*** Applied Machine Learning Fundamentals ***
Neural Networks / Deep Learning

Daniel Wehner, M.Sc.

SAP SE / DHBW Mannheim

Winter term 2022/2023

Find all slides on GitHub (DaWe1992/Applied_ML_Fundamentals)

Lecture Overview

Unit I	Machine Learning Introduction
Unit II	Mathematical Foundations
Unit III	Bayesian Decision Theory
Unit IV	Probability Density Estimation
Unit V	Regression
Unit VI	Classification I
Unit VII	Evaluation
Unit VIII	**Classification II**
Unit IX	Clustering
Unit X	Dimensionality Reduction

Agenda for this Unit

Section:

Introduction

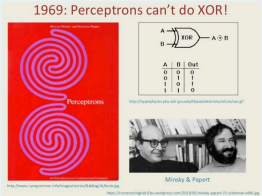
What is Deep Learning?

- 'Deep Learning' is a fancy new term for 'artificial neural networks'
- It is a **supervised** method and **model based**
- Artificial neural networks are inspired by the human brain
- Lots of different architectures exist:
  - **Multi-Layer perceptrons (MLPs)**
  - **Convolutional neural networks (CNNs, ConvNets)**
  - **Recurrent neural networks (LSTMs, GRUs, etc.)**
  - ...and many more...

History of Deep Learning

Early booming (1950s – early 1960s)

F. Rosenblatt suggests the **Perceptron** learning algorithm: `Click here!`



Setback I (mid 1960s – late 1970s)

M. Minsky and *S. Papert* (1969): Serious problems with perceptron algorithm: It cannot learn the **XOR problem**.

History of Deep Learning (Ctd.)

Renewed enthusiasm (1980s)

- New techniques available
- **Backpropagation** for deep nets

Setback II (1990s – mid 2000s)

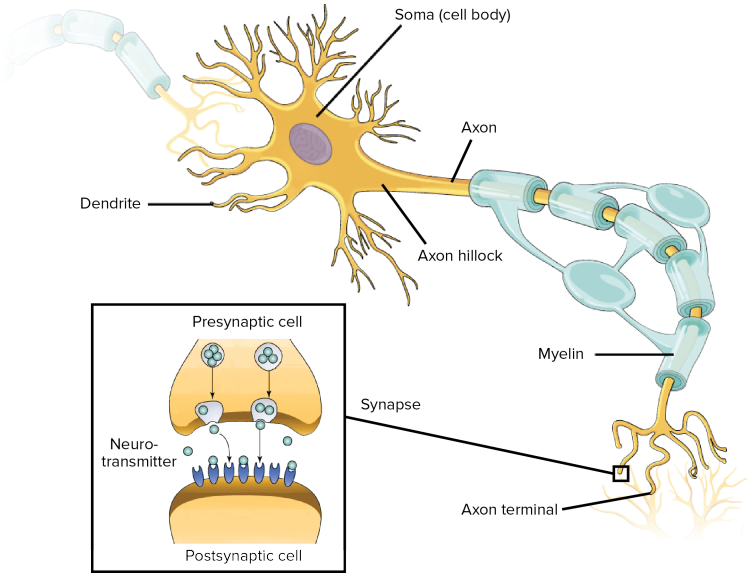
- Other techniques were considered superior (e. g. SVMs)
- CS journals rejected papers on neural networks

'Deep Learning' (since mid 2000)

More data, faster computers, better optimization techniques...

Biological Motivation

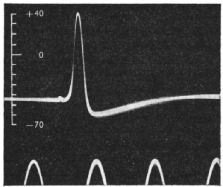
- All neurons are connected and form a complex **network**
- **Transmitter chemicals** within the fluid of the brain influence the **electrical potential** inside the body of the neurons
- If the **membrane potential** reaches some threshold, the neurons **fires**
  $\Rightarrow$ A pulse of fixed length is sent down the **axon**
- The axon connects the neuron with other neurons (via **synapses**)
- Probably there are 100 trillion (!!!) synapses in the human brain
- **Refractory period** after a neuron has fired


Soma (cell body)
Axon
Dendrite
Axon hillock
Myelin
Presynaptic cell
Synapse
Neuro-
transmitter
Axon terminal
Postsynaptic cell

How can we know this?

- *Santiago Ramón y Cajal* made neurons visible by applying **Golgi's method**
- Golgi's method uses the **Golgi stain** to colorize the neurons

- End of the 1940s, *Hodgkin* and *Huxley* started investigating the electrical properties of neurons on the squid's axon
- The right-hand-side image was the first **action potential** ever plotted



How do Humans / Animals learn?

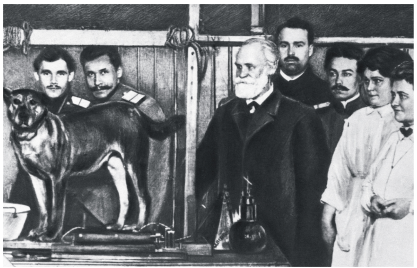
- **Idea:** Mechanism of learning is **association**
- **Hebbian learning:** If the firing of one neuron repeatedly assists in firing another neuron, their synaptic connection will be strengthened

*'When an axon of cell A is near enough to excite a cell B and repeatedly or persistently **takes part in firing it**, some growth process or **metabolic change** takes place in one or both cells such that A's **efficiency**, as one of the cells firing B, is **increased**.'*

*'The general idea is an old one, that any two cells or systems of cells that are **repeatedly active at the same** time will tend to become **'associated'**, so that activity in one facilitates activity in the other.'* *Hebb*

Classical / Pavlovian Conditioning

- Dog salivates when given food
- Food is an **unconditioned stimulus (US)**
- Salivation in response to food is **unconditioned response (UR)**
- Food is paired with the sound of a bell
- Bell is **conditioned stimulus (CS)**
- Bell will eventually elicit salivation event without food
- Salivation is **conditioned response (CR)**

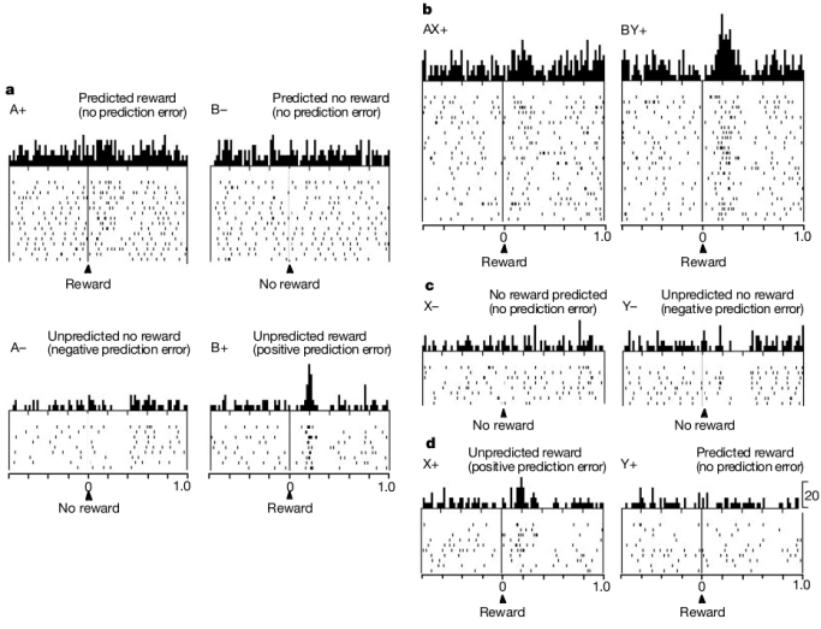


Classical / Pavlovian Conditioning (Ctd.)

Blocking

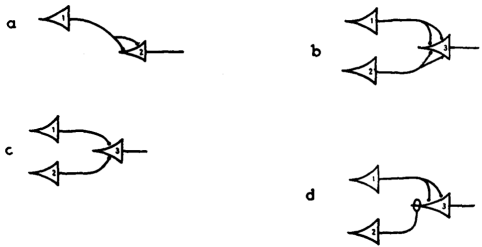
Group A	train N+	train LN+	test L-	$\Rightarrow$ no conditioning
Group B		train LN+	test L-	$\Rightarrow$ conditioning

- CS is a light (L), a noise (N), or a combination of both (LN)
- US is a mild shock that is paired with the CS in the training phase (+)
- Fear response is tested after training when only L is presented without shock (-)
- Group B shows conditioning; Group A does not: **N blocks L**
- This is hard to explain with Hebbian learning
- **Idea:** Learning only happens, if there is a **prediction error**


a
A+
Predicted reward
(no prediction error)
Reward
B–
Predicted no reward
(no prediction error)
No reward
A–
Unpredicted no reward
(negative prediction error)
0
1.0
No reward
B+
Unpredicted reward
(positive prediction error)
0
1.0
Reward
b
AX+
0
1.0
Reward
BY+
0
1.0
Reward
c
X–
No reward predicted
(no prediction error)
No reward
Y–
Unpredicted no reward
(negative prediction error)
No reward
d
X+
Unpredicted reward
(positive prediction error)
0
1.0
Reward
Y+
Predicted reward
(no prediction error)
20
0
1.0
Reward

Artificial Neurons [*McCulloch* and *Pitts*, 1943]

- In 1943, *McCulloch* and *Pitts* designed the first 'artificial neuron'
- These neurons can represent logical functions (e. g. b: `OR`, c: `AND`)



Section:

Perceptrons

Original Perceptron Algorithm [*Rosenblatt*, 1957]

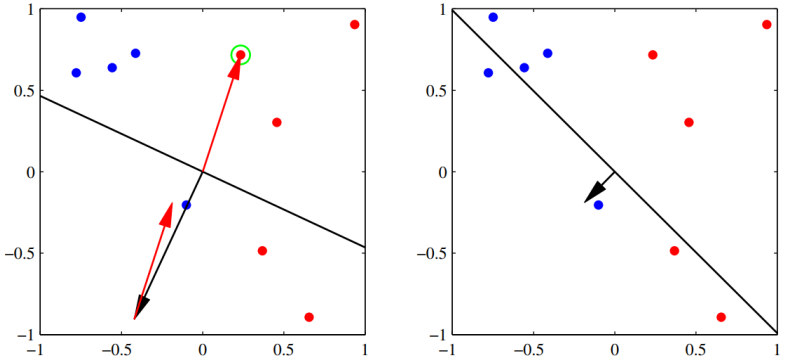
1. Initialize parameters $\boldsymbol{\theta} = \{\boldsymbol{w}, b\}$
2. $\forall \{\boldsymbol{x}^{(i)} \in \boldsymbol{X}, y^{(i)} \in \{-1, +1\}\}_{i=1}^{n}$ (until convergence):
   - 2a) If $\boldsymbol{x}^{(i)}$ is correctly classified, do nothing
   - 2b) Else if $y^{(i)} = +1$, update the paramters with:

$$\boldsymbol{w} \longleftarrow \boldsymbol{w} + \boldsymbol{x}^{(i)} \qquad b \longleftarrow b + 1$$

   - 2c) Else if $y^{(i)} = -1$, update the paramters with:

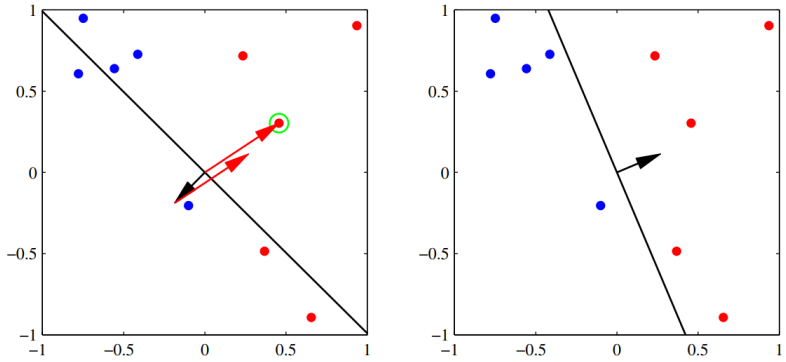
$$\boldsymbol{w} \longleftarrow \boldsymbol{w} - \boldsymbol{x}^{(i)} \qquad b \longleftarrow b - 1$$

Original Perceptron Algorithm (Ctd.)



cf. [2], p. 195

Original Perceptron Algorithm (Ctd.)



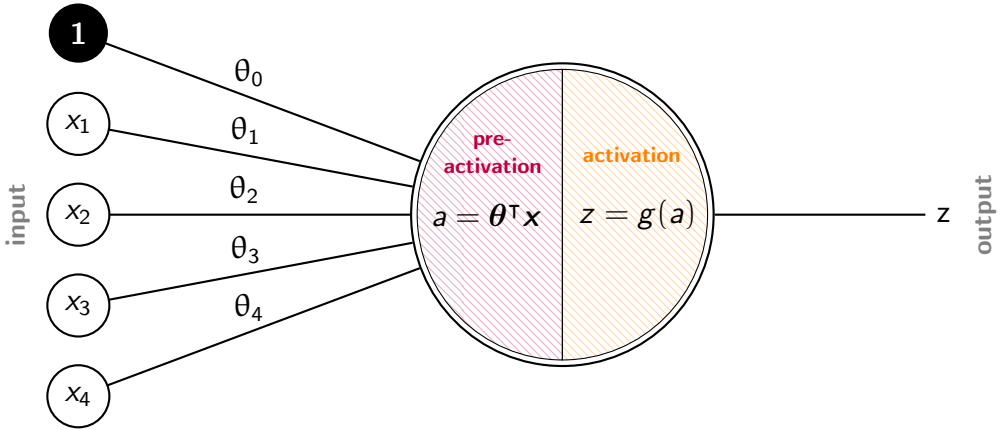
cf. [2], p. 195

Perceptron Convergence Theorem

Perceptron Convergence Theorem

If the training data is **linearly separable**, then the perceptron learning algorithm is going to **converge after a finite amount of time** and classifies **all training data examples correctly**.

The Architecture of a Neuron



Perceptron

- The neuron receives an input vector $\boldsymbol{x}$:

$$\boldsymbol{x} = (1, x_1, x_2, \ldots, x_m)^\intercal$$

- Each input signal is weighted by a factor θ_j: *(weight of synaptic strength)*

$$\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_m)^\intercal$$

- We compute the **pre-activation** and the **activation**:

$$a = \boldsymbol{\theta}^\intercal \boldsymbol{x} = \sum_{j=0}^{m} \theta_j x_j \qquad z = g(a) \qquad (1)$$

Perceptron (Ctd.)

- The simplest activation function is to use a threshold[1] ρ:

$$g_\rho(a) = \begin{cases} 0 & \text{if } a \leqslant \rho \\ 1 & \text{if } a > \rho \end{cases}$$

- Quick example: $\boldsymbol{x} = (1, 0, 0.5)^\intercal \qquad \boldsymbol{\theta} = (1, -0.5, -1)^\intercal \qquad \rho = 0$

$$a = \boldsymbol{\theta}^\intercal \boldsymbol{x} = 1 \cdot 1 + (-0.5) \cdot 0 + (-1) \cdot 0.5 = 0.5$$
$$z = g_{\rho=0}(0.5) = 1$$

[1] Not used, since not differentiable; alternatives later

Perceptron Learning

- Learning means choosing the correct weights $\boldsymbol{\theta}^*$ from a set of possible hypotheses $\mathcal{H}$ (**hypothesis space**):

$$\mathcal{H} = \{\boldsymbol{\theta} | \boldsymbol{\theta} \in \mathbb{R}^{m+1}\}$$

- How to learn the weights from a data set $\mathcal{D}$?
- **Algorithm outline:**
  1. Pick a training example $\boldsymbol{x} \in \mathcal{D}$
  2. Calculate the activation z for that training example
  3. Update the weights $\boldsymbol{\theta}$ based on the error

Perceptron Learning (Ctd.)

- How can we compute the error? $\Rightarrow$ We need a loss function $\mathcal{J}(\boldsymbol{\theta})$:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2 \tag{2}$$

- Again, we use **gradient descent**: Compute gradient and go into the negative direction of the gradient:

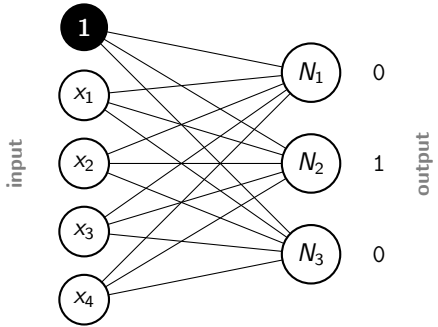
$$\boldsymbol{\theta}^{(t+1)} \longleftarrow \boldsymbol{\theta}^{(t)} - \alpha\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}) \tag{3}$$

$\Rightarrow$ cf. slides 'Regression'

Generalization to multiple Classes

- A single neuron can only distinguish two classes
- If there are more than two classes: Simply use more neurons[2]
- Use **one-hot encoding** for the classes and **softmax** as activation function (later)
- Example for three classes:

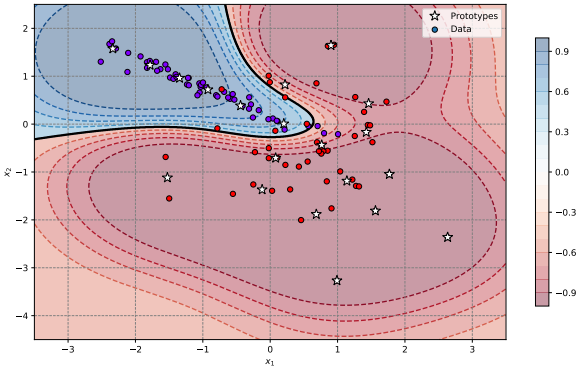
$\mathcal{C}_1$	1 0 0
$\mathcal{C}_2$	0 1 0
$\mathcal{C}_3$	0 0 1



[2]This construct is still referred to as a perceptron.

What about non-linear Data Sets?

- Perceptrons cannot learn non-linear boundaries
- Remember *Minsky* and *Papert*
- What can we do?
  1. Add feature mapping (cf. right)
  2. Add hidden layers **(Multi-Layer Perceptrons)**



Section:

Multi-Layer-Perceptrons (MLPs)

Multi-Layer Perceptrons

- In theory, a **Multi-Layer Perceptron (MLP)** can approximate any continuous function arbitrarily well
- An MLP with λ layers is a function $h : \mathbb{R}^m \to \mathbb{R}^\kappa$, parameterized by network parameters $\boldsymbol{\Theta}^{[1]}, \boldsymbol{\Theta}^{[2]}, \ldots, \boldsymbol{\Theta}^{[\lambda]}$
- In each layer, a non-linearity is applied: $g^{[1]}, g^{[2]}, \ldots, g^{[\lambda]}$

$$\boldsymbol{z}^{[1]}(\boldsymbol{x}) = g^{[1]}(\boldsymbol{\Theta}^{[1]}\boldsymbol{x})$$
$$\ldots$$
$$\boldsymbol{z}^{[\lambda]}(\boldsymbol{x}) = g^{[\lambda]}(\boldsymbol{\Theta}^{[\lambda]}\boldsymbol{z}^{[\lambda-1]}(\boldsymbol{x}))$$

Prediction:

$$y_p = \arg\max_k \boldsymbol{z}^{[\lambda]}(\boldsymbol{x})$$

MLP Learning

1. Start by randomly initializing the network weights
2. **Do not set an initial value of 0 for all weights! (Why?)**
   $\Rightarrow$ Initialize to small random numbers
3. Perform a forward pass through the network, i. e. make predictions
4. Using the predictions, compute a scalar loss value $\mathcal{J}(\boldsymbol{\Theta})$
5. Calculate the gradients of the loss w. r. t. each network parameter and update all parameters (gradient descent)

Backpropagation

- In order to update the weights, we first have to perform a **forward pass**:

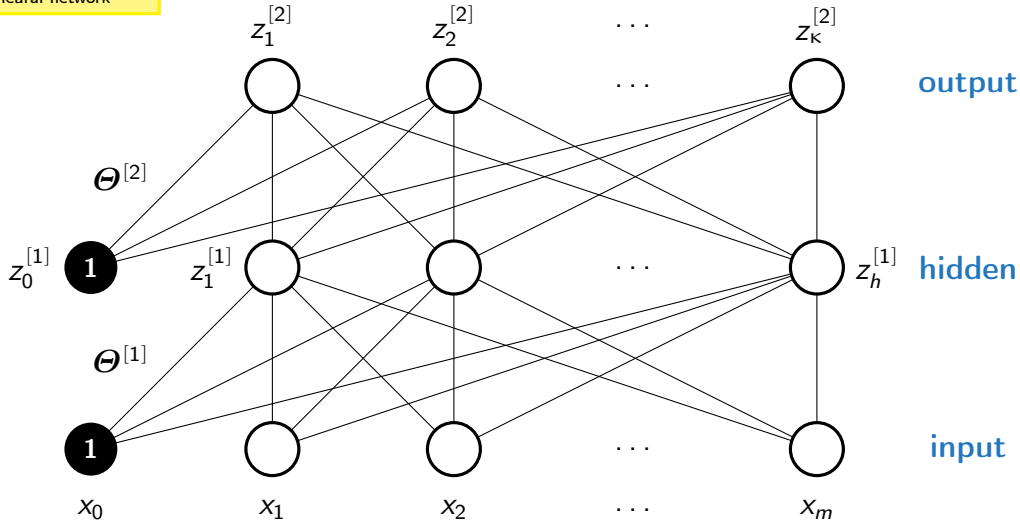
Let's consider $\lambda = 2$

$$z_k^{[2]}(\boldsymbol{x}) = g^{[2]}\left(\sum_{l=0}^{h} \Theta_{kl}^{[2]} g^{[1]}\left(\sum_{j=0}^{m} \Theta_{lj}^{[1]} x_j^{(i)}\right)\right)$$

$$z_l^{[1]}(\boldsymbol{x}) = g^{[1]}\left(\sum_{j=0}^{m} \Theta_{lj}^{[1]} x_j^{(i)}\right) \qquad \text{hidden activation}$$

- $g(\cdot) \equiv$ activation function, e. g. sigmoid, tanh, ReLU
- $\boldsymbol{\Theta}$ are the network parameters (to be learned)

This is a fully connected neural network



Backpropagation (Ctd.)

- Compute the network loss
- The loss function is given by: (assume square loss: $\ell = (z_k^{[2]}(\boldsymbol{x}^{(i)}) - y_k^{(i)})^2$)

$$\mathcal{J}(\boldsymbol{\Theta}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{J}^{(i)}(\boldsymbol{\Theta}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{\kappa}\ell(z_k^{[2]}(\boldsymbol{x}^{(i)}), y_k^{(i)})$$

- Compute the error gradient for the **output layer** w. r. t. $z_k^{[2]}$:

$$\frac{\partial\mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_k^{[2]}(\boldsymbol{x}^{(i)})} = \ell'(z_k^{[2]}(\boldsymbol{x}^{(i)}), y_k^{(i)}) \equiv \delta_k^{(i)}$$

Backpropagation (Ctd.)

- Compute the weight gradient for the **output layer**:

$$\begin{aligned}\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial \Theta_{kl}^{[2]}} &= \frac{\mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_k^{[2]}(\boldsymbol{x}^{(i)})} \frac{\partial z_k^{[2]}(\boldsymbol{x}^{(i)})}{\partial \Theta_{kl}^{[2]}} \\ &= \ell'(z_k^{[2]}(\boldsymbol{x}^{(i)}), y_k^{(i)}) \cdot g'^{[2]} \left(\sum_{t=0}^{h} \Theta_{kt}^{[2]} z_t^{[1]}(\boldsymbol{x}^{(i)}) \right) \cdot z_l^{[1]}(\boldsymbol{x}^{(i)}) \\ &= \delta_k^{(i)} \cdot g'^{[2]} \left(\sum_{t=0}^{h} \Theta_{kt}^{[2]} z_t^{[1]}(\boldsymbol{x}^{(i)}) \right) \cdot z_l^{[1]}(\boldsymbol{x}^{(i)})\end{aligned}$$

Backpropagation (Ctd.)

- Compute the error gradient for the **hidden layer**:

$$\begin{aligned}\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_l^{[1]}} &= \sum_{k=1}^{K} \frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_k^{[2]}(\boldsymbol{x}^{(i)})} \frac{\partial z_k^{[2]}(\boldsymbol{x}^{(i)})}{\partial z_l^{[1]}} \\ &= \sum_{k=1}^{K} \ell'(z_k^{[2]}(\boldsymbol{x}^{(i)}), y_k^{(i)}) \cdot g'^{[2]} \left(\sum_{t=0}^{h} \Theta_{kt}^{[2]} z_t^{[1]}(\boldsymbol{x}^{(i)}) \right) \cdot \Theta_{kl}^{[2]} \\ &= \sum_{k=1}^{K} \delta_k^{(i)} \cdot g'^{[2]} \left(\sum_{t=0}^{h} \Theta_{kt}^{[2]} z_t^{[1]}(\boldsymbol{x}^{(i)}) \right) \cdot \Theta_{kl}^{[2]} \equiv \widehat{\delta}_l^{(i)}\end{aligned}$$

Backpropagation (Ctd.)

- Compute the weight gradient for the **hidden layer**:

$$\begin{aligned}\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial \Theta_{lj}^{[1]}} &= \frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_l^{[1]}} \cdot g'^{[1]}\left(\sum_{t=0}^{m} \Theta_{lt}^{[1]} x_t^{(i)}\right) \cdot x_j^{(i)} \\ &= \widehat{\delta}_l^{(i)} \cdot g'^{[1]}\left(\sum_{t=0}^{m} \Theta_{lt}^{[1]} x_t^{(i)}\right) \cdot x_j^{(i)}\end{aligned}$$

- The weight derivatives are used in the gradient descent update rule, e. g.:

$$\Theta_{lj}^{[1]} \longleftarrow \Theta_{lj}^{[1]} - \alpha[\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})]/[\partial \Theta_{lj}^{[1]}]$$

Backpropagation Example

See blackboard!

Some Remarks

- **Check your gradients:**

$$\nabla h(x) \overset{!}{\approx} \frac{1}{\varepsilon} \cdot (h(x+\varepsilon) - h(x))$$

- **Hyper-parameter optimization is necessary:**
  - # hidden layers
  - # hidden units
  - activation functions
  - learning rate
  - batch size
  - # training epochs
  - regularization
  - ...
- Have a look at: `https://playground.tensorflow.org`

Sigmoid Function

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

- Value range: 0 to 1
- **Problem 1:** Gradient can become very small **(vanishing gradient problem)**
- **Problem 2:** Output is not zero-centered (makes optimization harder)

Sigmoid Function (Ctd.)

'Convergence is usually faster if the average of each input variable over the training set is close to zero. To see this, consider the extreme case where all the inputs are positive. Weights to a particular node in the first weight layer are updated by an amount proportional to δx where δ is the (scalar) error at that node and x is the input vector. When all of the components of an input vector are positive, all of the updates of weights that feed into a node will have the same sign (i. e. sign(δ)). As a result, these weights can only all decrease or all increase together for a given input pattern. Thus, if a weight vector must change direction it can only do so by zigzagging which is inefficient and thus very slow.' - **Yann LeCun et al., Efficient BackProp, 1998** (`http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf`)

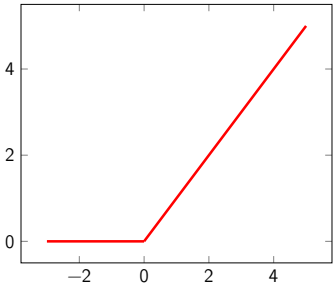
Tangent Hyperbolic (tanh)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad \tanh'(x) = 1 - \tanh(x)^2$$

- Value range: -1 to 1
- Zero-centered
- **Still suffers from the vanishing gradient problem**

Rectified Linear Unit (ReLU)

- $ReLU(x) = max(0, x)$
- $ReLU'(x) = \begin{cases} 0 & \text{if } x \leqslant 0 \\ 1 & \text{otherwise} \end{cases}$



ReLU (Ctd.)

- ReLU does not lead to vanishing gradients
- ReLU is very efficient to compute
- Use ReLU as the hidden layer activation function!
- **But:** Pay attention to the initialization of your parameters to avoid *'dying ReLUs'* (parameter settings where single neurons will always output 0)
- **Use batch-normalization!**

Softmax Activation

$$softmax_k(\boldsymbol{x}) = \frac{\exp\{x_k\}}{\sum_{j=1}^{K} \exp\{x_j\}}$$

- Softmax is a **global activation function** (i. e. it depends on the preactivations of the other units in the layer)
- It is used to squash the last layer's activations into a probability distribution, **such that the activations sum to 1**

Section:

Further Network Architectures and Word Embeddings

Convolutional Neural Networks (CNNs)

- A fully-connected MLP network has a lot of parameters, i. e. it might be **too complex** or **computationally inefficient** for some tasks
- The input position does not matter in all cases (e. g. word positions when classifying e-mails)
- An MLP always needs a fixed-size input vector, but **we might have variable-size input data** (e. g. images in different resolutions, text sequences of different lengths, etc.)
- **Solution:** Use **convolution**!

CNN: General Idea

'A convolutional neural network is designed to identify indicative local predictors in a large structure, and combine them to produce a fixed size vector representation of the structure, capturing these local aspects that are most informative for the prediction task at hand.' - **Yoav Goldberg**

- A CNN consists of multiple convolutional layers
- A convolutional layer consists of a filter and a non-linear activation function
- Pooling extracts the most important features **independent of their input position**

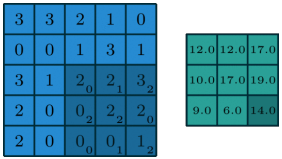
Convolution

$$(x \star g)[i] = \sum_{j=-w}^{w} x[i-j] \cdot g[j]$$

- $\star$ is the convolution operator
- x is the input to the convolutional layer
- i is the current position in the input
- w is the filter size / window size
- g is the convolution filter (kernel)

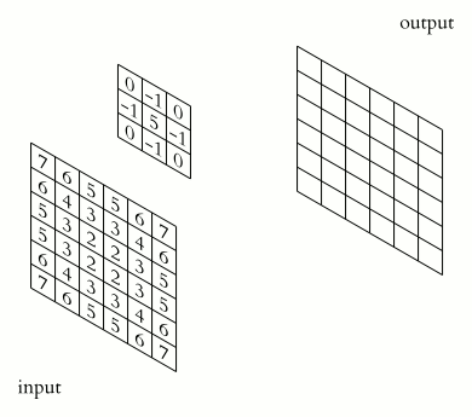
Convolution (Ctd.)

- We shift a filter across the input data (e. g. image pixels) with a certain stride and multiply the input with the filter weights at each position
- The resulting feature map is usually smaller in size



$$(x \star g)[i] = \sum_{j=-w}^{w} x[i-j] \cdot g[j]$$

Convolution Animation



Pooling

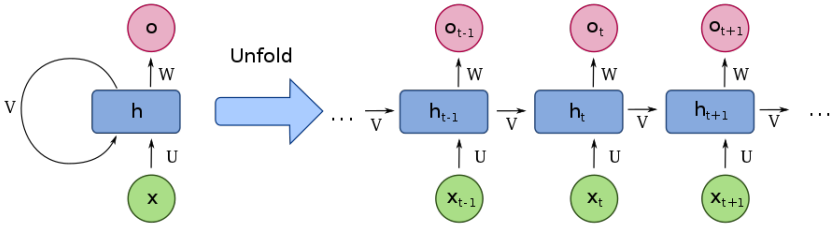
- The outputs of the convolutions are passed through non-linear activation functions (e. g. ReLU, tanh, ...)
- Pooling is applied to extract only the most important activations:
  - **max-pooling** (extracts the maximum activation)
  - **mean-pooling** (extracts the mean activation)
  - etc.
- **There are no parameters involved in the pooling operation**
- In image processing, pooling is necessary to reduce the dimensionality

Recurrent Neural Networks (RNNs)

- **Recurrent Neural Networks (RNNs)** can be used to process sequences
  - Sequence labeling (e. g. POS-tagging)
  - Sequence transduction (e. g. machine translation)
  - Sequence classification
- They are similar to feed-forward networks, but have a **recurrent loop in the computational graph**
- By unfolding the network, it can be trained exactly like a feed-forward MLP using backpropagation

RNN Architecture

$$\boldsymbol{h}_t = \sigma_h(\boldsymbol{U}\boldsymbol{x} + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h)$$
$$\boldsymbol{o}_t = \sigma_o(\boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b}_o)$$



RNN Extensions

- **Bi-directionality:**
  Run one RNN from left to right and another one from right to left and concatenate the hidden states $\overrightarrow{\boldsymbol{h}_t}$ and $\overleftarrow{\boldsymbol{h}_t}$
- **Gating:**
  Gated Recurrent Units (GRUs) and Long-Short Term Memory (LSTM) networks modify the recurrent unit to better control what is stored in the hidden state h and to prevent a vanishing gradient
- **Skip-connections** through time

Inside an LSTM [Hochreiter/Schmidhuber, 1997]

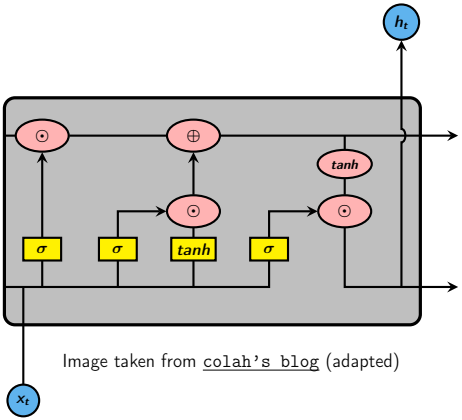


Image taken from colah's blog (adapted)

Forget gate

$$\boldsymbol{f}_t = \sigma(\boldsymbol{\Theta}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_f)$$

Input gate

$$\boldsymbol{i}_t = \sigma(\boldsymbol{\Theta}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_i)$$

$$\widetilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{\Theta}_C \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_C)$$

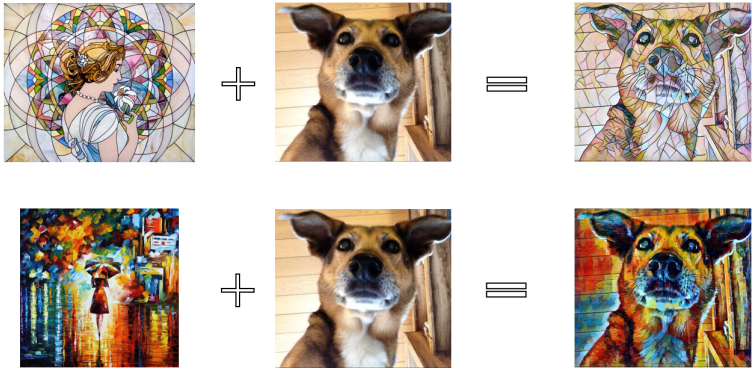
$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{C}}_t$$

Output gate

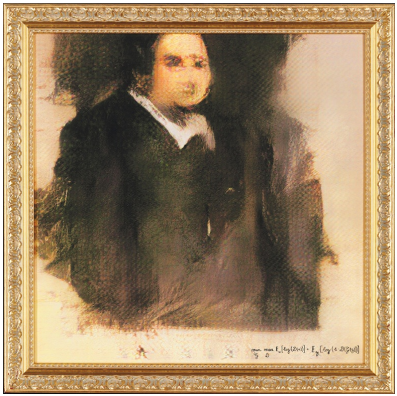
$$\boldsymbol{o}_t = \sigma(\boldsymbol{\Theta}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_o)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$$

Use Case: Neural artistic Style Transfer

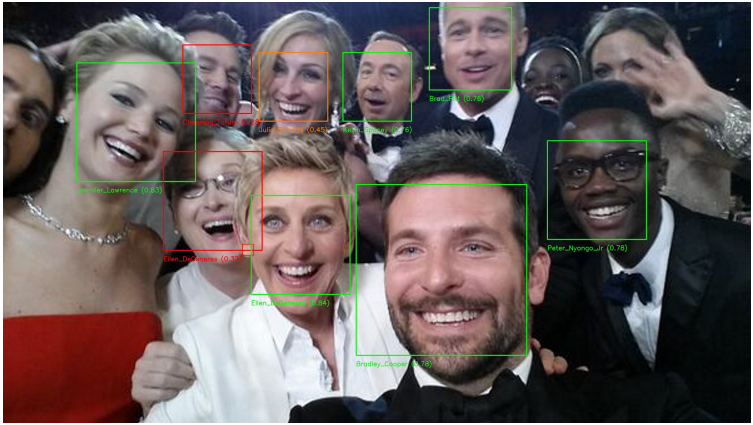


Use Case: Image Generation (GAN)



Use Case: Watermark Removal

Use Case: Face Recognition



Use Case: Meme Generation (seriously...)

Dank Learning: Generating Memes Using Deep Neural Networks

Abel L. Peirson V
Department of Physics
Stanford University
alpv95@stanford.edu

E. Meltem Tolunay
Department of Electrical Engineering
Stanford University
meltem.tolunay@stanford.edu

Why Vector Representations of natural Language?

- **Why?**
  - Most machine learning algorithms (e. g. neural nets) cannot handle text input
  - Therefore, a numeric representation has to be devised
- **How? – First ideas:**
  - Strawman idea: Represent each word $w \in \mathcal{V}$ as a one-hot vector
  - E. g. $\mathcal{V} = \{\text{this, great, be}\}$:

$$\boldsymbol{v}_{\text{this}} = [1, 0, 0]^\intercal \qquad \boldsymbol{v}_{\text{great}} = [0, 1, 0]^\intercal \qquad \boldsymbol{v}_{\text{be}} = [0, 0, 1]^\intercal$$

  - **Problems:** $\boldsymbol{v}_{w_i} \in \mathbb{R}^{|\mathcal{V}|}$ and semantics are not modeled

Distributional Hypothesis [Harris, 1954] and [Firth, 1957]

- Other approaches are based onto the **distributional hypothesis**:
  - [Harris, 1954]: Words are similar if they occur in similar contexts:

    'The fact that, for example, not every adjective occurs with every noun can be used as a measure of meaning difference. (...) In other words, difference in meaning correlates with difference in distribution.'

  - [Firth, 1957]:

    'You shall know a word by the company it keeps.'

- It is possible to represent words by its context! Such models are called **count models** [Baroni et al., 2014]

Distributional Hypothesis (Ctd.)

- Famous example by [McDonald and Ramscar, 2001]:

  *He filled the **wampimuk**, passed it around and we all drank some.*

  $\Rightarrow$ jar, cup, glass, ...

Distributional Hypothesis (Ctd.)

- Famous example by [McDonald and Ramscar, 2001]:

*He filled the **wampimuk**, passed it around and we all drank some.*

$\Rightarrow$ jar, cup, glass, ...

*We found a little hairy **wampimuk** sleeping behind the tree.*

$\Rightarrow$ cat, bear, raccoon, ...

- **Based on the context the invented word 'wampimuk' is associated with a different meaning**

Count Models

- Set a window size m and count # of times a word occurs in that window
- We get a $|\mathcal{V}| \times |\mathcal{V}|$ count matrix $\boldsymbol{M}$
- Similar approach: **Latent semantic analysis** [Deerwester et al., 1990]

Example by R. Socher:

- I like deep learning .
- I like NLP .
- I enjoy flying .

Window size: $m = 1$								
Counts	**I**	**like**	**enjoy**	**deep**	**learning**	**NLP**	**flying**	**.**
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Count Models (Ctd.)

- Each column (or row) can be used as a word vector
- **Observations:**
  1. Due to the counts the word vector now captures the context/semantics better
  2. The vector is still huge! (size of the vocabulary)
- **Singular value decomposition (SVD)** is performed in order to reduce the dimensionality of the matrix $\boldsymbol{M}$:

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}}$$

- We can now use matrix $\boldsymbol{U}$ as a reduced version

word2vec [Mikolov et al., 2013]

- Why not learning compact word representations in the first place?
- T. Mikolov et al. proposed **word2vec**
- **Unsupervised learning** using an **auxiliary task**
- The auxiliary task is inspired by the concept of **language models**, e. g. [Bengio et al., 2003]
  - A language model predicts the next word given the previous words
  - E. g.: *A dog is a man's best* _____ .

CBOW vs. Skip-Gram

Mikolov et al. propose two architectures:

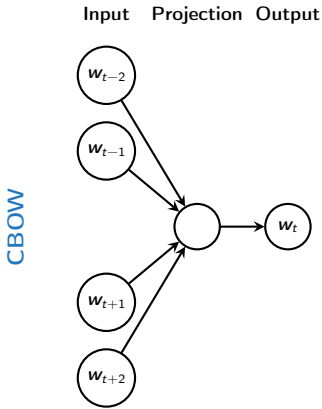
- **CBOW (Continuous bag-of-words):**
  Predict the middle word given the context, e. g.:

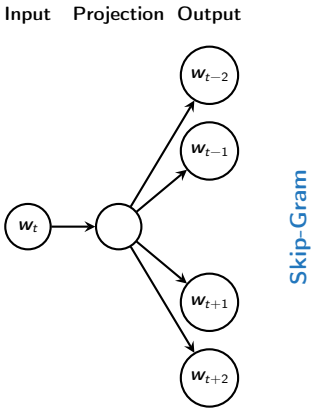
  same procedure _____ every year

- **Skip-Gram:**
  Predict the context words given the middle word, e. g.:

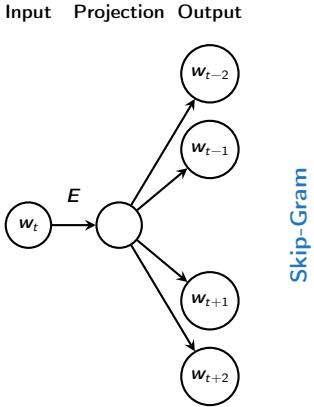
  _____ _____ as _____ _____

CBOW vs. Skip-Gram (Ctd.)





Skip-Gram in more Detail



- The input/labels are **one-hot vectors** (also for CBOW)
- $\boldsymbol{E} \in \mathbb{R}^{d \times |\mathcal{V}|}$ is a **shared embedding matrix**
- **Identity activation** in the projection layer
- $\boldsymbol{wE} = \boldsymbol{v} \in \mathbb{R}^{d \times 1}$ is the embedding of word $\boldsymbol{w}$ (since $\boldsymbol{w}$ is one-hot)

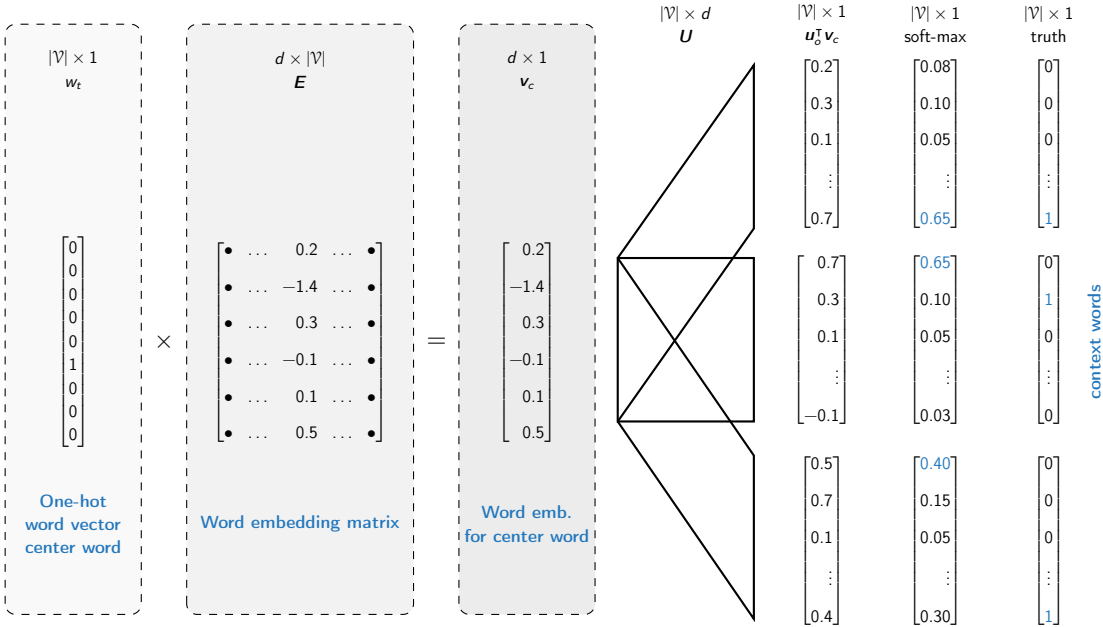


Image taken from Manning (adapted): https://www.youtube.com/watch?v=ERibwqs9p38

Skip-Gram Training Objective

- Skip-Gram optimizes the **negative log-likelihood**:

$$\mathcal{J}(\boldsymbol{\theta}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m \leqslant j \leqslant m \\ j \neq 0}} \log p(w_{t+j}|w_t;\boldsymbol{\theta})$$

- The probability is given by the soft-max:

$$p(o|c) = \frac{\exp\{\boldsymbol{u}_o^\intercal \boldsymbol{v}_c\}}{\sum_{w=1}^{|\mathcal{V}|}\exp\{\boldsymbol{u}_w^\intercal \boldsymbol{v}_c\}}$$

- t is the index into the corpus, o and c are indices into the vocabulary

Evaluation of Word Embeddings

- As Mikolov et al. found out, the embeddings do not only capture **syntactic regularities**...
- ...but also **semantic aspects** of the words
- **How did the authors get this insight?**

$\Rightarrow$ Definition of a set of 9 syntactic and 5 semantic questions/tasks, e. g.:

- Adjective to adverb (e. g. `bad` $\Leftrightarrow$ `badly`)
- Comparative (e. g. `great` $\Leftrightarrow$ `greater`)
- Nation adjective (e. g. `France` $\Leftrightarrow$ `French`)

Evaluation of Word Embeddings (Ctd.)

- An instance of a syntactic question (Comparative):

  *What is the word that is similar to **small** in the same sense as **biggest** is similar to **big**?*

- One very famous semantic example (Man-Woman):

  ***Man** relates to **king** like **woman** does to...?*

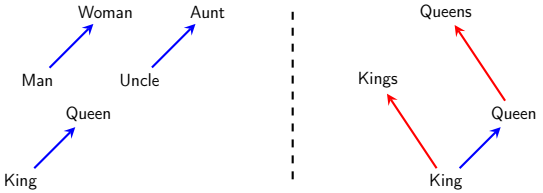
Evaluation of Word Embeddings (Ctd.)

- Such questions can be answered by performing basic algebraic operations:

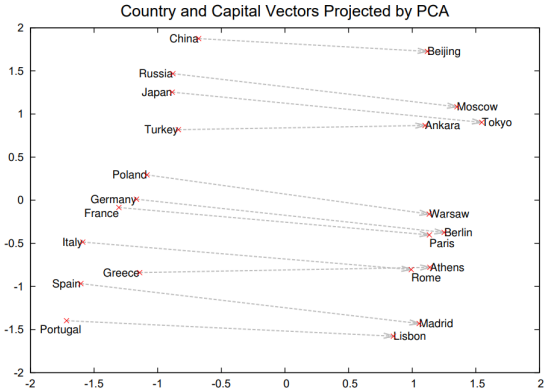
$$\boldsymbol{v}_{biggest} - \boldsymbol{v}_{big} + \boldsymbol{v}_{small} \approx \boldsymbol{v}_{smallest}$$

$$\boldsymbol{v}_{king} - \boldsymbol{v}_{man} + \boldsymbol{v}_{woman} \approx \boldsymbol{v}_{queen}$$

- A visual example:



Result of Country Capital Task



Section:

Wrap-Up

Summary

- Neural Networks are powerful models for pattern recognition
- The perceptron can classify all training examples correctly, **iff the training data is linearly separable**
- **MLPs are universal function approximators**
- Backpropagation is a recursive procedure based on the **chain rule of calculus** to obtain the gradients for neural network learning
- Different neural network architectures like CNNs and RNNs exist for solving different kinds of problems

Self-Test Questions

1. What is the relation between neural networks and logistic regression?
2. What is a perceptron? Which problems can it solve and which not?
3. Why do we often use multiple layers instead of a simple perceptron?
4. How does backpropagation work? How does a neural network learn?
5. What are advantages and disadvantages of using neural networks?
6. What are CNNs and RNNs? For which tasks are they suitable?
7. How can words be represented in vectorial form?

What's next...?

Unit I	Machine Learning Introduction
Unit II	Mathematical Foundations
Unit III	Bayesian Decision Theory
Unit IV	Probability Density Estimation
Unit V	Regression
Unit VI	Classification I
Unit VII	Evaluation
Unit VIII	**Classification II**
Unit IX	Clustering
Unit X	Dimensionality Reduction

Recommended Literature and further Reading I

[1] Deep Learning
Ian Goodfellow et al. MIT Press. 2016.
→ Link, cf. chapters 6 *Deep Feedforward Networks*, especially chapter 6.5

[2] Pattern Recognition and Machine Learning
Christopher Bishop. Springer. 2006.
→ Link, cf. chapter 5, especially chapter 5.3

[3] Backpropagation calculus
Grant Sanderson. YouTube. 2017.
→ Link

Recommended Literature and further Reading II

[4] Simple Backpropagation in NumPy
Andrew Ng et al. Stanford CS229. 2019.
→ Link

Meme of the Day

Thank you very much for the attention!

Topic: *** Applied Machine Learning Fundamentals *** Neural Networks / Deep Learning
Term: Winter term 2022/2023

Contact:
Daniel Wehner, M.Sc.
SAP SE / DHBW Mannheim
daniel.wehner@sap.com

Do you have any questions?