Transfer Learning for Biomedical Relation Extraction

Introduction to Deep Learning for NLP

(Slides partially taken from Ulf Leser)

Mario Sänger (WBI, HU Berlin)

saengema@informatik.hu-berlin.de

Outline

- Classification methods
  - Nearest Neighbour
  - Support vector machine

- Introduction to Deep Learning
  - Motivation
  - Feed forward networks
  - Outlook (RNNs, CNNs)

- Topic presentation

Classification Methods

Nearest Neighbour

Support Vector Machine

Linear Classifiers

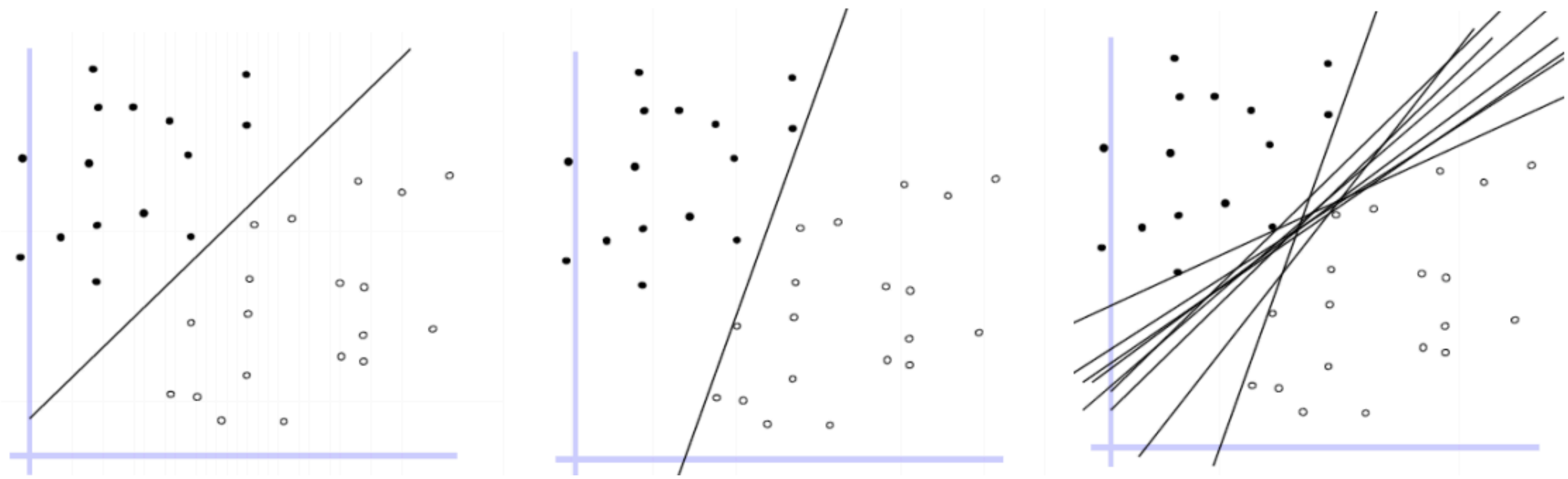
- Many common classifiers are (log-)linear classifiers
  - Naïve Bayes, Perceptron, Linear and Logistic Regression, Maximum Entropy, Support Vector Machines
- If applied on a binary classification problem, all these methods somehow compute a hyperplane which (hopefully) separates the two classes
  - Despite similarity, noticeable performance differences exist – Which feature space is used?
  - Which of the infinite number of possible hyperplanes is chosen?
  - How are non-linear-separable data sets handled?

Characteristics of text data

- High dimensionality: 100k+ features
- Sparsity: Feature values are almost all zero
- Most documents are very far apart (i.e., not strictly orthogonal, but only share very common words)
- Consequence: Most document sets are well separable
  - This is part of why linear classifiers are quite successful in this domain
- The trick is more of finding the “right” separating hyperplane instead of just finding (any) one

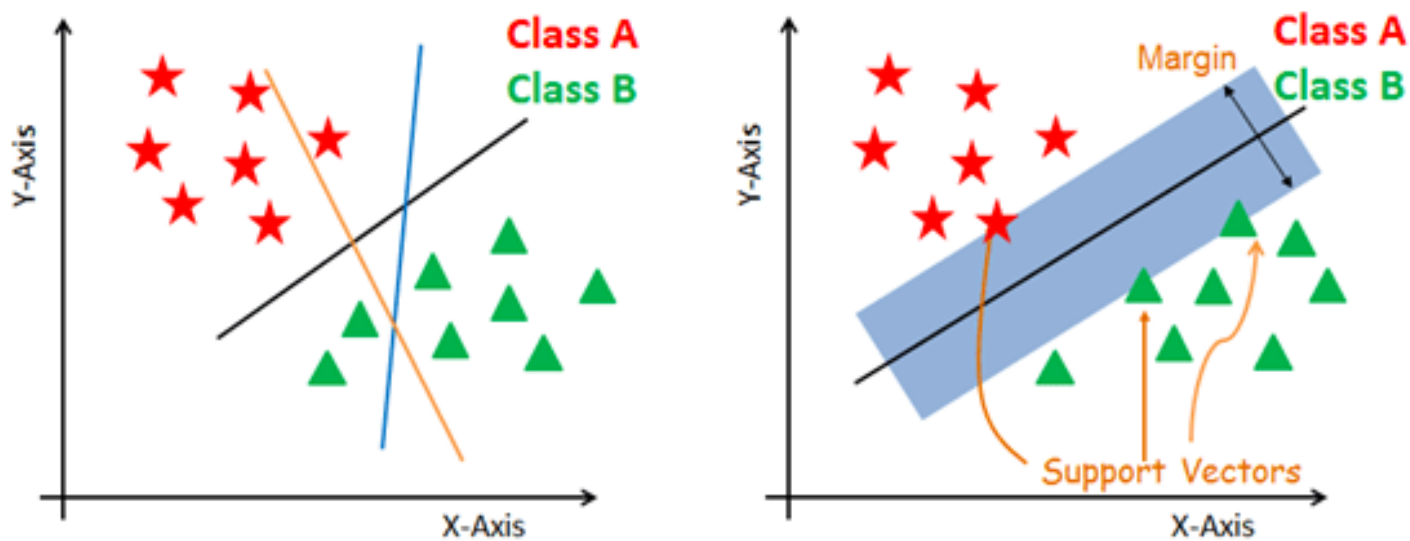
Example: Linear Classifiers – 2D

- Hyperplane separating classes in high dimensional space
- But which?



Support Vector Machine (SVM) - Idea

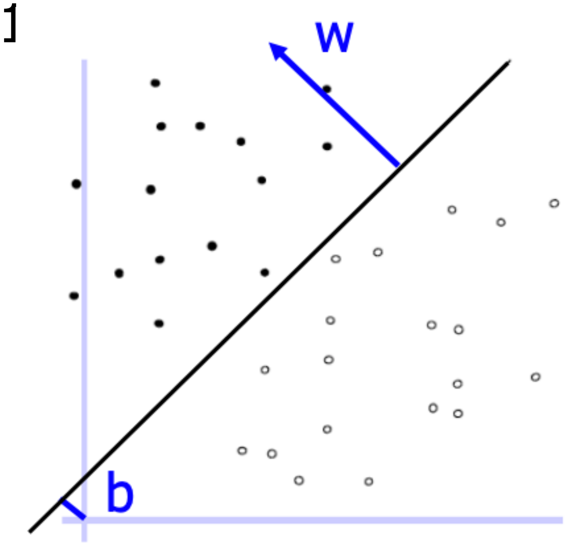
- SVMs: Hyperplane which maximizes the margin
  - I.e., is as far away from any data point as possible
  - Cast in a linear optimization problem and solved efficiently
  - Classification only depends on support vectors – efficient
    - Points most closest to hyperplane



http://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1526288454/index2_ub1uzd.png

Separating Hyperplanes

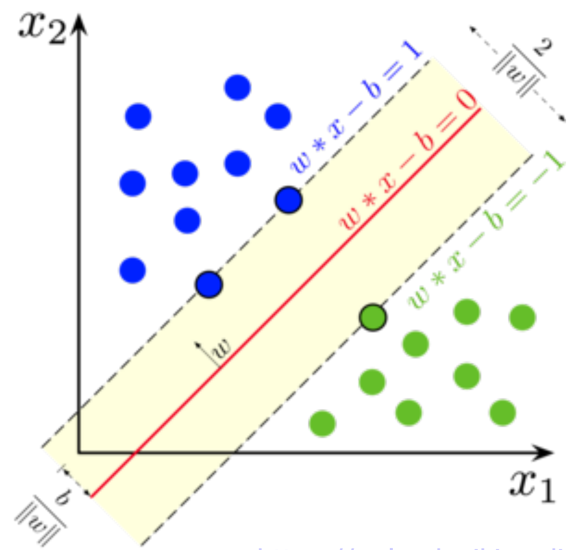
- Assume a set of training samples x_i with labels $y_i \subseteq \{-1,+1\}$
  - Training samples xi are real-valued vectors
- Assume, for now, that the classes are linearly separable
  - There exists a hyperplane h such that all instances above h have label +1, all instances below h have label -1



- A hyperplane h can be characterized by ortho-normal vector w and a bias b: All points v with <w*v>+ b=0
  - – <> is the dot product, i.e. $\Sigma w_i * v_i$
- Thus, we seek a pair w,b such that $\forall i$: $y_i = sgn(+b)$
- Problem: If one such hyperplane exists, then there are infinitely many (infinitesimally different)

Maximum Margin Classifier

- SVM seeks the one which maximizes the margin, i.e., find $h=(w,b)$ such that $m = \min_i |<w*x_i>+b|$ is maximal
  - Compute distance between h and all x_i ; the minimal distance is the width of the margin; find h such that this margin m is maximal
- The margin is actually defined by two parallel hyperplanes h1, h2; one defined by the closest points with y=1, one by the closest negative points. Thus, there are x' and x" with
  - h1: $<w*x'>+b=1$ and
  - h2: $<w*x''>+b=-1$



https://upload.wikimedia.org/wikipedia/commons/thumb/7/72/SVM_margin.png/600px-SVM_margin.png

- The distance between h1 and h2 is $2/||w||$
- Thus: A minimal $||w||$ gives maximal margin

Optimization Problem

- Of course, h must also separate the two classes
- This gives the following constrained optimization problem:

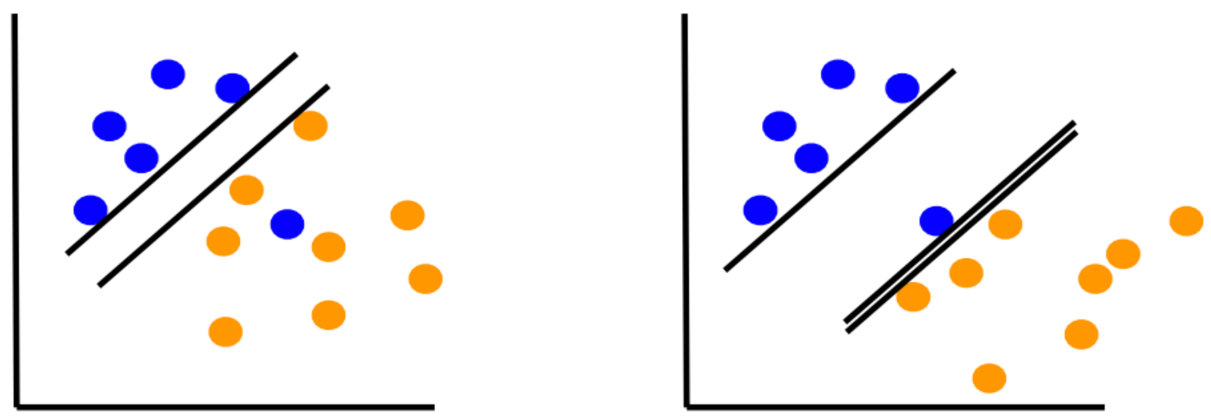
Minimize ||w||

under the constraint: $\forall i: y_i (\langle w * x_i \rangle + b) \geq 1$

- Not yet done: Most data sets are not linearly separable

Misclassification

- We need to account for instances that are misclassified
  - There should not be many, though
  - Might even be useful if the data set is linearly separable
  - We need a parameter defining how hard to "punish" training instances that are misclassified



Slack Variables

- For each training instance (x_i, y_i), we introduce a slack variable ξ_i which measures the error wrt. the correct side of the hyperplane
  - i.e.: $\forall i$: $y_i (<w*x_i>+b) + \xi_i \geq 1$
  - Ideally, ξ_i is zero for all instances i

- New constraint: The sum of all errors ξ_i should be minimal
- New constrained optimization problem:

Minimize $||w|| + C*\Sigma\xi_i$

under the constraint: $\forall i$: $y_i (+b)+\xi_i \geq 1$

C-Parameter

- New constrained optimization problem:

$$\text{Minimize } ||w|| + C * \Sigma\xi_i$$

$$\text{under the constraint: } \forall i\text{: } y_i\ (+b)+\xi_i \geq 1$$

- C controls the influence of misclassification
  - Large C: leads to few misclassifications and small margins
  - Small C: lead to more misclassifications and larger margins

Solutions

- Fortunately, this is a convex optimization problem and usually can be solved efficiently
  - But training with millions of dimensions and thousands of training instances still may take considerable time
- Classification (a new x) is fast: Compute sgn(<w*x>+b)
- "Support Vector" machine: The hyperplane only depends on the instances at the border of the margin; these are called "support vectors"
  - Original Paper: History: Victor Vapnik, "The Nature of Statistical Learning Theory", 1995

Properties of SVM

- SVMs provide a strong baseline for many NLP problems
  - State-of-the-art for long time in text classification
- Can cope with millions of dimensions
  - Might require long training time
- Classification is rather fast
- Quite robust to overfitting
- SVM are quite good "as is", but tuning possible
- Several free implementations exist: SVMlight, libSVM, …

Deep Learning for NLP

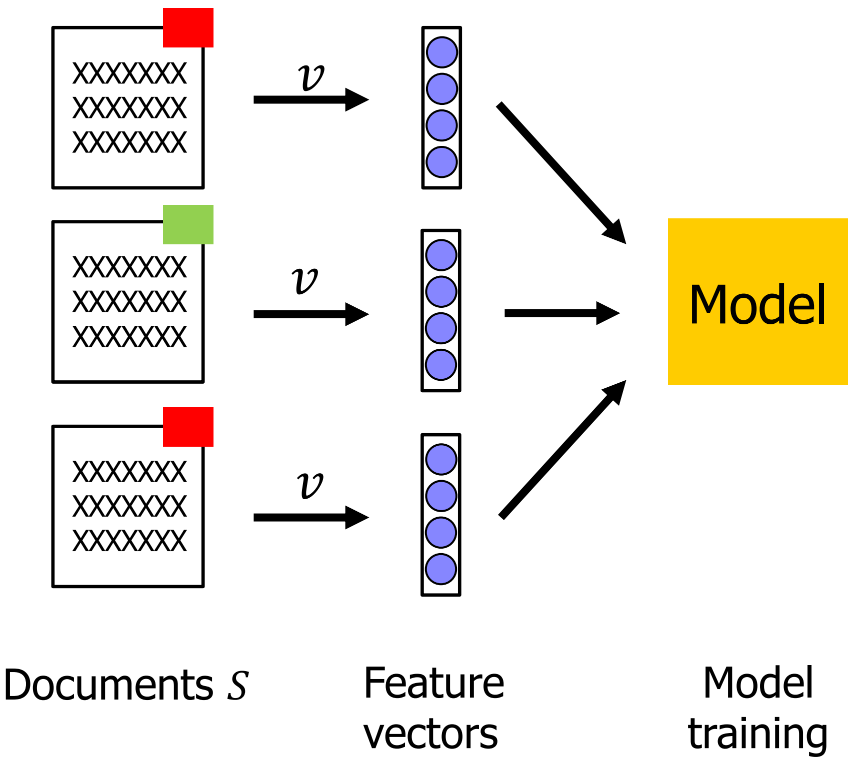
Motivation

Feed forward networks

Outlook

Recall: Supervised Learning

- Given a set D of documents and a set of classes C. A classifier is a function $f: D \rightarrow C$



- Problems
  - Finding enough training data
  - Finding the best pre-processing (tokenization, case, POS tag set …)
  - Finding the best features
  - Finding a good classifier (~ assigning as many docs as possible to their correct class)

How to find good features?

- Many AI tasks can be solved by designing the right set of features to extract and apply a (simple) ML approach
  - However, for many tasks is difficult to know what features should be extracted
  - Can take decades for an entire community of researchers

- Example: Identify cars in photographs
  - We know cars have wheels – presence of a wheel maybe a good feature
  - Unfortunately it is hard to describe a wheel in terms of pixels
  - Simple geometric shape – but it's image may be complicated by shadows falling on it, the sun glaring off the metal parts, ...

How to find good features?

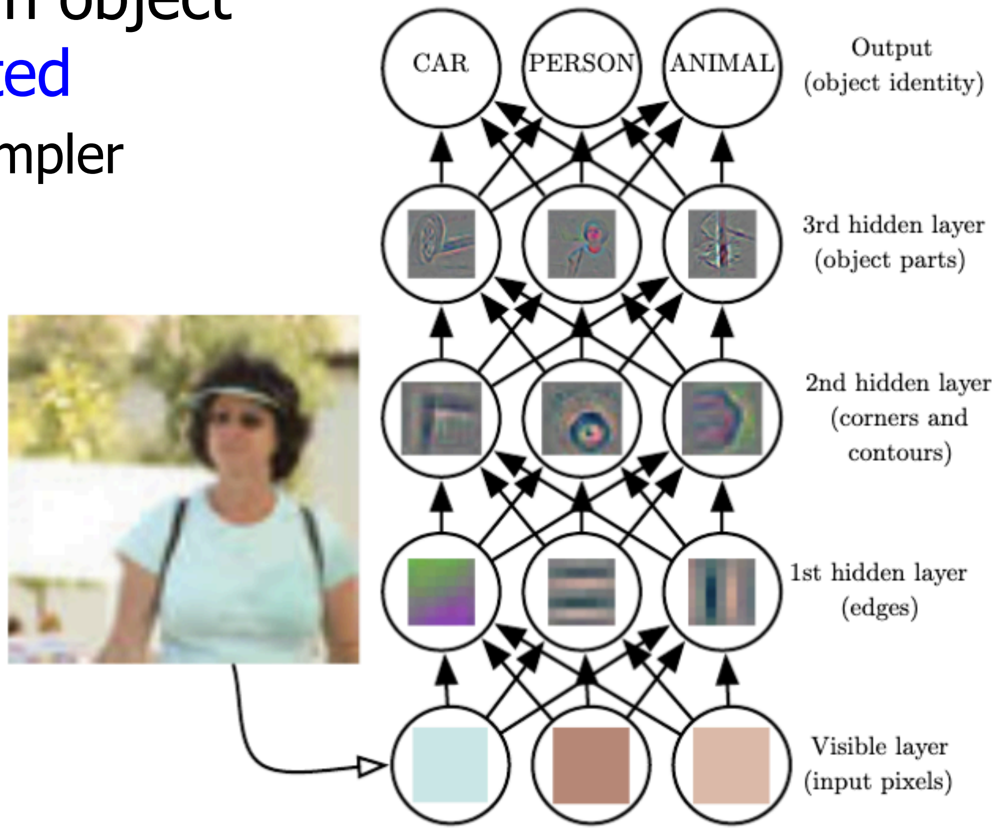
- Our goal usually is to separate the factors of variation that explain the observed data
  - Factors refer to separate sources of influence in this context
  - Such factors are often not quantities that are directly observed
- Example: Photographs of cars
  - Plethora of factors exists: the position of the car, its color, the angle or brightness of the sun, ...
  - Many factors influence every single piece of data (pixel) we have
    - E.g. Individual pixels in an image of a red car might be very close to black at night
- Most applications require to disentangle these factors
  - Discard the factors we don't care about

Representation Learning / Deep Learning

- Of course, it is very difficult to extract such high-level features / factors from raw data
  - Need very sophisticated (nearly human-level) understanding of the raw data
- One solution to the problem: representation learning (RL)
  - Use machine learning to discover not only the mapping from representation to output but also the representation itself
  - Representation learning ~ feature learning
- Deep Learning: A RL technique that learns representations that are expressed by simpler representations
  - Build more complex concepts out of simpler concepts

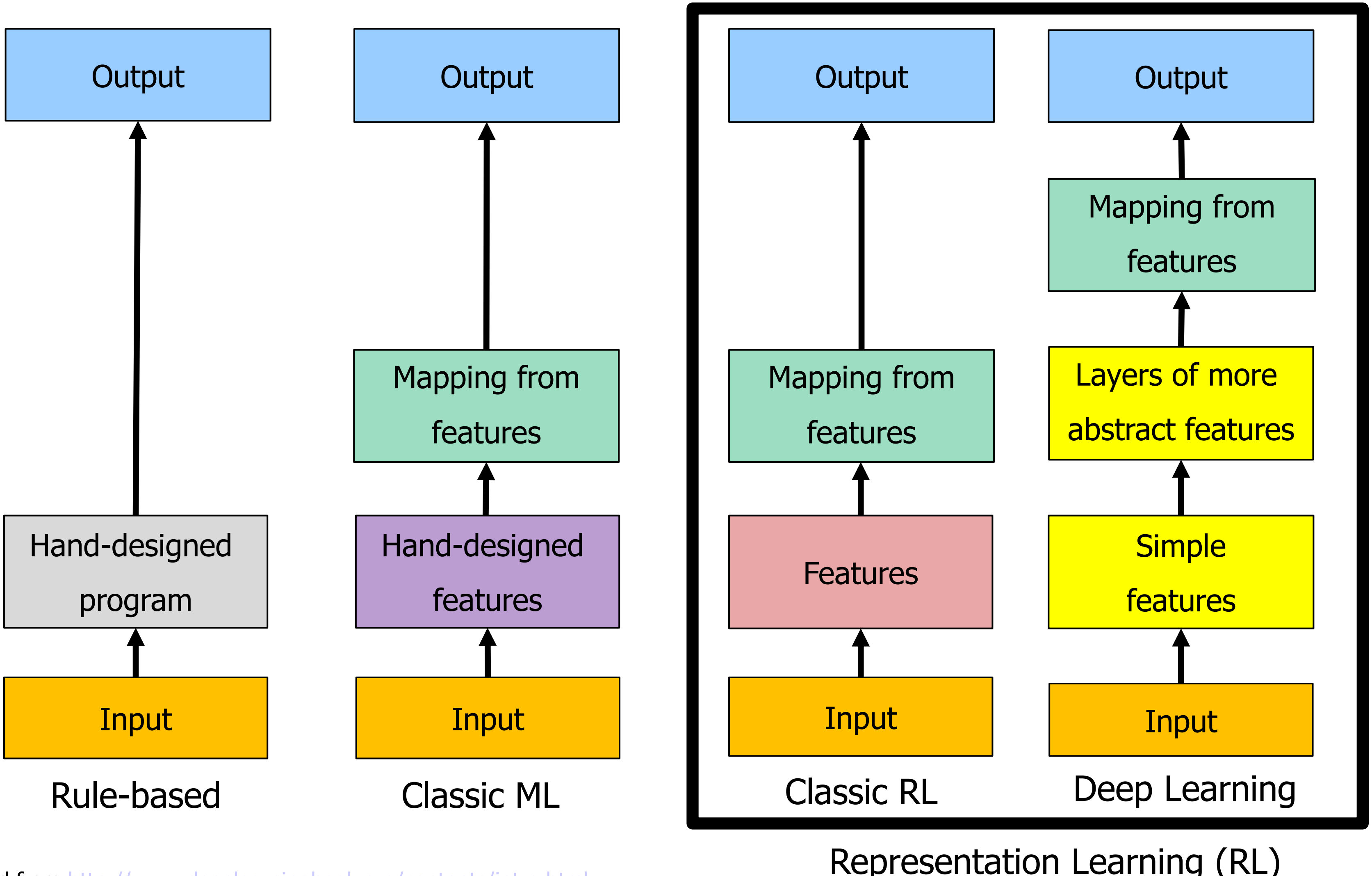
Example: image classification

- Mapping from pixels to an object identity is very complicated
  - Instead, use a series of simpler nested mappings
- Every layer builds a higher abstractions based on the former layer's output
- Final layer uses most abstract representations to make the prediction



http://www.deeplearningbook.org/contents/intro.html

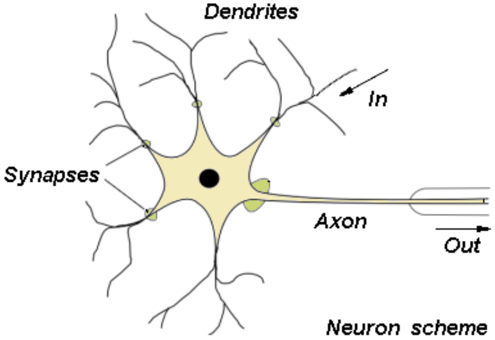
Comparison of different AI systems

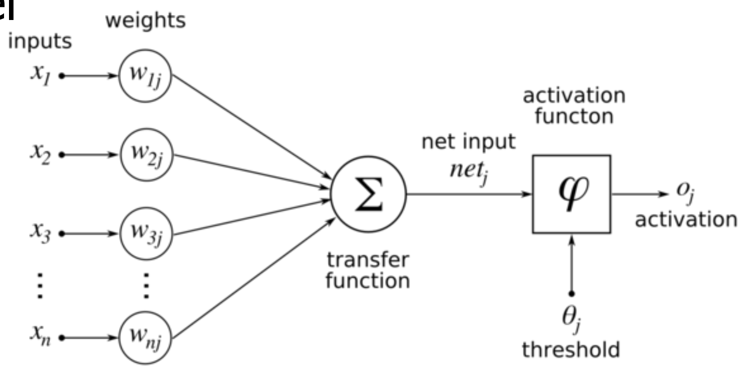


Adapted from http://www.deeplearningbook.org/contents/intro.html

Deep Learning / neural networks

- A method for non-linear classification
  - Long history - but also forgotten for a long time
    - First works range back to the 1950s / 60s
    - Extremely hyped since about 2005
  - Basic concepts inspired by biological networks
    - But, it isn't the goal to simulate / model these networks





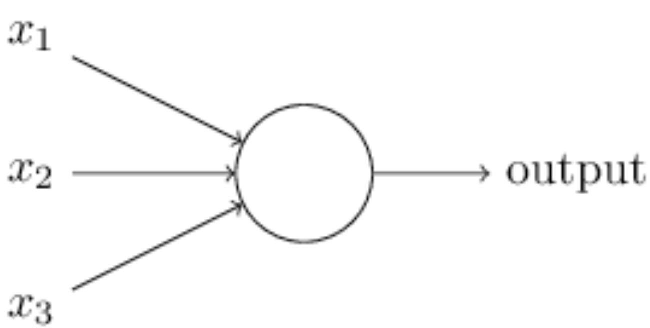
- Today: state-of-the-art in machine translation, image recognition, gaming, machine reading, ...

Deep learning / neural networks

- Two trends have contributed to the breakthrough
  - Build deeper networks – more and wider hidden layers capture more signals
    - It is not true that “more is always better”
    - Still much art (not science) in tuning hyper-parameters
  - Learn on much more data
    - Deep learning is only good if a lot training data is available
    - Include unsupervised data – pre-training to obtain good initial weights
  - Both require much longer training times – prohibitive in the past
- Today: Optimized algorithms, stronger machines, accelerators (GPU), distributed learning, pre-trained models, …

First start: Perceptron

- A perceptron takes a sequence of binary inputs $x_1, x_2, \ldots$ and produces a single binary output
  - Weights: for each input x_i there is associated weight w_i representing the importance of x_i to the output
  - Output (~ activation): Determine whether the weighted sum $\sum_i w_i x_i$ is over a threshold value t
  - Basic idea: weighing up evidence to make a decision



$$output = \begin{cases} 0 & if \sum_i w_i x_i \leq t \\ 1 & if \sum_i w_i x_i > t \end{cases}$$

First start: Perceptron

- Example: Go to a music festival or not?
  - Input variables
    - x_1 = Is the weather good? (No=0 / Yes=1)
    - x_2 = Do your friends want to accompany you? (No=0 / Yes=1)
    - x_3 = Is the venue near public transport? (No=0 / Yes=1)
  - Suppose:
    - You love music so much that you're happy to go to the festival even if your friends are uninterested and the festival is hard to get to
    - But you really loathe bad weather, and there's no way you'd go to the festival if the weather is bad
  - You can use a perceptron to model this kind of decision-making
    - For example: $w_1 = 6$, $w_2 = 2$, $w_3 = 2$ and a threshold $t = 5$
    - Models exact this decision-making process

First start: Perceptron

- By varying the weights w_1, w_2, w_3 and the threshold t, we can get different models of decision-making
  - For example, suppose we instead chose a threshold of $t = 3$
  - Now: you should go to the festival whenever the weather was good ...
  - ... *or* when both the festival was near public transit *and* your friends are willing to join you

- Conclusion: perceptron can weigh up different kinds of evidence in order to make decisions

First start: perceptron

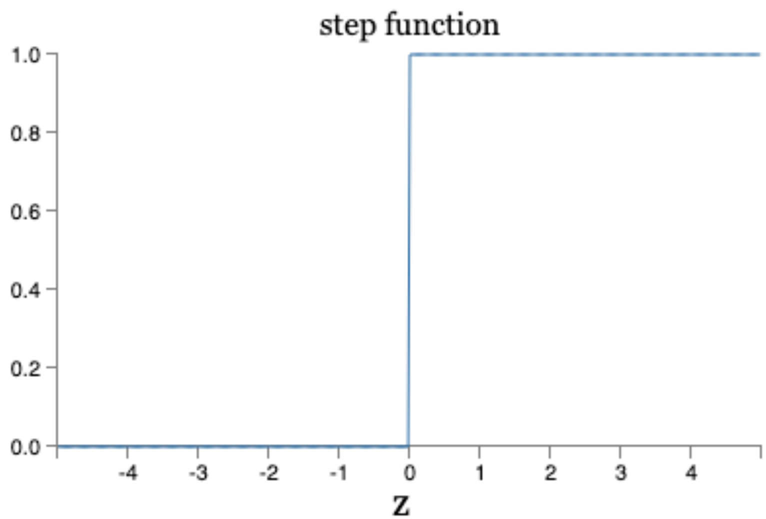
- We can simplify the perceptions description
  - Write $\sum_i w_i x_i$ as a dot product, $\mathrm{w} \cdot x = \sum_i w_i x_i$
    - w and x are vectors whose components are the weights and inputs respectively
  - Move threshold to other side – known as bias b
    - We set $b = -t$
  - We can now calculate the output

$$output = \begin{cases} 0 & if\ \mathrm{w} \cdot x + b \leq 0 \\ 1 & if\ \mathrm{w} \cdot x + b > 0 \end{cases}$$

- Can be modelled via sign-function: $\mathrm{sgn}(\mathrm{w} \cdot x + b)$

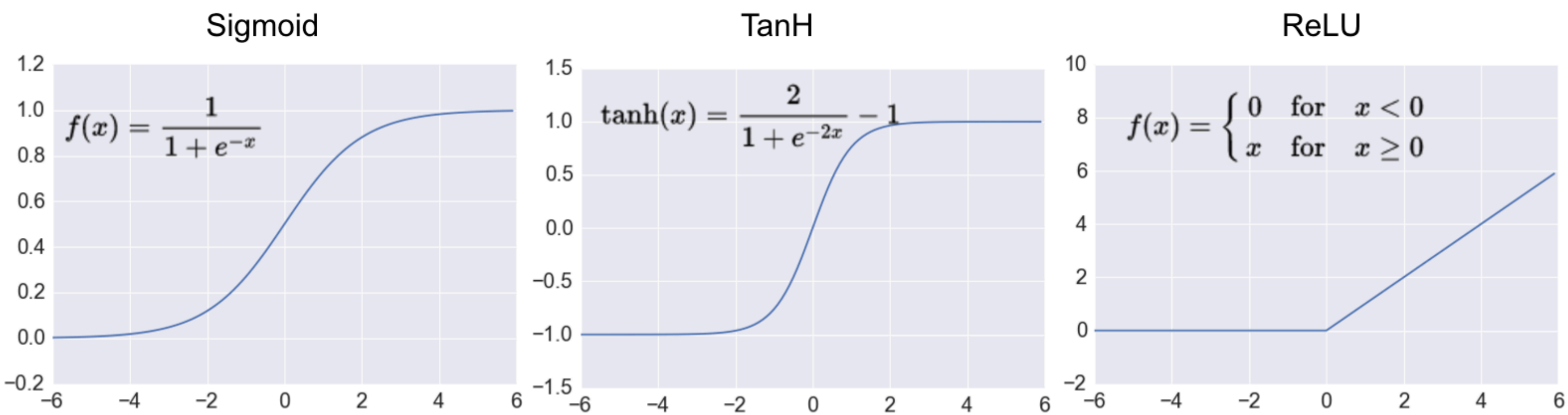
Activation functions

- So far, we modelled the output of our perceptron using the step function $\text{sgn}(z)$
  - Small changes of some weights w_i can lead to great changes in the output (i.e. flip from 0 to 1)
  - This makes it difficult to learn the weights



Activation functions

- Use activation functions with a continuous value range
  - Small changes in the weights and biases cause only a small change in their output
  - Often: activations saturate for very large and/or small values

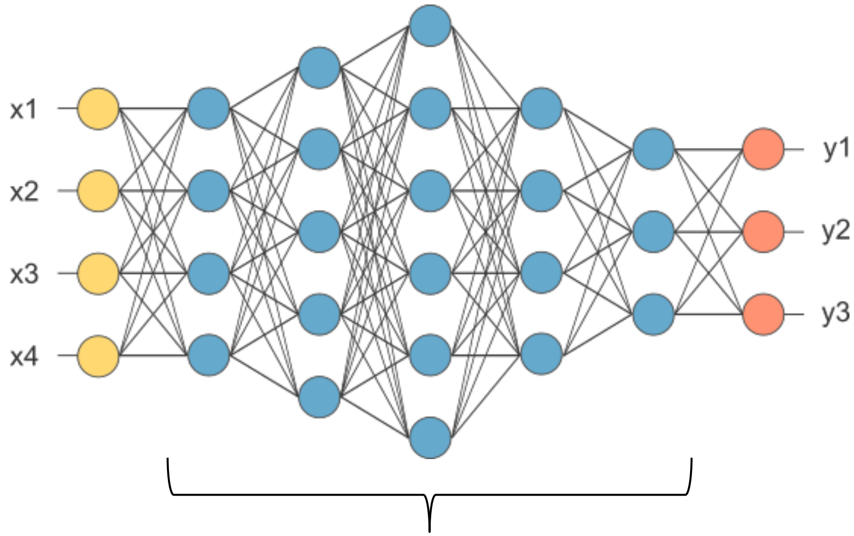


http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Structure of neural networks

- Neurons are organized and stacked in layers
  - The neurons of each layer work on the activations from the former layer
  - Each layer learns more complex abstractions (~ decisions) of the input based on the former layer's abstractions

Input layer



Output layer

Hidden layers

http://www.opennn.net/

Example: Text classification

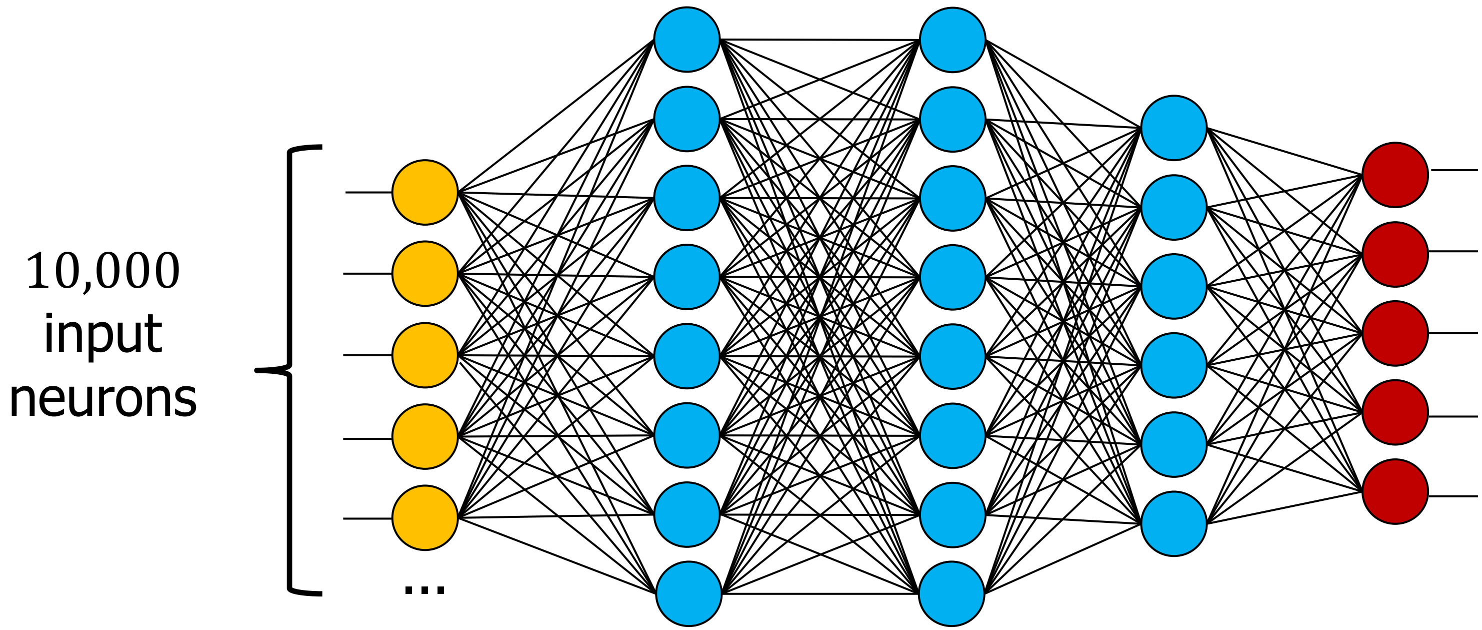
- Let's suppose, we have a corpus of news articles and we want to perform automatic categorization of these articles
  - We want to distinguish articles from five different categories: politics, economy, culture, lifestyle, sport

- Assume we have a set S of labelled examples (x_i, y_i)
  - x_i : TF-IDF vector of text from article i (details next slide)
  - y_i : The gold standard label for article i
    - In following we will often refer to the label as one-hot encoded vector

Example	politics	economy	culture	lifestyle	sport
$y(x_1) = economy$	0	1	0	0	0
$y(x_2) = science$	0	0	0	1	0

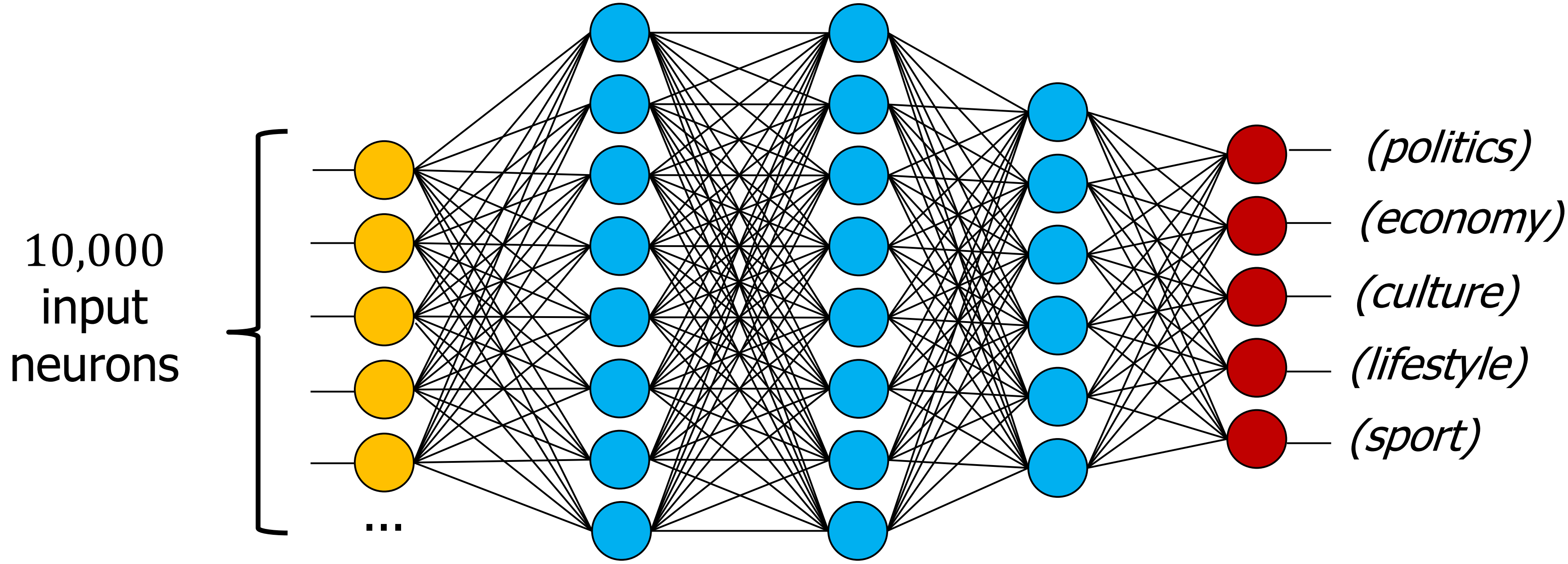
Example: Text classification

- Input: TF-IDF vectors of the articles
  - Let's say we have a vocabulary with 10.000 terms
  - Each component of the vector is modelled as separate input neuron



Example: Text classification

- Output: One of the five classes politics, economy, culture, lifestyle, sport
  - Each class gets one dedicated neuron in the output layer
  - We select the output neuron which fires resp. has the highest activation as prediction



Cost function

- We want to find weights and biases so that the output from the network approximates y_i for all training inputs x_i as good as possible

- To quantify how well we're achieving this goal we define a cost function (loss / objective):

$$C(w, b) = \frac{1}{2n} \sum_i \|y_i - a_i\|^2$$

  - w and b all weights and biases of the network
  - n = |S| is number of training examples
  - || || is the length of the vector

Cost function

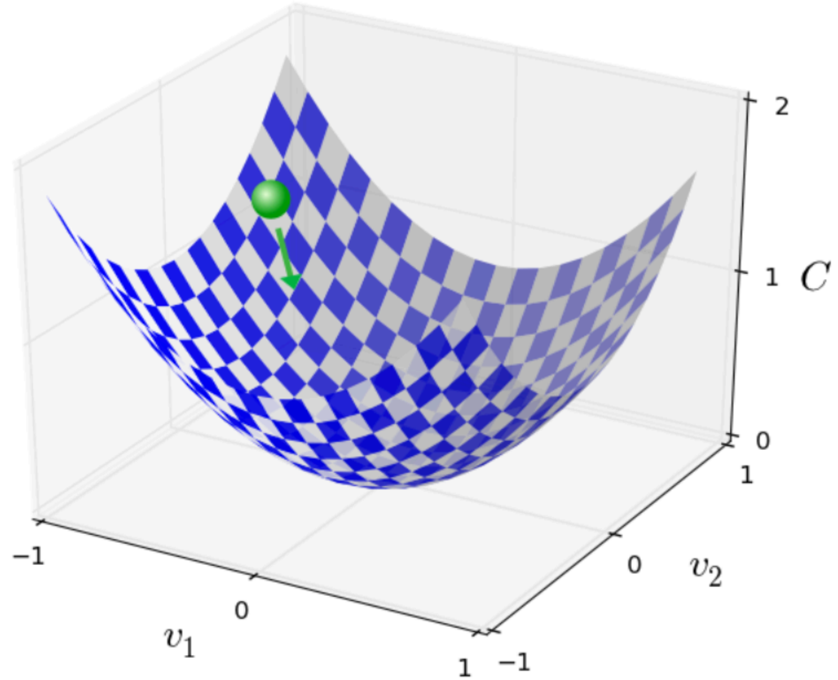
- Quadratic cost function (mean squared error)

$$C(w,b) = \frac{1}{2n}\sum_i \|y_i - a_i\|^2$$

- Properties
  - The costs are always non-negative
  - $C(w,b)$ becomes small $C(w,b) \approx 0$ when y_i is approximately equal to the network output a_i for all instances
  - In contrast, a large $C(w,b)$ means that output a_i is not close to y_i for many instances
- Training objective: minimizing the cost $C(w,b)$
  - But, how to achieve this?

How can we optimize the costs?

- For simplicity say we have a cost function $C(v_1, v_2)$ only depending on two parameter v_1 and v_2
- Image we start at a random initialization of v_1 and v_2
- We want to find the global minimum of $C(v_1, v_2)$
  - Moving down the slope from the (random) starting point
  - Calculate the gradient and move in the opposite direction



http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

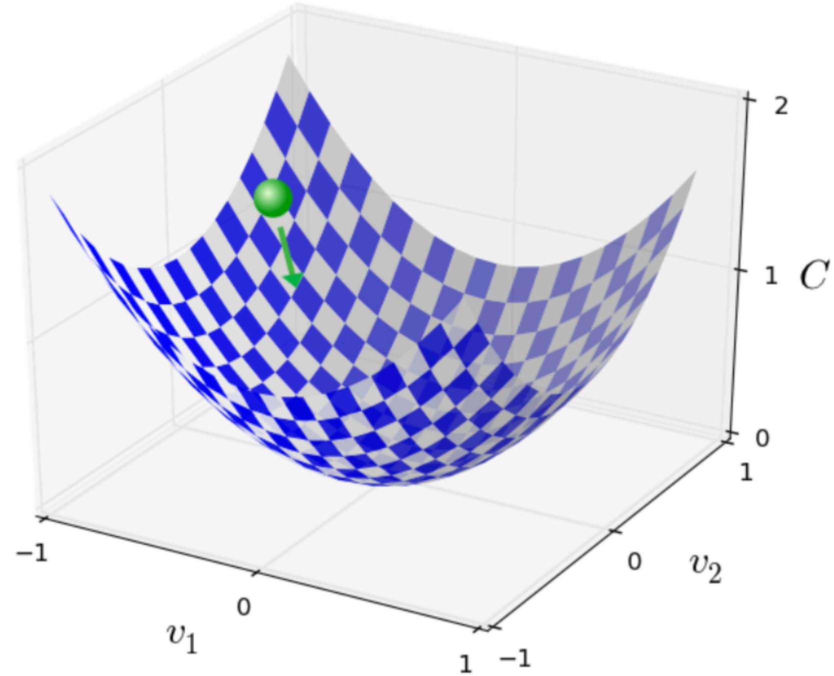
How can we optimize the costs?

- We move a small amount Δv_1 and Δv_2 in direction of v_1/v_2
  - We can represent Δv_1 and Δv_2 as vector $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$
- According to calculus the costs change are:

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2$$

- We define the gradient of C to be the vector of partial derivates

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$$



http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

How can we optimize the costs?

- We can write the cost changes as:

$$\Delta C \approx \nabla C \cdot \Delta v$$

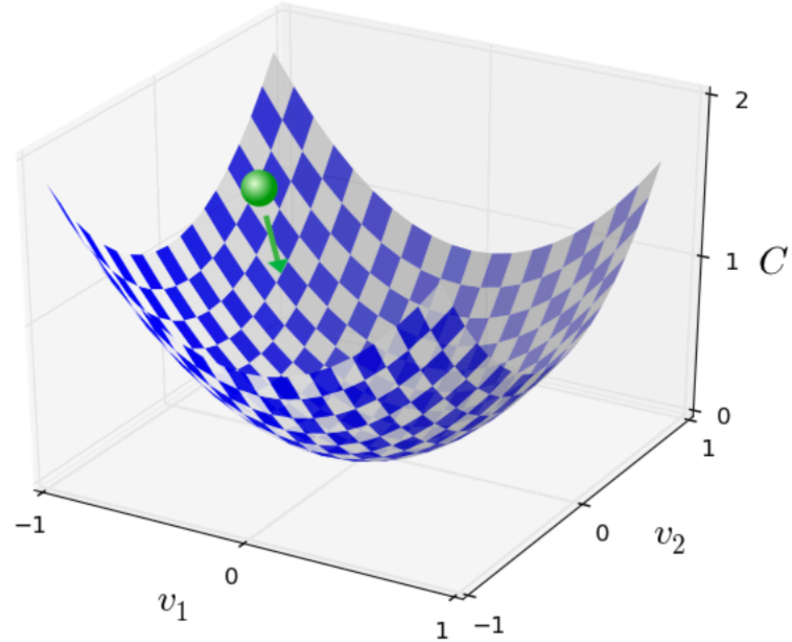
- This helps us to choose Δv so as to make ΔC negative:

$$\Delta v = -\eta \nabla C$$

  - $-\eta$ is a small, positive parameter (learning rate)

- Our cost changes are now:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$$



http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

How can we optimize the costs?

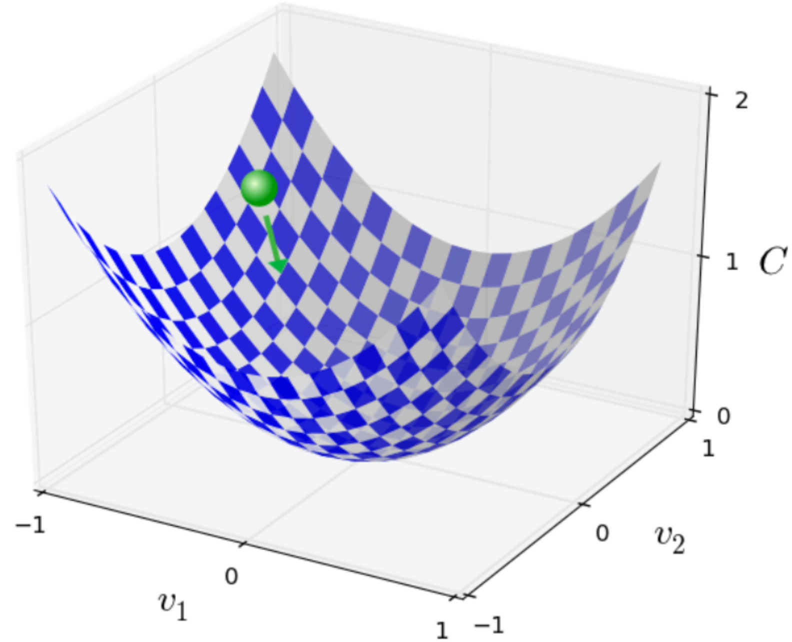
- Our cost changes are now:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$$

- Because $\|\nabla C\|^2 \geq 0$ this guarantees that $\Delta C \leq 0$
  - C will always decrease and never increase

- We can describe the update of our variables v:

$$v \rightarrow v' = v - -\eta \nabla C$$



http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

How can we optimize the costs?

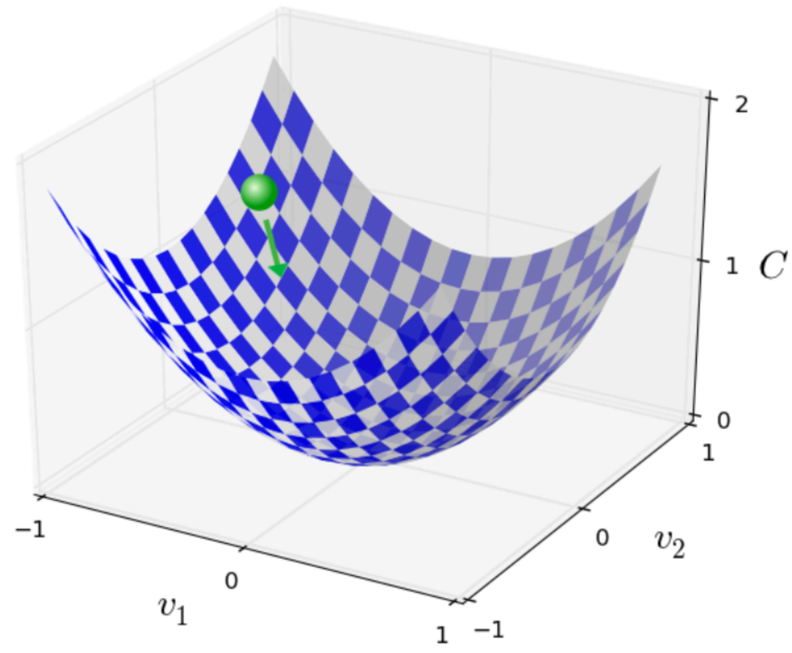
- Our cost changes are now:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2$$

- We can describe the update of our variables v:

$$v \rightarrow v' = v - -\eta \nabla C$$

- Good selection of η is important
  - Too small: learns very slow
  - Too big: jumps from valley side to valley side (C may increase)



http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

Gradient descent (GD)

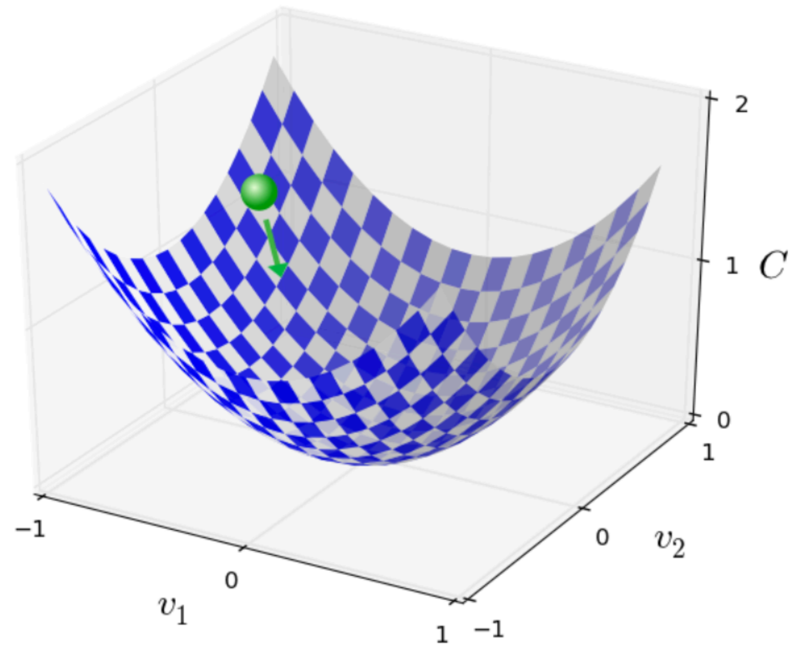
- We can easily extend out considerations to cost functions C with m parameter $v_1, v_2, \ldots, v_m$
- Small change in v is given by $\Delta v = (\Delta v_1, \Delta v_2, \ldots, \Delta v_m)^T$
- Gradient vector is given by

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, \ldots, \frac{\partial C}{\partial v_m}\right)^T$$

- Parameter update:

$$\Delta v = -\eta \nabla C$$

$$v \rightarrow v' = v - -\eta \nabla C$$



http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

GD and neural networks

- How can we apply gradient descent to learn in a neural network?
- Idea: Use GD to find the weights w and biases b which minimize our cost function
  - Minimizing the cost by changing all weights w_k and biases b_l according to:

$$w_k \rightarrow \ {w_k}' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow \ {b_l}' = b_l - \eta \frac{\partial C}{\partial b_l}$$

Gradient descent in practice

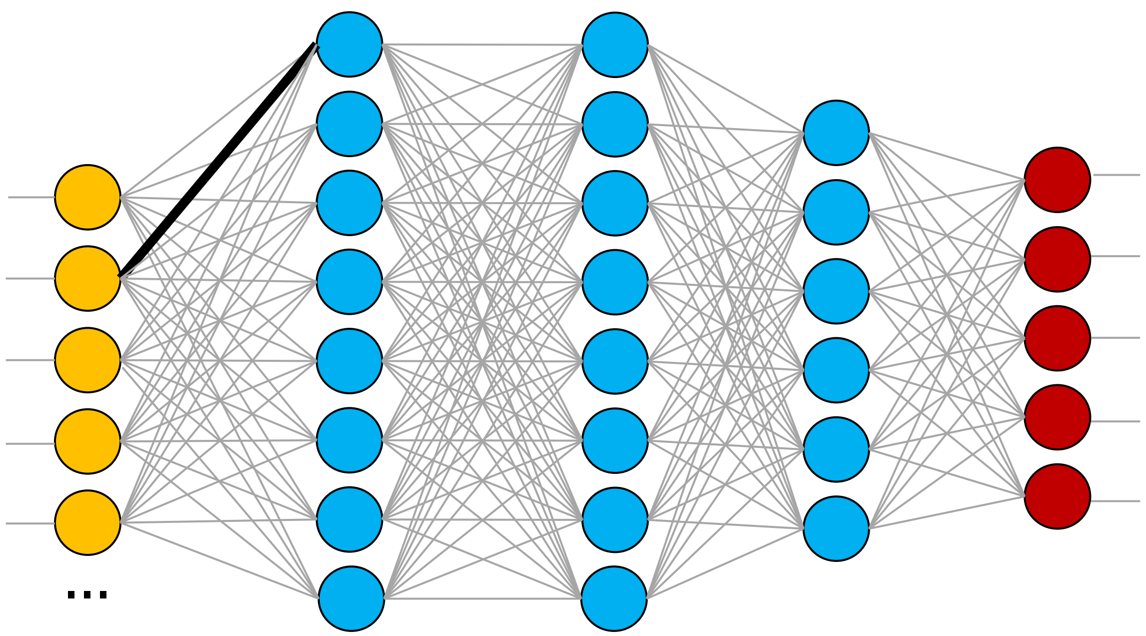
- However, there several challenges in applying the gradient descent rule (for neural networks)
- For example: Notice that our cost function averages the costs over all individual training samples

$$C(w,b) = \frac{1}{2n}\sum_i \|y_i - a_i\|^2$$

  - In practice, we need to compute the gradients ∇C_i separately for each training samples x_i and then average them
  - This can take a long time if the training set is (very) large
- Stochastic gradient descent:
  - Estimate ∇C by computing ∇C_X for a small sample of training instances X (mini-batch)

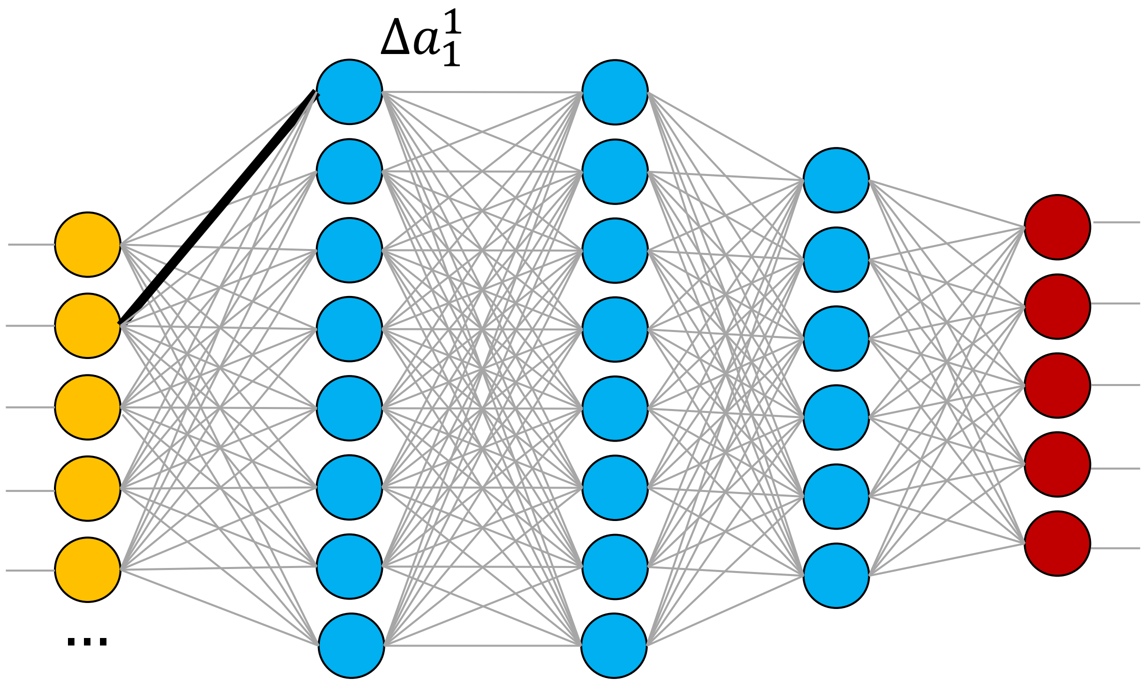
Be aware of the network structure!

- Let‘s imagine we make a small change to some weight



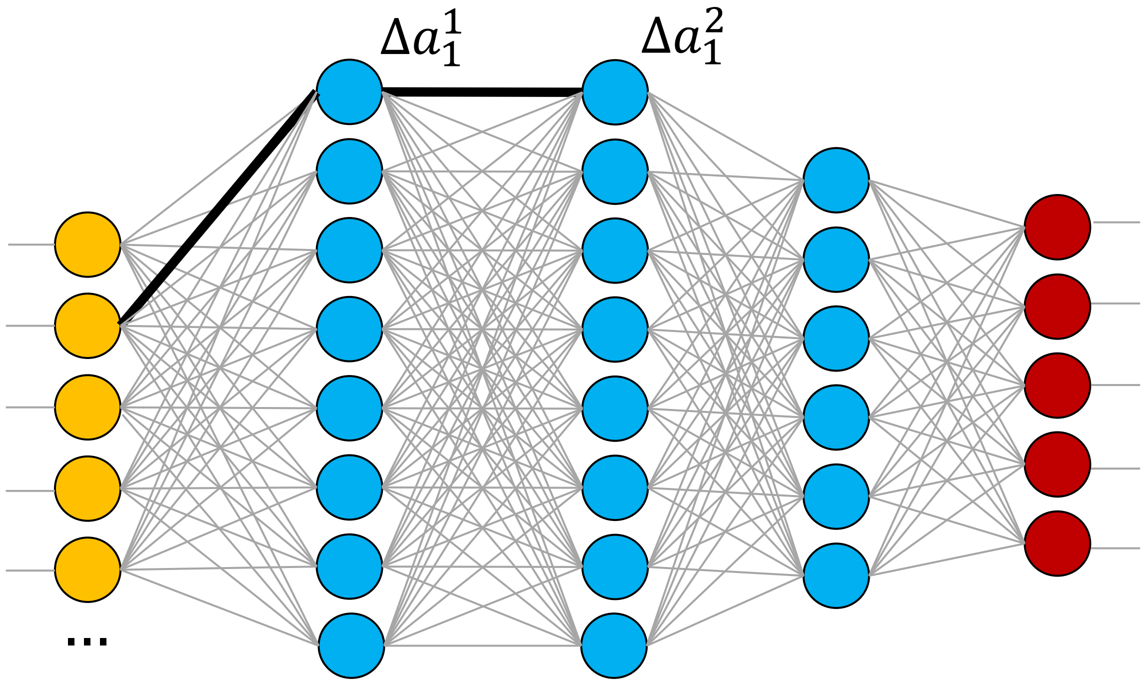
Be aware of the network structure!

- Let‘s imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron



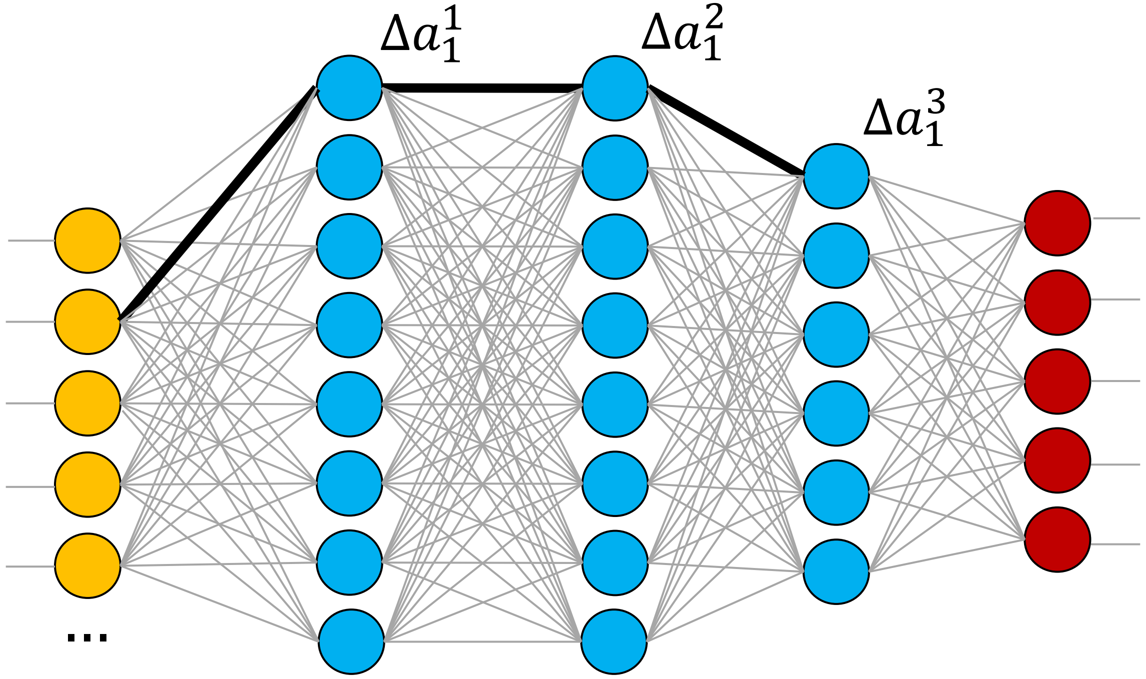
Be aware of the network structure!

- Let‘s imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron
  - ... this triggers updates of all subsequent layers / neurons



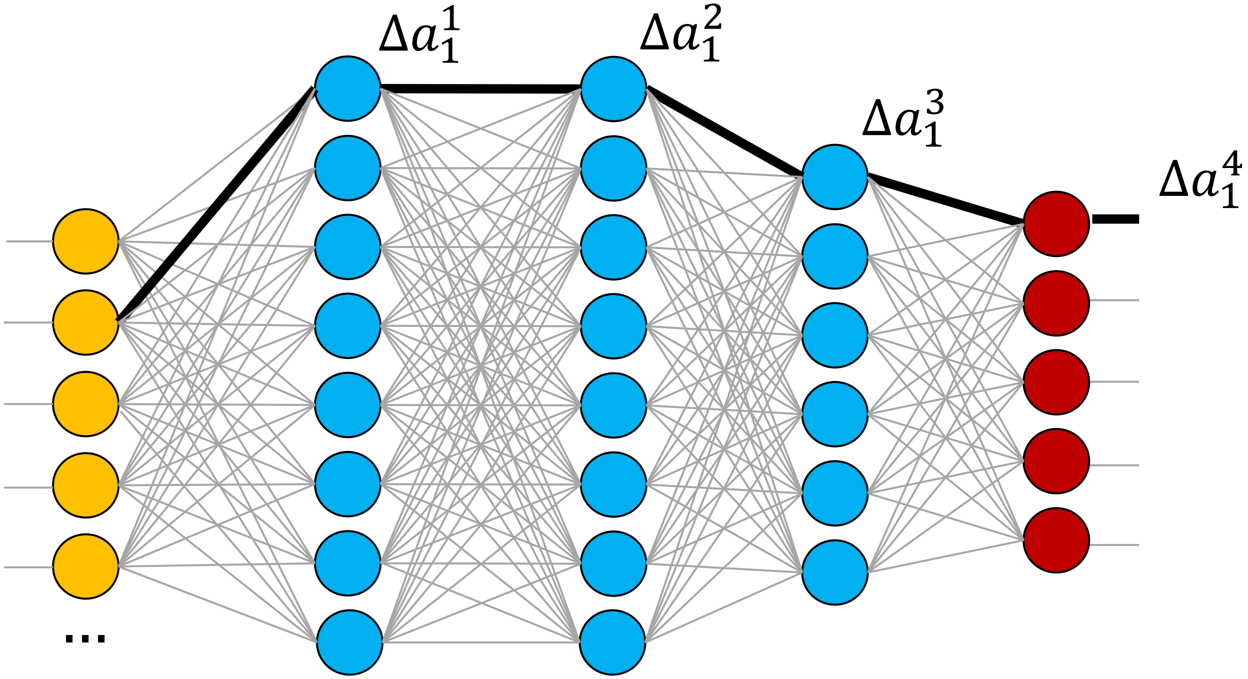
Be aware of the network structure!

- Let‘s imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron
  - ... this triggers updates of all subsequent layers / neurons



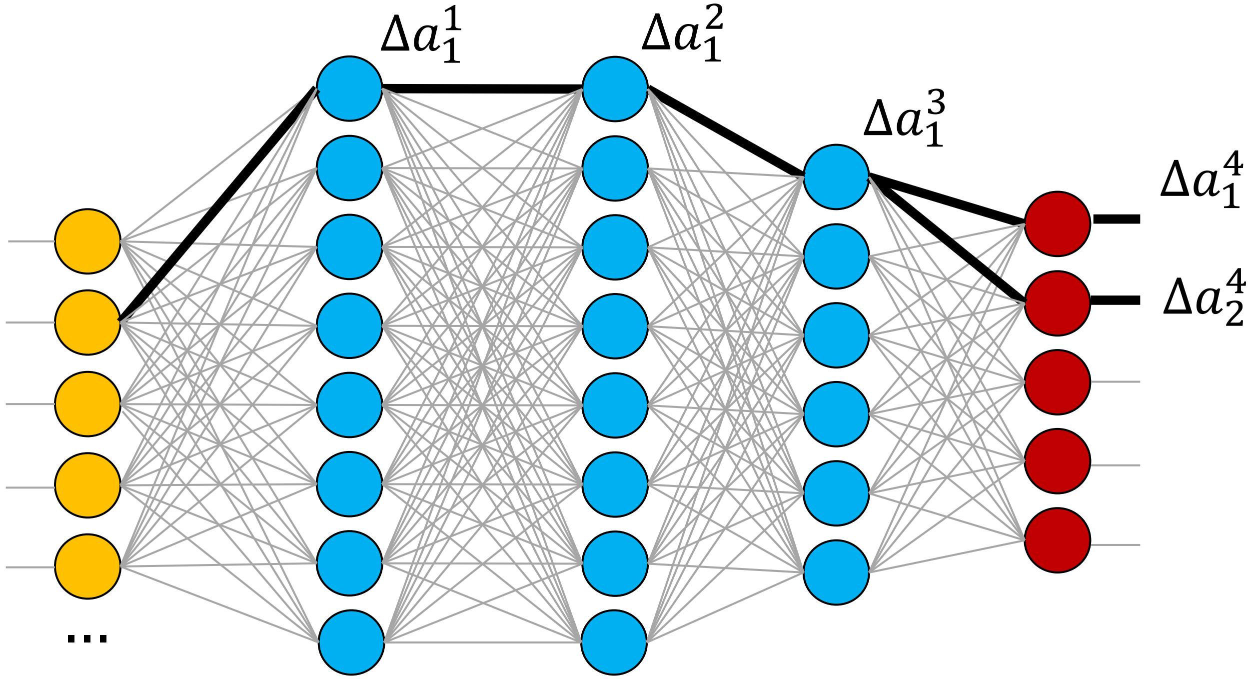
Be aware of the network structure!

- Let‘s imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron
  - ... this triggers updates of all subsequent layers / neurons
  - ... and eventually the result of the cost function changes



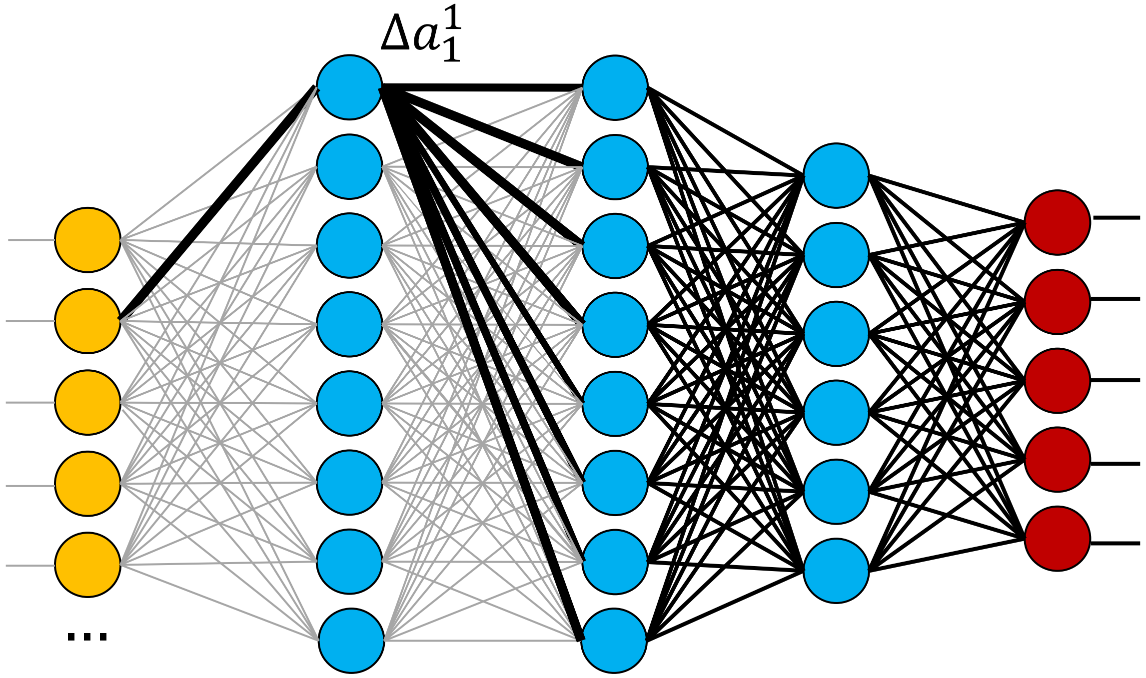
Be aware of the network structure!

- Let‘s imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron
  - ... this triggers updates of all subsequent layers / neurons
  - ... and eventually the result of the cost function changes



Be aware of the network structure!

- Let‘s imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron
  - ... this triggers updates of all subsequent layers / neurons
  - ... and eventually the result of the cost function changes

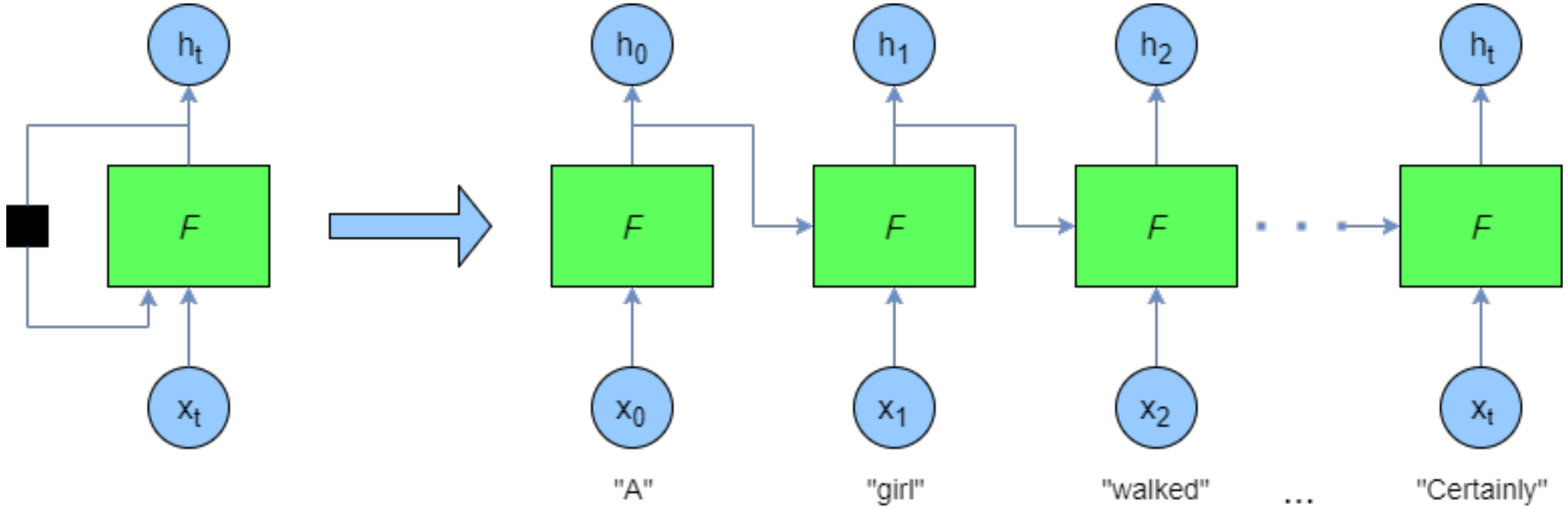


Be aware of the network structure!

- Let's imagine we make a small change to some weight
  - This leads to a change in the activation of the subsequent neuron
  - ... this triggers updates of all subsequent layers / neurons
  - ... and eventually the result of the cost function changes
- We have to consider all paths, if changing some weight, to calculate the impact of the change to the cost function
- Solution: Backpropagation algorithm
  - Got famous 1986 by paper of Rummelhart, Hinton and Williams
  - Efficient algorithm to compute partial derivates $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_l}$
  - Only use one forward and one backward to calculate parameter updates

Recurrent neural networks (RNN)

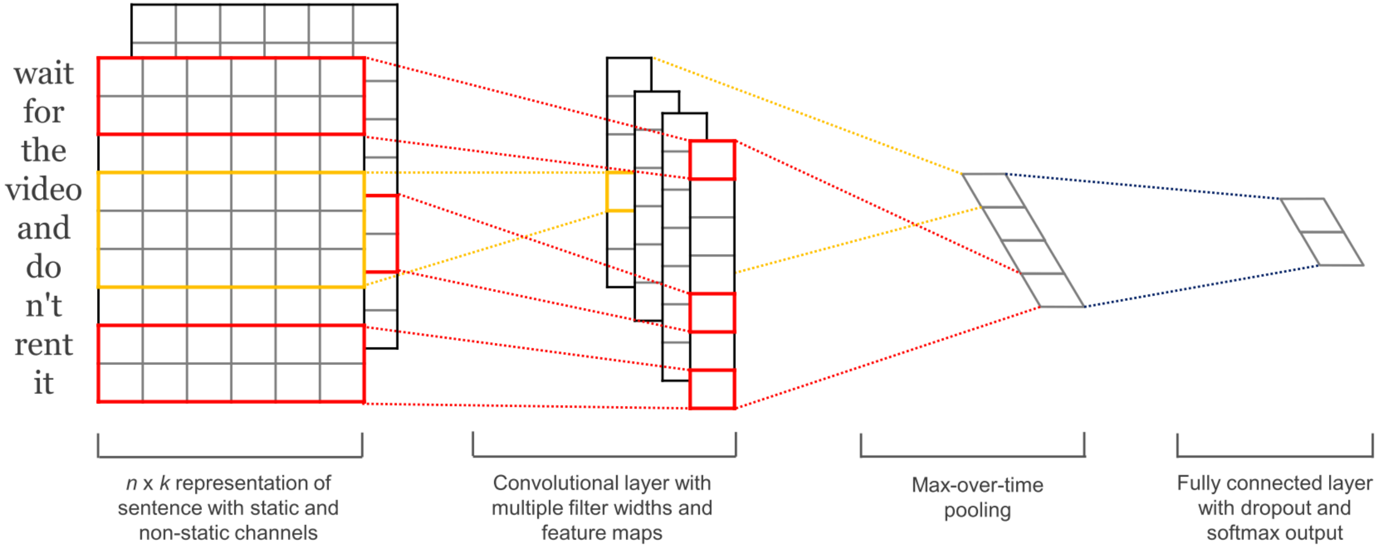
- RNNs: Class of ANNs with connections between nodes form a directed graph along a temporal sequence
  - Contain feedback loops to explicitly model temporal dynamic behaviour (e.g. the sequence of words in a sentence)



https://adventuresinmachinelearning.com/wp-content/uploads/2017/09/Recurrent-neural-network.png

Convolutional neural networks (CNNs)

- CNNs are regularized versions of multi-layer perceptrons
  - Convolutional layers convolve the input and pass its result to the next layer
  - Advantage: shared-weight architecture and translation invariant



Topic presentation

Overview

Nr.	Typ	Thema	Schwierigkeit
1	STL	(Bio-) BERT for Relation Extraction *Lin, Chen, et al. "A BERT-based Universal Model for Both Within-and Cross-sentence Clinical Temporal Relation Extraction." Proceedings of the 2nd Clinical Natural Language Processing Workshop. 2019.*	Niedrig-Mittel
2	STL	(Bio-) ELMO for Relation Extraction *Peng, Yifan, Shankai Yan, and Zhiyong Lu. "Transfer Learning in Biomedical Natural Language Processing: An Evaluation of BERT and ELMo on Ten Benchmarking Datasets." arXiv preprint arXiv:1906.05474 (2019).*	Mittel
3	STL	CNNs with pre-trained biomedical word embeddings *Liu, Shengyu, et al. "Drug-drug interaction extraction via convolutional neural networks." Computational and mathematical methods in medicine 2016 (2016).*	Niedrig
4	STL	CNN(+RNN) with biomedical word embeddings *Vu, Ngoc Thang, et al. "Combining recurrent and convolutional neural networks for relation classification." arXiv preprint arXiv:1605.07333 (2016).*	Niedrig-Mittel

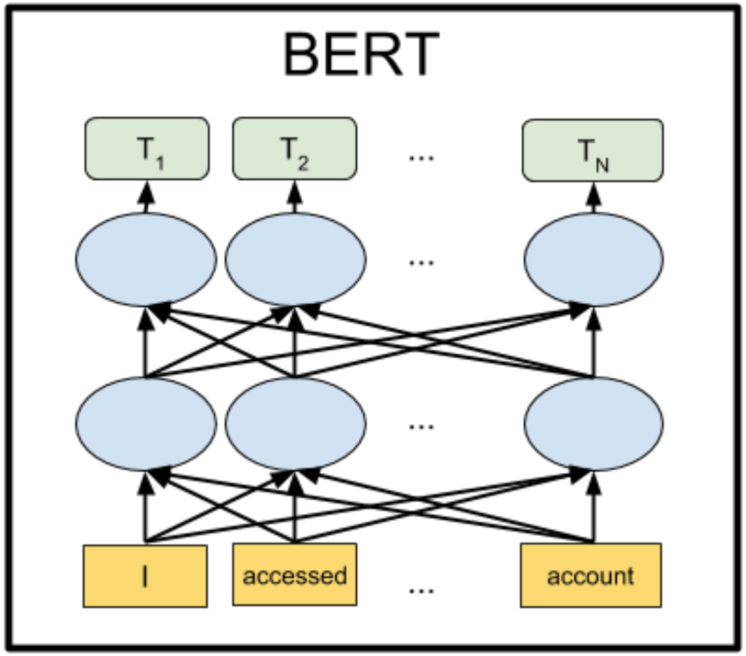
Overview

5	MTL	Hierarchical multi-task learning *Sanh, Victor, Thomas Wolf, and Sebastian Ruder. "A hierarchical multi-task approach for learning embeddings from semantic tasks." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019.*	Hoch
6	STL	Pre-trained transformer networks for RE *Alt, Christoph, Marc Hübner, and Leonhard Hennig. "Improving relation extraction by pre-trained language representations." arXiv preprint arXiv:1906.03088 (2019).*	Mittel-Hoch
7	STL	CNN with pre-trained biomedical word embeddings and linguistic features *Choi, Sung-Pil. "Extraction of protein–protein interactions (PPIs) from the literature by deep convolutional neural networks with various feature embeddings." Journal of Information Science 44.1 (2018): 60-73.*	Hoch

T1: (Bio-BERT) for Relation Extraction

- BERT: State-of-the-art pre-trained language model
  - Different models pre-trained on Wikipedia or biomedical text available

- Task and challenges:
  - Evaluate BERT for biomedical RE
  - Compare BERT models pre-trained on different corpora (in- vs. out-of-domain)
  - Complex model, but good (easy to adapt) implementations available

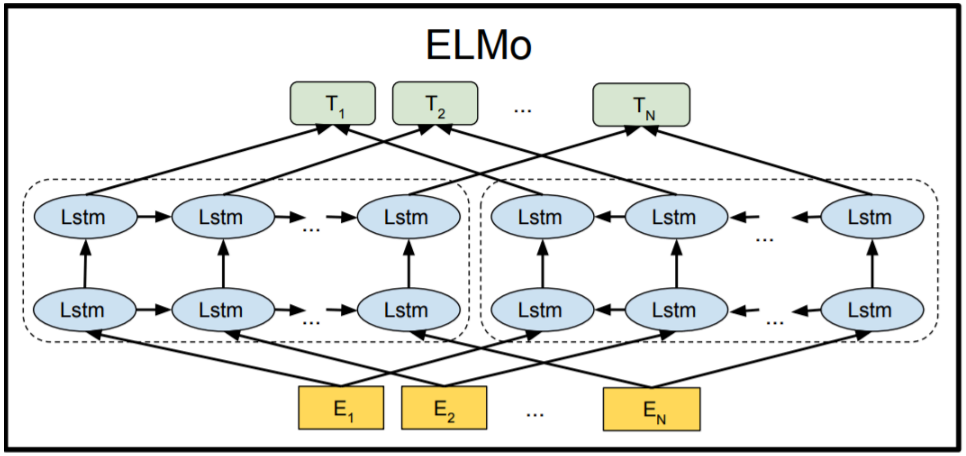


https://habrastorage.org/getpro/habr/post_images/2bd/0ba/1c4/2bd0ba1c4fb80fe4d771f555168c9ff0.png

http://www.charliepunk.de/WebRoot/Store19/Shops/61747719/5228/9113/D83C/F82D/1BDD/C0A8/28BB/6D9B/kuehlschrankmagnet-bert-sesamstrasse_ml.jpg

T2: (Bio-) ELMo for Relation Extraction

- ELMo: State-of-the-art pre-trained language model

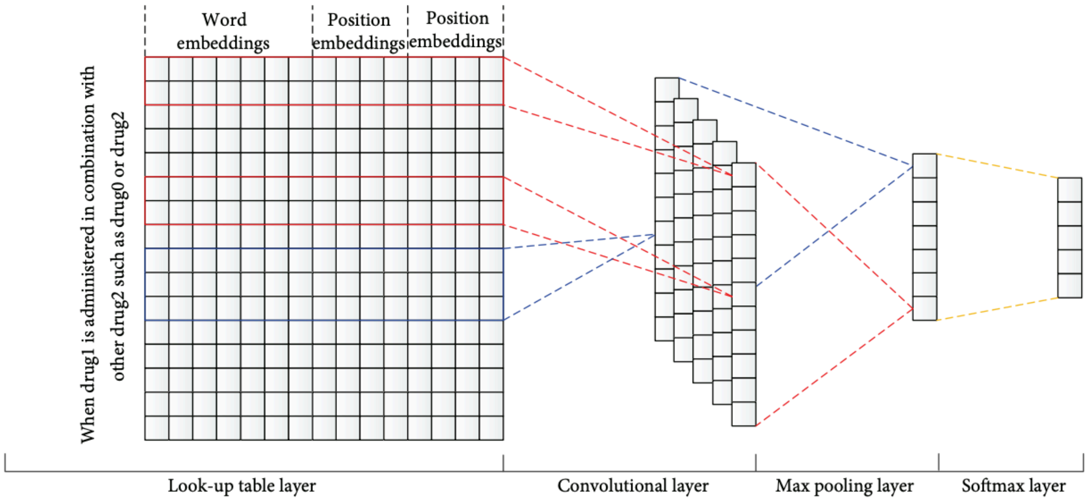


https://miro.medium.com/max/2128/1*ko2Ut74J_oMxF4jSo1VnCg.png

- Task and challenges:
  - Evaluate and compare ELMo models pre-trained on different corpora (in- vs. out-of-domain)
  - Complex model, but good (easy to adapt) implementations available

T3: CNN with pre-trained biomed. embeddings

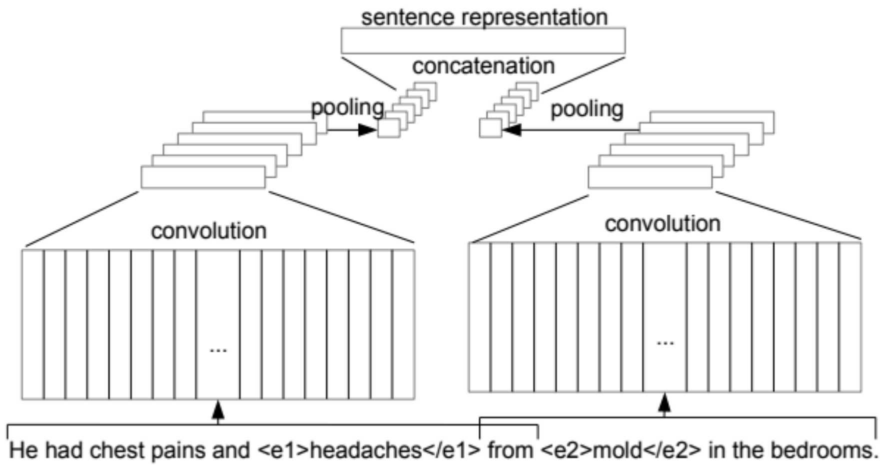
- Liu et al. use CNNs with word and positional embeddings



- Task and challenges
  - Get familiar with CNNs and re-implement the approach
  - Evaluate the benefit of pre-trained biomedical embeddings

T4: CNN with biomedical embeddings

- Vu et al. use separate CNNs for different parts of the sentence

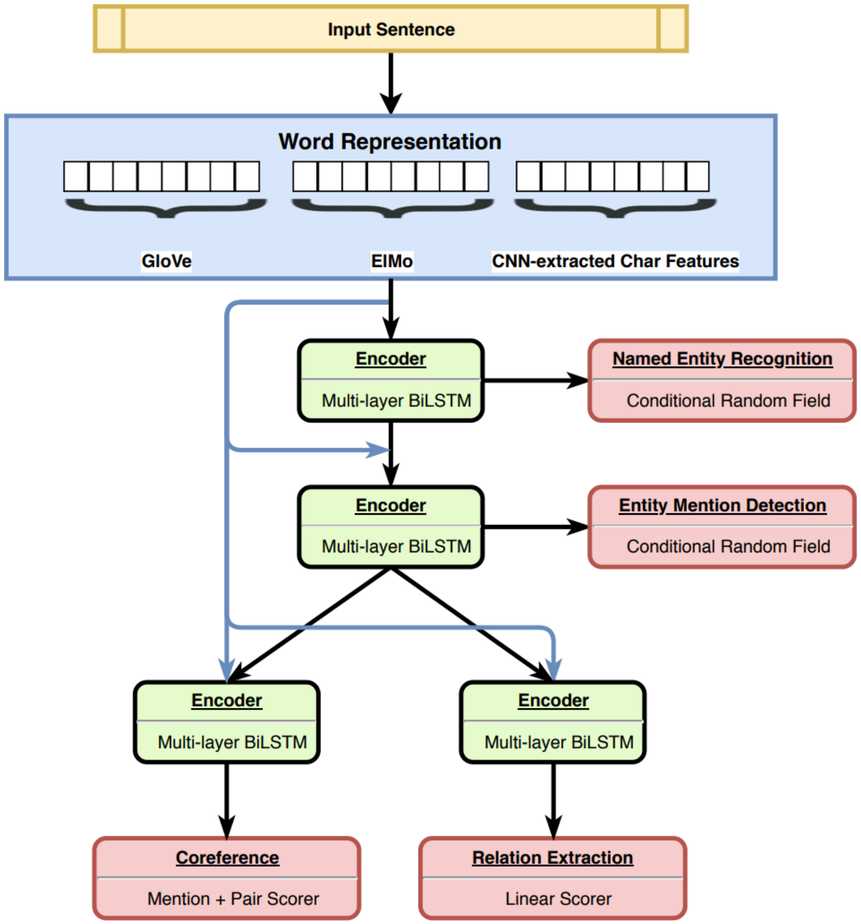


- Task and challenges
  - Get familiar with CNNs and re-implement the approach
  - Evaluate the benefit of pre-trained biomedical embeddings

T5: Hierarchical multi-task learning

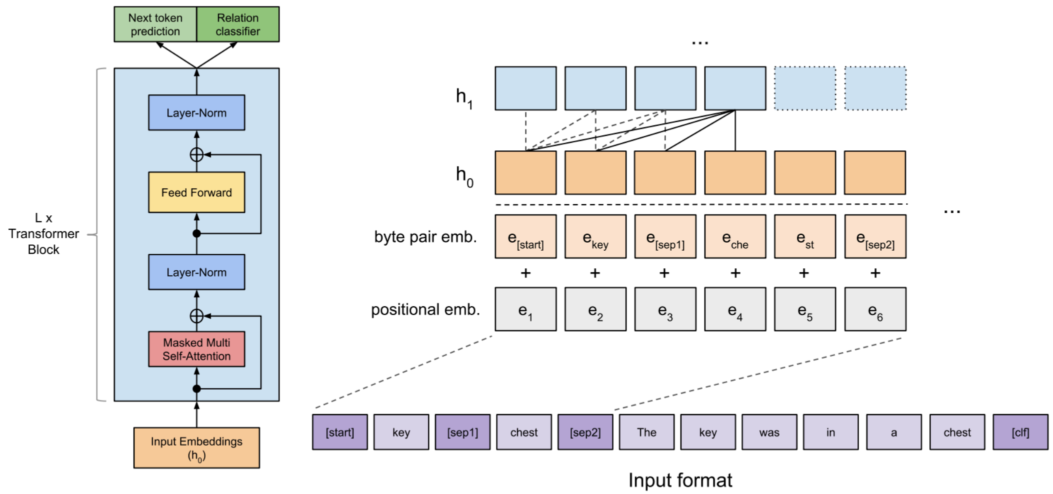
- Sanh et al. propose a hierarchical multi-task architecture
  - Network learns to perform NER, EMD, CoRef, RE simultaneously

- Task and challenges:
  - Re-implement and adapt approach for biomedical RE
  - Find suitable auxiliary tasks and data



T6: Pre-trained transformer networks for RE

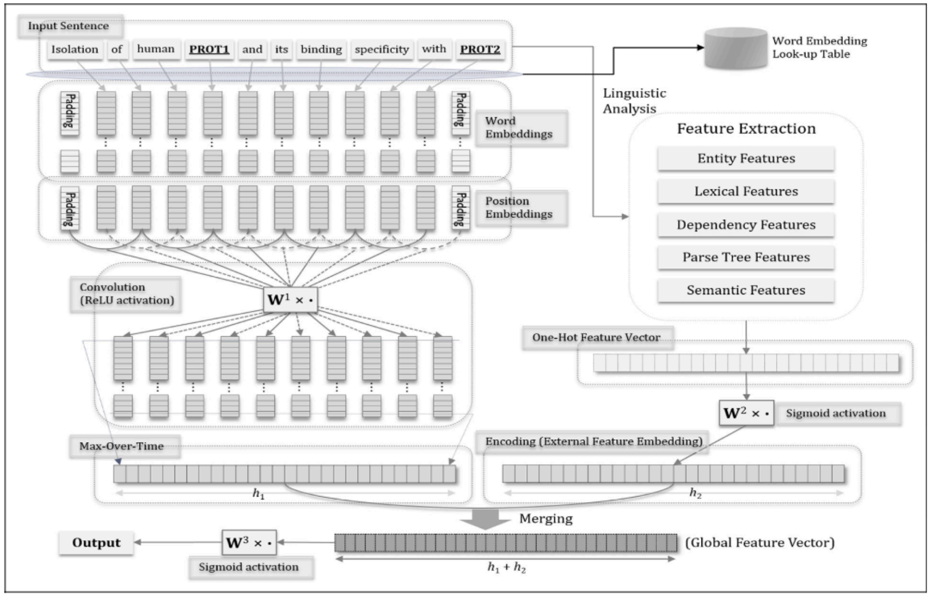
- Alt et al. use a pre-trained transformer model and adapt it to relation extraction



- Task and challenges:
  - Evaluate the transformer model for biomedical RE
  - Complex model, but good (easy to adapt) implementations available

T7: CNN with word embeddings and ling. features

- Choi uses CNNs with word embeddings and hand-crafted linguistic features for RE



- Task and challenges
  - Get familiar with CNNs and re-implement the approach
  - Evaluate the benefit of pre-trained biomedical embeddings

Questions?

Thank you for your attention!