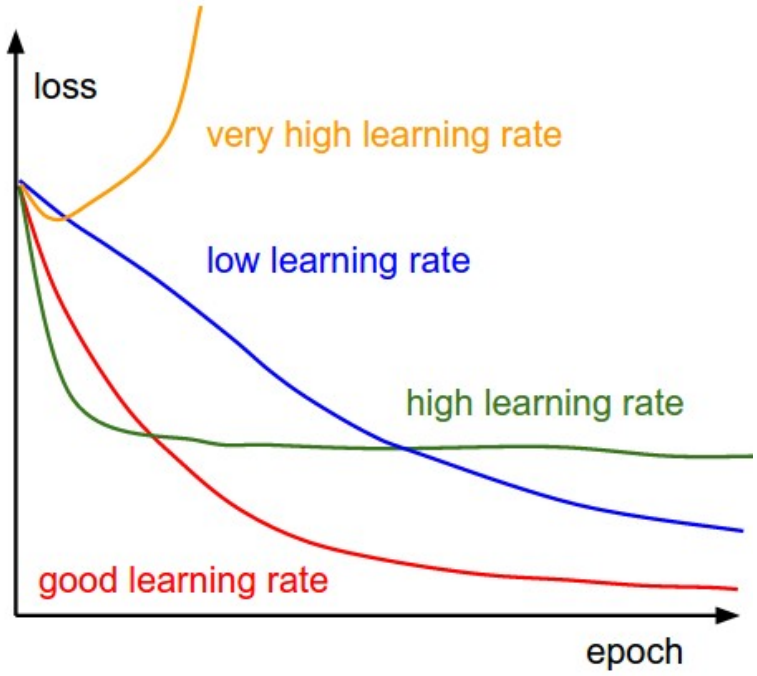
Choice of the step size



- If the step size is chosen too small, then training is very slow
- If the step size is too large, we might overstep minima and never converge
- Ideally, the step size is adapted over time. There are many such approaches (AdaGrad, RMSProp, Gradient descent with momentum,...)

http://prog3.com/sbdm/blog/google19890102/article/details/50276775

Choice of the step size



http://cs231n.github.io/neural-networks-3/

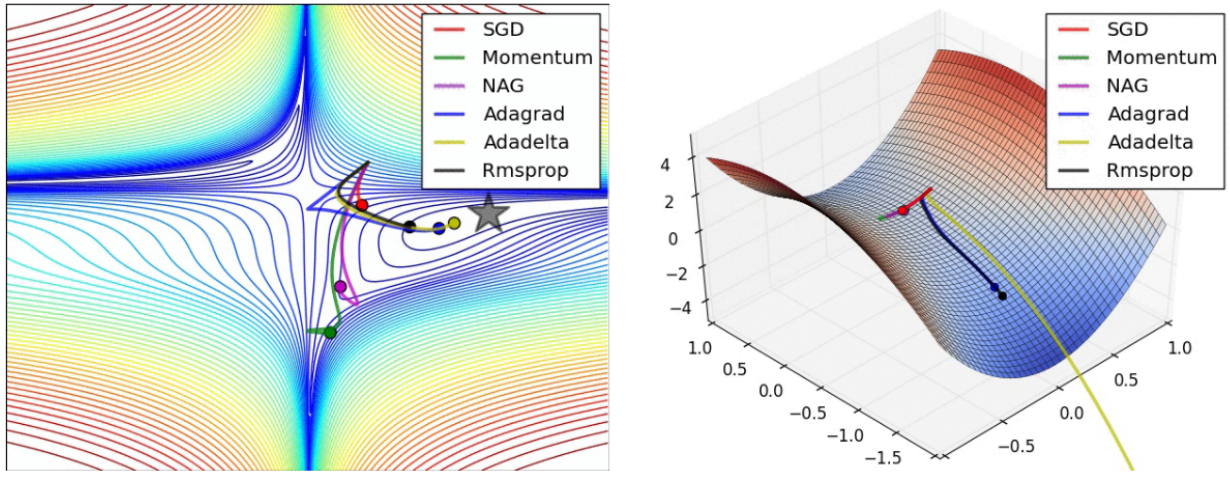
Visualization of different methods

E.g. Gradient descent with momentum:

$$\mathbf{m}_n = \mu \mathbf{m}_{n-1} - \alpha \nabla \mathcal{L}$$
$$W^{n+1} = W^n + \mathbf{m}_n$$

We call μ the momentum parameter. Typical values for μ are 0.9 or 0.99 and the choice $\alpha = 1 - \mu$.

`http://cs231n.github.io/assets/nn3/opt2.gif`
`http://cs231n.github.io/assets/nn3/opt1.gif`



Stochastic gradient descent

Under mild conditions, convergence follows from the Robbins-Siegmund theorem. Roughly:

If the loss function is (pseudo-)convex and the step sizes α_k of step k in stochastic gradient descent satisfy

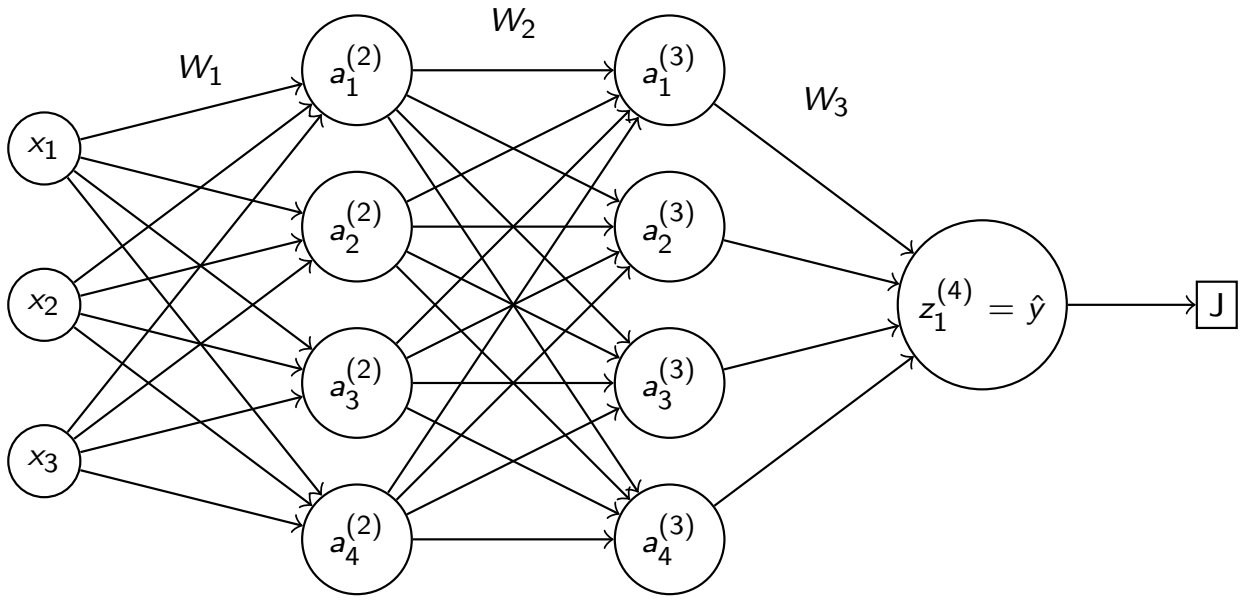
$$\sum \alpha_k = \infty, \text{ i.e., they are large enough,}$$

$$\text{and } \sum \alpha_k^2 < \infty, \text{ i.e., they are small enough,}$$

then the stochastic gradient descent algorithm converges to a (local) minimum almost surely.

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Forwardpropagation



Forward propagation (in vector notation):

$a^{(1)} = x$
$z^{(2)} = W_1 \cdot a^{(1)}$
$a_2 = \sigma(z_2)$
$z^{(3)} = W_2 \cdot a^{(2)}$
$a_3 = \sigma(z_3)$
$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$

Given one training example (x, y)

$$J((x, y), W) = \frac{1}{2}(\hat{y} - y)^2$$

can be computed algorithmically by forward propagation

Backpropagation

Forward propagation:

$a^{(1)} = x$
$z^{(2)} = W_1 \cdot a^{(1)}$
$a^{(2)} = \sigma(z^{(2)})$
$z^{(3)} = W_2 \cdot a^{(2)}$
$a^{(3)} = \sigma(z^{(3)})$
$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$
$(W_k = [w_{i,j}^{(k)}]_{i,j}, W = (W_1, W_2, W_3))$

$$\frac{\partial J((x,y),W)}{\partial w_{1,2}^{(1)}} = ???$$

Backpropagation

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = W_1 \cdot a^{(1)}$$
$$a^{(2)} = \sigma(z^{(2)})$$
$$z^{(3)} = W_2 \cdot a^{(2)}$$
$$a^{(3)} = \sigma(z^{(3)})$$
$$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$$
$$(W_k = [w_{i,j}^{(k)}]_{i,j}, W = (W_1, W_2, W_3))$$

$$\frac{\partial J((x,y),W)}{\partial w_{1,2}^{(1)}} = ???$$

$$J((x,y),W) = \frac{1}{2}\left[(W_3 \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x) - y\right]^2$$

$$= \frac{1}{2}\left[\sum_{i=1}^{4} w_{1,i}^{(3)} \sigma\left(\sum_{j=1}^{4} w_{i,j}^{(2)} \sigma\left(\sum_{k=1}^{3} w_{j,k}^{(1)} x_k\right)\right) - y\right]^2$$

Backpropagation

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = W_1 \cdot a^{(1)}$$
$$a^{(2)} = \sigma(z^{(2)})$$
$$z^{(3)} = W_2 \cdot a^{(2)}$$
$$a^{(3)} = \sigma(z^{(3)})$$
$$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$$
$$(W_k = [w_{i,j}^{(k)}]_{i,j}, W = (W_1, W_2, W_3))$$

$$\frac{\partial J((x,y),W)}{\partial w_{1,2}^{(1)}} = ???$$

$$J((x,y),W) = \frac{1}{2}\left[(W_3 \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x) - y\right]^2$$

$$= \frac{1}{2}\left[\sum_{i=1}^{4} w_{1,i}^{(3)} \sigma\left(\sum_{j=1}^{4} w_{i,j}^{(2)} \sigma\left(\sum_{k=1}^{3} w_{j,k}^{(1)} x_k\right)\right) - y\right]^2$$

The idea of backpropagation is to use use the chain rule and propagate the changes backwards through the net to efficiently compute the gradient of the loss function with respect to the weight parameters.

Backpropagation

Forward propagation:

$a^{(1)} = x$
$z^{(2)} = W_1 \cdot a^{(1)}$
$a^{(2)} = \sigma(z^{(2)})$
$z^{(3)} = W_2 \cdot a^{(2)}$
$a^{(3)} = \sigma(z^{(3)})$
$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$
$(W_k = [w_{i,j}^{(k)}]_{i,j}, W = (W_1, W_2, W_3))$

$$\frac{\partial J((x,y),W)}{\partial w_{1,2}^{(1)}} = ???$$

$$J((x,y),W) = \frac{1}{2}\left[(W_3 \circ \sigma \circ W_2 \circ \sigma \circ W_1)(x) - y\right]^2$$

$$\frac{\partial J}{\partial w_{1,2}^{(1)}}[(x,y),W] = \frac{\partial J}{\partial z^{(2)}}[z^{(2)},y] \cdot \frac{\partial z^{(2)}}{\partial w_{1,2}^{(1)}}[W_1,x]$$

The idea of backpropagation is to use use the chain rule and propagate the changes backwards through the net to efficiently compute the gradient of the loss function with respect to the weight parameters.

Backpropagation

Forward propagation:

$a^{(1)} = x$
$z^{(2)} = W_1 \cdot a^{(1)}$
$a_2 = \sigma(z_2)$
$z^{(3)} = W_2 \cdot a^{(2)}$
$a_3 = \sigma(z_3)$
$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$

$$\text{Given} \quad h(x) = (f \circ g)(x) = f(g(x))$$

then

$$h'(x) = (f \circ g)'(x) = f'(g(x)) \cdot g'(x)$$

Backpropagation

Forward propagation:

$a^{(1)} = x$
$z^{(2)} = W_1 \cdot a^{(1)}$
$a_2 = \sigma(z_2)$
$z^{(3)} = W_2 \cdot a^{(2)}$
$a_3 = \sigma(z_3)$
$z^{(4)} = W_3 \cdot a^{(3)} = \hat{y}$

$$\text{Given} \quad h(x) = (f \circ g)(x) = f(g(x))$$

$$J(W) = J(W_3, W_2, z^{(2)}(W_1))$$

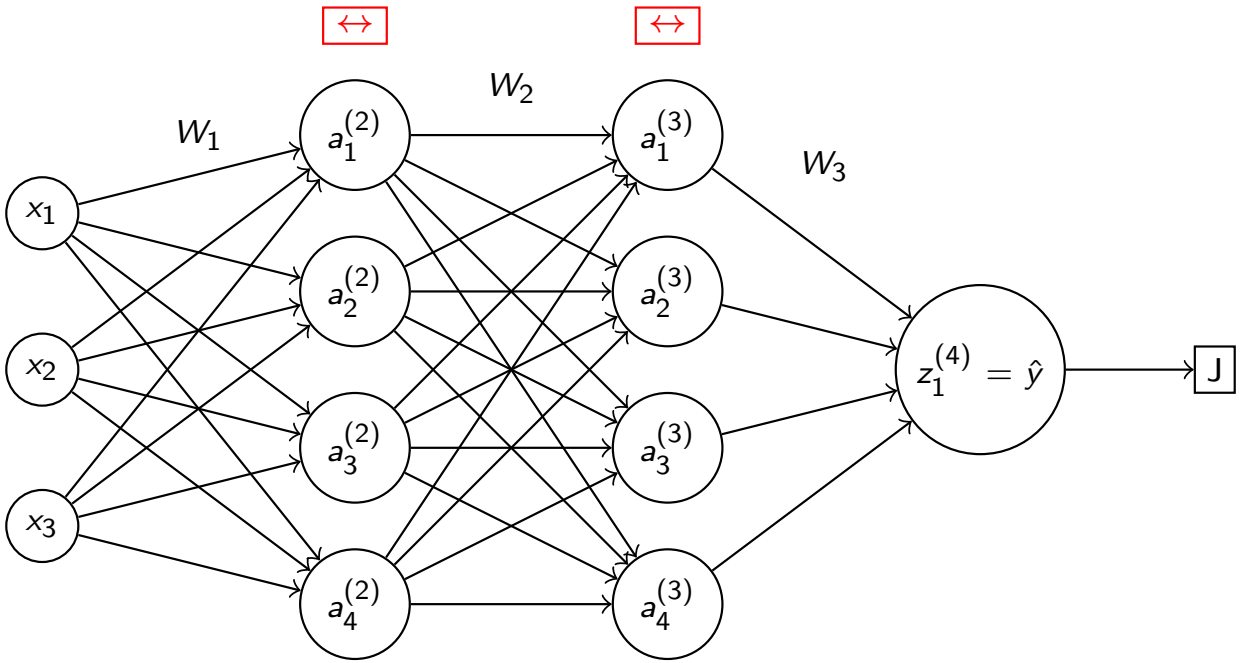
then

$$h'(x) = (f \circ g)'(x) = f'(g(x)) \cdot g'(x)$$

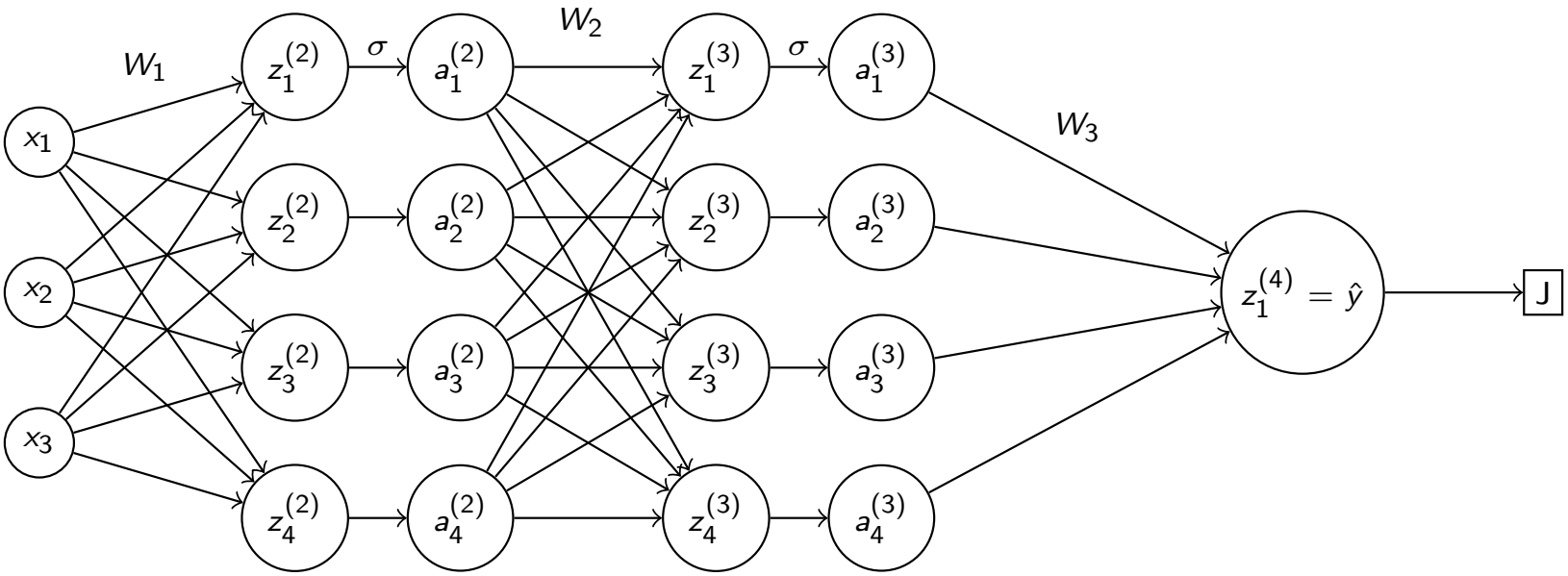
$$\frac{\partial J(W)}{\partial w_{1,2}^{(1)}} = \frac{\partial J(W_3, W_2, z^{(2)})}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}(W_1)}{\partial w_{1,2}^{(1)}}$$

Backpropagation

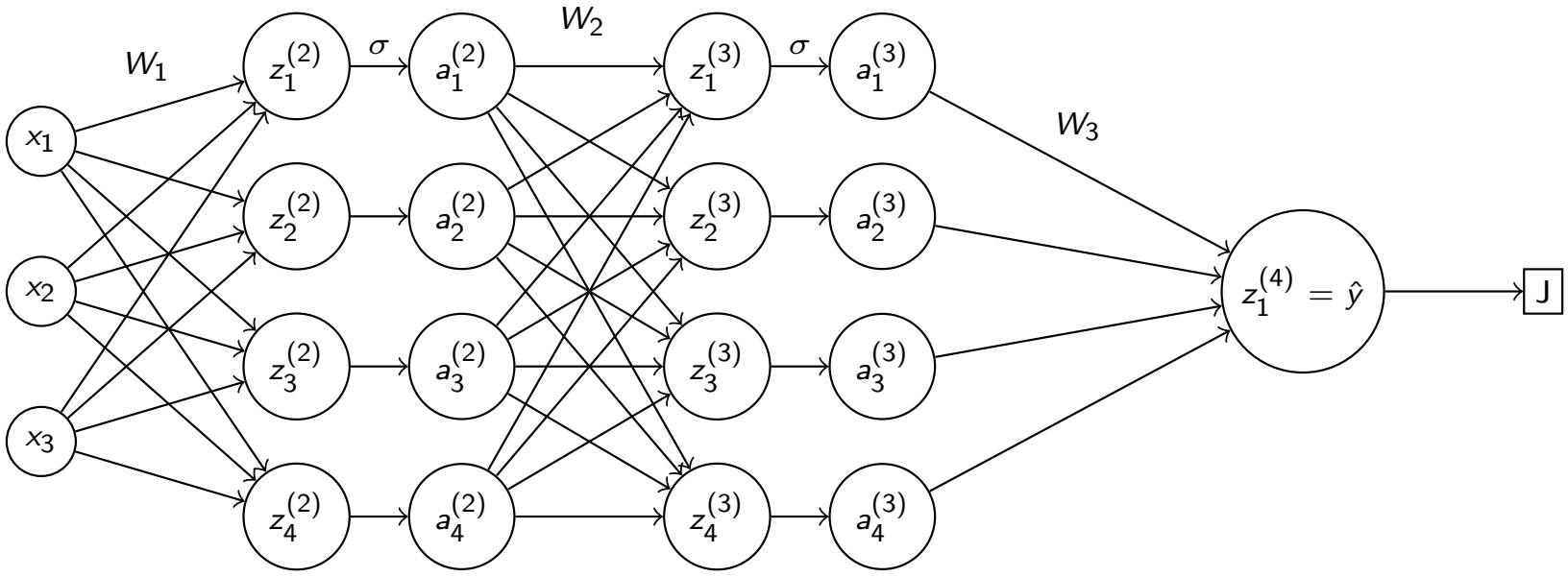
We split the nodes into before and after the activation.



Backpropagation

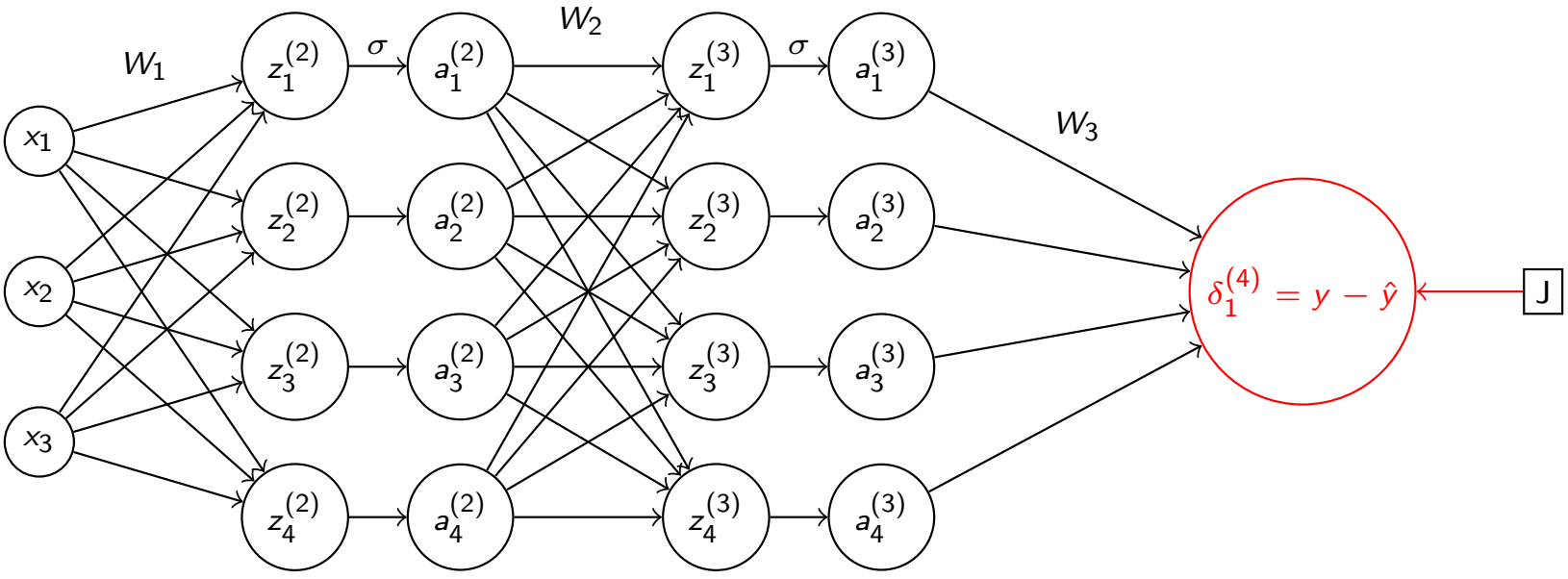


Backpropagation



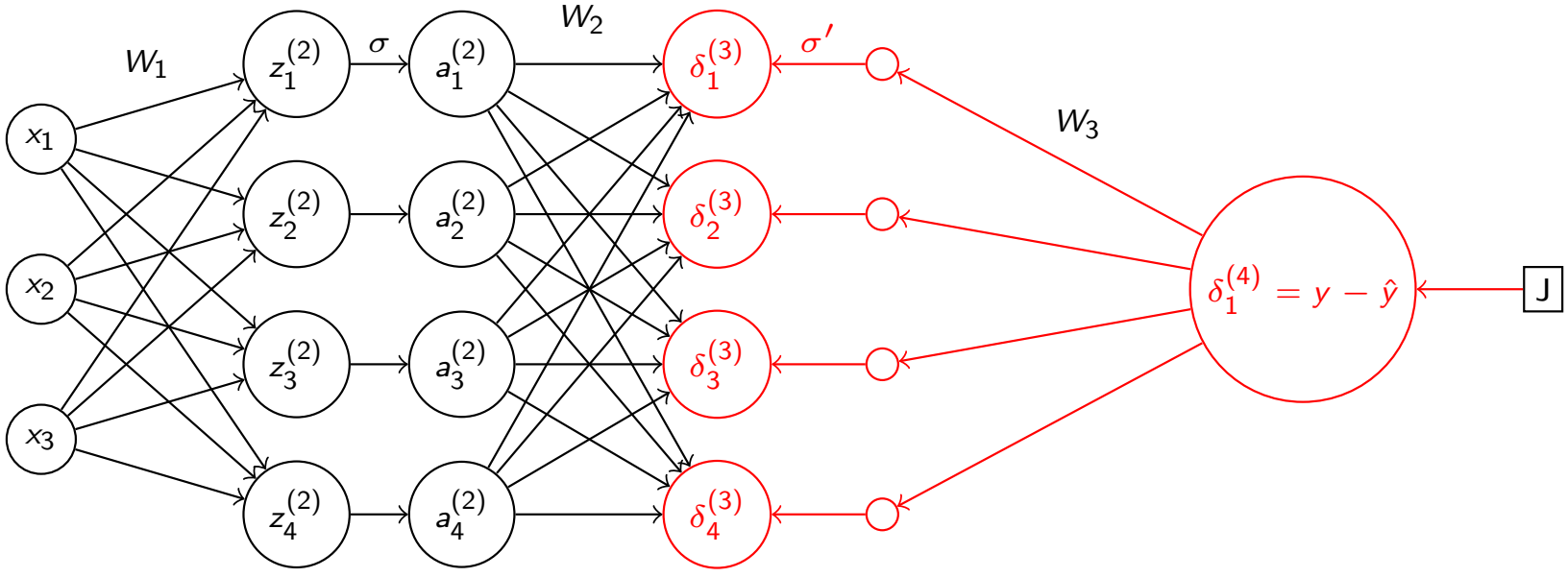
Define $\delta_j^{(l)} := \frac{\partial J}{\partial z_j^{(l)}} =$ the change of loss with change of incoming signal at the j-th node in layer l.

Backpropagation



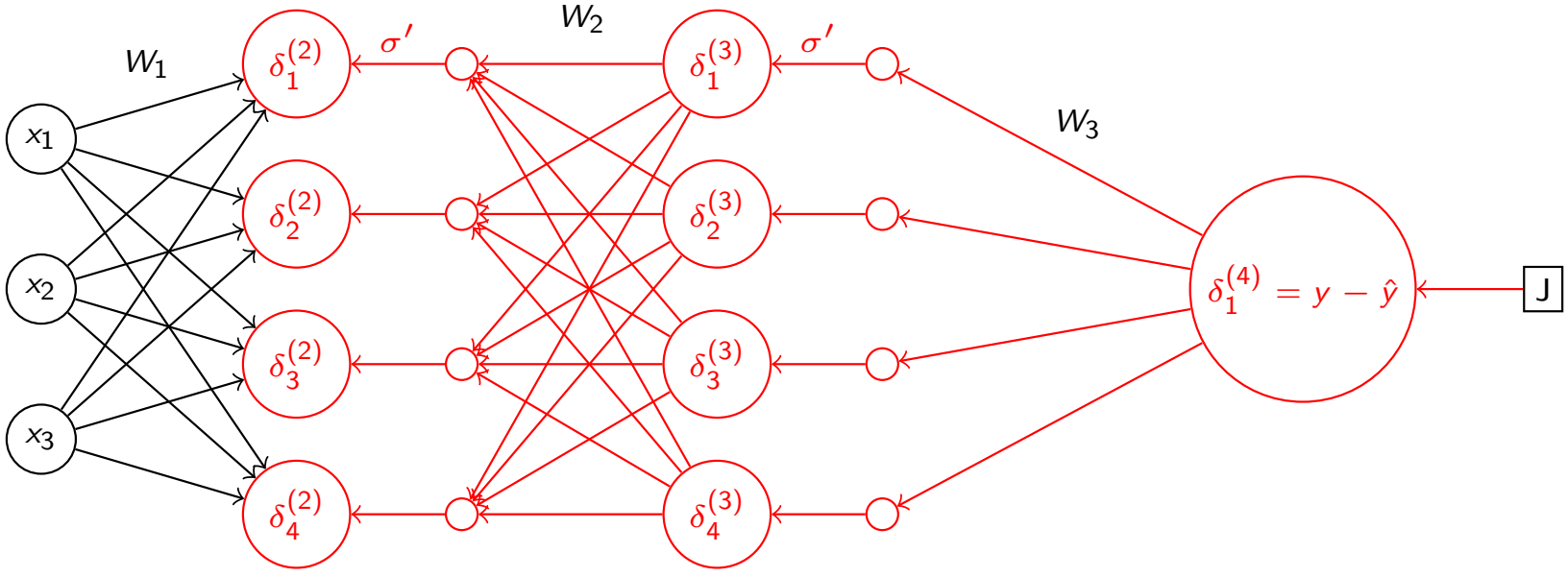
$$\delta_1^{(4)} = \frac{\partial J}{\partial z_1^{(4)}} = \frac{\partial J}{\partial \hat{y}} = y - \hat{y}$$

Backpropagation



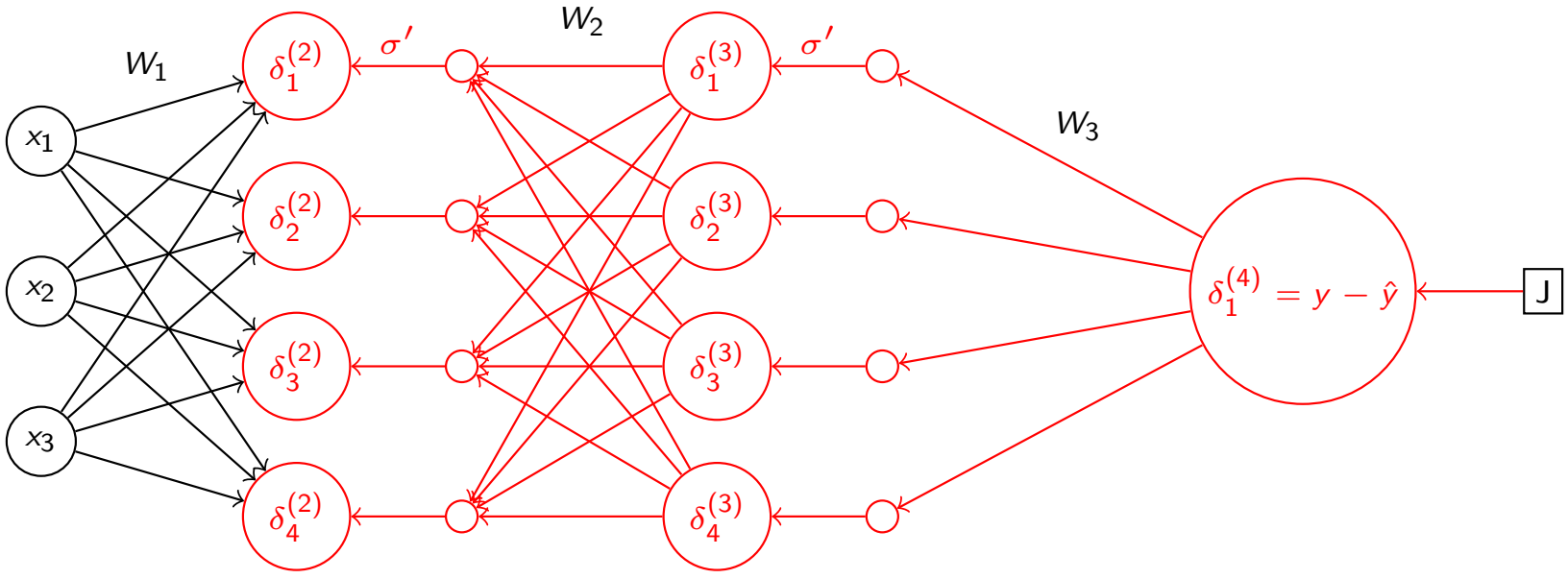
$$\delta_j^{(3)} = \frac{\partial J}{\partial z_j^{(3)}} = \underbrace{\frac{\partial J}{\partial z_1^{(4)}}}_{\delta_1^{(4)}} \cdot \underbrace{\frac{\partial z_1^{(4)}}{\partial a_j^{(3)}}}_{w_{1,j}^{(3)}} \cdot \underbrace{\frac{\partial a_j^{(3)}}{\partial z_j^{(3)}}}_{a_j^{(3)} \cdot (1 - a_j^{(3)})}$$

Backpropagation



$$\delta_j^{(2)} = \frac{\partial J}{\partial z_j^{(2)}} = \sum_{k=1}^{4} \underbrace{\frac{\partial J}{\partial z_k^{(3)}}}_{\delta_k^{(3)}} \cdot \underbrace{\frac{\partial z_k^{(3)}}{\partial a_j^{(2)}}}_{w_{k,j}^{(2)}} \cdot \underbrace{\frac{\partial a_j^{(2)}}{\partial z_j^{(2)}}}_{a_j^{(2)} \cdot (1 - a_j^{(2)})}$$

Backpropagation



$$\Rightarrow \frac{\partial J}{\partial w_{1,2}^{(1)}} = \frac{\partial J}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{1,2}^{(1)}} = \delta_1^{(2)} \cdot x_2$$

Backpropagation

- For each layer type we implement the forward and backward passes.

Backpropagation

- For each layer type we implement the forward and backward passes.
- We "propagate forward" the input to compute the activations for the given parameters.

Backpropagation

- For each layer type we implement the forward and backward passes.
- We "propagate forward" the input to compute the activations for the given parameters.
- We "propagate backward" the derivative of the loss with respect to the layers

Backpropagation

- For each layer type we implement the forward and backward passes.
- We "propagate forward" the input to compute the activations for the given parameters.
- We "propagate backward" the derivative of the loss with respect to the layers
- We compute the partial derivatives of J with respect to the weight parameters by using the chain rule and derivatives of the next layer.

Backpropagation

- For each layer type we implement the forward and backward passes.
- We "propagate forward" the input to compute the activations for the given parameters.
- We "propagate backward" the derivative of the loss with respect to the layers
- We compute the partial derivatives of J with respect to the weight parameters by using the chain rule and derivatives of the next layer.
- This approach highly modular and we can easily add new layers and change the architecture.

Backpropagation-Algorithm (vectorized)

Given training set $\{(x_i, y_i) \mid i = 1, \ldots, N\}$, to compute one update of weights using backpropagation with a batch size of m for a network with L layers:

$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \Delta_{i,j}^{(l)} := 0$

for $s = 1, \ldots, N$:

$a^{(1)} := x_s$

Perform forward propagation to compute every $a^{(l)}$

$\delta^{(L)} := a^{(L)} - y_s$

for $j = 1, \ldots, L-2$

$\delta^{(L-j)} \leftarrow (W_{L-j}^T \delta^{(L-j+1)}) .* a^{(L-j)}(1 - a^{(L-j)})$

$\Delta^{(L-j)} \leftarrow \Delta^{(L-j)} + \delta^{(L-j+1)}(a^{(L-j)})^T$

$dW = \frac{1}{m}\Delta, \; W \leftarrow W - \alpha \cdot dW.$

Implementing Backpropagation

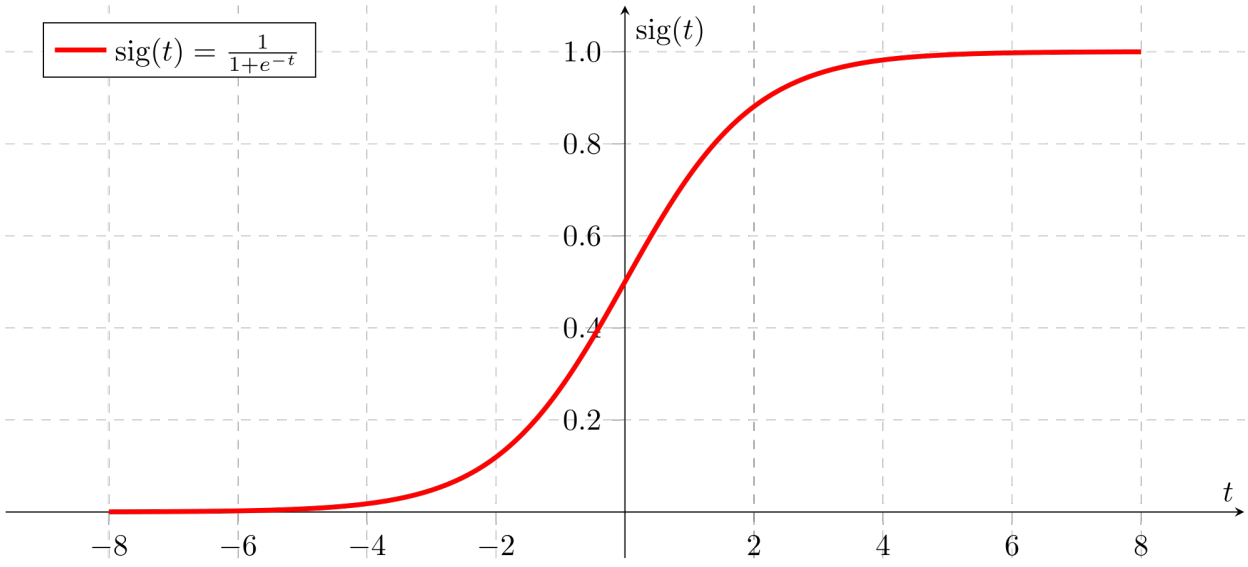
- You will derive and implement it for a fully connected layer, "relu layer" and softmax loss.
- Important to check that it is correct.

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + \mathbf{e}_i\epsilon) - f(\mathbf{x} - \mathbf{e}_i\epsilon)}{2\epsilon}$$

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

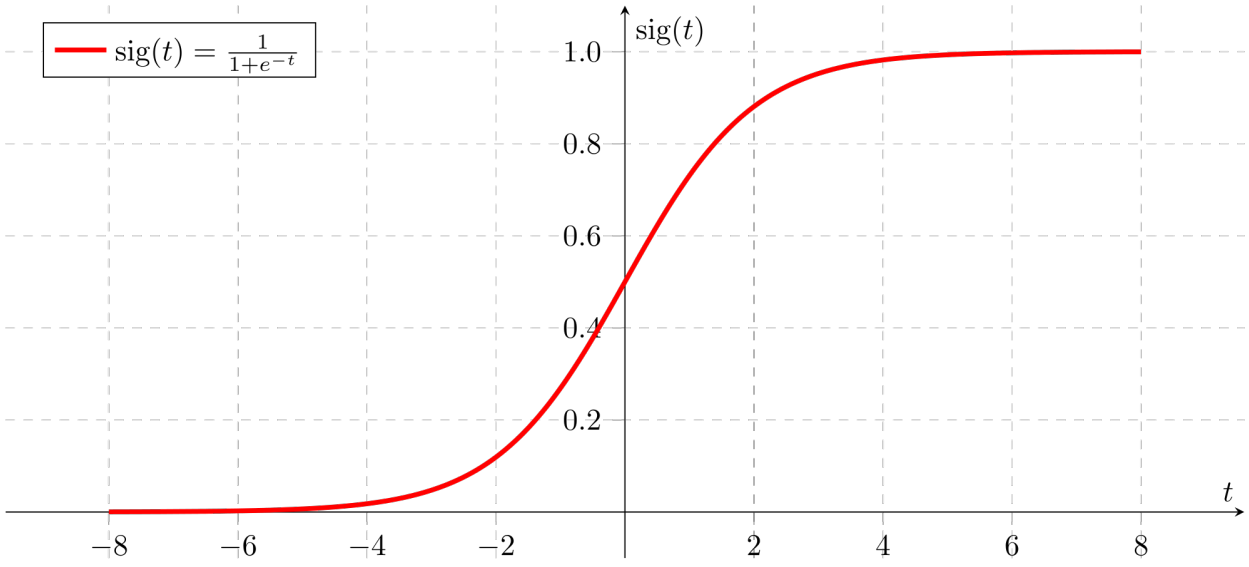
Activation functions

- Note that for high and low numbers, the derivative of the sigmoid function is very small. This could lead to convergence problems.



Activation functions

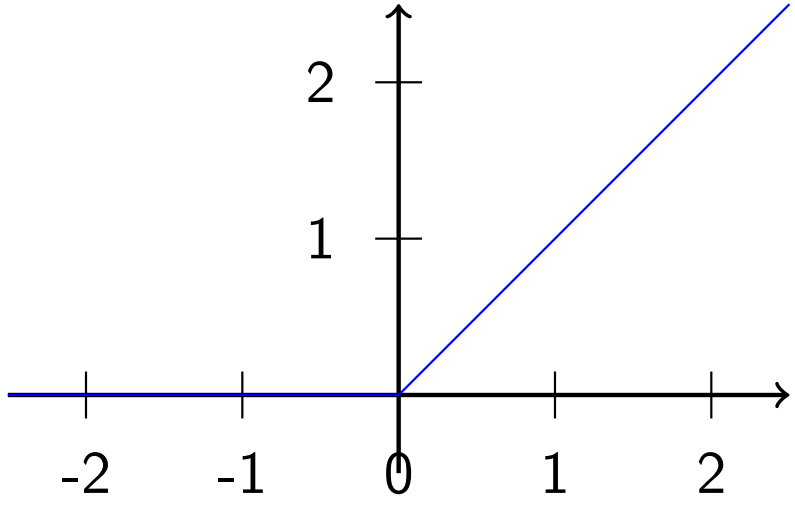
- Note that for high and low numbers, the derivative of the sigmoid function is very small. This could lead to convergence problems.
- We can use activation functions different from the sigmoid function.
- In practice, the following choices have shown to behave quite well.



Activation functions

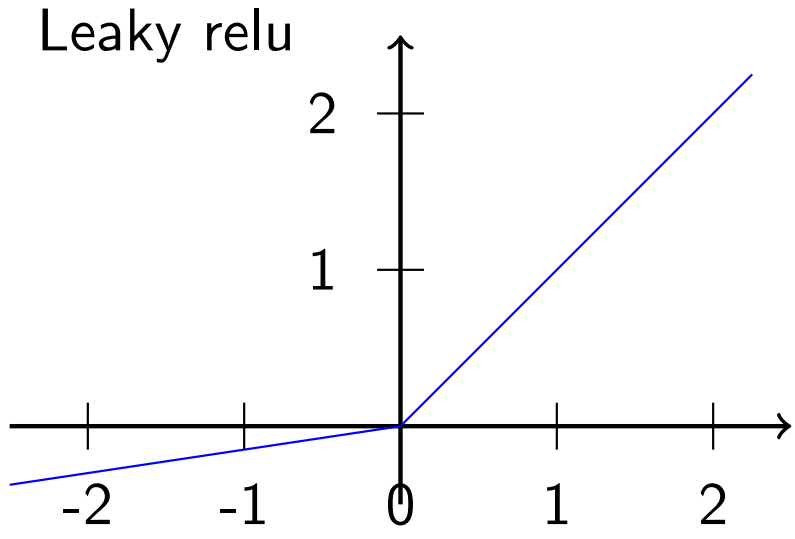
- Note that for high and low numbers, the derivative of the sigmoid function is very small. This could lead to convergence problems.
- We can use activation functions different from the sigmoid function.
- In practice, the following choices have shown to behave quite well.

Rectified linear units (relu)



Activation functions

- Note that for high and low numbers, the derivative of the sigmoid function is very small. This could lead to convergence problems.
- We can use activation functions different from the sigmoid function.
- In practice, the following choices have shown to behave quite well.



Non-linearities

- The network should be highly non-linear as a function from input to output.
- Use layers that apply some non-linear function element-wise.
- Relu: $y = max(0, x)$
- Sigmoid: $y = \sigma(x) = \frac{1}{1+e^{-x}}$, range $[0, 1]$
- Tanh: $y = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, range $[-1, 1]$
- Use relu in the project!
- For deep nets, use relu. For recurrent nets, use a clever combination of sigmoids and tanhs.

Regularization

- Since NNs tend to overfit, in practice, we often use a regularizer for the parameters. For example, the function to minimize is chosen as

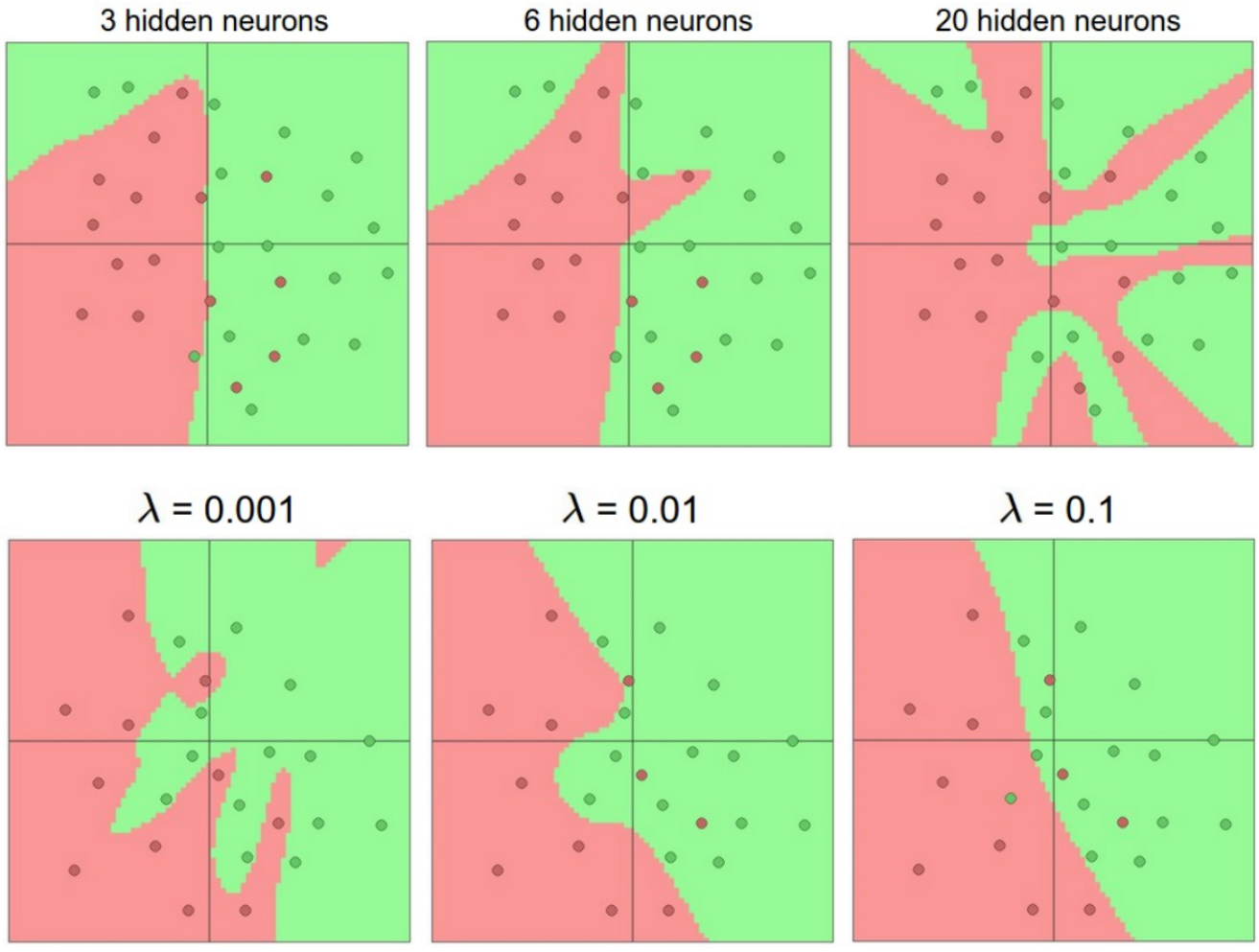
$$\frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}^{(i)}, y^{(i)}; W) + \frac{\lambda}{2}\|W\|^2$$

- where λ is the weight decay parameter. By using that $\frac{\partial}{\partial W}\left(\frac{\lambda}{2}\|W\|^2\right) = \lambda W$, gradient descent is changed to

$$W \leftarrow W - \alpha\left(\frac{\partial L}{\partial W} + \lambda W\right)$$

- and gradient descent with momentum is changed to

$$\mathbf{m}_n = \mu\mathbf{m}_{n-1} + (1-\mu)\frac{\partial L}{\partial \boldsymbol{\theta}}$$
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha(\mathbf{m}_n + \lambda\boldsymbol{\theta}_n)$$



http://cs231n.github.io/neural-networks-1/

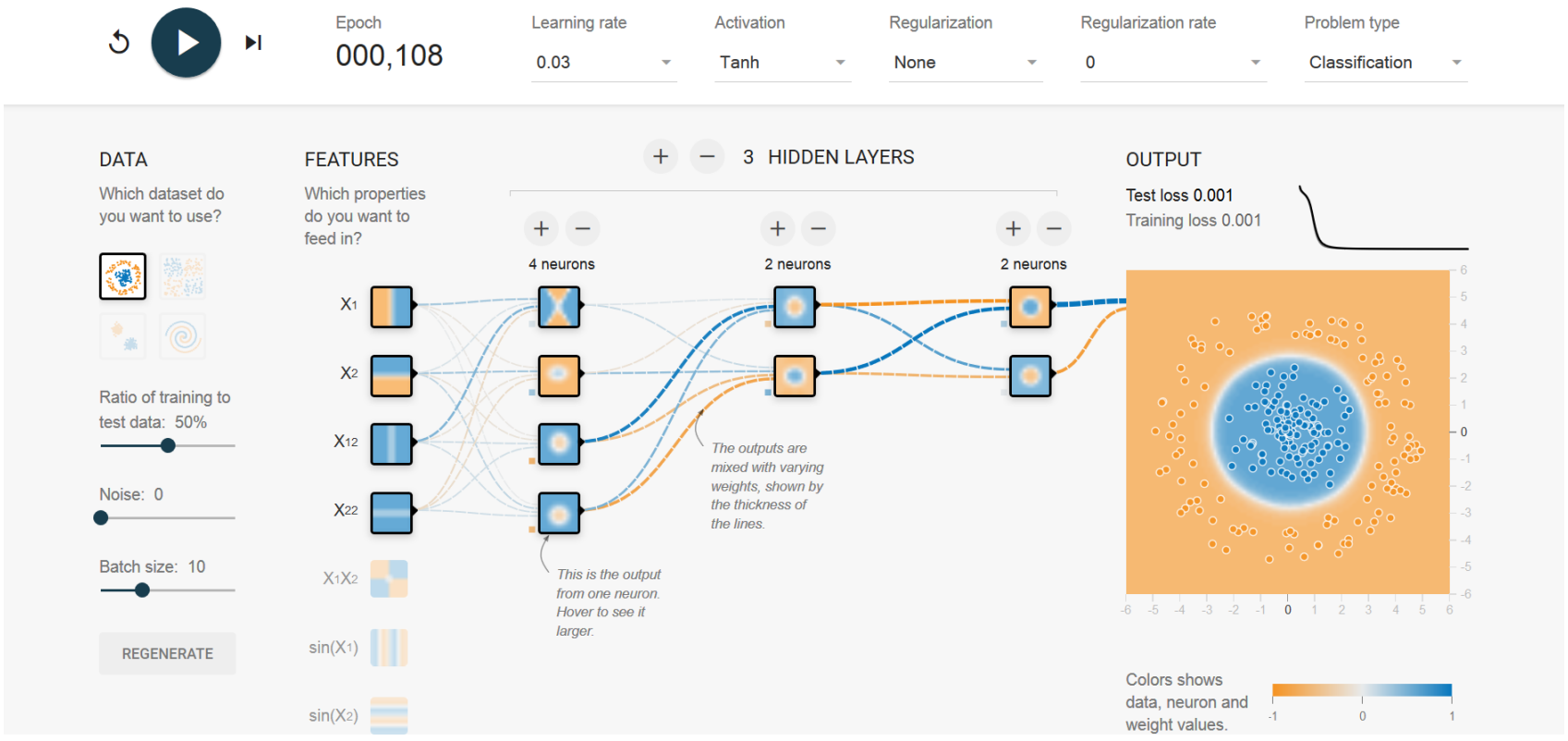
Practical Training Tips

- Is the test accuracy much less than the training accuracy? Increase weight decay.
- Tune the learning rate by changing it by a factor 2 or $1/2$ depending on the behavior of the loss.
- More layers increase the model capacity, but beware of overfitting.
- Initialize parameters randomly. A common way is to count how many ingoing connections n_{in} there are and then initialize the parameters by sampling random values from a Gaussian with mean 0 and standard deviation $\sqrt{\frac{2}{n_{in}}}$.
- Do not start training from scratch every time. Reuse the model using `save` and `load` in Matlab.
- Decrease the learning rate and call `training` again with new settings if the loss and accuracy plateaus.

Training with a GPU

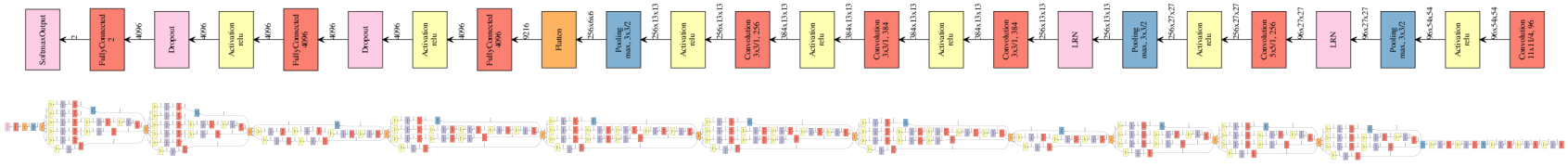
Demo

playground.tensorflow.org



Deep neural networks

- We can stack many layers for big problems. They consist of 10 layers or more. We speak of **deep learning**, which has become one of the most dominant buzz-words of the field.
- Some networks contain hundreds of layers and millions of weight parameters.



http://josephpcohen.com/w/visualizing-cnn-architectures-side-by-side-with-mxnet/

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Why are neural networks powerful?

- Use stochastic gradient descent with small batch sizes and the backpropagation algorithm

Why are neural networks powerful?

- Use stochastic gradient descent with small batch sizes and the backpropagation algorithm
- Then NNs can process a lot of data in considerably little time.

Why are neural networks powerful?

- Use stochastic gradient descent with small batch sizes and the backpropagation algorithm
- Then NNs can process a lot of data in considerably little time.
- NNs can take advantage in the era of "Big Data" and its massive amounts of data

Why are neural networks powerful?

Theorem: Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous, non-constant, bounded and monotonically increasing function. Let $K \subseteq \mathbb{R}^m$ be a compact subset, and let $\mathrm{C}(K)$ denote the set of continuous functions from K into $\mathbb{R}$.

Why are neural networks powerful?

Theorem: Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous, non-constant, bounded and monotonically increasing function. Let $K \subseteq \mathbb{R}^m$ be a compact subset, and let $\mathrm{C}(K)$ denote the set of continuous functions from K into $\mathbb{R}$.

Then, for any given $g \in \mathrm{C}(K)$ and any $\epsilon > 0$, there exists some natural number N and a hidden layer with N many nodes and coefficient matrices $W_1 \in \mathbb{R}^{m \times N}$, $W_2 \in \mathbb{R}^{N \times 1}$ and bias vector $b \in \mathbb{R}^N$ such that

$$f(x) = W_2 \cdot \sigma(W_1 x + b)$$

approximates g uniformly up to ϵ, i.e.,
$||f - g||_\infty = \max_{x \in K} |f(x) - g(x)| \leq \epsilon$.

Why are neural networks powerful?

Theorem: Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous, non-constant, bounded and monotonically increasing function. Let $K \subseteq \mathbb{R}^m$ be a compact subset, and let $C(K)$ denote the set of continuous functions from K into $\mathbb{R}$.

Then, for any given $g \in C(K)$ and any $\epsilon > 0$, there exists some natural number N and a hidden layer with N many nodes and coefficient matrices $W_1 \in \mathbb{R}^{m \times N}$, $W_2 \in \mathbb{R}^{N \times 1}$ and bias vector $b \in \mathbb{R}^N$ such that

$$f(x) = W_2 \cdot \sigma(W_1 x + b)$$

approximates g uniformly up to ϵ, i.e.,
$||f - g||_\infty = \max_{x \in K} |f(x) - g(x)| \leq \epsilon$.

In other words, any function can be approximated on a compact set arbitrarily well by a neural network with only one hidden layer, if the hidden layer is deep enough.

Why are neural networks powerful?

- The required number of neurons in the hidden layer depends on the function and the value of ϵ.
- A network with fixed layer sizes can only model a subspace of all continuous functions with fixed dimensionality given by the number of parameters.
- The vector space of all continuous functions has infinite dimensions. Hence, arbitrarily large hidden layer sizes are needed.
- In practice, deep shallow networks work better than narrow wide networks.

Why are neural networks powerful?

- Before the era of deep learning, features/representations were manually constructed.

Why are neural networks powerful?

- Before the era of deep learning, features/representations were manually constructed.
- A central concept of deep learning is that neural networks automatically learn good representations.

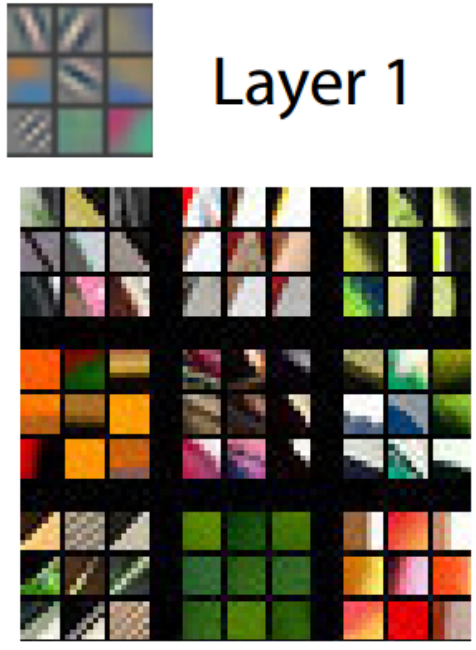
Why are neural networks powerful?

- Before the era of deep learning, features/representations were manually constructed.
- A central concept of deep learning is that neural networks automatically learn good representations.
- Lower layers extract basic features (like edge detectors for images), while higher layers compose them to complex features (like space invariant object detectors)

Why are neural networks powerful?

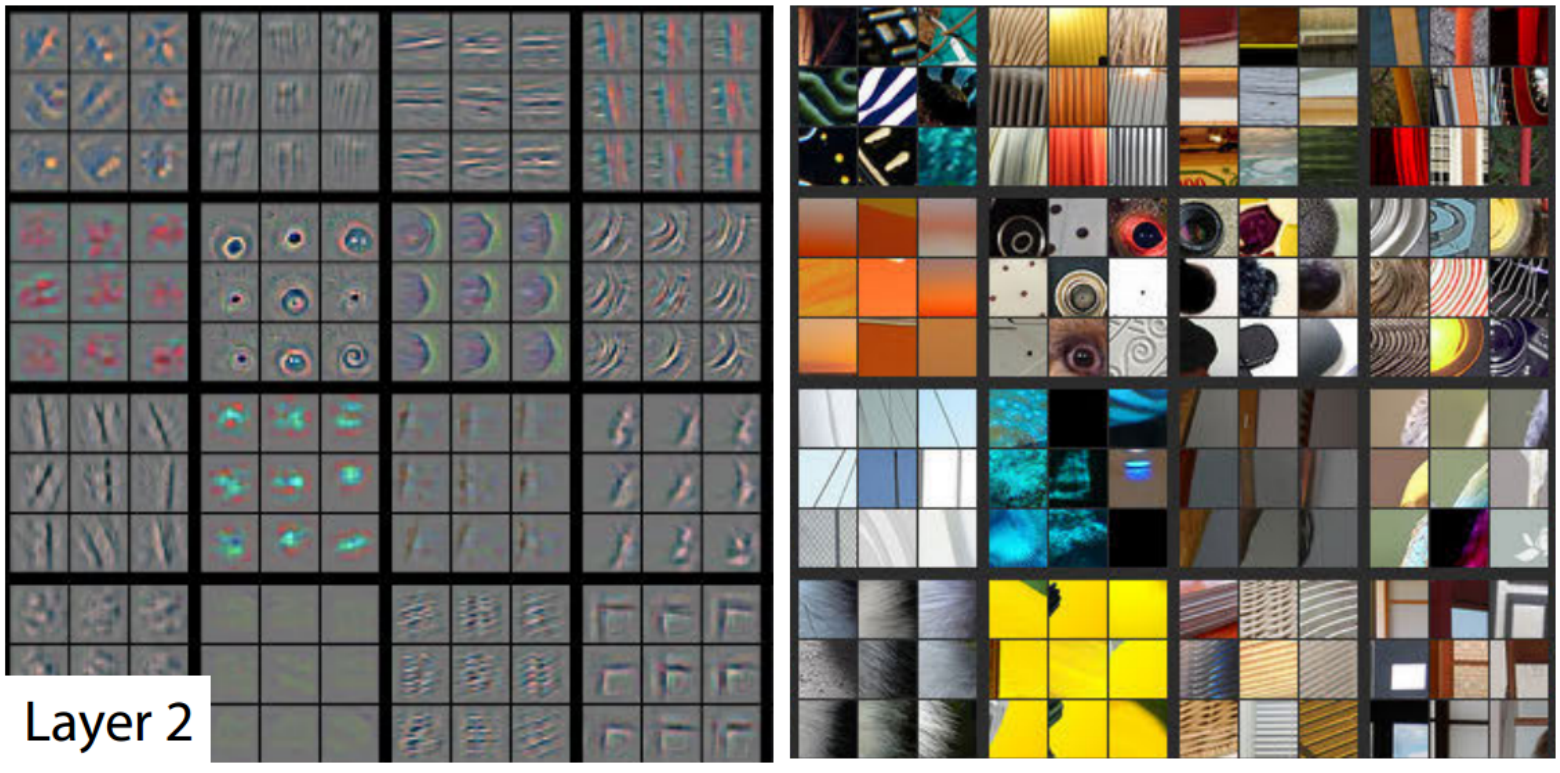
- Before the era of deep learning, features/representations were manually constructed.
- A central concept of deep learning is that neural networks automatically learn good representations.
- Lower layers extract basic features (like edge detectors for images), while higher layers compose them to complex features (like space invariant object detectors)
- This is in rough correspondence with our understanding of how the visual cortex processes images.

Automated feature learning



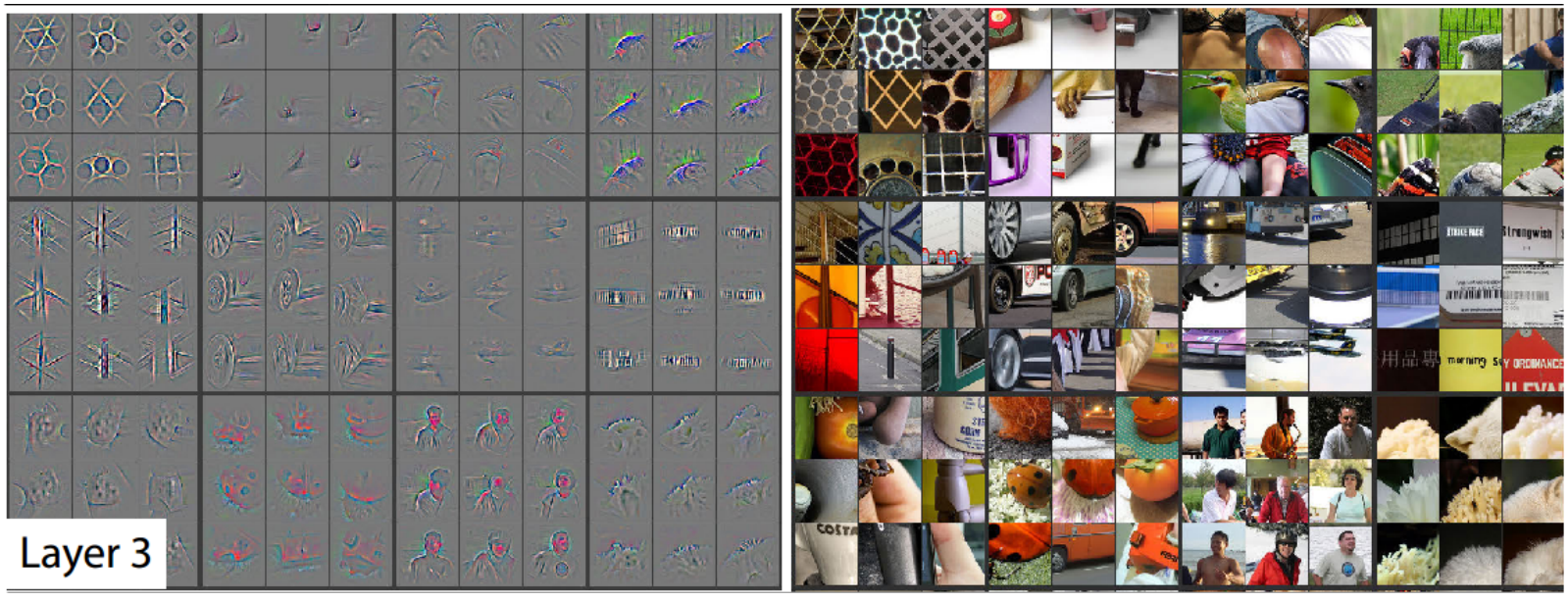
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Automated feature learning



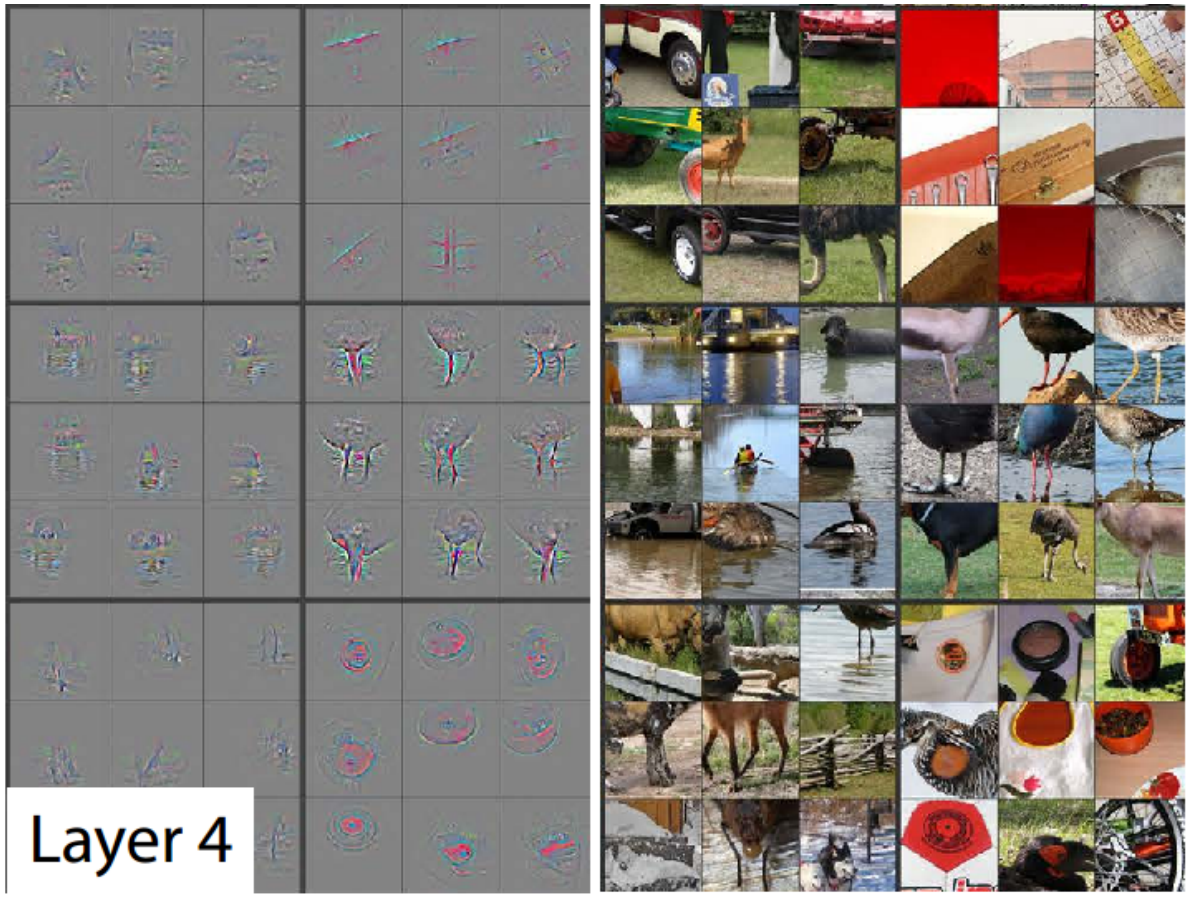
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Automated feature learning



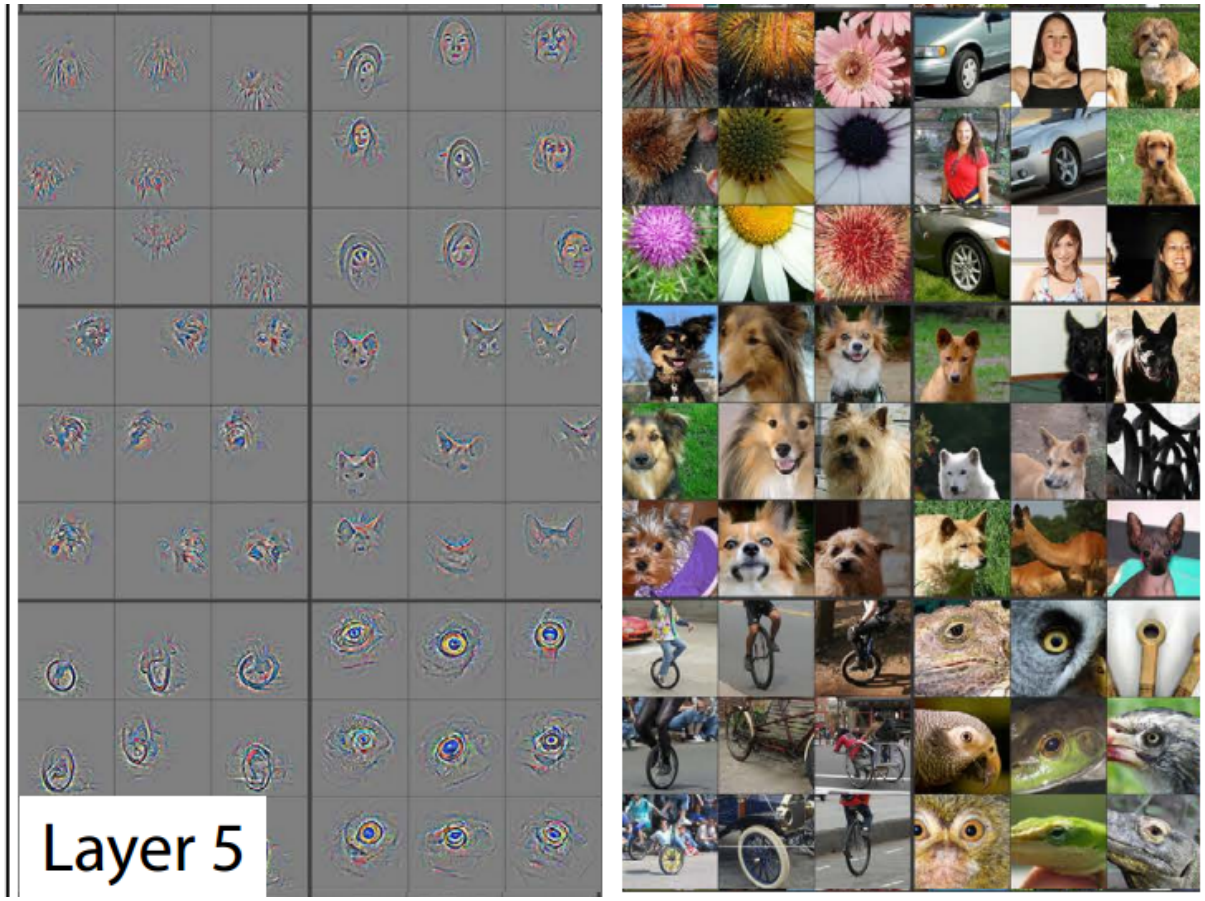
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Automated feature learning



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Automated feature learning



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

- Introduction
- Neural networks
  - The biological motivation
  - The model
- How to train a neural network
  - Loss functions
  - Gradient descent
  - Stochastic gradient descent
  - Choice of step size
  - Backpropagation
- Modeling choices
  - Activation functions
  - Regularization
  - Making NNs "deep"
- Why are neural networks so powerful?
- Limitations of NNs

Limitations of NN - local minima

- The model class of neural networks can approximate any function on compact sets.
- But there is no way to make full use out of the universal approximation property. Since the neural network function is non-convex, the learning algorithm will usually not find the globally best model, but only a locally optimal one.

Limitations of NN - overfitting

- If they are deep enough, neural networks can perfectly fit noisy data. Their learning rule actively minimizes the emprical loss.
- Thus, neural networks are endangered to overfit.
- We can try to avoid overfitting by using regularization or by "early stopping". In the latter case, we do not train to convergence, but monitor the performance on a separate validation step and stop training when the error on the validation set starts to increase.

Limitations of NN-Black box

- However, despite the huge success of deep learning, our understanding of how higher level features are composed in deep architectures is very limited. In safety-sensitive applications this can be a major issue.
- This issue is sometimes called the "black box of neural networks".

Outlook

- A powerful alternative to fully connected layers are **convolutional layers**, which are in particular a good modeling choice for image processing.
- To process sequential input data, we study a neural network structure called **Recurrent Neural Networks**.