Statistical Natural Language Processing

Neural networks

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

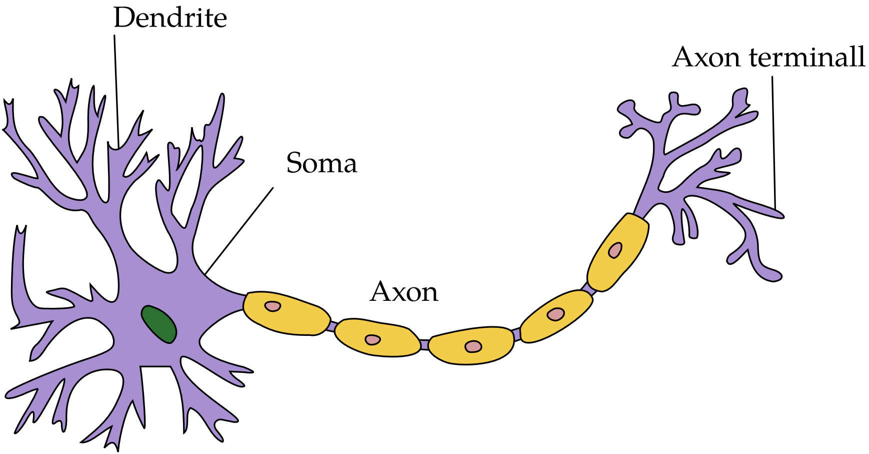
Summer Semester 2017

Artificial neural networks

- Artificial neural networks (ANNs) are machine learning models inspired by biological neural networks
- ANNs are powerful non-linear models
- Power comes with a price: there are no guarantees of finding a global minimum of the error function
- ANNs have been used in ML, AI, Cognitive science since 1950's – with some ups and downs
- Currently they are the driving force behind the popular '*deep learning*' methods

The biological neuron

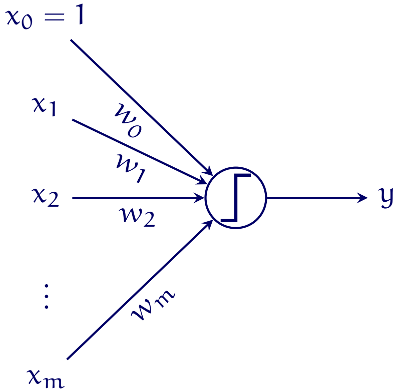
(showing a picture of a real neuron is mandatory in every ANN lecture)



Artificial and biological neural networks

- ANNs are *inspired* by biological neural networks
- Similar to biological networks, ANNs are made of many simple processing units
- Despite the similarities, there are many differences ANNs do not mimic biological networks
- ANNs are a practical statistical machine learning methods

Recap: the perceptron



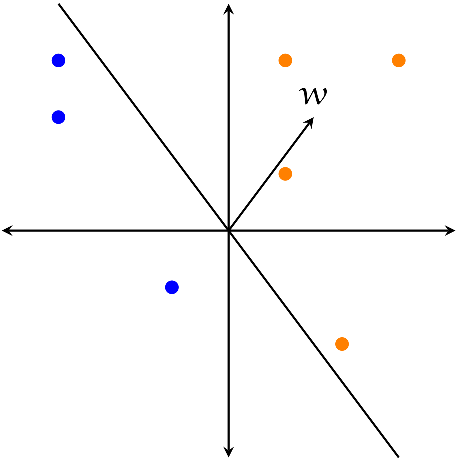
$$y = f\left(\sum_{j}^{m} w_j x_j\right)$$

where

$$f(x) = \begin{cases} +1 & \text{if} \quad \boldsymbol{wx} > 0 \\ -1 & \text{otherwise} \end{cases}$$

where $\boldsymbol{x} = (x_1, \ldots, x_m)$,
$\boldsymbol{w} = (w_1, \ldots, w_m)$
In ANN-speak $f(\cdot)$ is called an *activation function*.

Recap: perceptron algorithm



- Perceptron algorithm minimizes the function

$$J(w) = \sum_i \max(0, -\boldsymbol{w}\boldsymbol{x}_i y_i)$$

- The online version picks an misclassified example, and sets

$$w \leftarrow w + \boldsymbol{x}_i y_i$$

- Algorithm is guaranteed to converge if classes are linearly separable

Recap: logistic regression

- Logistic regression estimates $P(y \mid \boldsymbol{x})$

$$P(y \mid \boldsymbol{x}) = p^{y}(1-p)^{1-y} \quad \text{where } p = \frac{1}{1+e^{-\boldsymbol{w}\boldsymbol{x}}}$$

- The function

$$\frac{1}{1+e^{-\boldsymbol{w}\boldsymbol{x}}}$$

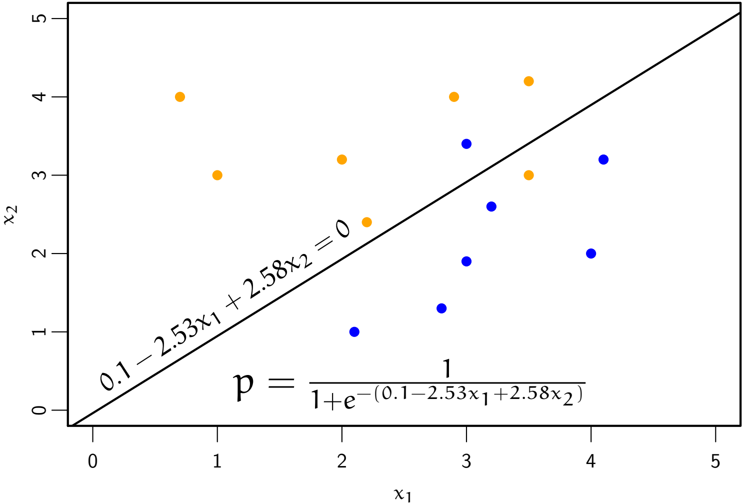
  is called the logistic sigmoid function

- To find the solution, we minimize the minus log likelihood

$$-\log \mathcal{L}(\boldsymbol{w}) = \sum_{i} y_i \log p + (1-y_i)\log(1-p)$$

- The above objective function is simply cross entropy

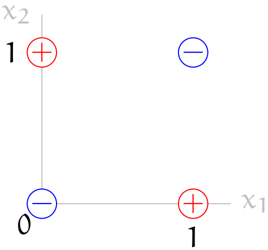
Logistic regression is also a linear classifier



Note: the decision boundary is $\boldsymbol{wx} = 0$

Linear separability

- A classification problem is said to be *linearly separable* if one can find a linear discriminator
- A well-known counter example is the logical XOR problem



There is no line that can separate positive and negative classes.

Can a linear classifier learn the XOR problem?

Can a linear classifier learn the XOR problem?

- We can use non-linear basis functions

$$w_0 + w_1 x_1 + w_2 x_2 + w_2 \phi(x_1, x_2)$$

  is still linear in w for any choice of $\phi(\cdot)$

- For example, adding the product $x_1 x_2$ as an additional feature would allow a solution like: $x_1 + x_2 - 2x_1 x_2$

x_1	x_2	$x_1 + x_2 - 2x_1x_2$
0	0	0
0	1	1
1	0	1
1	1	0

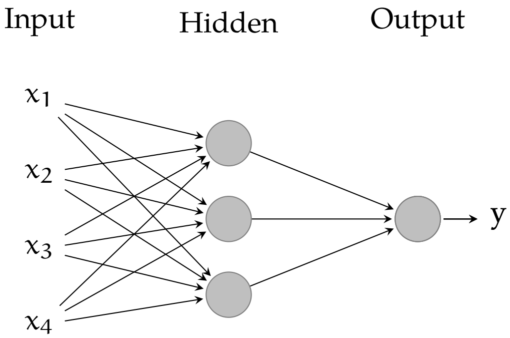
- Choosing proper basis functions like $x_1 x_2$ is called *feature engineering*

Multi-layer perceptron

- The simplest modern ANN architecture is called multi-layer perceptron (MLP)
- (MLP) is a *fully connected*, *feed-forward* network consisting of perceptron-like units
- Unlike classical perceptron, the units in an MLP use a continuous activation function
- The MLP can be trained using gradient-based methods
- The MLP can represent many interesting machine learning problems
  - It can be used for both regression and classification

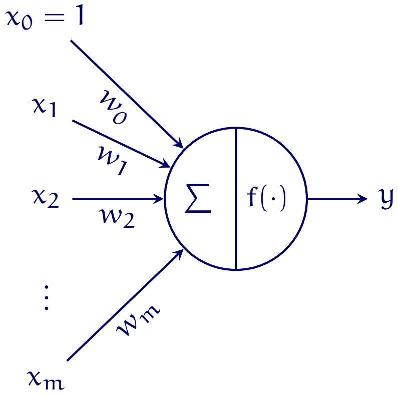
Multi-layer perceptron

the picture



Each unit takes a weighted sum of their input,
and applies a (non-linear) *activation function*.

An artificial neuron



- The unit calculates a weighted sum of the inputs

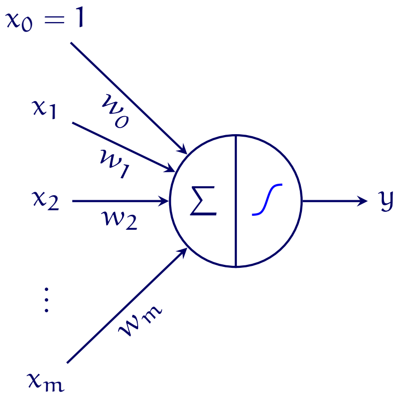
$$\sum_j^m w_j x_j = \boldsymbol{wx}$$

- Result is a linear transformation
- Then the unit applies a non-linear activation function $f(\cdot)$
- Output of the unit is

$$y = f(\boldsymbol{wx})$$

Artificial neuron

an example



- A common activation function is *logistic sigmoid* function

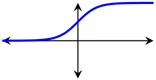
$$f(x) = \frac{1}{1 + e^{-x}}$$

- The output of the network becomes

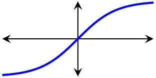
$$y = \frac{1}{1 + e^{-\boldsymbol{w}\boldsymbol{x}}}$$

Activation functions in ANNs
hidden units

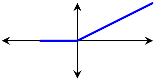
- The activation functions in MLP are typically continuous (differentiable) functions
- For hidden units common choices are



Sigmoid (logistic) $\frac{1}{1+e^{x}}$



Hyperbolic tangent (tanh) $\frac{e^{2x}-1}{e^{2x}+1}$



Rectified linear unit (relu) $max(0, x)$

Activation functions in ANNs

output units

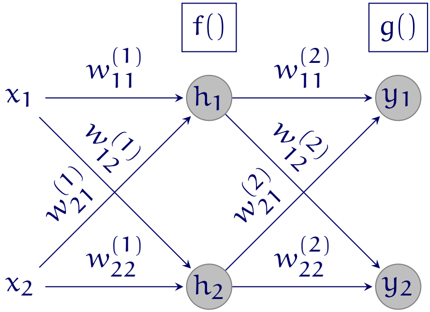
- The activation functions of the output units depends on the task
  - For regression, identity function
  - For binary classification, logistic sigmoid

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-\boldsymbol{w}\boldsymbol{x}}} = \frac{e^{\boldsymbol{w}\boldsymbol{x}}}{1 + e^{-\boldsymbol{w}\boldsymbol{x}}}$$

  - For multi-class classification, softmax

$$P(y = k \mid x) = \frac{e^{\boldsymbol{w}_k \boldsymbol{x}}}{\sum_j e^{\boldsymbol{w}_j \boldsymbol{x}}}$$

MLP: a simple example

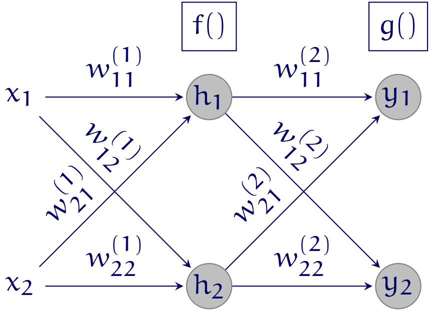


$$h_j = f\left(\sum_i w_{ij} x_i\right)$$

$$y_k = g\left(\sum_j w_{jk} h_j\right)$$

$$y_k = g\left(\sum_j w_{jk} f\left(\sum_i w_{ij} x_i\right)\right)$$

MLP: a simple example



- Alternatively, we can write the computations in matrix form

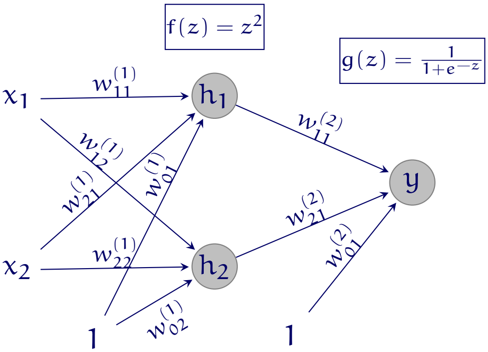
$$\mathbf{h} = f(W^{(1)}\mathbf{x})$$

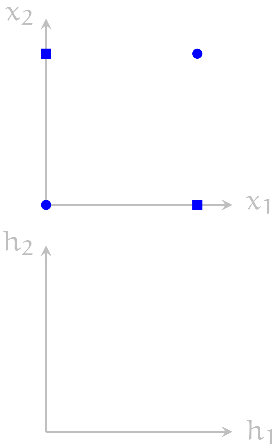
$$\begin{aligned}\mathbf{y} &= g(W^{(2)}\mathbf{h}) \\ &= g\left(W^{(2)}f(W^{(1)}\mathbf{x})\right)\end{aligned}$$

- This corresponds to a series of transformations followed by element-wise (non-linear) function applications

Solving non-linear problems with ANNs

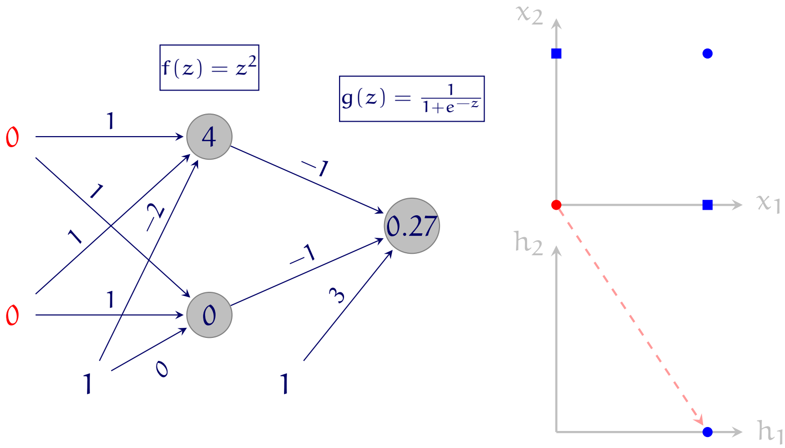
a solution to XOR problem





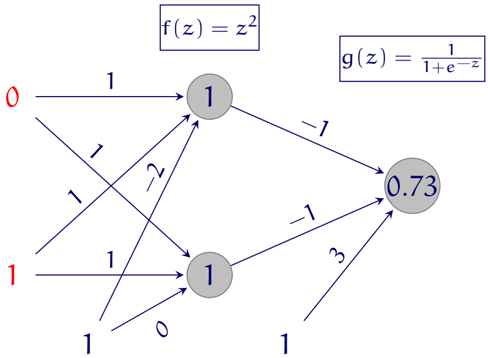
Solving non-linear problems with ANNs

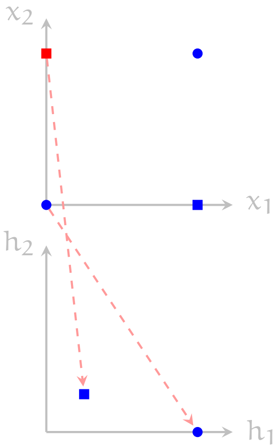
a solution to XOR problem



Solving non-linear problems with ANNs

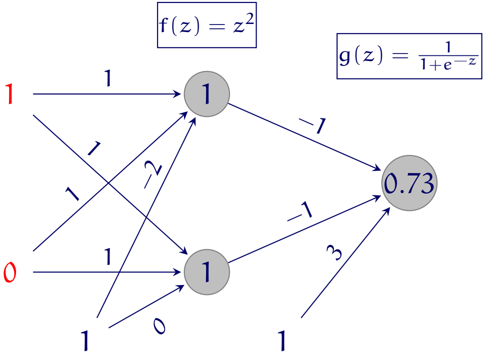
a solution to XOR problem

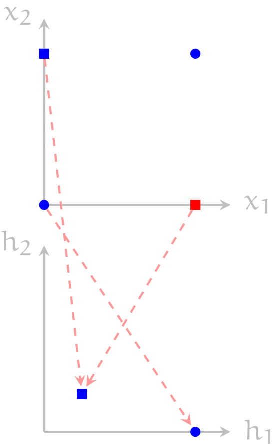




Solving non-linear problems with ANNs

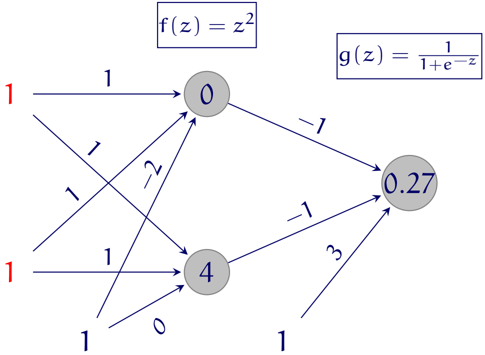
a solution to XOR problem

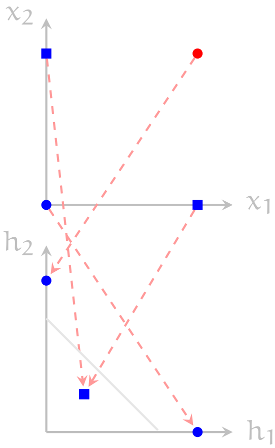




Solving non-linear problems with ANNs

a solution to XOR problem

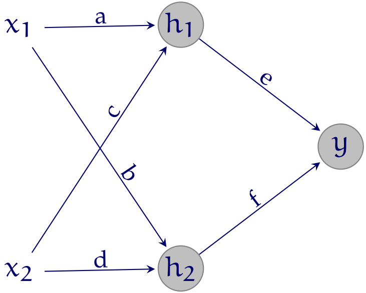




Is this different from non-linear basis functions?

Non-linear activation functions are necessary

Without non-linear activation functions, the ANN is equivalent to a linear model.



$$
\begin{aligned}
h_1 &= ax_1 + cx_2 \\
h_2 &= bx_1 + dx_2 \\
y &= eh_1 + fh_2 \\
&= (ea + fb)x_1 + (ec + fd)x_2
\end{aligned}
$$

y is still a linear function of x_i

Where do non-linearities come from?

non-linearities are abundant in nature, it is not only the XOR problem

In a linear model, $y = w_0 + w_1 x_1 + \ldots + w_k x_k$

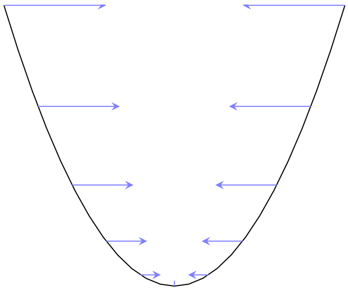
- The outcome is *linearly-related* to the predictors
- The effects of the inputs are *additive*

This is not always the case:

- Some predictors affect the outcome in a non-linear way
  - The effect may be strong or positive only in a certain range of the variable (e.g., reaction time change by age)
  - Some effects are periodic (e.g., many measures of time)
- Some predictors interact
  '*not bad*' is not '*not*' + '*bad*' (e.g., for sentiment analysis)

Finding the minimum of a loss functions

- Derivative of a function points to the largest direction of change
- Derivative is 0 at minima/maxima
- To find the minimum (or maximum) of error function $f(x)$, we solve $f'(x) = 0$, for x
- If no analytic solution exist, we search for the minimum iteratively
- $-f'(x)$ for any x points towards the minimum



Gradient descent: a refresher

- The general idea is to approach a minimum of the error function in small steps

$$\boldsymbol{w} \leftarrow w - \eta \nabla J(\boldsymbol{w})$$

  - ∇J is the gradient of the loss function, it points to the direction of the maximum increase
  - η is the learning rate

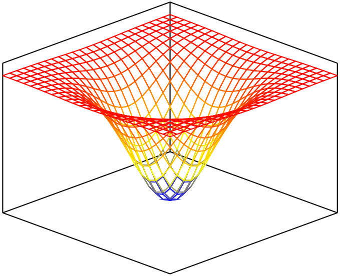
- The updates can be performed

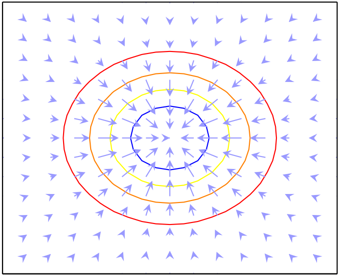
batch for the complete training set

on-line after every training instance
– this is known as *stochastic gradient descent* (SGD)

mini-batch after small fixed-sized batches

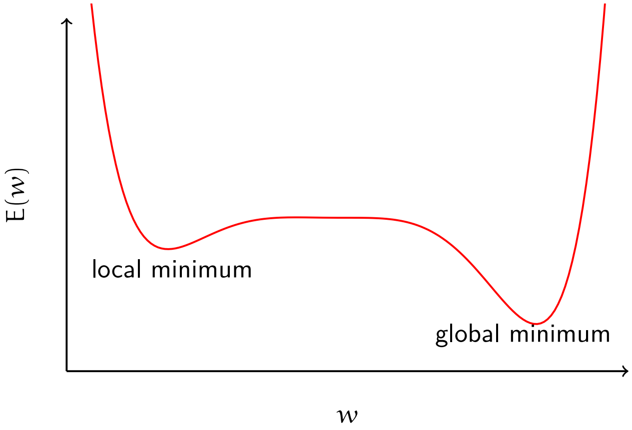
Gradient descent: the picture





$$\nabla f(x_1, \ldots, x_n) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

Global and local minima



A function is *convex* if there is only one global minimum.

Error functions in ANN training

depends on the task

- If we assume Gaussian noise, a natural choice is the minimizing the sum of squared error

$$E(w) = \sum_i (y_i - \hat{y}_i)^2$$

- For binary classification, we use *cross entropy*

$$E(w) = -\sum_i y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

- Similarly, for multi-class classification, also cross entropy

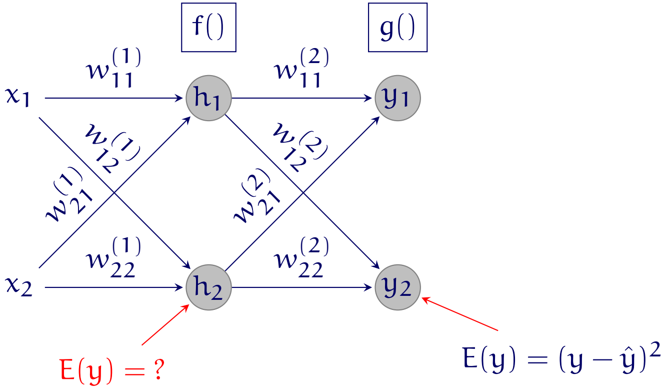
$$E(w) = -\sum_i \sum_k y_{i,k} \log \hat{y}_k$$

In any practical ANN, the loss function will not be convex.

Learning in ANNs

- ANNs implement complex functions: we need to use optimization methods (e.g., gradient descent) to train them
- Typically error functions for ANNs are not convex, gradient descent will find a local minimum
- Optimization requires updating multiple layers of weights
- Assigning credit (or blame) to each weight during learning is not trivial
- An effective solution to the last problem is the *backpropagation* algorithm

Learning in multi-layer networks: the problem



We want a way to update non-final weights based on final error.

Backpropagation

- The final output of the network is computed by calculating the output of each layer and passing it to the next (*forward propagation*)
- Weight updates on the final layer is easy: we need the relevant component of the gradient:

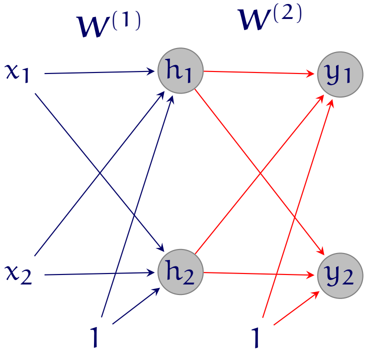
$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}}$$

- For the non-final weights we make use of chain rule of derivatives

$$\text{if } F(w) = f(g(w)), \qquad F'(x) = f'(g(w))g'(w)$$

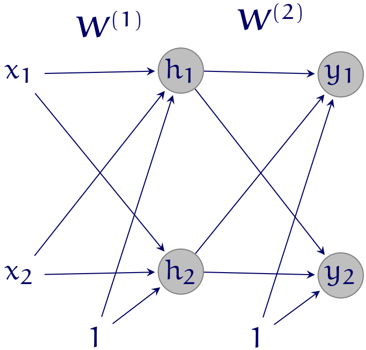
- Backpropagation propagates the error from output units to the input weights using the chain rule of derivatives

Backpropagation: visualization



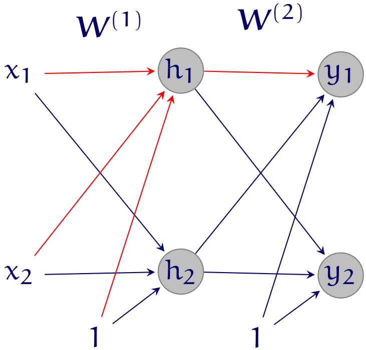
- Updating weights $W^{(2)}$ are easy: we can use gradient descent directly

Backpropagation: visualization



- Updating weights $W^{(2)}$ are easy: we can use gradient descent directly
- We update weights $W^{(1)}$ using the chain rule

Backpropagation: visualization



- Updating weights $W^{(2)}$ are easy: we can use gradient descent directly
- We update weights $W^{(1)}$ using the chain rule
- Backpropagation algorithm uses dynamic programming to do this efficiently

Regularization in neural networks

- As in linear models, we can use L1 and L2 regularization by adding a regularization term to the error function (known as weight decay). For example,

$$J(w) = E(w) + \|\boldsymbol{W}\|$$

- There are other ways to fight overfitting
  - With *early stopping*, one stops the training before it reaches to the smallest training error
  - With *dropout*, random units (with all of their connections) are dropped during training
  - Injecting noise at the output, as a way to (implicitly) model the noise in the target classes/values

Adapting learning rate

- The choice of learning rate η is important

too small slow convergence

too big overshooting - may fluctuate around the minimum, or even jump away

- The idea is adapt the learning rate during learning
- A common trick is adding a momentum: if we move in the same direction a long time accelarate

$$\Delta w_{ij}(t) = \eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1)$$

- There are many adaptive optimization algorithms: Adagrad, Adadelta, RMSprop, Adam, …

How many layers, units

- A network with single hidden layer is said to be *a universal approximator*: it can approximate any continuous function with arbitrary precision
- However, in practice multiple interconnected layers are useful and commonly used in modern ANN models
- The choice of layers, in general the architecture of the system, depends on the application

A bit of history

1950-60 ANNs (perceptron) become popular:
lots of excitement in AI, cognitive science

1970s Not much interest

- criticims on perceptron: linear separability

1980s ANNs became popular again

- backpropagation algorithm
- multi-layer networks

1990s ANNs had again fallen ‘out of fashion’

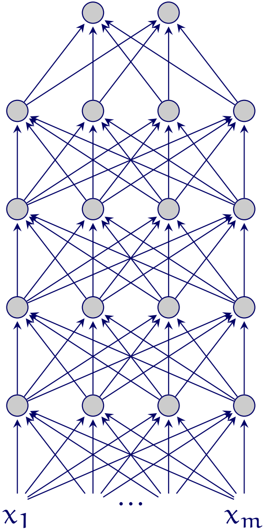
- Engineering: other algorithms (such as SVMs) performed generally better
- From the cognitive science perspective: ANNs are difficult to interpret

present ANNs (again) enjoy popularity and popularity with the name ‘deep learning’

Summary, so far…

- ANNs are non-linear machine learning methods
- they can be used for both regression and classification
- they are trined with *backpropaation* algorithm
- ANN loss functions are not convex, what we find is a local minimum

Deep feed-forward networks



- Deep neural networks (>2 hidden layers) have recently been successful in many tasks
- They are particularly useful in problems where layers/hierarchies of features are useful
- They often use sparse connectivity and shared weights
- We will review two important architectures: CNNs and RNNs

Training deep networks
difficulties

- Training deep netowrks is more difficult
- A common practical problem is unstable gradients: the gradients may vanish, or explode
- Often we have lots of hyper parameters:
  - the number of layers
  - For each layer:
    - what architecture to use (dense, CNN, RNN, …)
    - activation function(s)
    - regularizaiton method / parameters
    - optimiaztion algorithm
    - initialization
    - …

Why now?

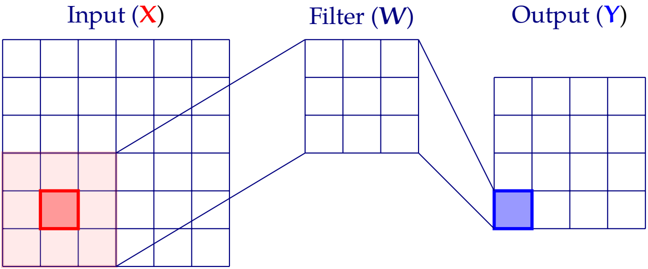
- Increased computational power, especially advances in graphical processing unit (GPU) hardware
- Availability of large amounts of data
  - mainly unlabeled data (more on this later)
  - but also labeled data through 'crowd sourcing' and other sources
- Some new developments in theory and applications

Convolutional networks

- Convolutional networks are particularly popular in image processing applications
- They have also been used with success some NLP tasks
- Unlike feed-forward networks we have discussed so far,
  - CNNs are not fully connected
  - The hidden layer(s) receive input from only a set of neighboring units
  - Some weights are shared
- A CNN learns features that are *location invariant*
- CNNs are also computationally less expensive compared to fully connected networks

Convolution in image processing

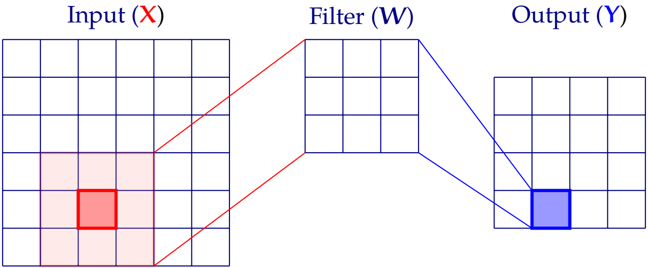
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

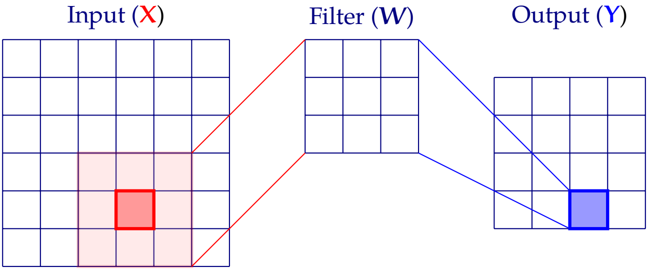
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

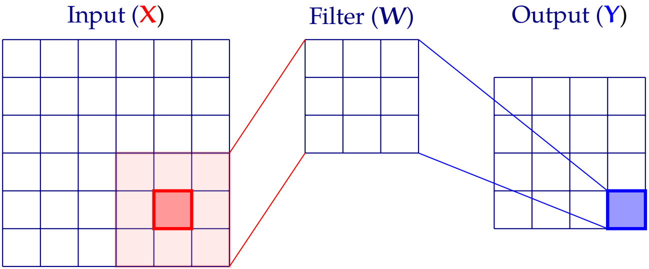
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

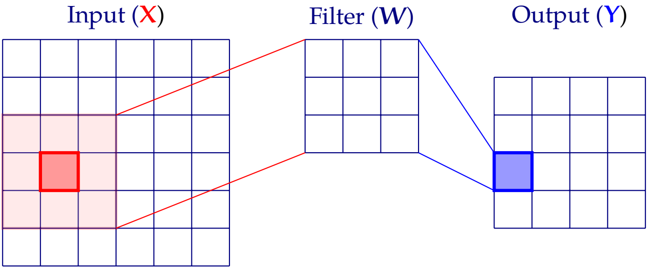
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

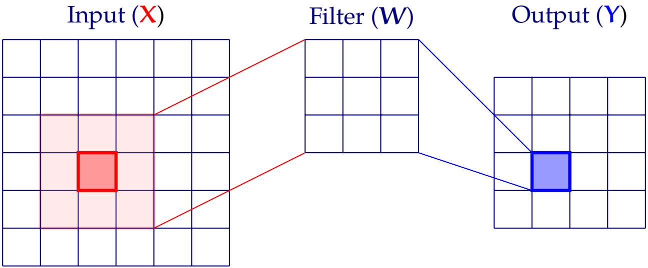
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

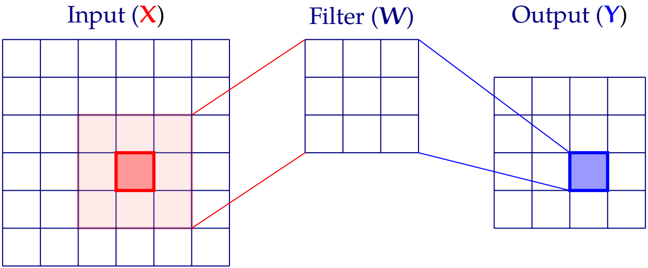
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

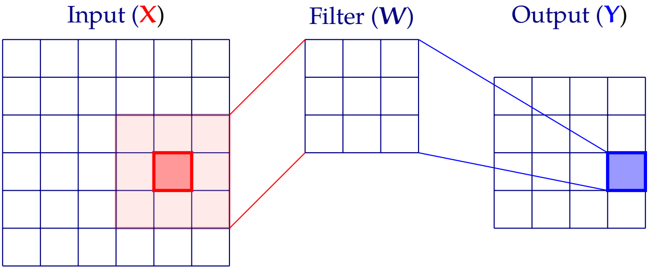
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

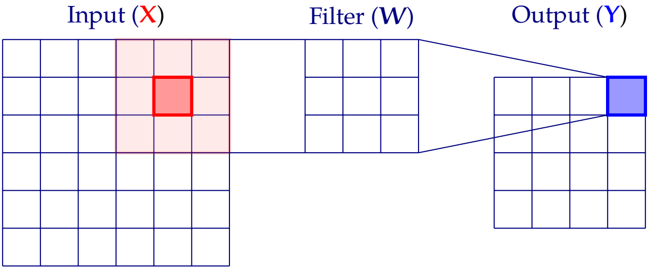
- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Convolution in image processing

- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, …
- The idea is to transform each pixel with a function of the local neighborhood



$$y = \sum_i w_i x_i$$

Example convolutions

- Blurring

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

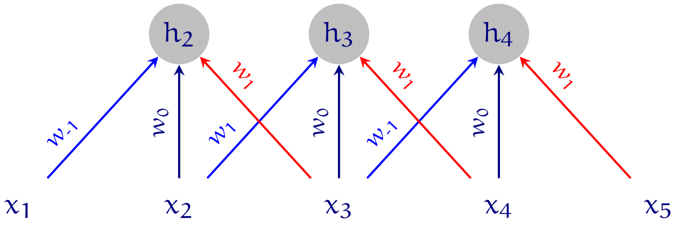
- Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Learning convolutions

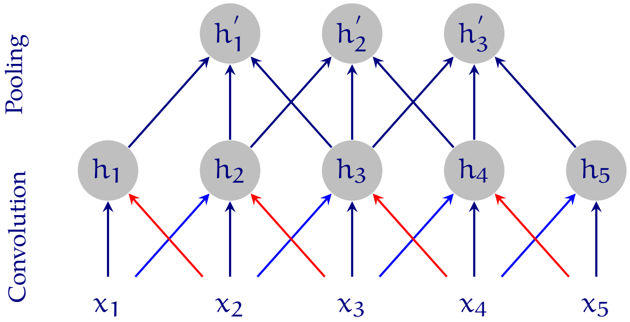
- Some filters produce features that are useful for classification (e.g., images, or sentences)
- In machine learning we want to *learn* the convolutions
- Typically, we learn multiple convolutions, each resulting in a different feature map
- Repeated application of convolutions allow learning higher level features
- The last layer is typically a standard fully-connected classifier

Convolution in neural networks



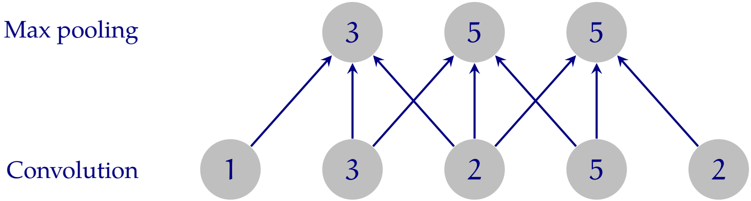
- Each hidden layer corresponds to a local window in the input
- Weights are shared: each convolution detects the same type of features

Pooling



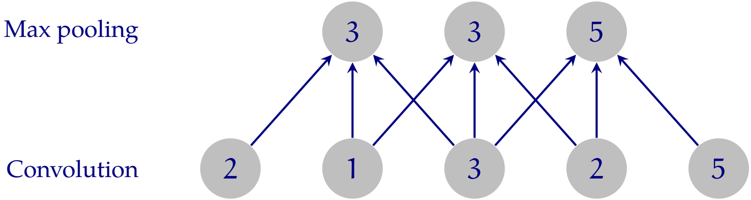
- Convolution is combined with *pooling*
- Pooling 'layer' simply calculates a statistic (e.g., max) over the convolution layer
- Location invariance comes from pooling

Pooling and location invariance



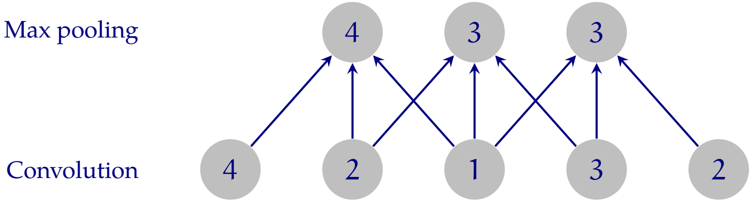
- Note that the numbers at the pooling layer are stable in comparison to the convolution layer

Pooling and location invariance



- Note that the numbers at the pooling layer are stable in comparison to the convolution layer

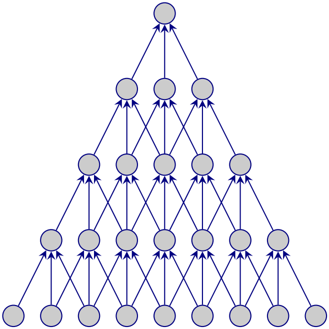
Pooling and location invariance



- Note that the numbers at the pooling layer are stable in comparison to the convolution layer

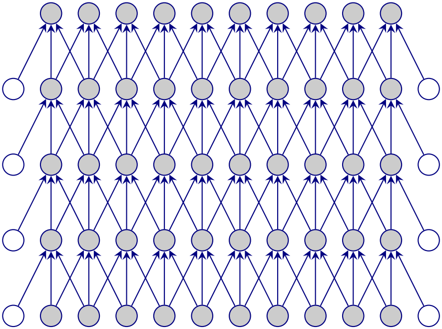
Padding in CNNs

- With successive layers of convolution and pooling, the size of the later layers shrinks



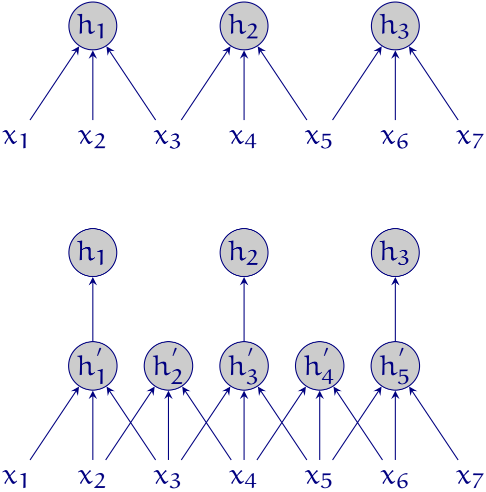
Padding in CNNs

- With successive layers of convolution and pooling, the size of the later layers shrinks
- One way to avoid this is *padding* the input and hidden layers with enough number of zeros



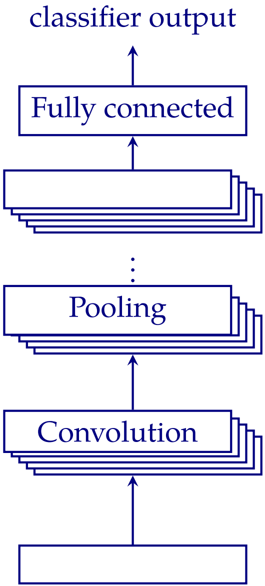
Strided convolution

- It is common to use *strided* convolution in some applications
- This is equivalent to convolution + down sampling

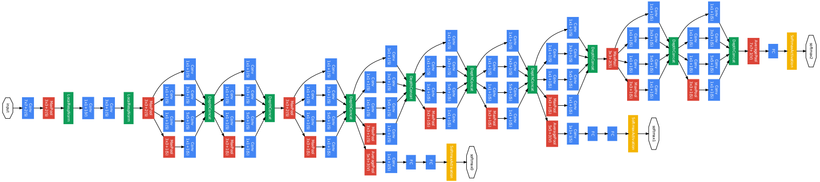


CNNs: the bigger picture

- At each convolution/pooling step, we often want to learn multiple feature maps
- After a (long) chain of hierarchical features maps, the final layer is typically a fully-connected layer (e.g., softmax for classification)



Real-world examples are complex



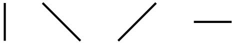
The real-world NNs tend to be complex (with some repetition)

- Many layers
- Large amount of branching

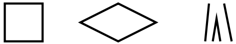
* Diagram describes an image classification network, GoogLeNet (Szegedy et al. 2014).

CNNs in natural language processing

- The use of CNNs in image applications is clear:
  - the first convolutional layer learns local features, e.g.,edges

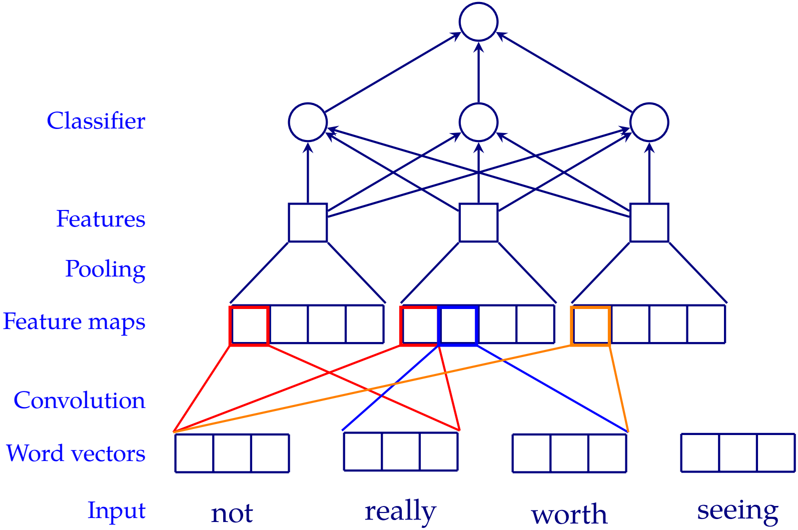


  - successive layers learn more complex features that are combinations of these features



- In NLP, it is a bit less straight-forward
  - CNNs are typically used in combination with word vectors
  - The convolutions of different sizes correspond to (word) ngrams of different sizes
  - With pooling, CNNs produce summaries of documents or sentences similar to BoW approach

An example: sentiment analysis



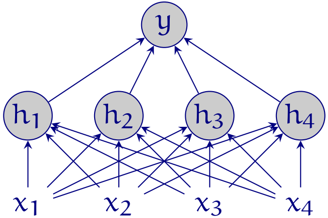
Convolutional networks: summary

- Convolutional networks use sparse connectivity with weight sharing
- The resulting network is computationally more efficient (compared to fully-connected networks)
- They are suitable for inputs with local features with (some) location invariance
- CNNs are very popular in image classification / object detection
- They are also used in NLP, particularly for document/sentence classification

Recurrent neural networks

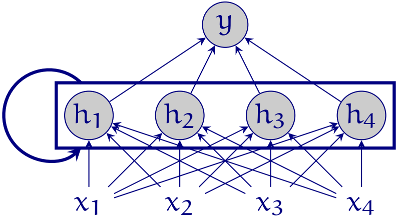
- Feed forward networks (also CNNs)
  - can only learn associations
  - they do not have memory of earlier inputs: they cannot handle sequences
- Recurrent neural networks are NN solution for sequence learning
- This is achieved by recursive loops in the network

Recurrent neural networks



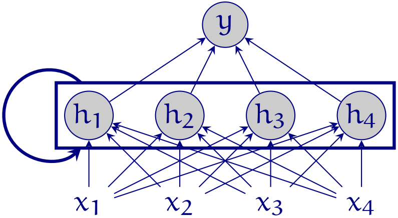
- Recurrent neural networks are similar to the standard feed-forward networks

Recurrent neural networks



- Recurrent neural networks are similar to the standard feed-forward networks
- They include loops that use previous output (of the hidden layers) as well as the input

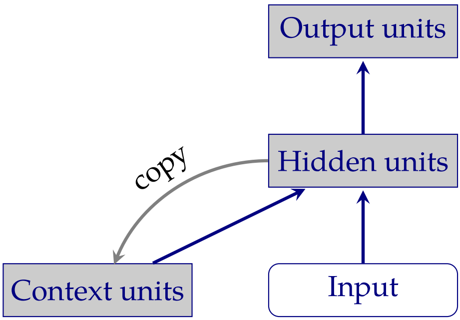
Recurrent neural networks



- Recurrent neural networks are similar to the standard feed-forward networks
- They include loops that use previous output (of the hidden layers) as well as the input
- Forward calculation is straightforward, learning becomes somewhat tricky

A simple version: SRNs

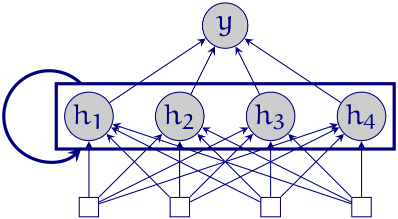
Elman (1990)



- The network keeps previous hidden states (context units)
- The rest is just like a feed-forward network
- Training is simple, but cannot learn long-distance dependencies

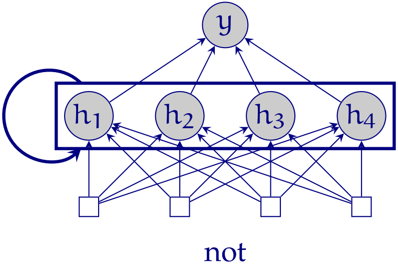
Processing sequences with RNNs

- RNNs process sequences one unit at a time
- The earlier input affects the output through the recurrent links



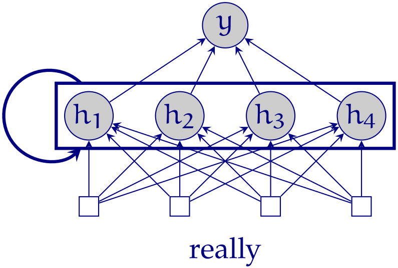
Processing sequences with RNNs

- RNNs process sequences one unit at a time
- The earlier input affects the output through the recurrent links



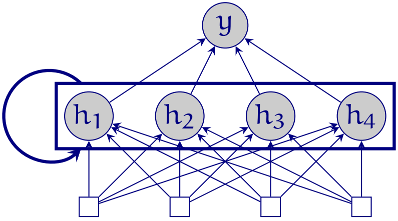
Processing sequences with RNNs

- RNNs process sequences one unit at a time
- The earlier input affects the output through the recurrent links



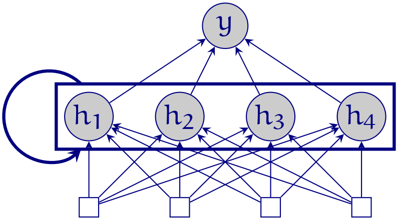
Processing sequences with RNNs

- RNNs process sequences one unit at a time
- The earlier input affects the output through the recurrent links

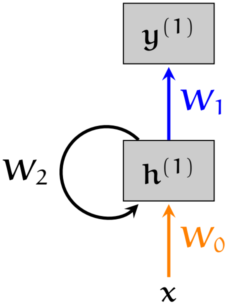


Processing sequences with RNNs

- RNNs process sequences one unit at a time
- The earlier input affects the output through the recurrent links



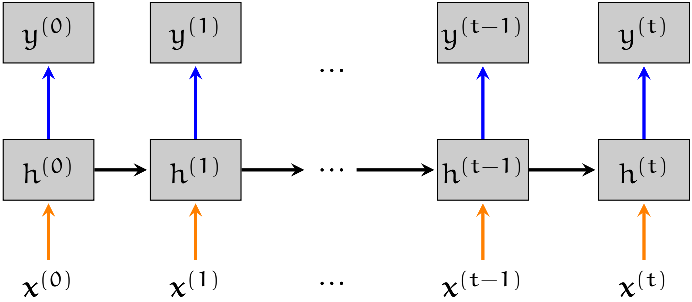
Learning in recurrent networks



- We need to learn three sets of weights: W_0, W_1 and W_2
- Backpropagation in RNNs are at first not that obvious
- The main difficulty is in propagating the error through the recurrent connections

Unrolling a recurrent network

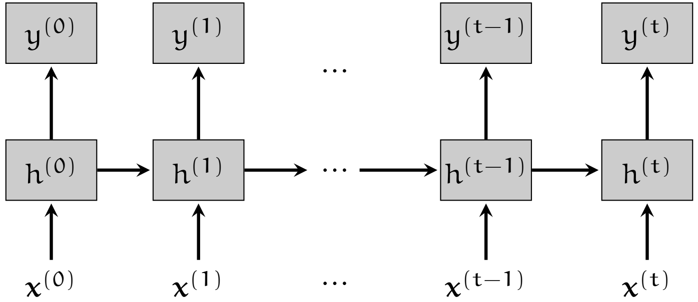
Back propagation through time (BPTT)



Note: the weights with the same color are shared.

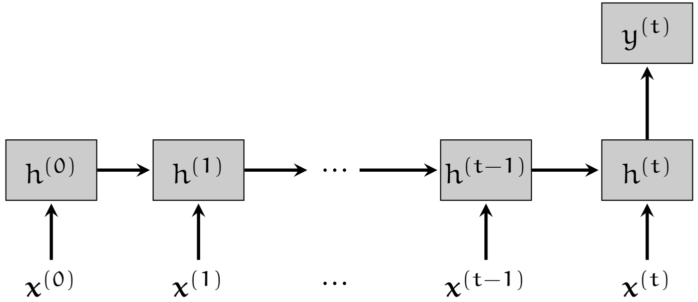
RNN architectures

Many-to-many (e.g., POS tagging)



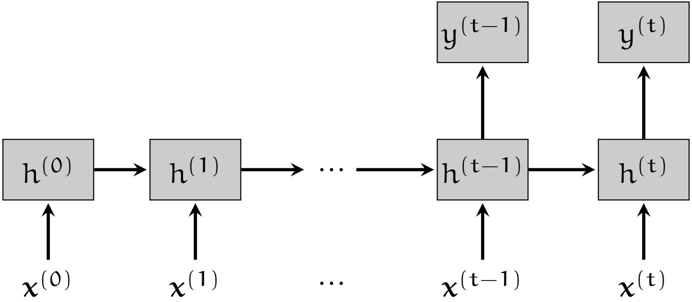
RNN architectures

Many-to-one (e.g., document classification)

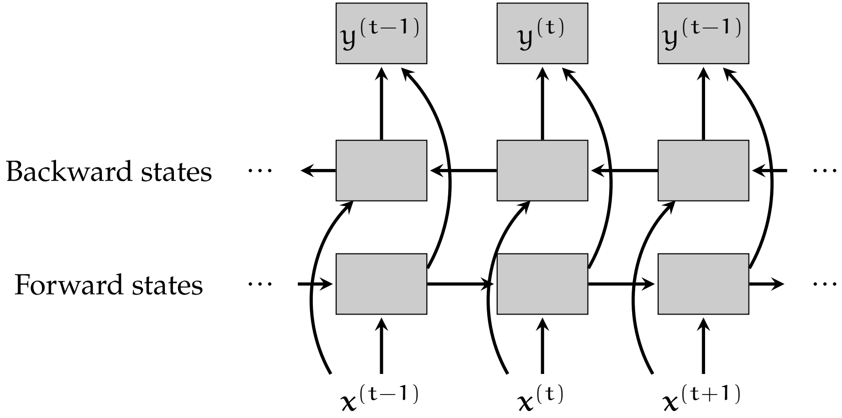


RNN architectures

Many-to-one with a delay (e.g., machine translation)



Bidirectional RNNs



RNNs as language models

- RNNs can function as language models
- We can train RNNs using unlabeled data for this purpose
- During training the task of RNN is to predict the next word
- Depending on the network configuration, an RNN can learn dependencies at a longer distance
- The resulting system can generate sequences

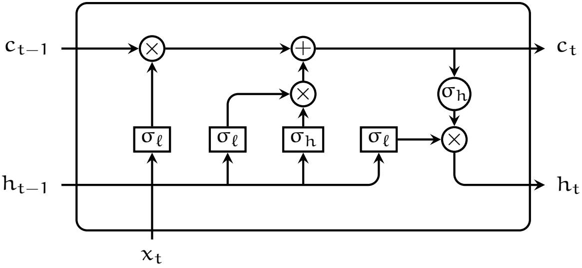
Recommended reding:

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Unstable gradients revisited

- We noted earlier that the gradients may *vanish* or *explode* during backpropagation in deep networks
- This is especially problematic for RNNs since the effective dept of the network can be extremely large
- Although RNNs can theoretically learn long-distance dependencies, this is affected by unstable gradients problem
- The most popular solution is to use *gated* recurrent networks

Gated recurrent networks



- Most modern RNN architectures are 'gated'
- The main idea is learning a mask that controls what to remember (or forget) from previous hidden layers
- Two popular architectures are
  - Long short term memory (LSTM) networks (above)
  - Gated recurrent units (GRU)

Summary

- ANNs are powerfull non-linear learners
  - based on some inspiration from bioloical NNs
  - using many simple processing units
  - built on linear models (logistic regression)
- For non-linear problems we need non-linear activation functions, and at least one hidden layer
- Deep networks use more than one hidden layer
- Common (deep) ANN architectures include:

CNN location invariance
RNN sequence learning

Summary

- ANNs are powerfull non-linear learners
  - based on some inspiration from bioloical NNs
  - using many simple processing units
  - built on linear models (logistic regression)
- For non-linear problems we need non-linear activation functions, and at least one hidden layer
- Deep networks use more than one hidden layer
- Common (deep) ANN architectures include:

CNN location invariance
RNN sequence learning

Next:

Fri exercises (continued)
Mon parsing