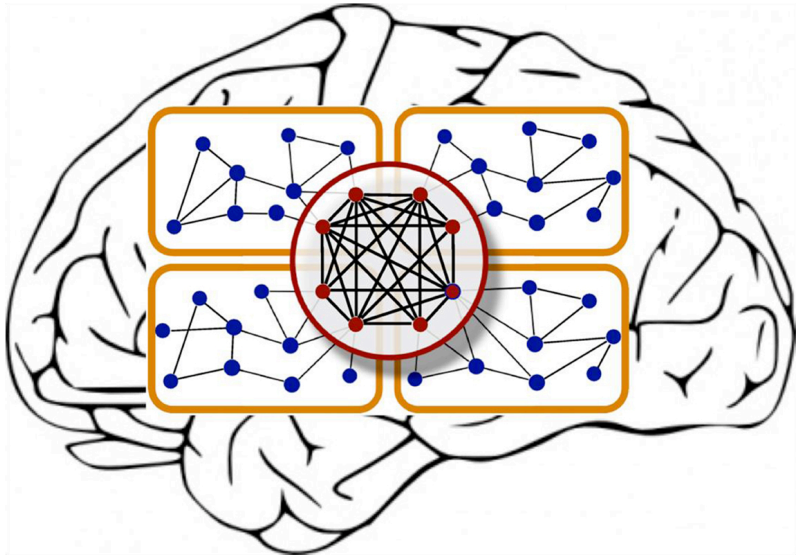
Deep Learning for NLP

Alessandro Moschitti

Deep Learning

… Artificial Neural Networks with many layers.



Representation learning: attempts to automatically learn good features or representations.

Deep learning: attempt to carry out learning by learning multiple levels of representation of increasing complexity and abstraction.

A Deep Architecture

Output Layer

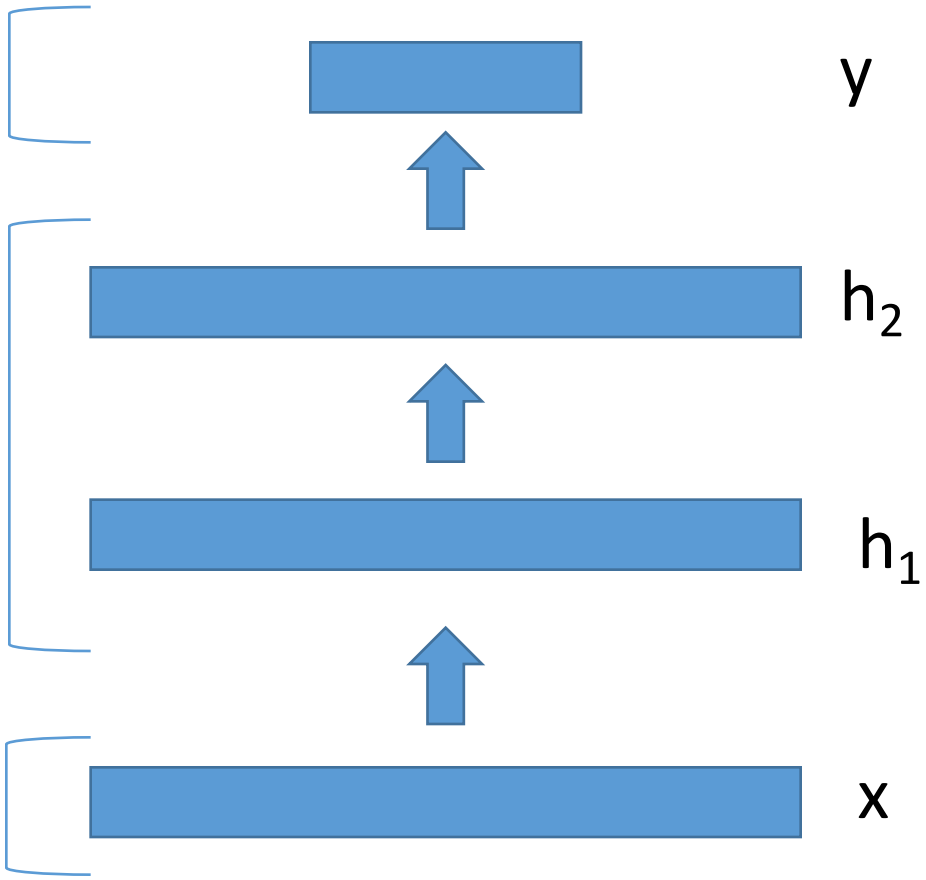
Predicts the supervised target

Hidden Layers

Learns more abstract representations

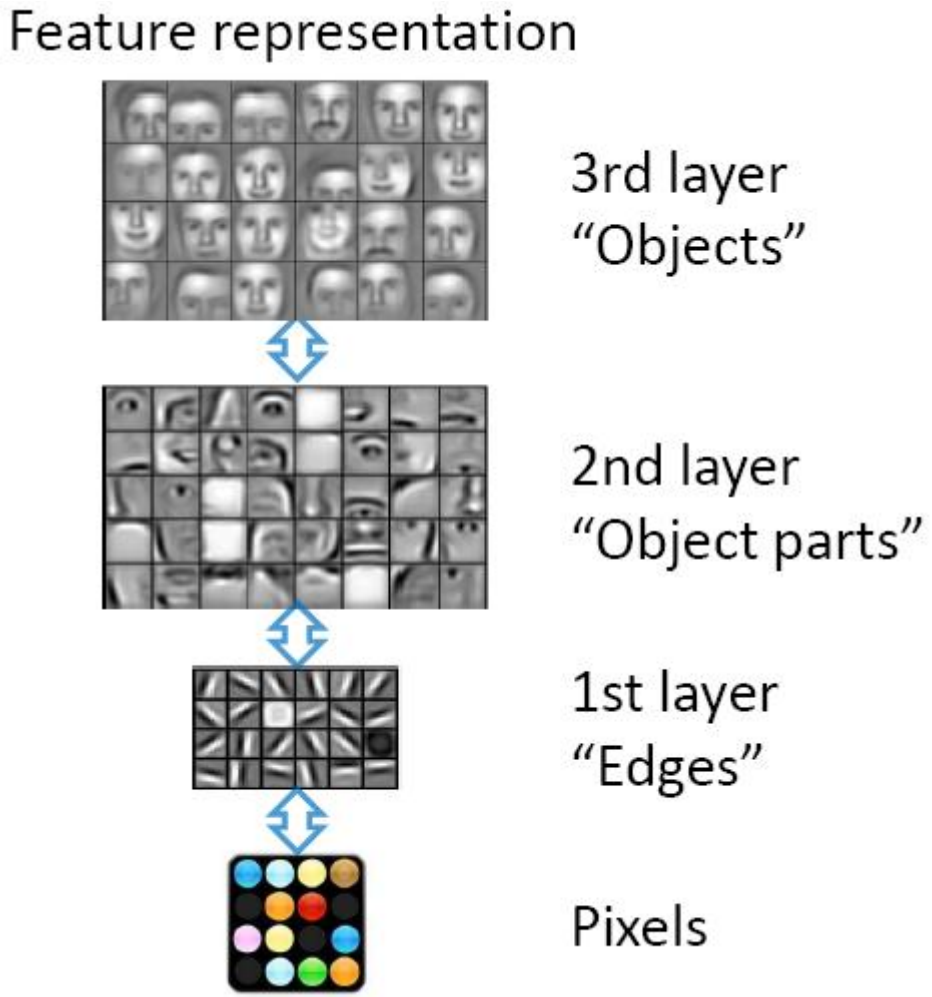
Input Layer

Encode raw inputs



Learning multiple levels of representation

Neural networks learn representation of increasing complexity at each layer.



Representation Learning

- Hand-crafting features it is time-consuming.
- Manually engineered features are task dependent, over-specified or incomplete.
- Deep learning provides a way of automatically generate representations for learning and reasoning.

Outline – 1/2

- Introduction to Neural Networks
  - Perceptron
  - Multi Layer Perceptron
  - Computation Graph
  - Stochastic Gradient Descent
  - Back propagation
- Distributed Representation
  - Words Embedding
  - Unsupervised pre-training

Outline – 2/2

- Convolutional Sentence Models
  - Convolution
  - Pooling
  - Applications
- Recurrent Neural Networks
  - RNN
  - Long-Short Term Memory (LSTM)
- Other Architectures
  - Seq2Seq
  - Similarity Networks

Introduction to Neural Networks

The classification function of perceptron

- The equation of a hyperplane is

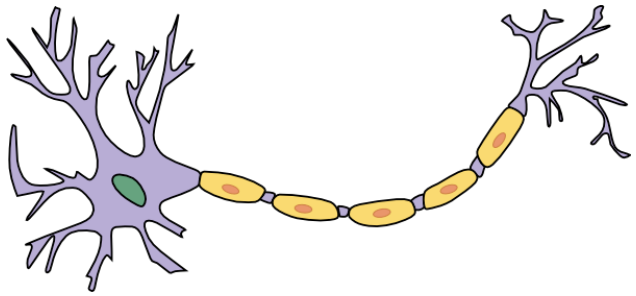
$$f(\vec{x}) = \vec{x} \cdot \vec{w} + b = 0, \quad \vec{x}, \vec{w} \in \Re^n, b \in \Re$$

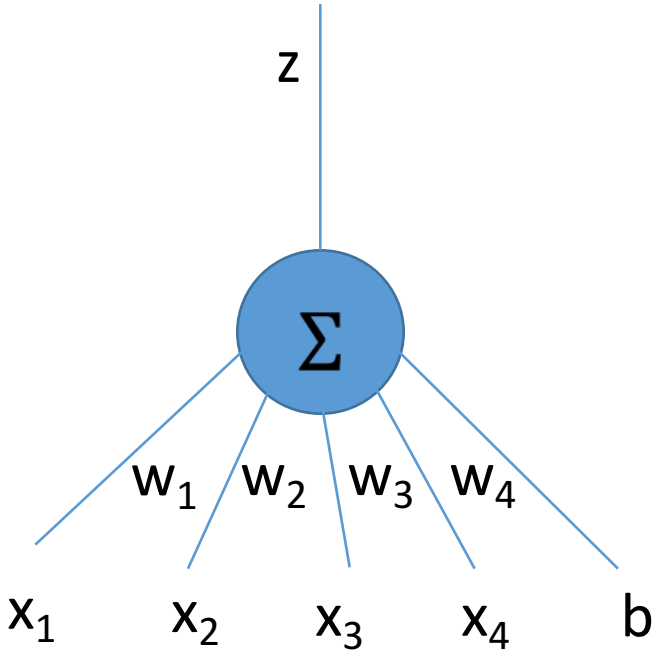
- $\vec{x}$ is the vector representing the classifying example
- $\vec{w}$ is the gradient of the hyperplane (learned model)
- The classification function is $h(\vec{x}) = \text{sign}(f(\vec{x}))$

Neural Network Basics

A single Neural Network Neuron is a Computational unit with *n=4* inputs and 1 output and parameters ***w***, *b*

$$z = \boldsymbol{\sigma}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

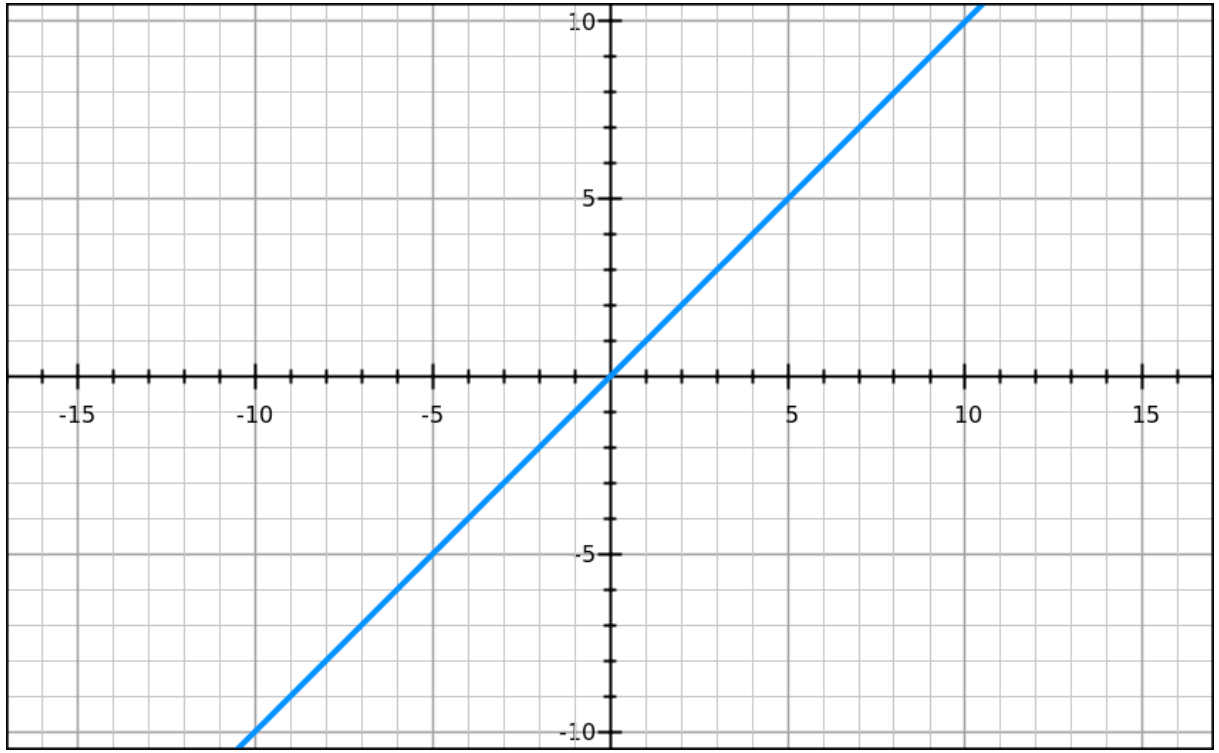




Activation Functions - Linear

- The simplest activation function is the linear activation function.
- Le linear activation function act as identity

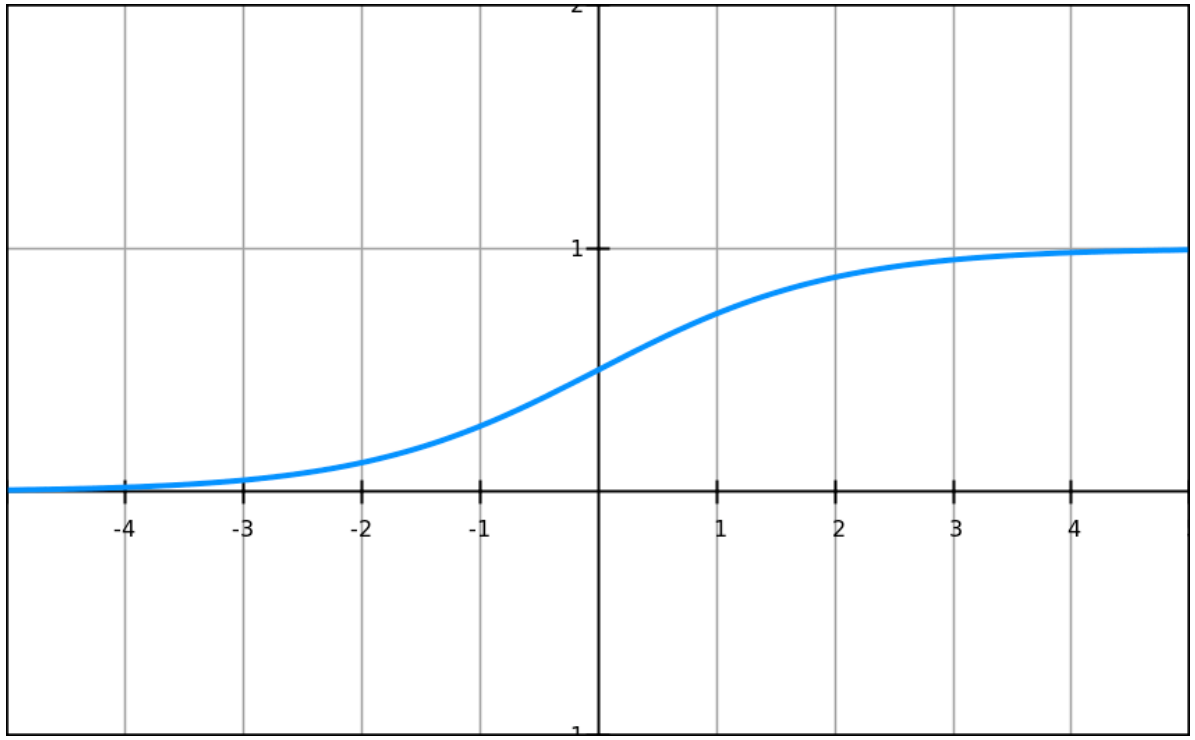
$$\boldsymbol{\sigma}(z) = z$$



Activation Functions - Sigmoid

- The sigmoid activation function maps the input into the [0, 1] range
- This property allows to model probabilities

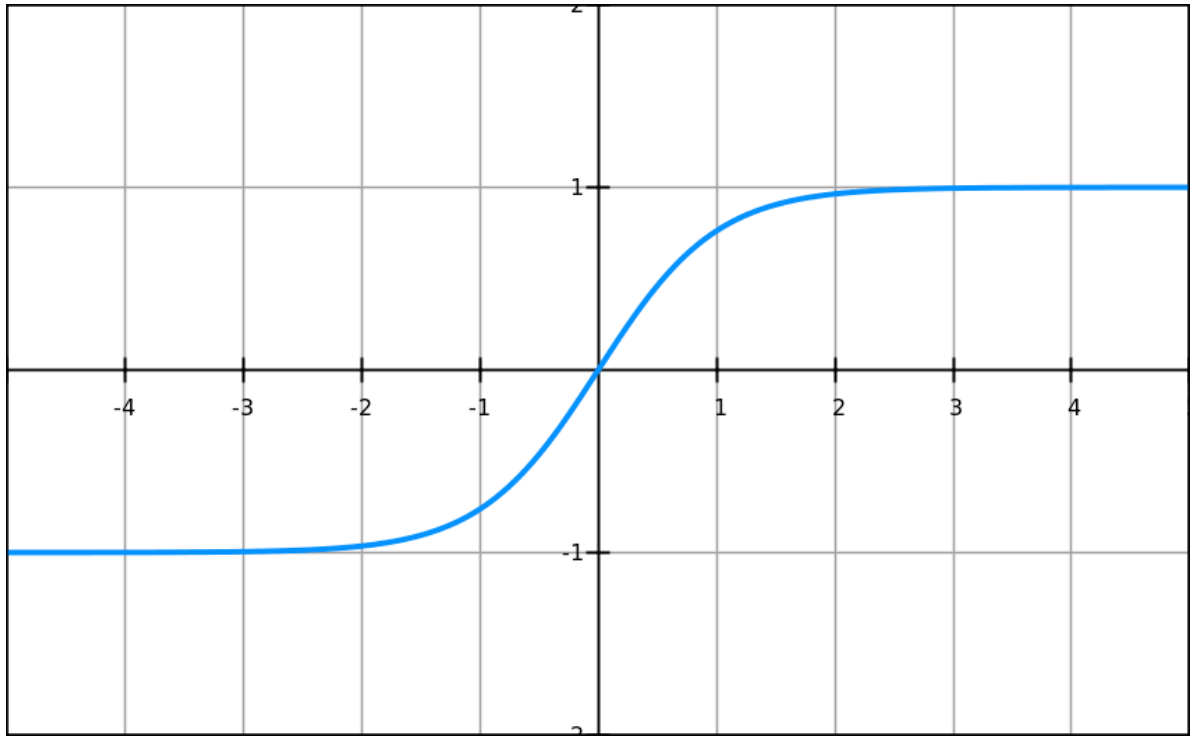
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Activation Functions - Tanh

- Similar to the Sigmoid function, but maps in the range [-1, 1]
- Nowadays, it is one of the most common activation functions

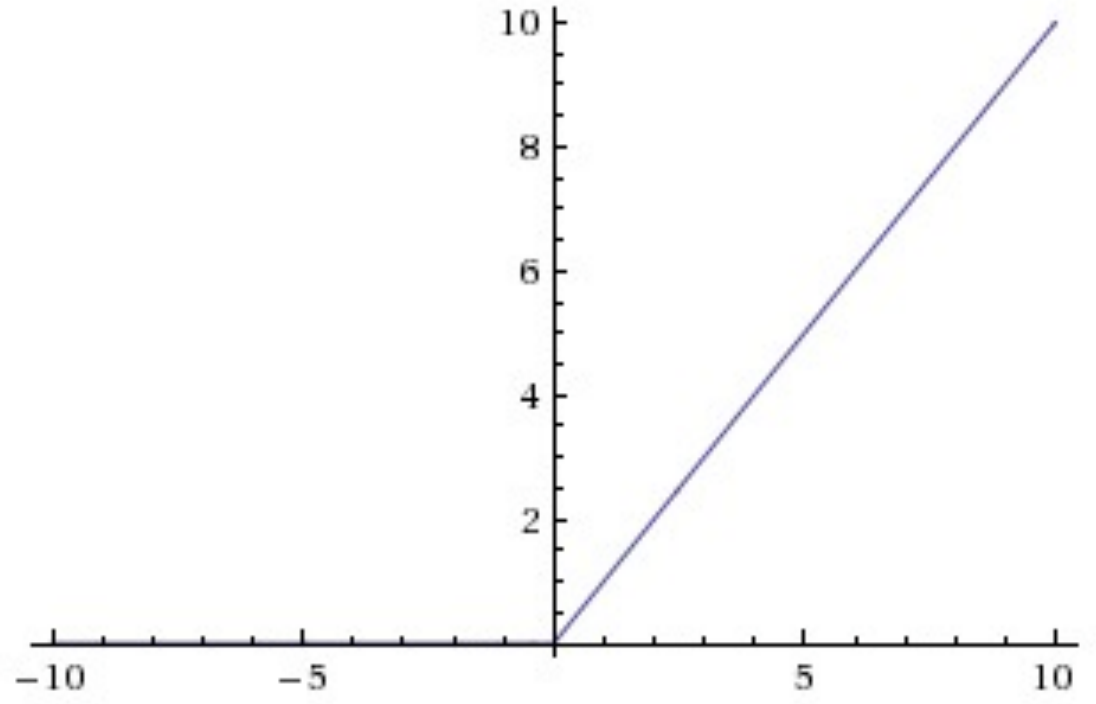
$$\boldsymbol{\sigma}(z) = \tanh(z)$$



Activation Functions - ReLU

Rectified Linear Unit (ReLU) favor sparsity (many points set to zeros) of activations and have a simple derivative

$$\boldsymbol{\sigma}(z) = \max(0, z)$$



Activation Functions - Softmax

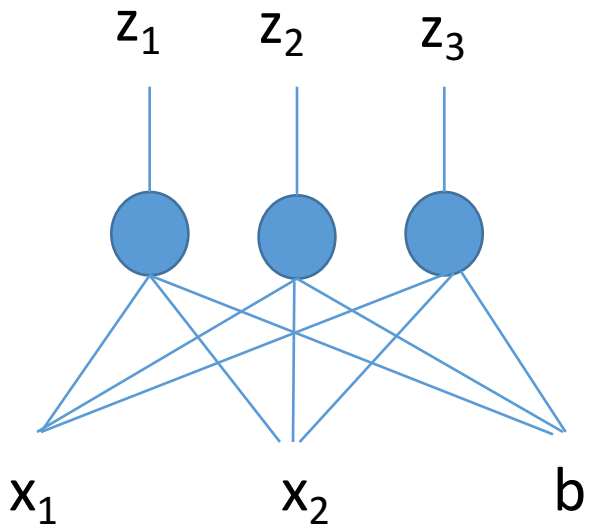
- It models categorical probability distribution
- It is mainly used for multiclass classification
- It is usually applied on a list of activation (layers are discussed later)
- The vector in output sums to one so it can be used to model categorical probabilities

$$\sigma_i(z) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

From Neurons to Layers

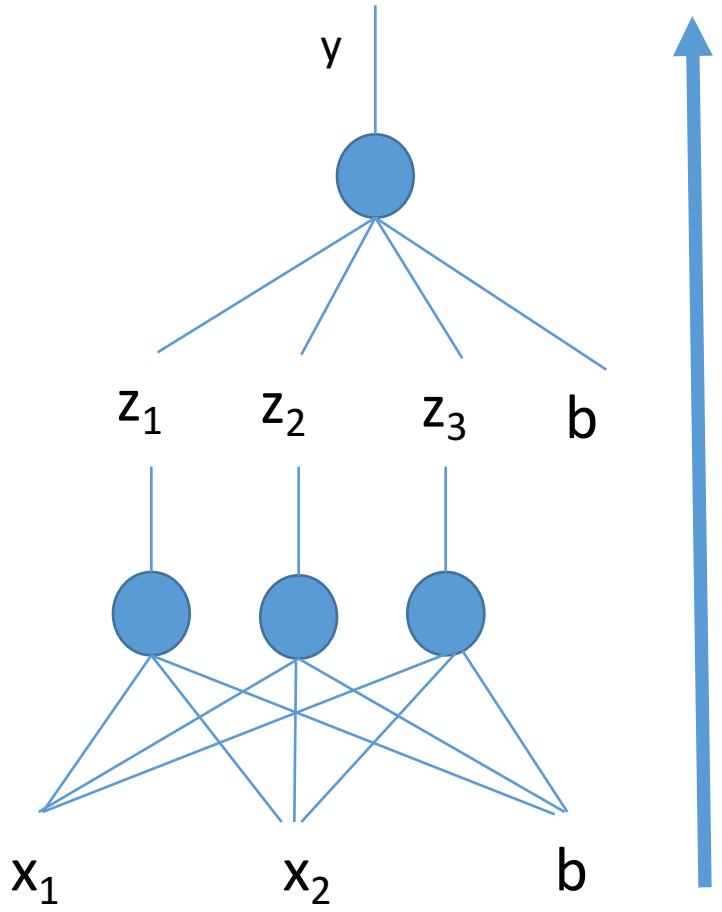
- Artificial Neurons are composed of several layers:
- A layer non linearly transforms the input

$$\boldsymbol{z} = \boldsymbol{\sigma}(\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b})$$



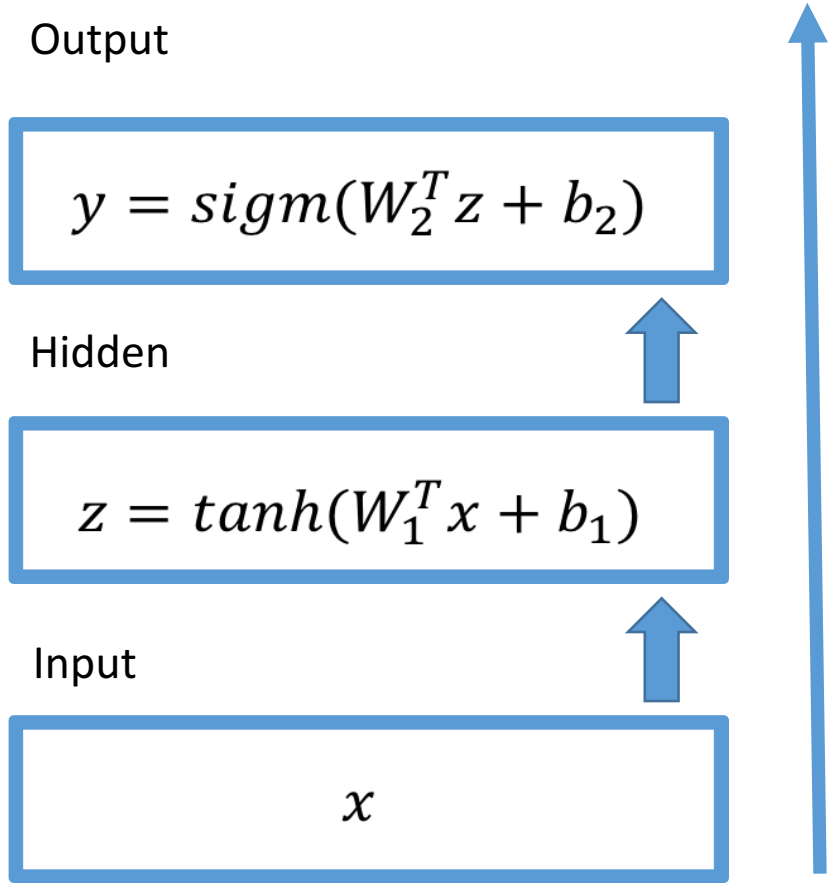
Multi Layer Perceptron

- A multi layer perceptron (MLP) stacks, fully connected layers
- The NNs non linearly transform the input at each layer until the final layer that computes the output
- **A composition of linear function is a linear function itself**
- Non linear activation functions enable learning any function

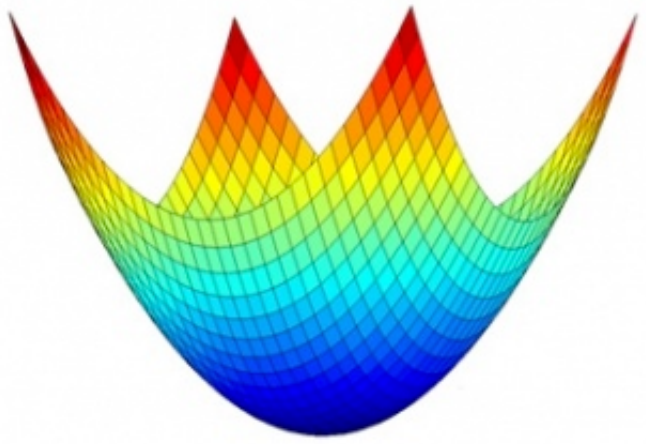


Multi Layer Perceptron – Modular Approach

- We can compose layer in a modular way
- The input layer is a simple vector representation of the input example
- Hidden layers can use different non-linearities.
- For binary classification the output layer uses *sigmoid* activation



Training Neural Networks



A single supervised layer is trained by

computing the derivatives of the error with respect to the weight.

- The error (loss function) is computed between the output of the network and the true label of the input example.
- For each example the weights are updated in the direction of the gradient (Stochastic Gradient Descent - SGD)

$$w_t := w_{t-1} - \eta \nabla_w L(y, \tilde{y})$$

Perceptron training on a data set (on-line algorithm)

$\vec{w}_0 \leftarrow \vec{0}; b_0 \leftarrow 0; k \leftarrow 0; R \leftarrow \max_{1 \le i \le l} \| \vec{x}_i \|$

Repeat

  for i = 1 to m

    if $y_i(\vec{w}_k \cdot \vec{x}_i + b_k) \le 0$ then

      $\vec{w}_{k+1} = \vec{w}_k + \eta y_i \vec{x}_i$

      $b_{k+1} = b_k + \eta y_i R^2$

      k = k + 1

    endif

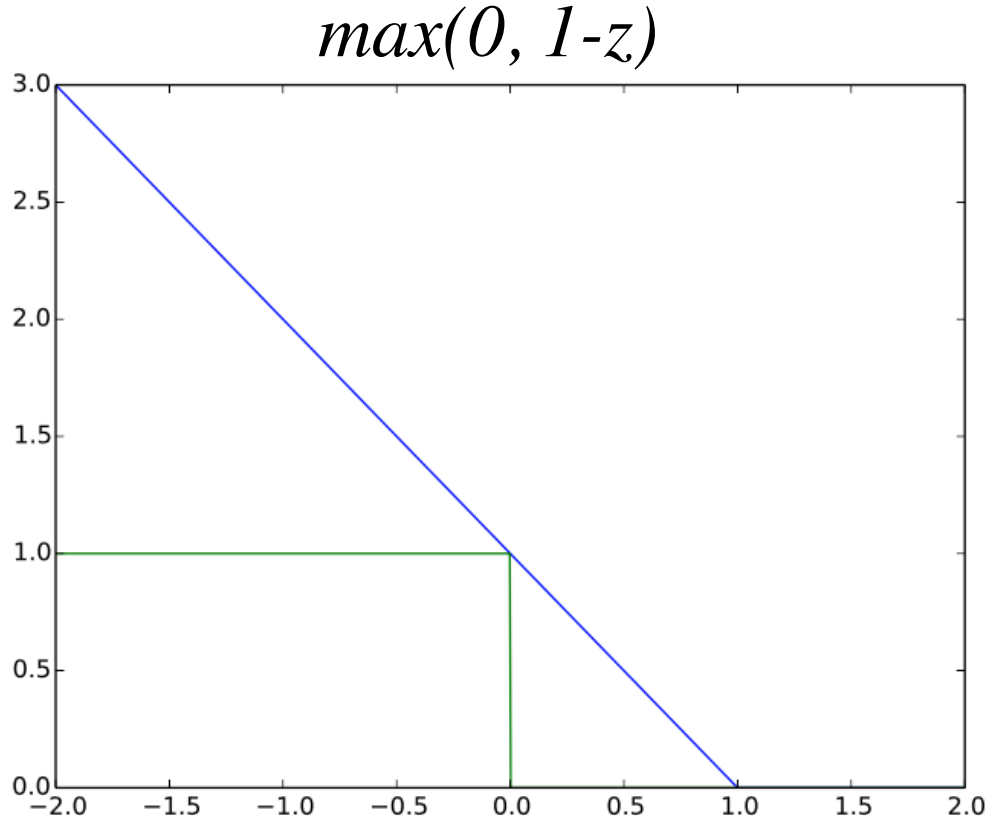
  endfor

until no error is found

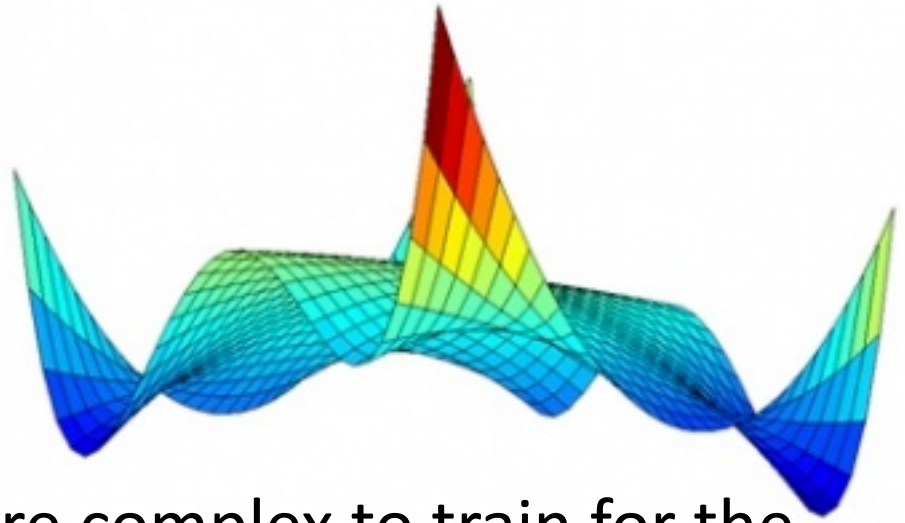
return k, $(\vec{w}_k, b_k)$

Is it a gradient descent method?

- Loss function: the so-called hinge loss
  - $loss(y) = max(0, 1\text{-}y*label)$
- Using the classification function of perceptron:
  - $loss(y) = max(0, 1\text{-}\boldsymbol{w}*\boldsymbol{x}*label)$
- Let us apply derivative wrt w
  - $\boldsymbol{x}*label$
- Thus we get: $\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{x}*label$



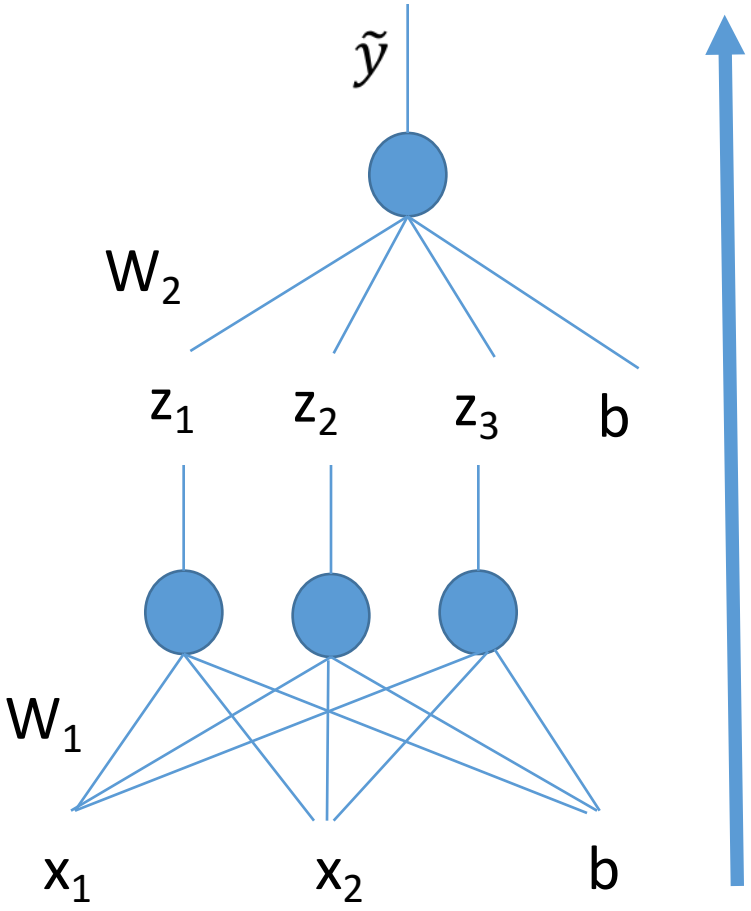
Training Neural Networks



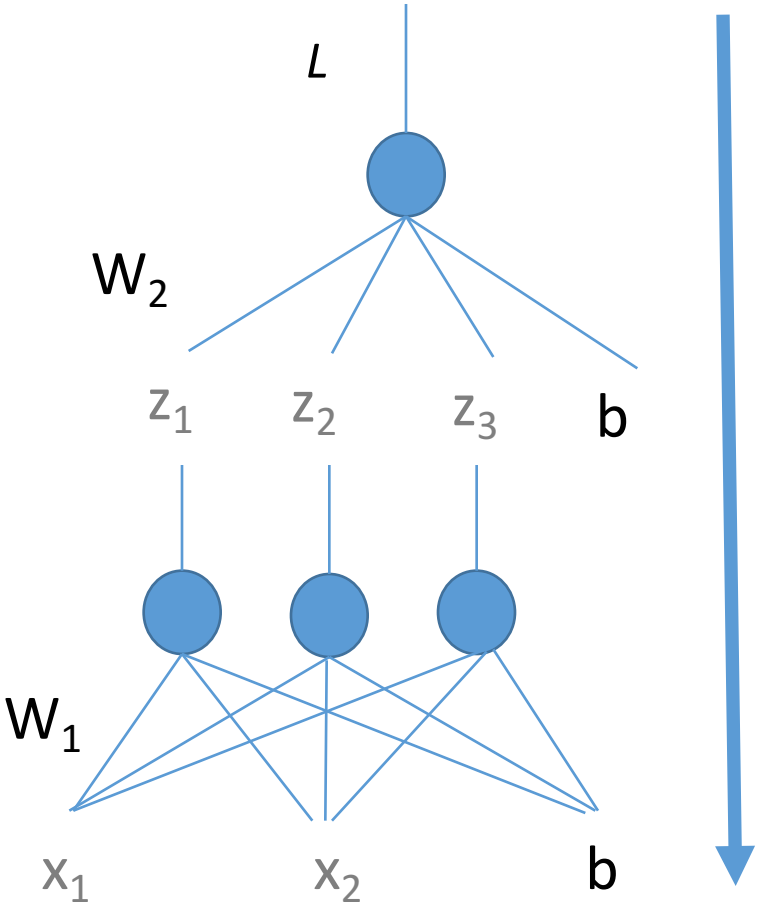
- A neural network with multiple layers is more complex to train for the presence of non linear functions, which make the optimization surface non convex.
- Despite this problem, we use the same techniques as for convex optimization (without guaranties of reaching global minima)
- Back propagating the gradients through the network (using the chain rules of derivatives)

Back Propagation Algorithm

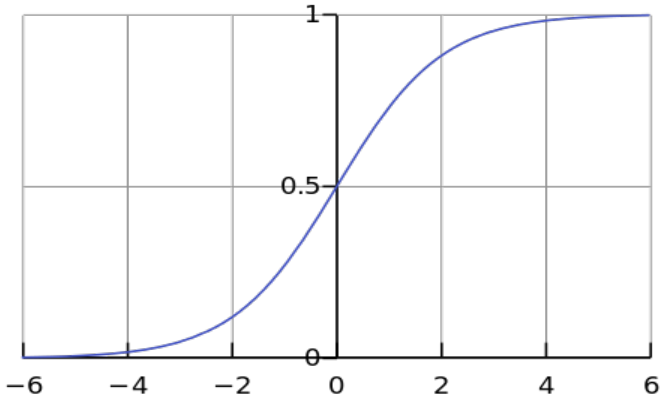
Forward Pass



Backward Pass



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



-0.06

W1

-2.5

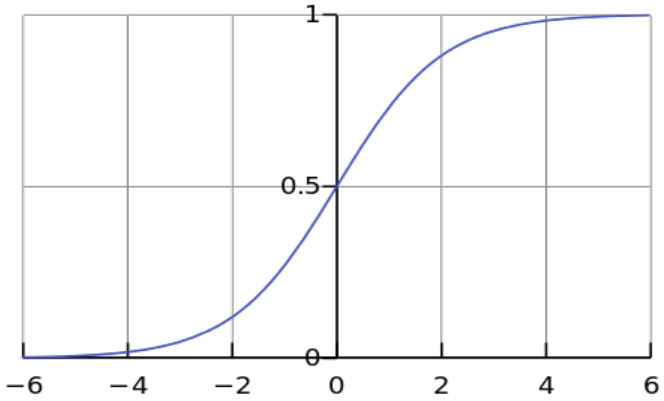
W2

W3

1.4

f(x)

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



-0.06

2.7

-2.5

-8.6

***f*(*x*)**

0.002

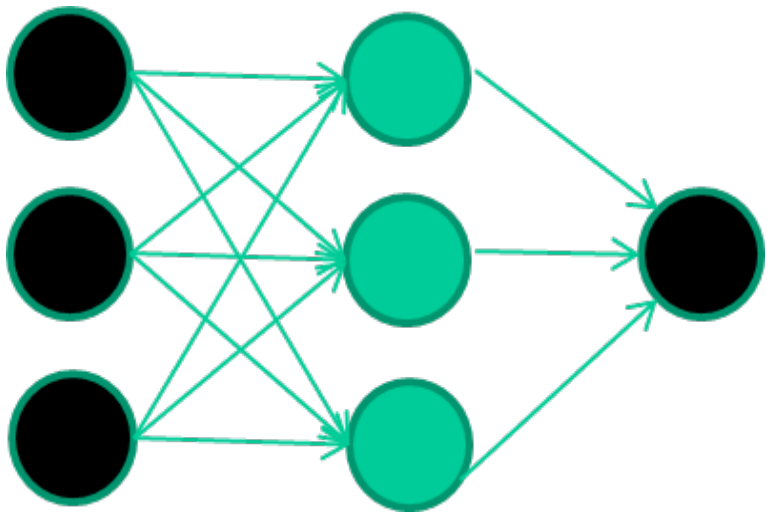
1.4

$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$

A dataset

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

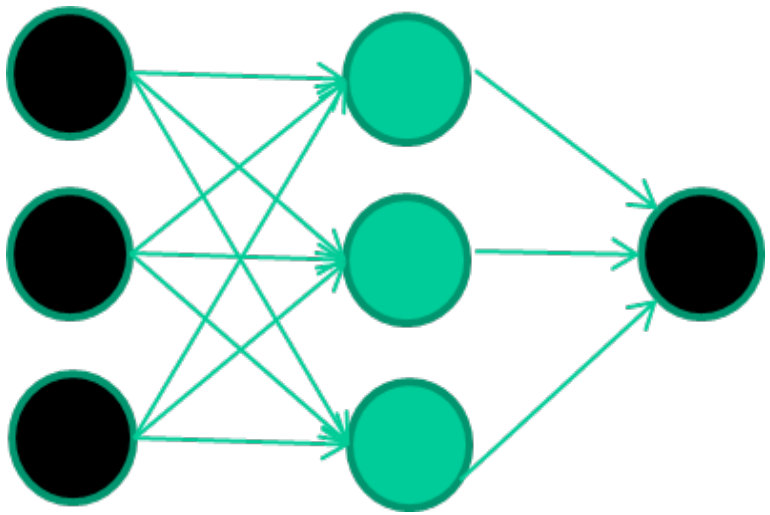
etc …



Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

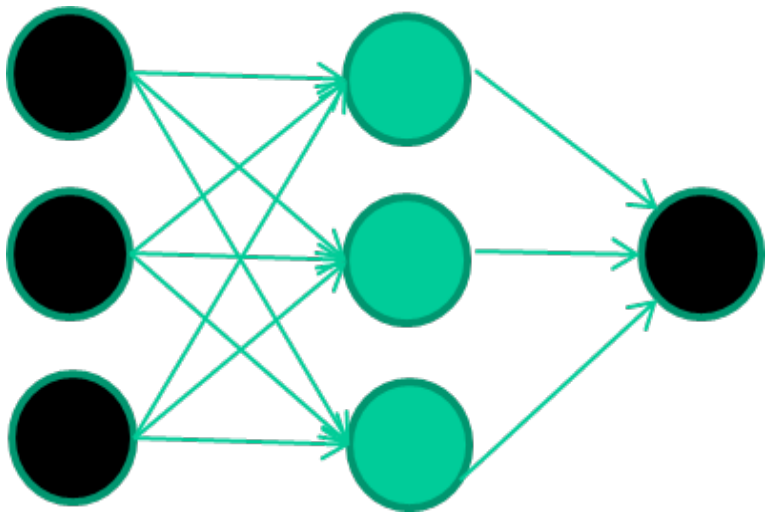


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

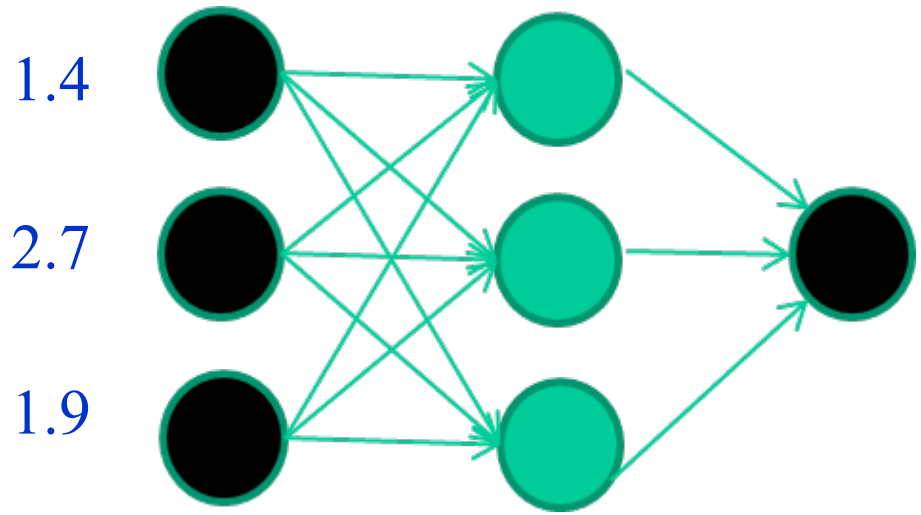
Initialise with random weights



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern

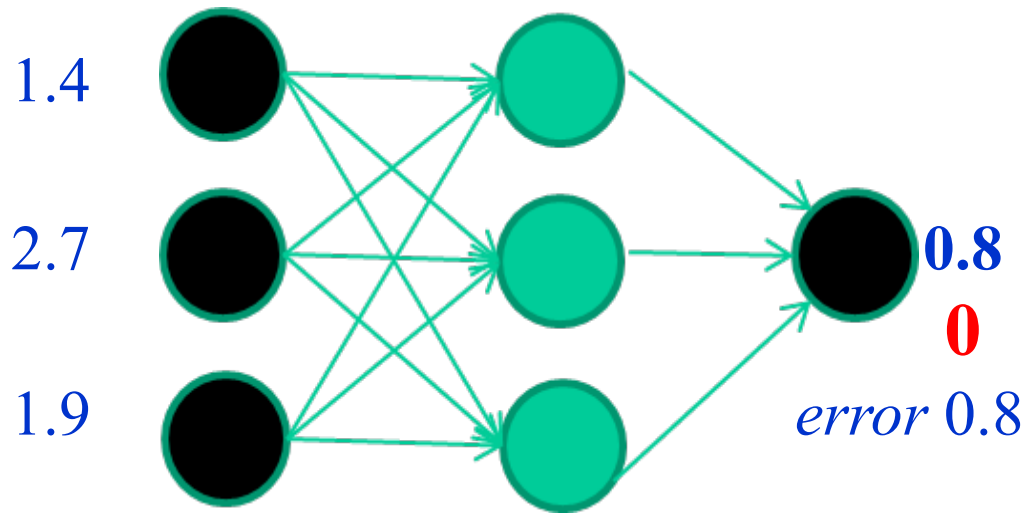


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Feed it through to get output

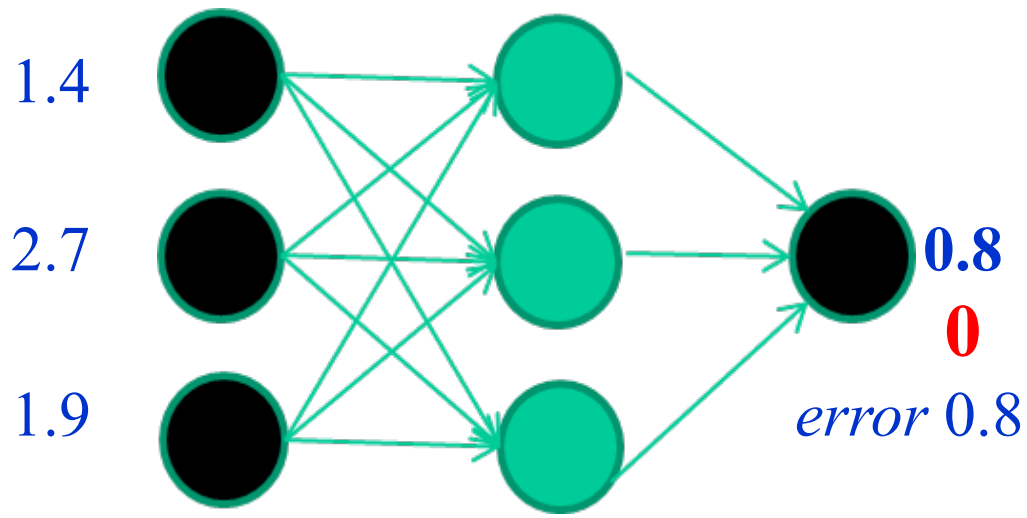


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Compare with target output

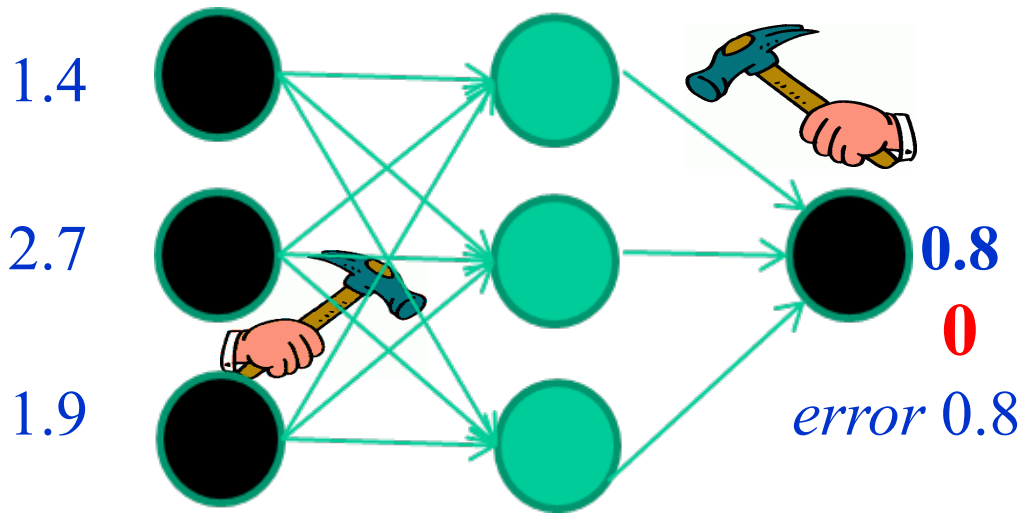


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Adjust weights based on error

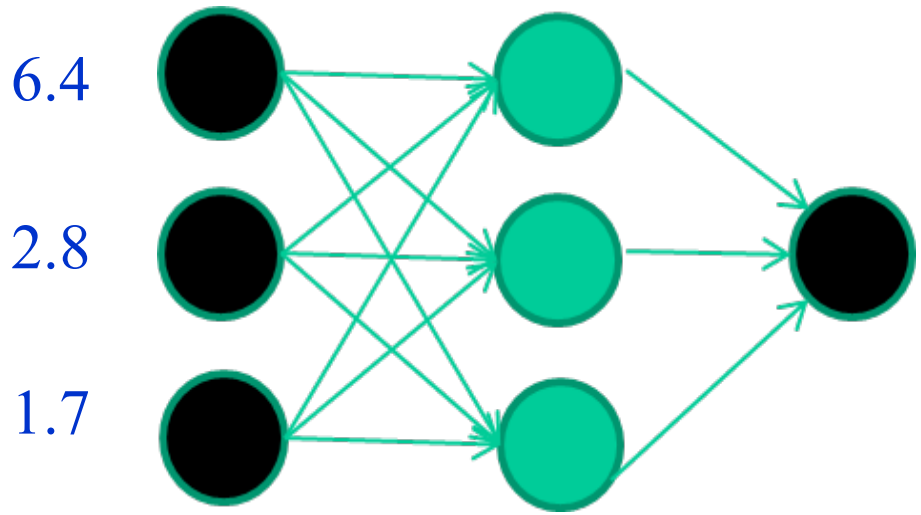


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Present a training pattern

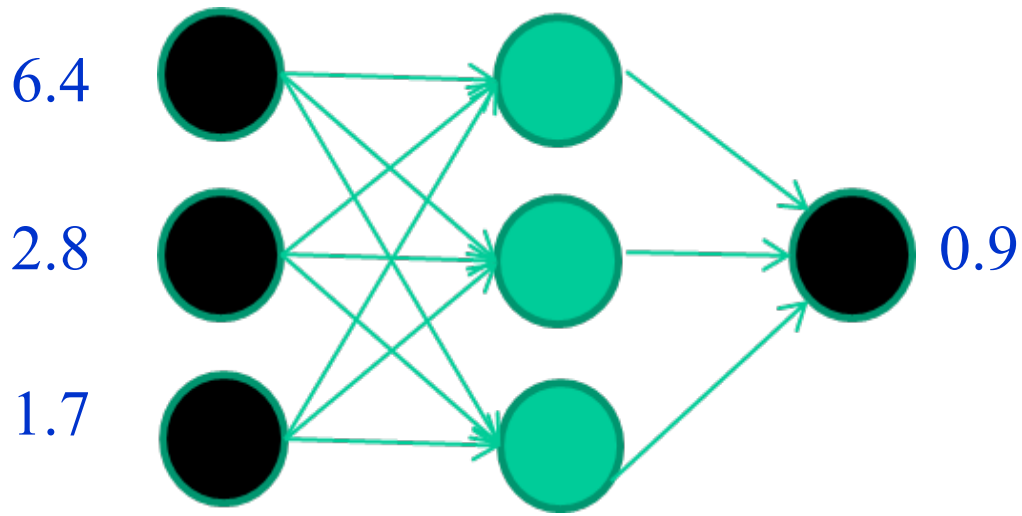


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Feed it through to get output

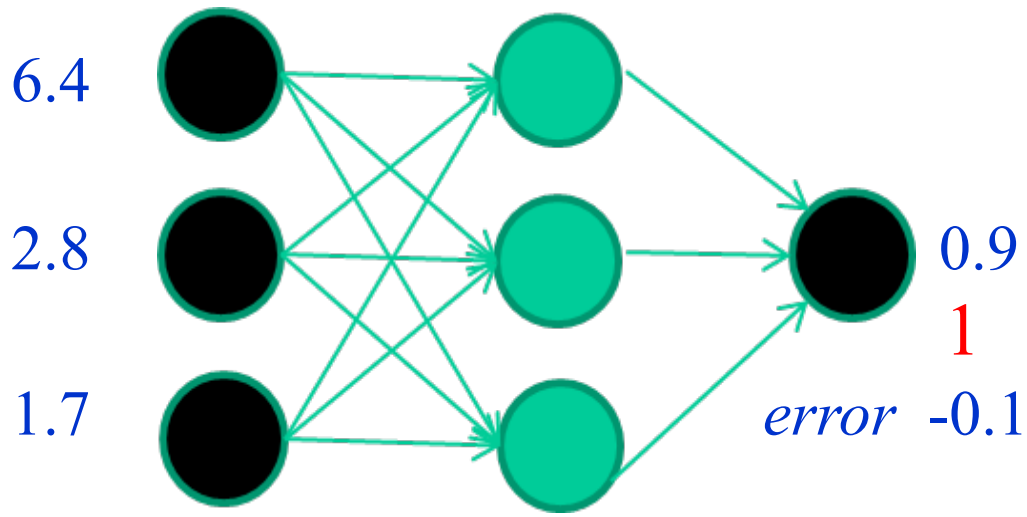


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Compare with target output

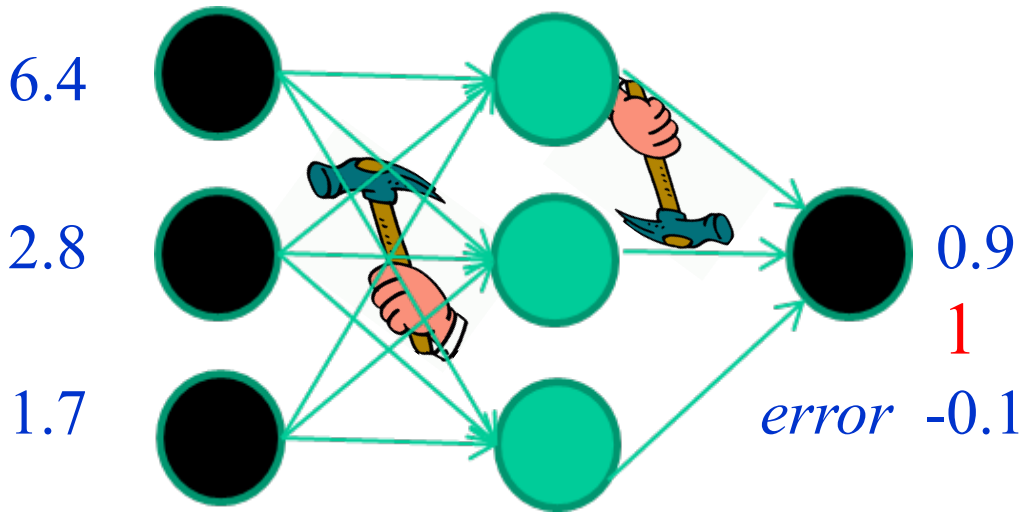


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

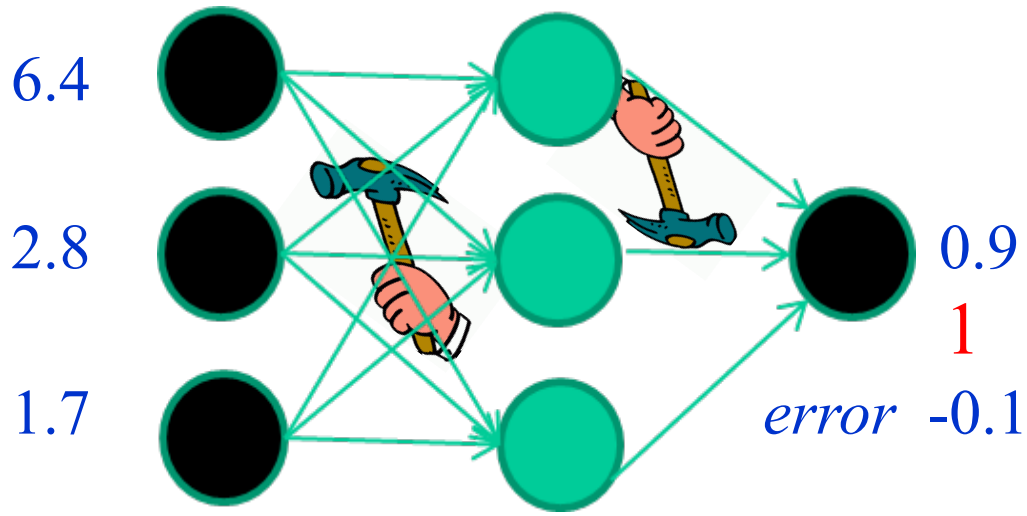
Adjust weights based on error



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

And so on ….

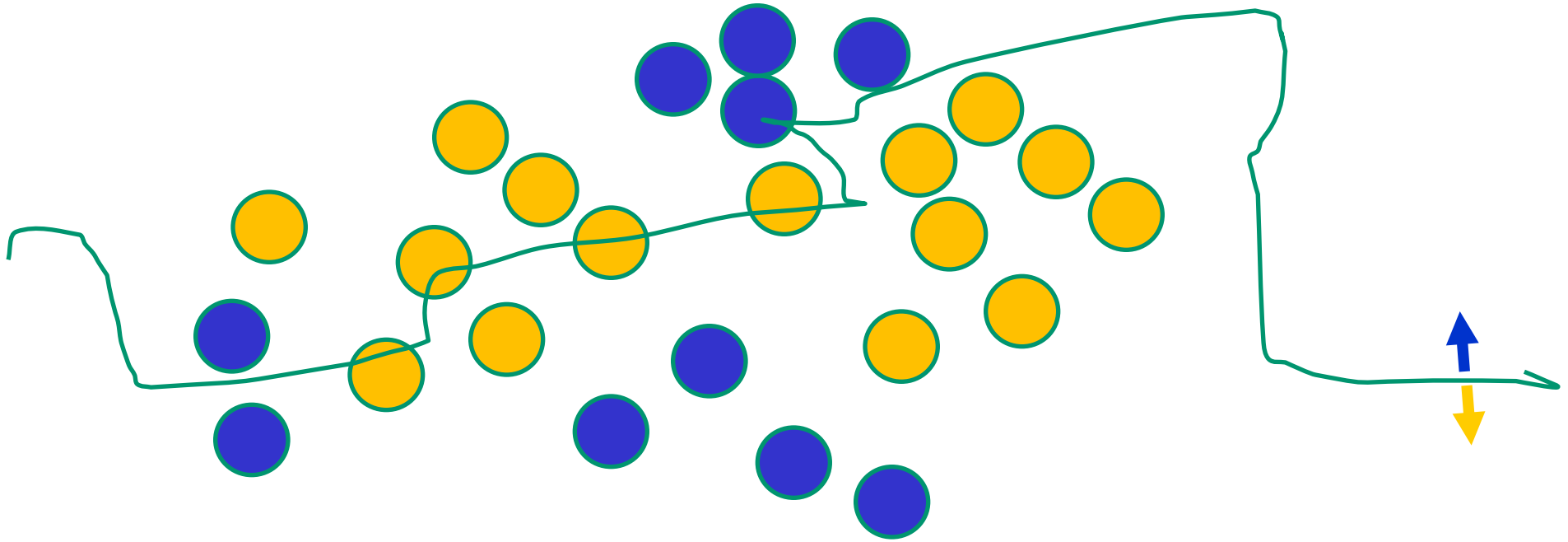


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

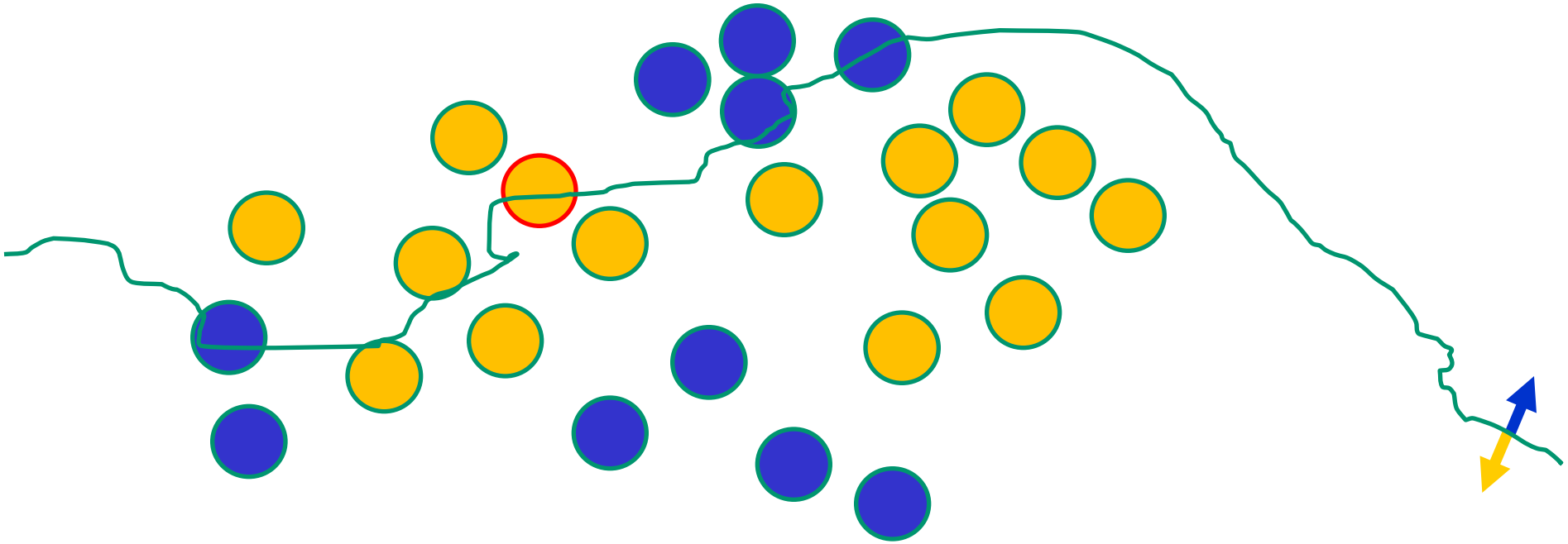
The decision boundary perspective…

Initial random weights



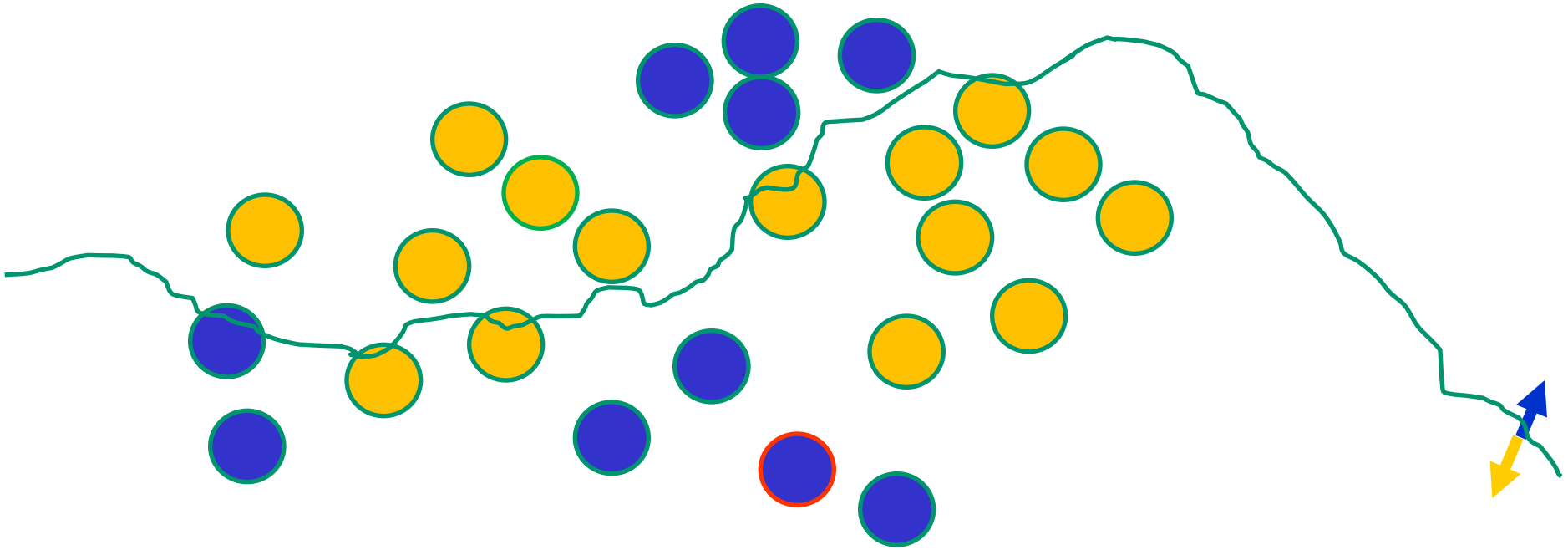
The decision boundary perspective…

Present a training instance / adjust the weights



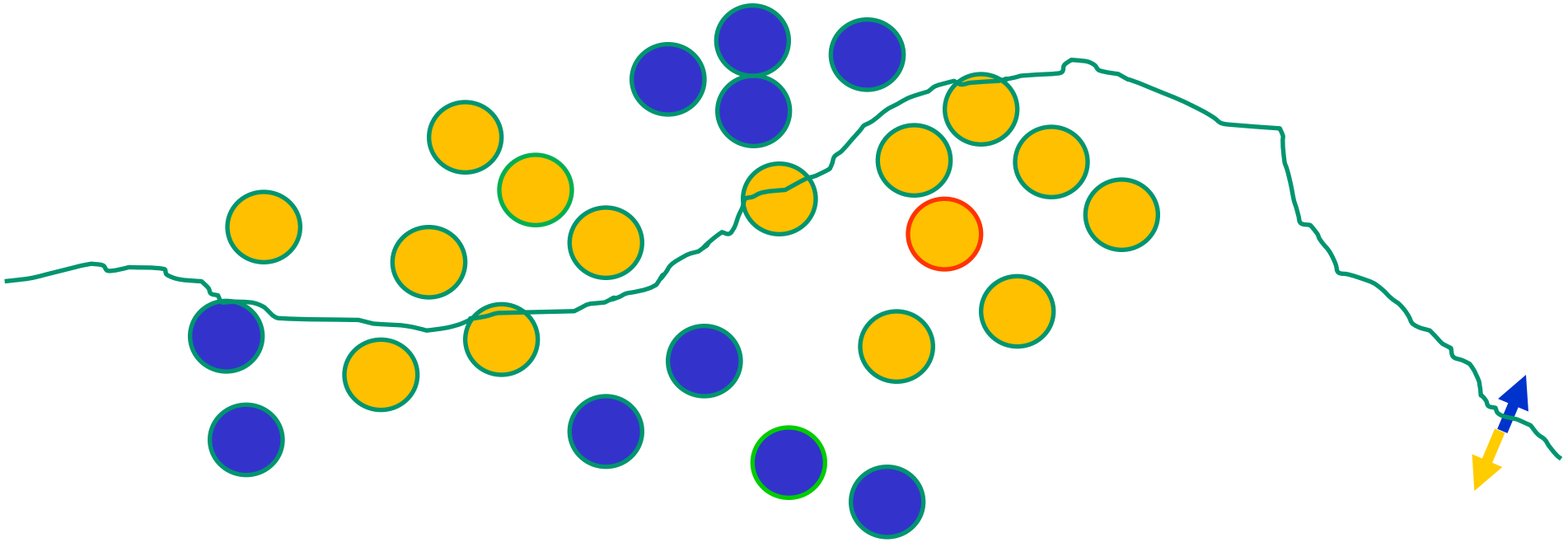
The decision boundary perspective…

Present a training instance / adjust the weights



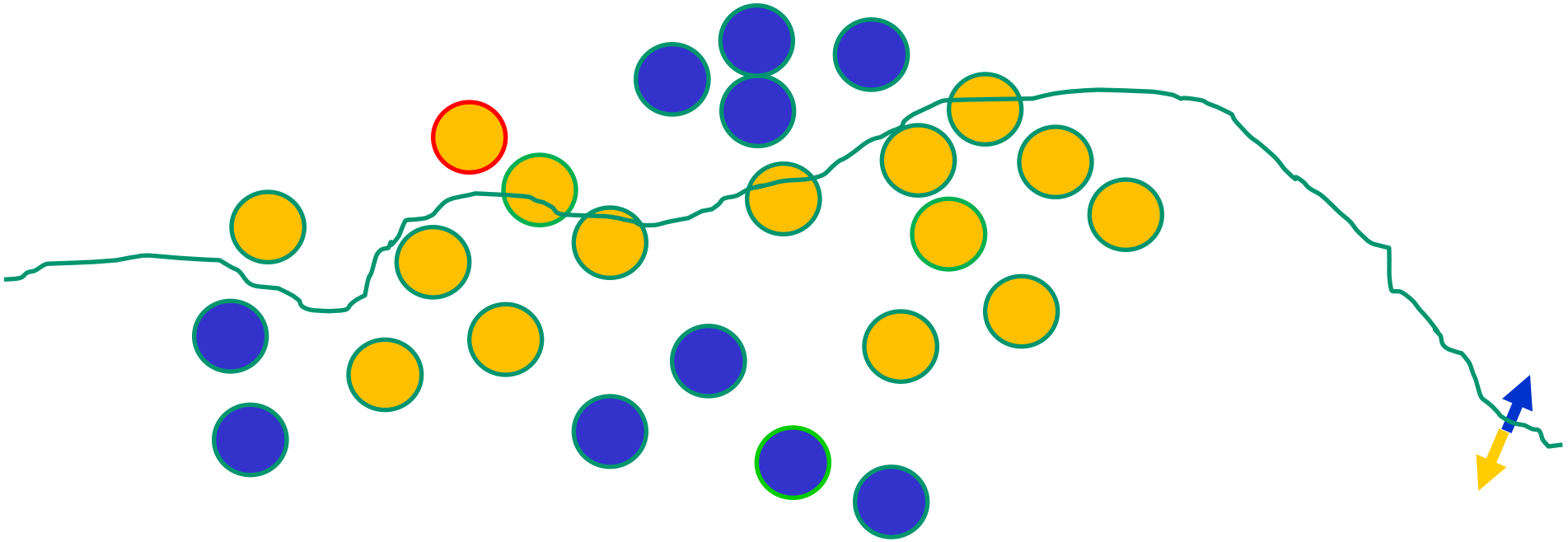
The decision boundary perspective…

Present a training instance / adjust the weights



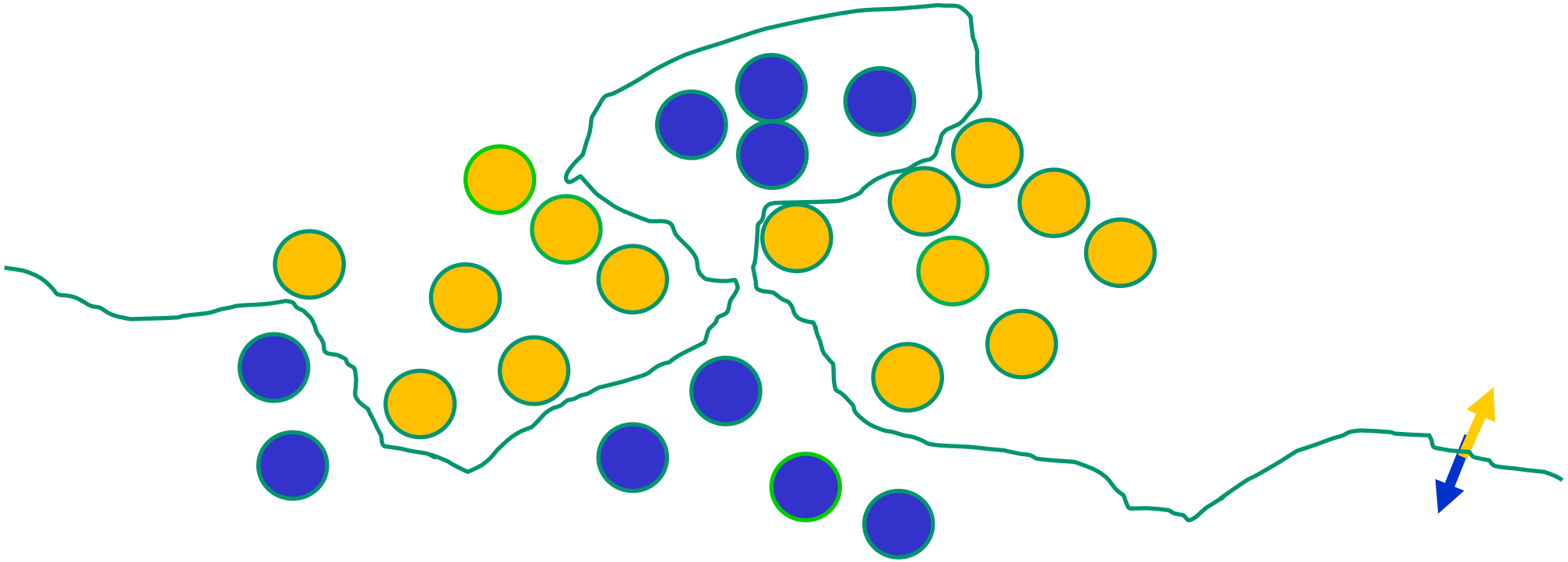
The decision boundary perspective…

Present a training instance / adjust the weights



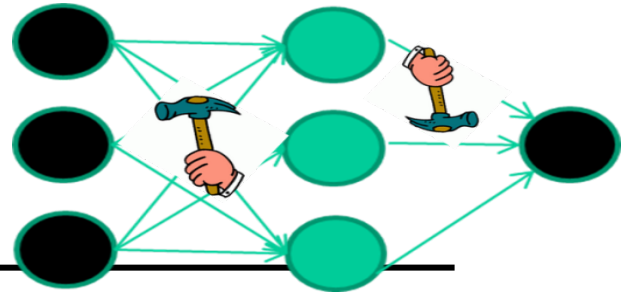
The decision boundary perspective…

Eventually ….



The point I am trying to make

- Weight-learning algorithms for NNs are dumb
- They work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others
- But, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications

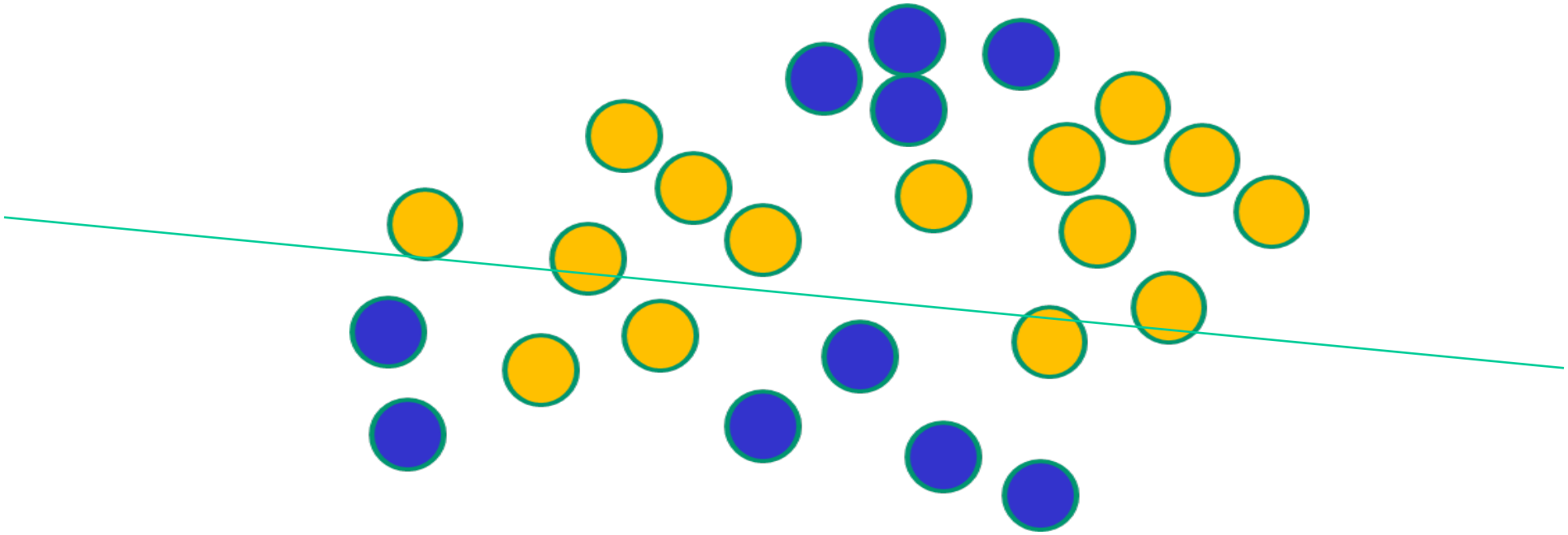


Some other points

- If $f(x)$ is non-linear, a network with 1 hidden layer can, in theory, learn perfectly any classification problem
- A set of weights exists that can produce the targets from the inputs
- The problem is finding them

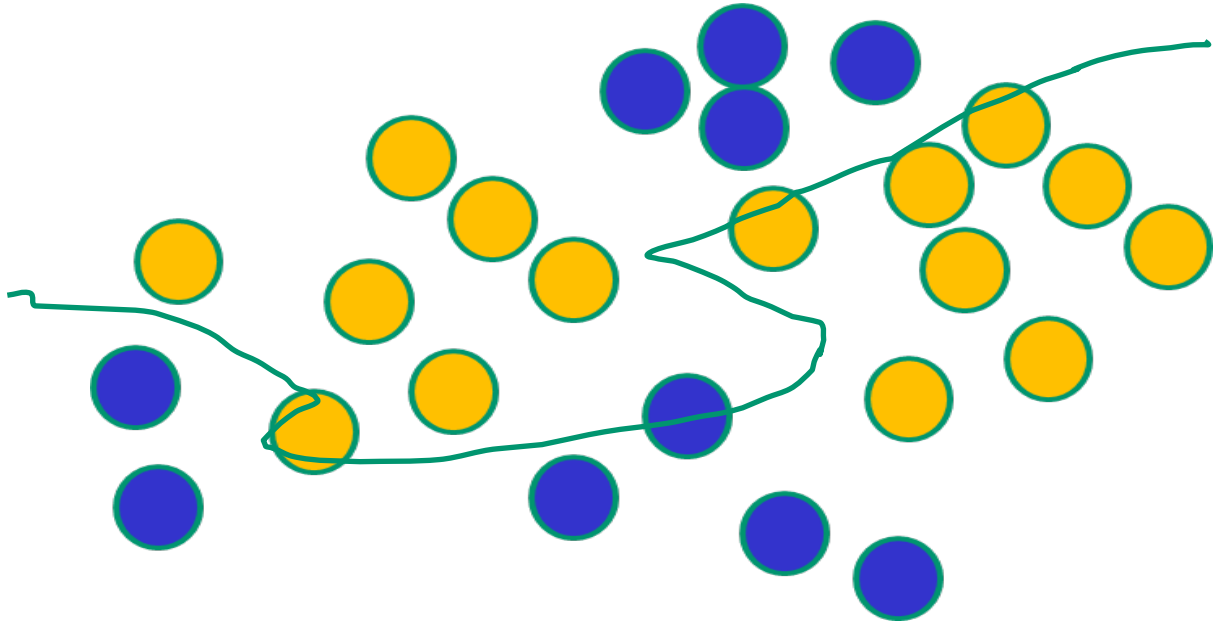
Some other ‘by the way’ points

If $f(x)$ is linear, the NN can **only** draw straight decision boundaries (even if there are many layers of units)



Some other ‘by the way’ points

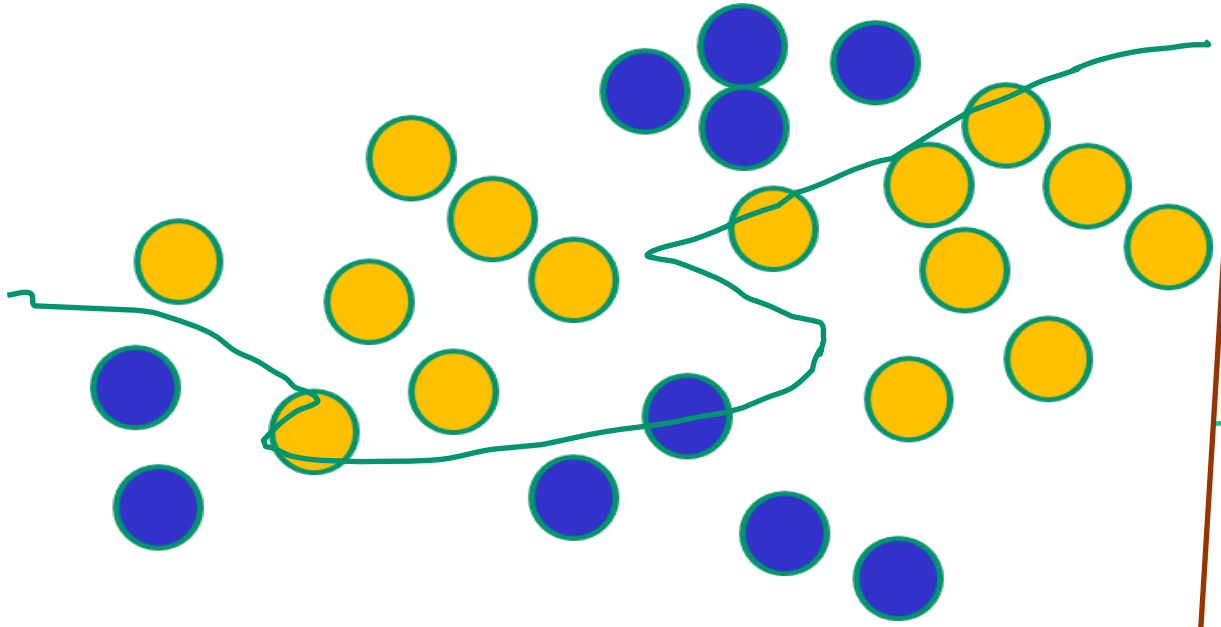
NNs use nonlinear *f*(x) so they can draw complex boundaries, but keep the data unchanged

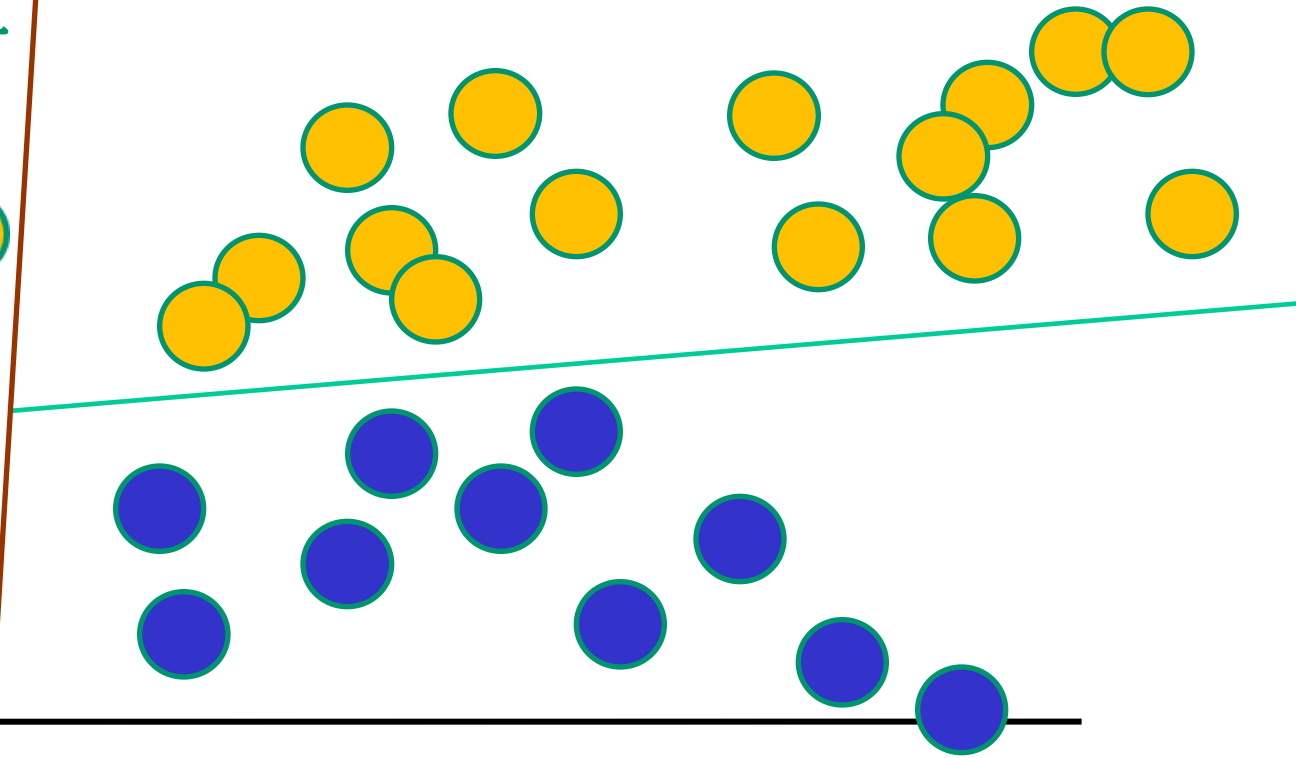


Some other ‘by the way’ points

NNs use nonlinear *f*(x) so they can draw complex boundaries, but keep the data unchanged

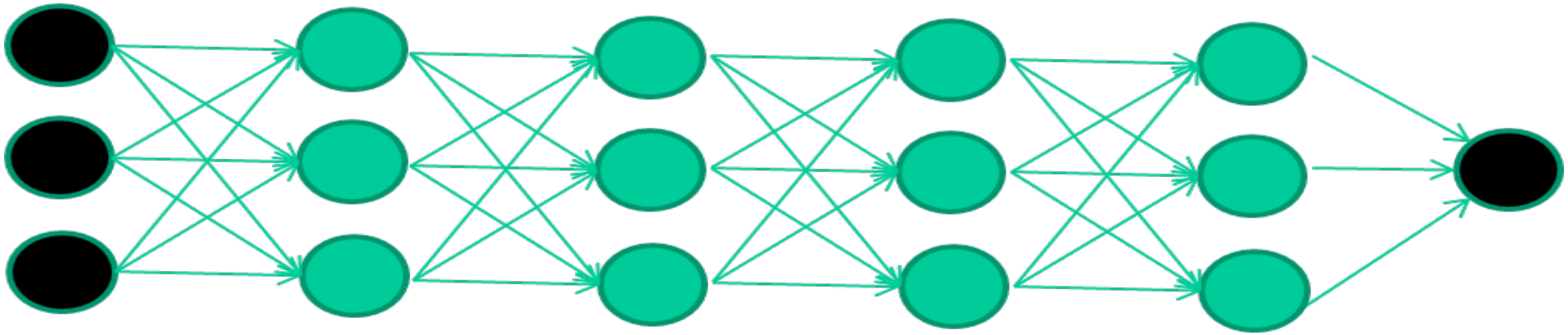
SVMs only draw straight lines, but they transform the data first in a way that makes that OK



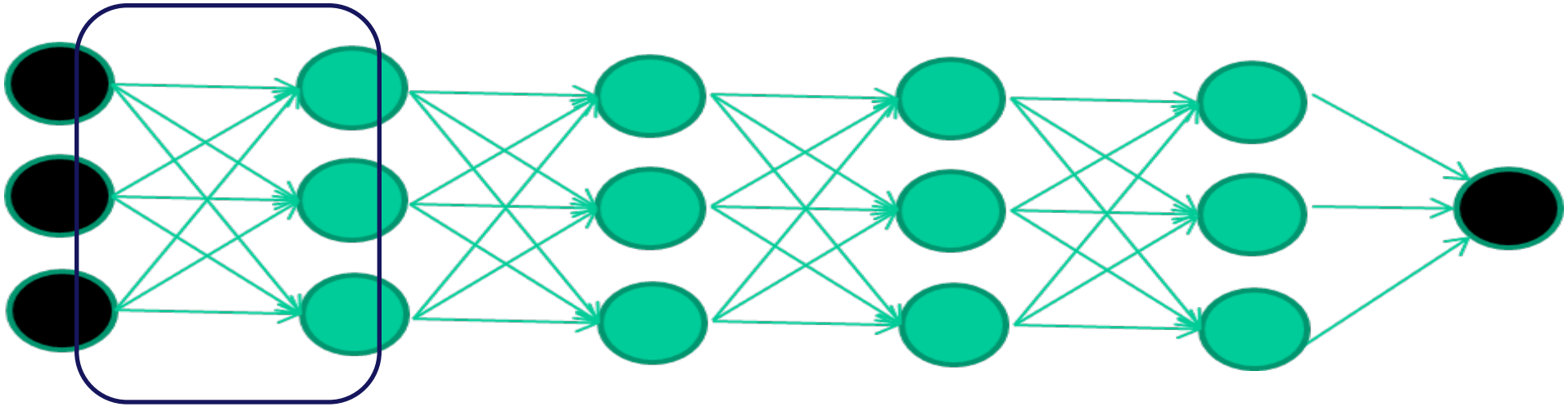


Multiple layers: deep learning

The new way to train multi-layer NNs…

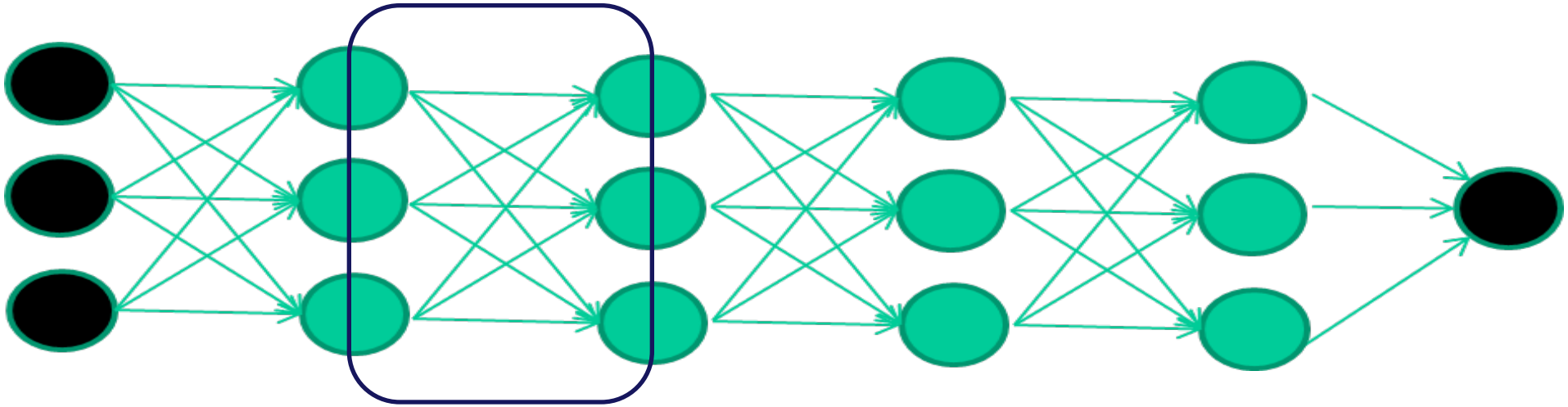


The new way to train multi-layer NNs…



Train **this** layer first

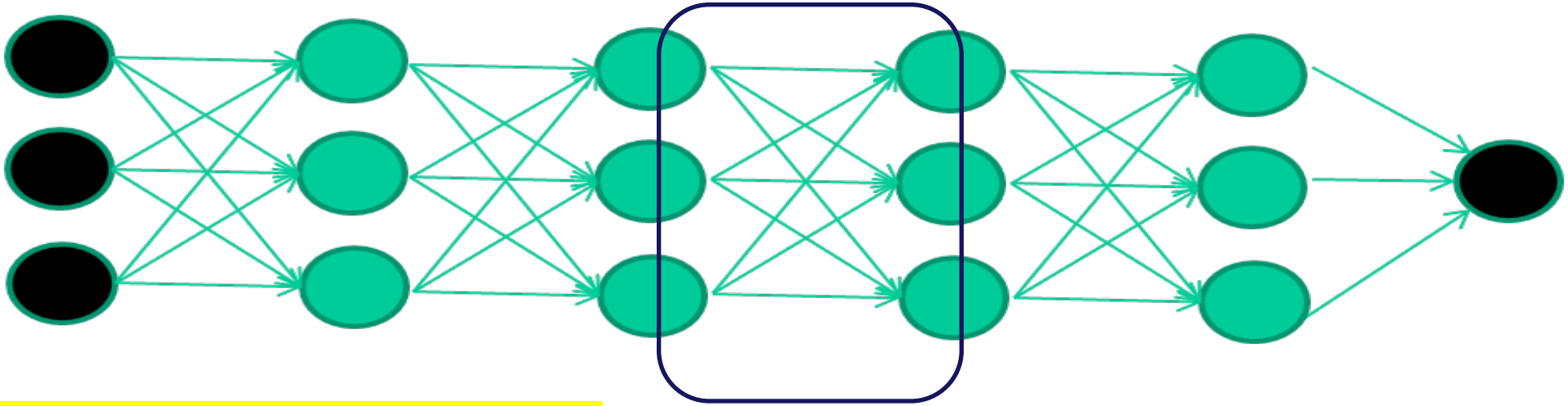
The new way to train multi-layer NNs…



Train **this** layer first

then **this** layer

The new way to train multi-layer NNs…

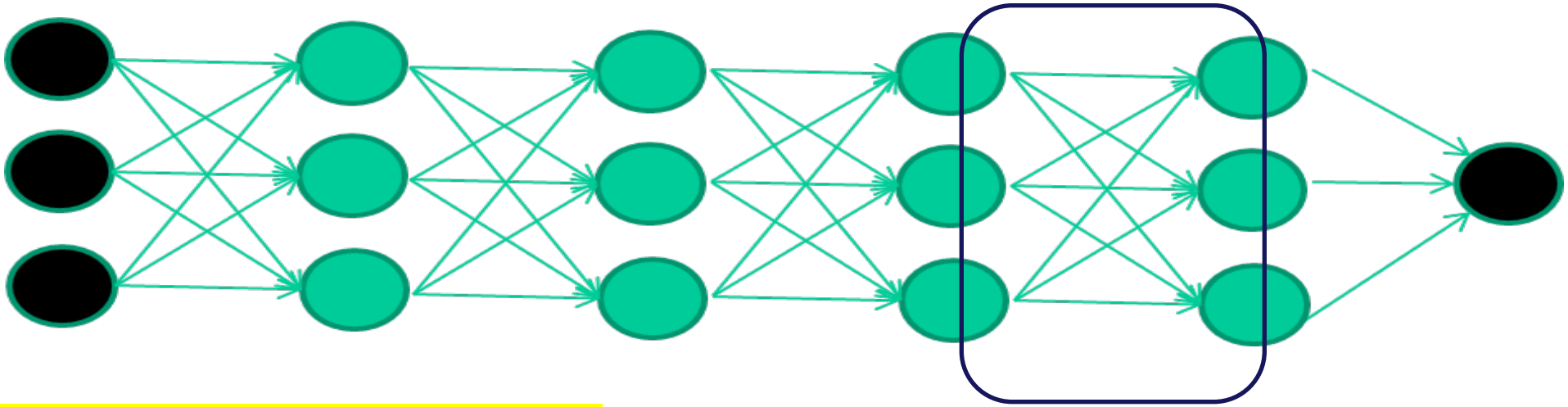


Train **this** layer first

then **this** layer

then **this** layer

The new way to train multi-layer NNs…



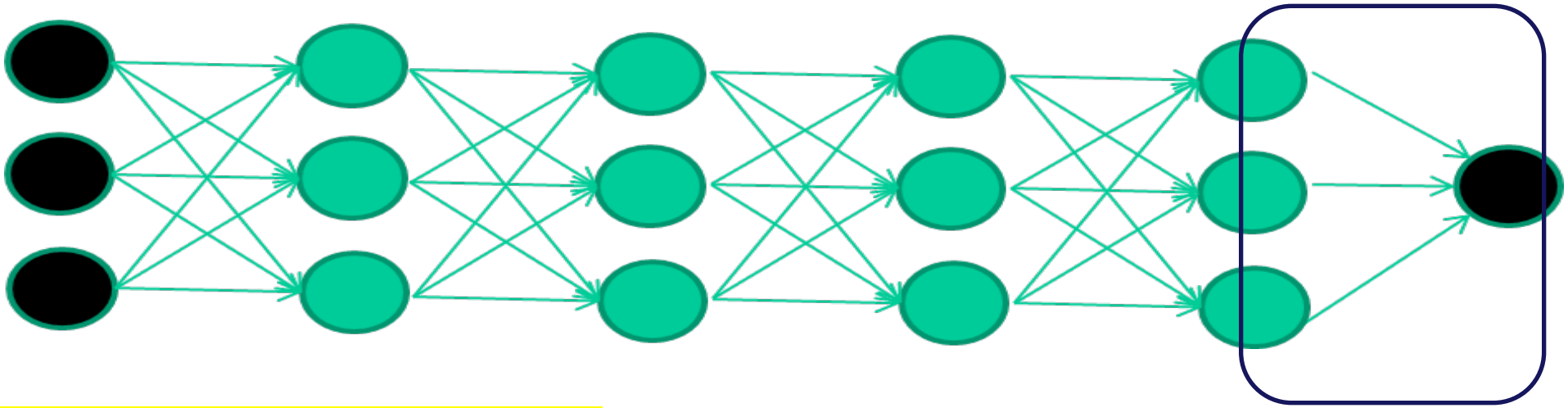
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

The new way to train multi-layer NNs…



Train **this** layer first

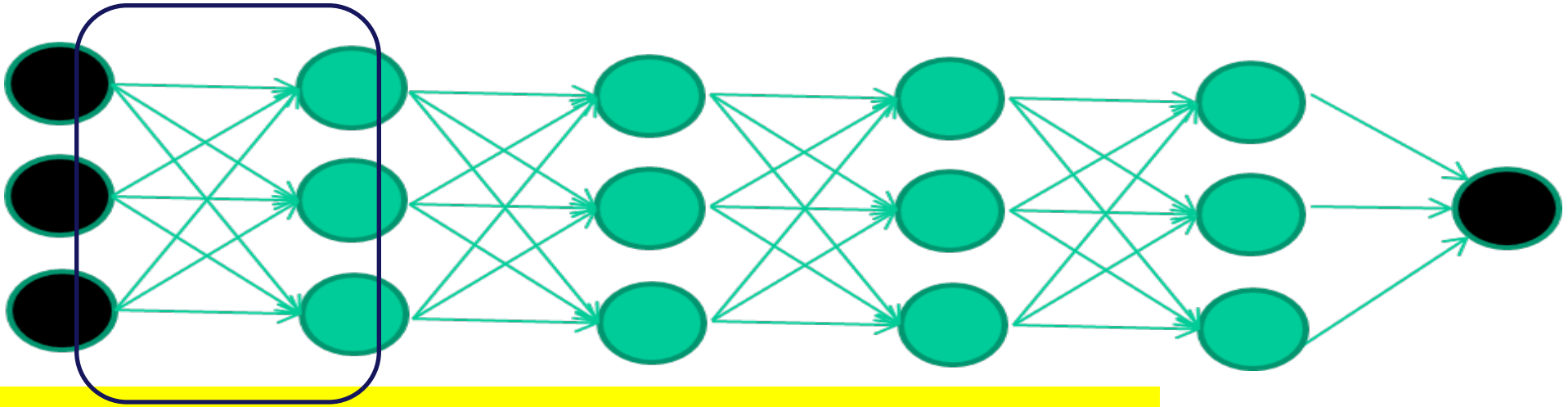
then **this** layer

then **this** laver

then **this** laver

finally **this** layer

The new way to train multi-layer NNs…

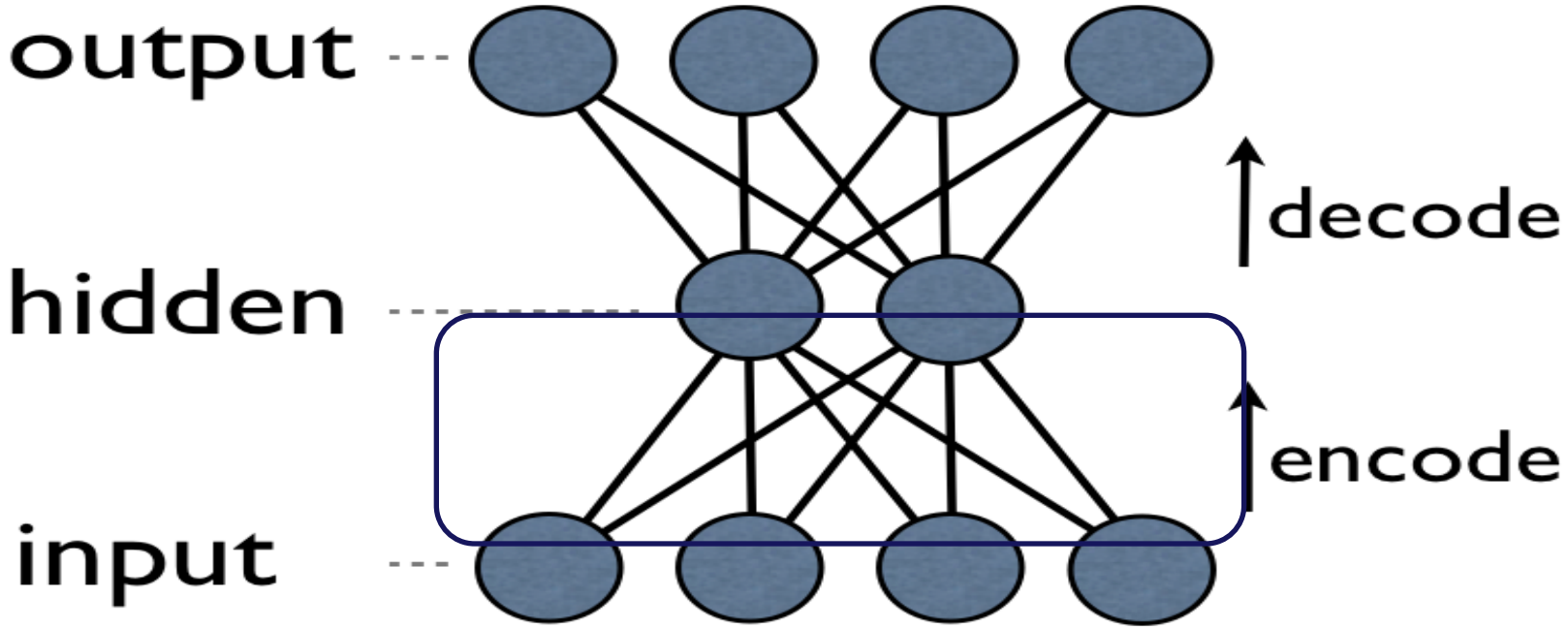


EACH of the (non-output) layers is trained to be an

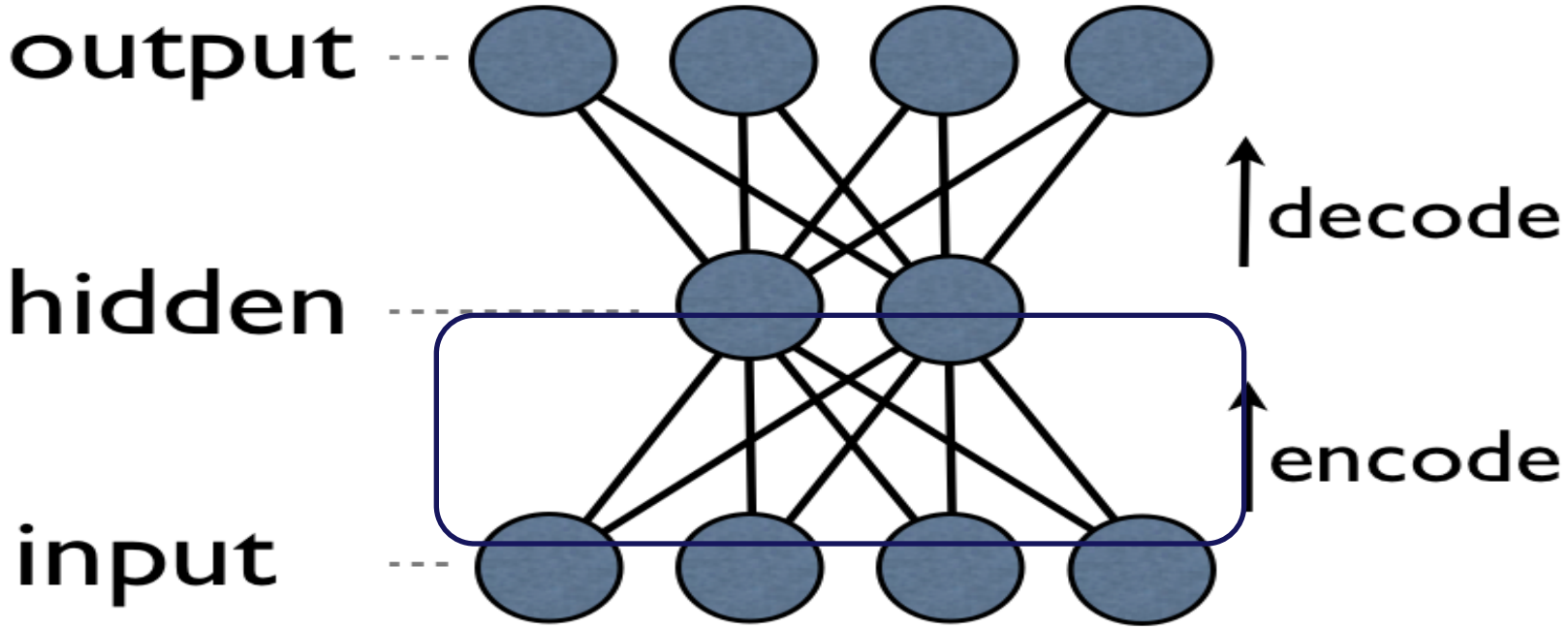
auto-encoder

Basically, it is forced to learn good features that describe what comes from the previous layer

An auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

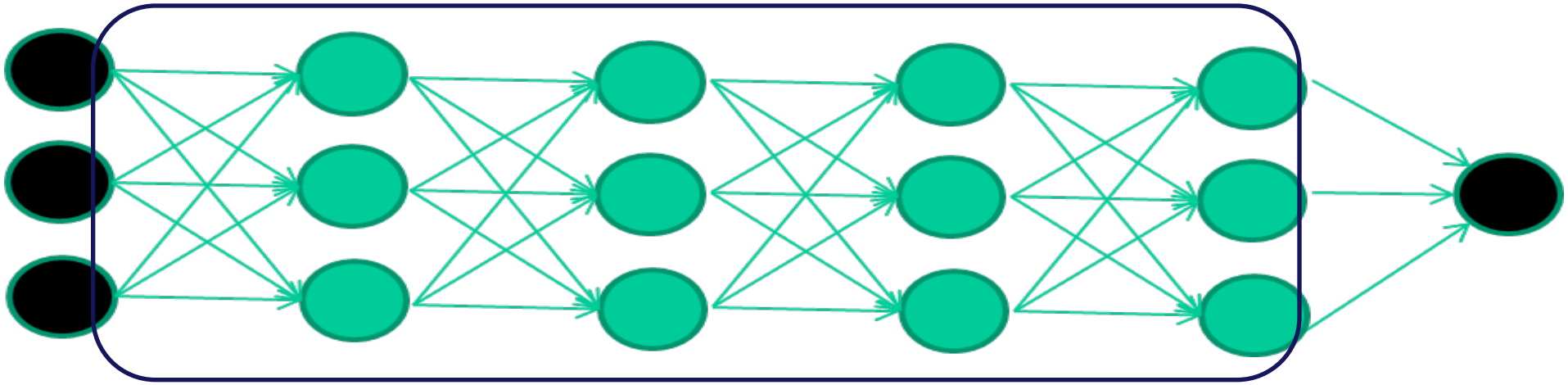


an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

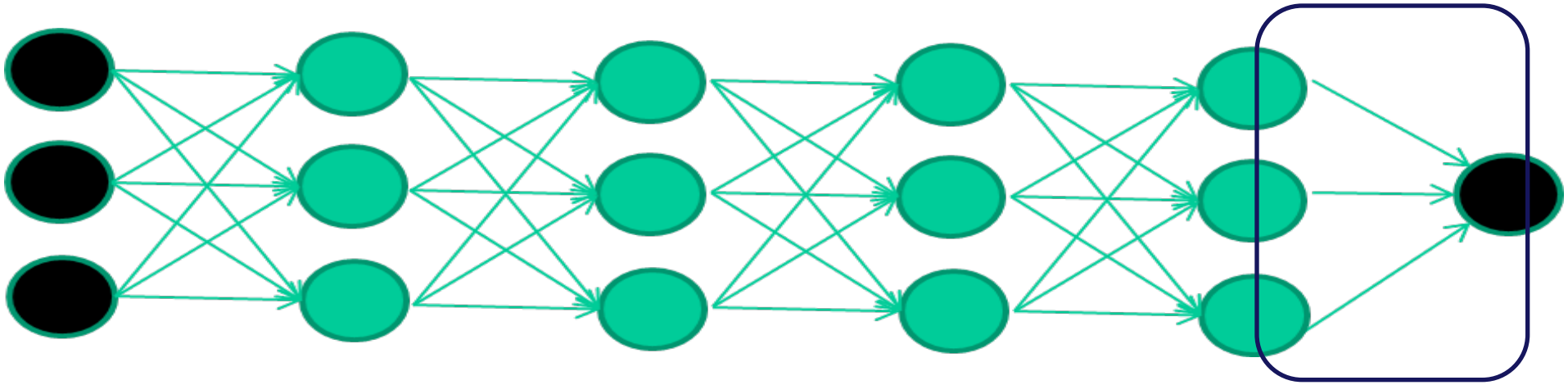


By making this happen with (many) fewer units than the inputs, this forces the 'hidden layer' units to become good feature detectors

Intermediate layers are each trained to be auto encoders (or similar)



Final layer trained to predict class based on outputs from previous layers



More in depth on the math

(http://mt-class.org/jhu/slides/lecture-nn-computation-graphs.pdf)

Neural Networks for NLP

Standard Word Representations

- Until now words encoded as atomic symbols: **class, lesson, book**
- In vector space, words are represented in a vector with one "1" and a lot of zeroes
  - this may result in more than 50k-dimensional vectors, even for small datasets
- With "**one-hot**" representations "class" and "lessons" are considered different.

The need of distributed representations

Standard NLP techniques can easily fail due to atomic symbol representations, e.g., lexical gap

One-hot representation:

Car = (0, 0, 0, …, 0, 1, 0)
Train = (0, 1, 0, ..., 0, 0, 0)

Car != Train

Distributed representation:

Car = (0.1, 0.2, -0.2, 0.3, 0.4, 0.1)
Train = (0.1, 0.3, -0.1, 0.2, 0.3, 0.1)

Car ~ Train

Distributed Representations Of Words

- You can get a lot of value by representing a word by means of its neighbors:

  “You shall know a word by the company it keeps” (J. R. Firth 1957)

  The teacher’s **class** is about Deep Learning

  Tom is teaching a **lesson** on Deep Learning

- Words around “class” and “lesson” provides their semantic characterization.

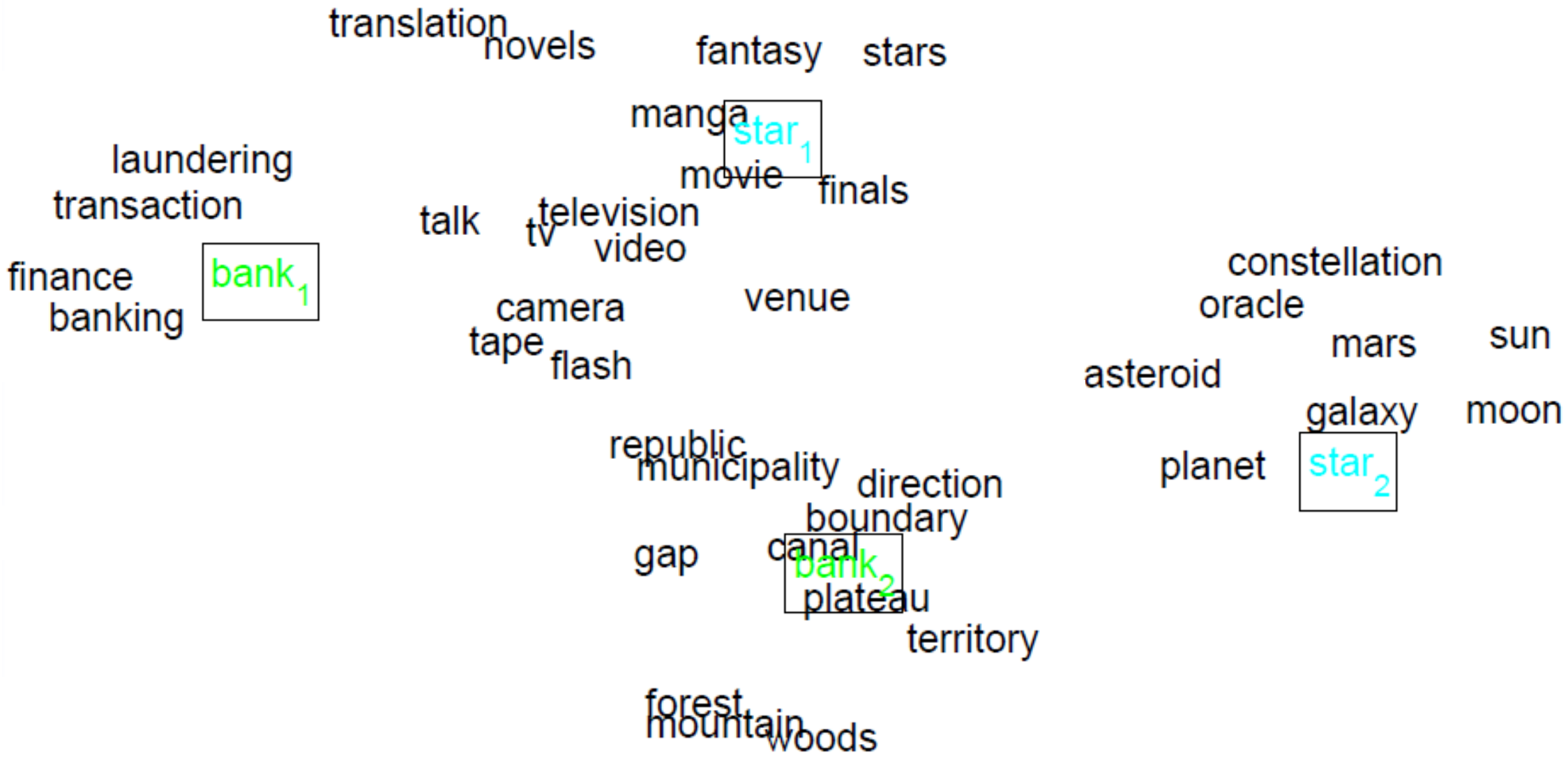
Neural Word Embeddings

- Represent words as dense a vector with weights trained with the prediction of a probabilistic model (Collobert & Weston 2008, Turian et al. 2010, Mikolov et al 2013).

$$Class = \begin{pmatrix} 0.3 \\ -0.2 \\ 0.1 \\ 0 \\ 0.9 \\ -0.8 \\ 0.05 \end{pmatrix}$$

- Eg. Probability of a word given the context (CBOW) or vice versa (Skip Gram)

Bidimensional visualization of Word Embeddings



Properties of Word Embeddings

Powerful tool to encode syntactic and semantic information about words !!

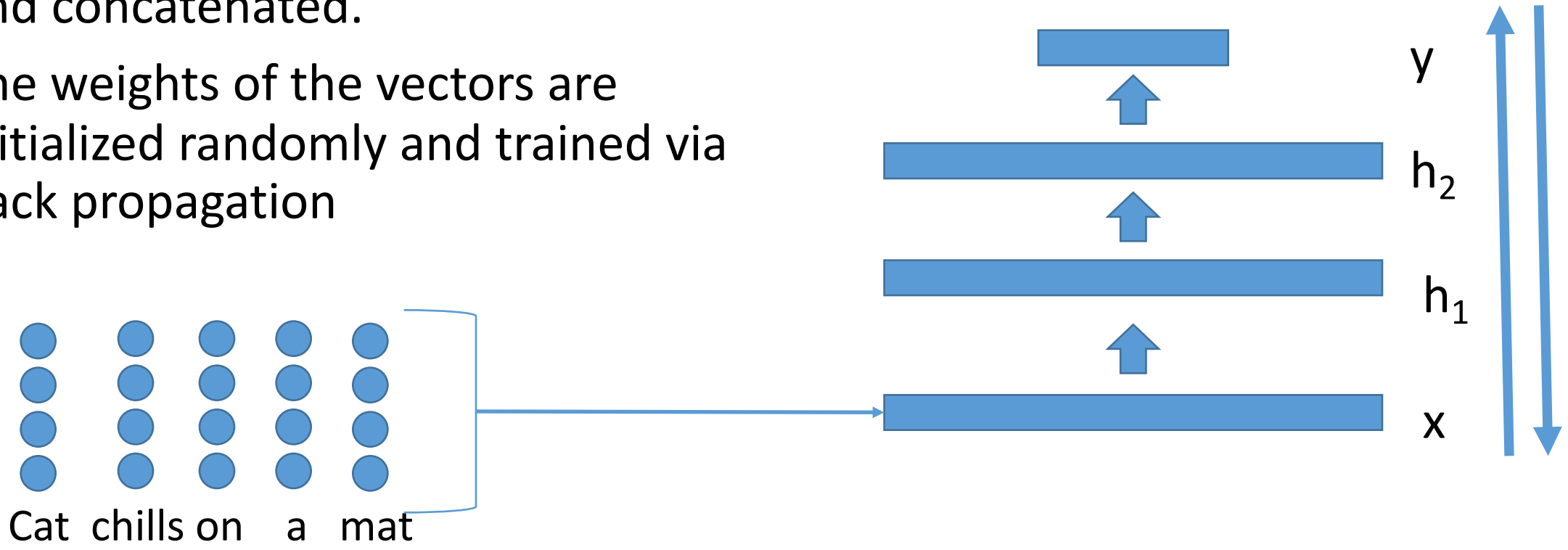
Analogy tests can be easily solved by doing vector subtraction in the embedding space:

- Syntactically: $x_{car} - x_{cars} \approx x_{apple} - x_{apples}$
- Semantically: $x_{shirt} - x_{clothing} \approx x_{chair} - x_{forniture}$

And we can go further: $x_{king} - x_{man} + x_{woman} \approx x_{queen}$

Training Word Embeddings

- Words are encoded in dense vectors and concatenated.
- The weights of the vectors are initialized randomly and trained via back propagation



Deep Learning models

- Rely only on words as input
- Words are represented as vectors aka embeddings
- Learn compositional rules to represent sentences from the input words only
- Captures syntactic/semantic information
- Merge feature extraction and learning steps
- State-of-the-art results on many sentence classification tasks [Nal et al., 2014; Kim, 2014]

A Deep Architecture

Output Layer

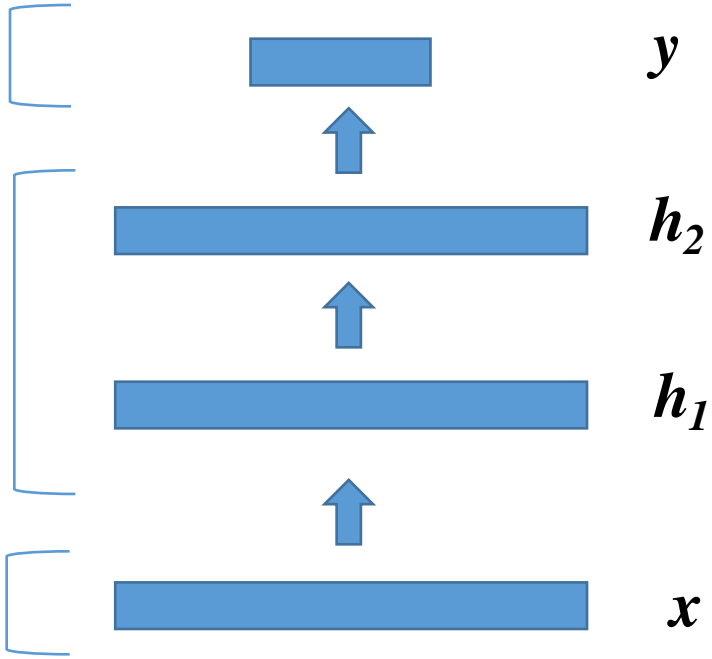
Predicts the supervised target

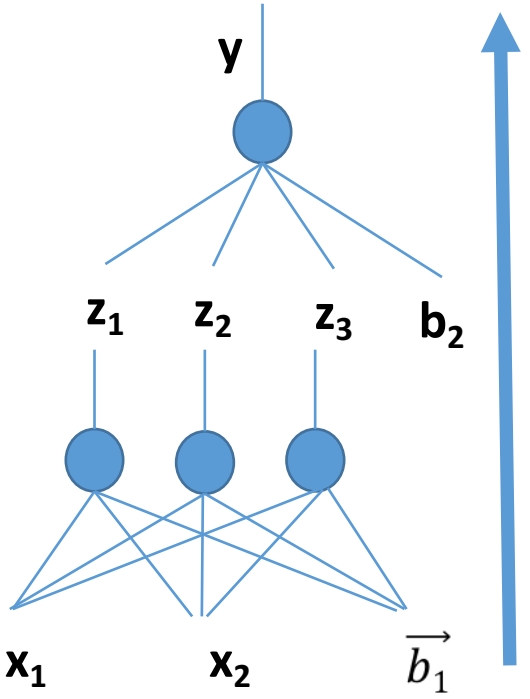
Hidden Layers

Learns more abstract representations

Input Layer

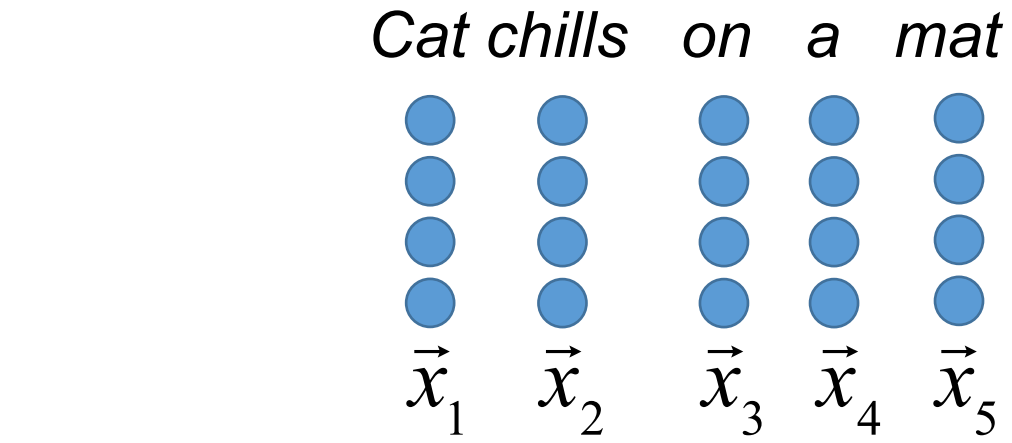
Encodes raw inputs





Word Representation in Neural Networks

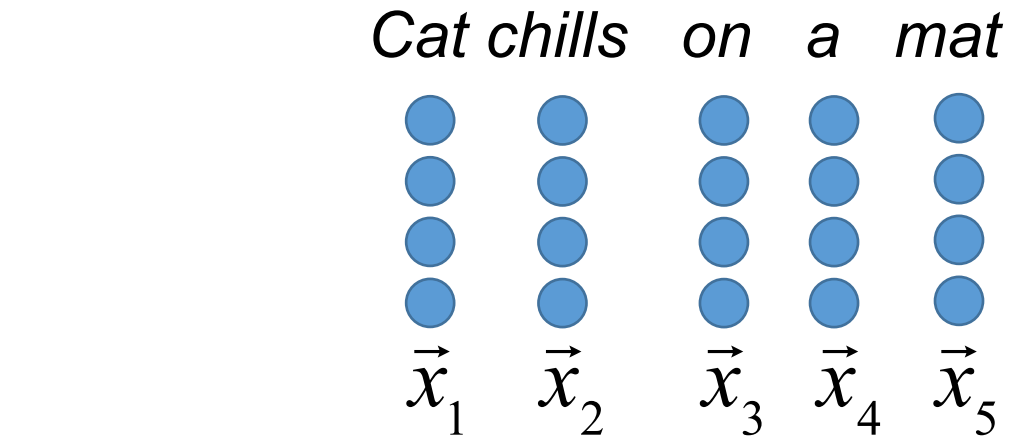
- Given a sentence, e.g.,



- Each words is represented with an fixed size vector

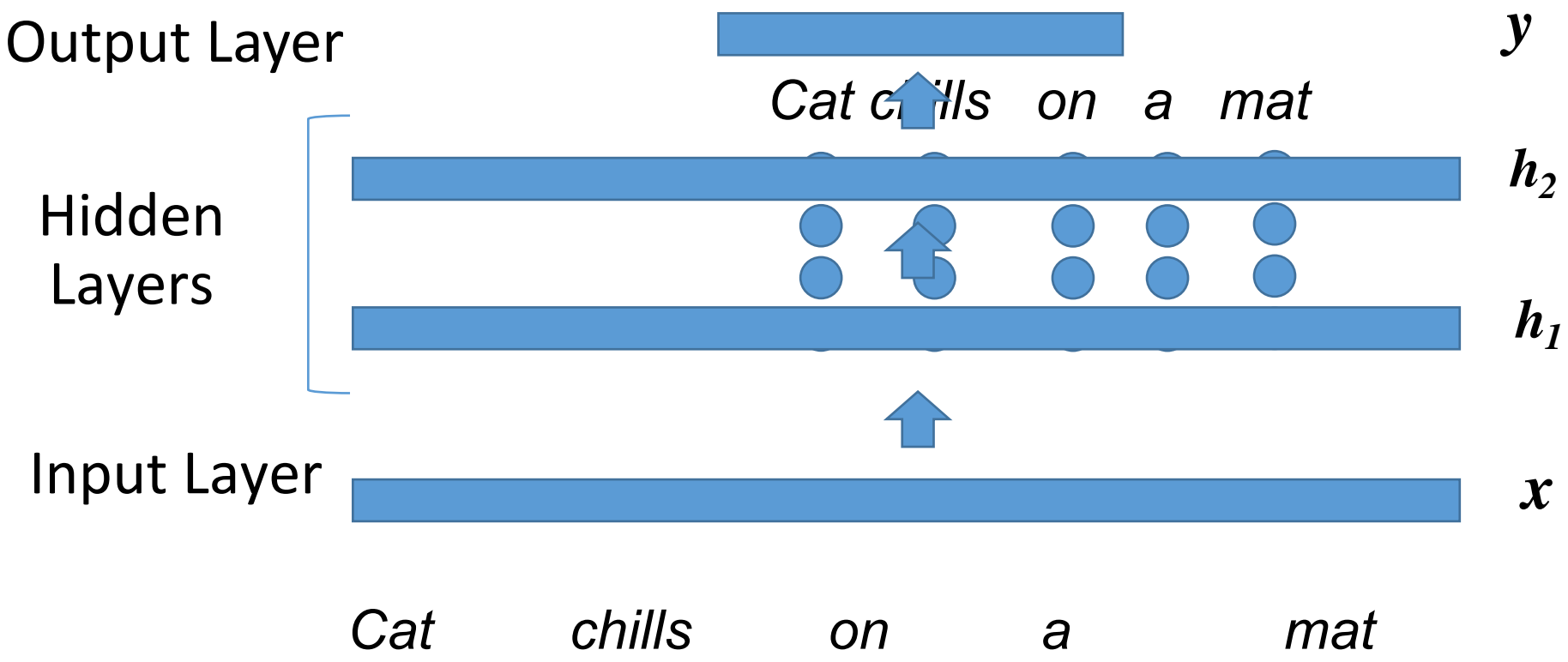
Word Representation in Neural Networks

- Given a sentence, e.g.,



- These vectors are concatenated and fed to the first hidden layer of the Network

Word Representation in Neural Networks



Word Embeddings

- Continuous Bag-Of-Words (CBOW)
  - SkipGram
  - GloVe
- Word embeddings are trained on lemmatized words which are suffixed by the first letter of their POS tag, e.g., be::v
- Training corpora:
  - ukWaC
  - 50 million English tweets from Twitter API

Unsupervised Pre-Training

Problem – embedding trained on a supervised task are:

- Task dependent
- Not accurate as typically there is little data
- Easy to overfit

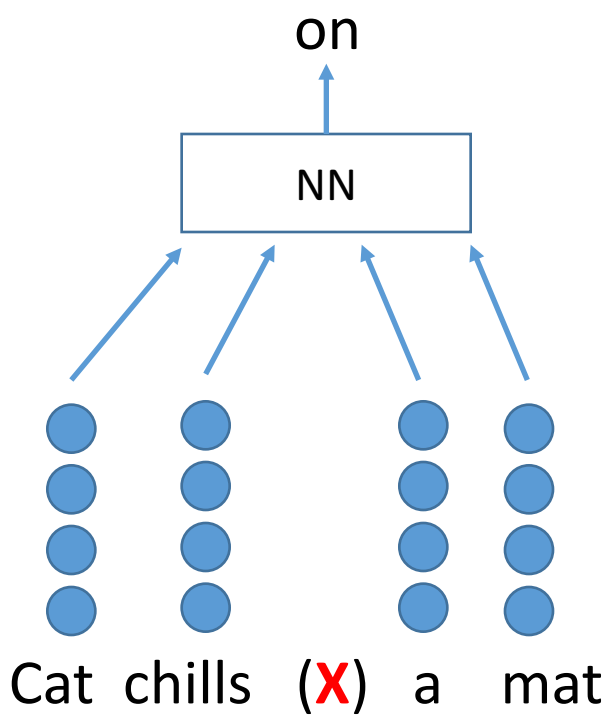
Solution – train a model on a general task that can be trained on a huge dataset.

- Task independent
- More accurate (due to the size of the dataset)
- Generalize better

Word2Vec Tool

- Tool developed by Mikolov to generate word embeddings.
- Two artificial general tasks:
  - CBOW: predict a word given the context
  - SkipGram: predict a context given the word
- Allows to train embedding on very large datasets (e.g., Wikipedia)

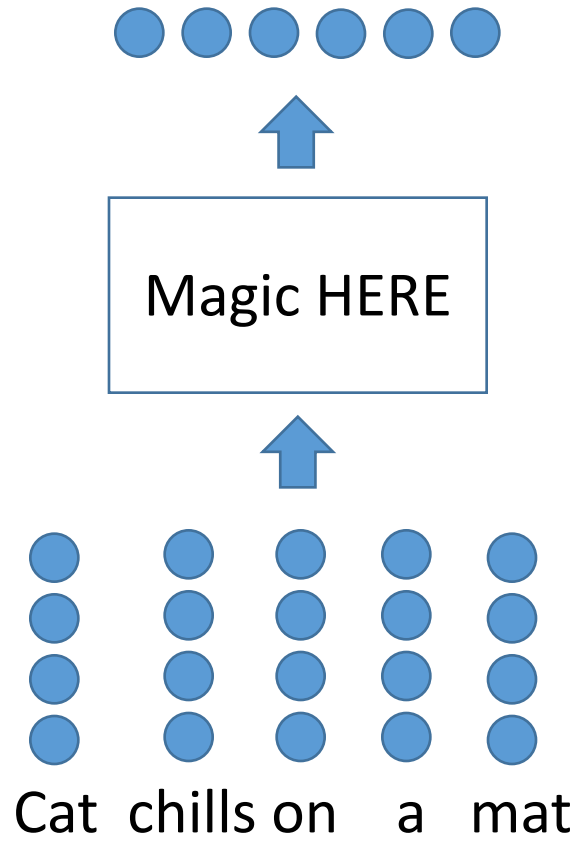
CBOW model



From Word To Sentence Embeddings

Words can be encoded as vectors

But what about sentences?

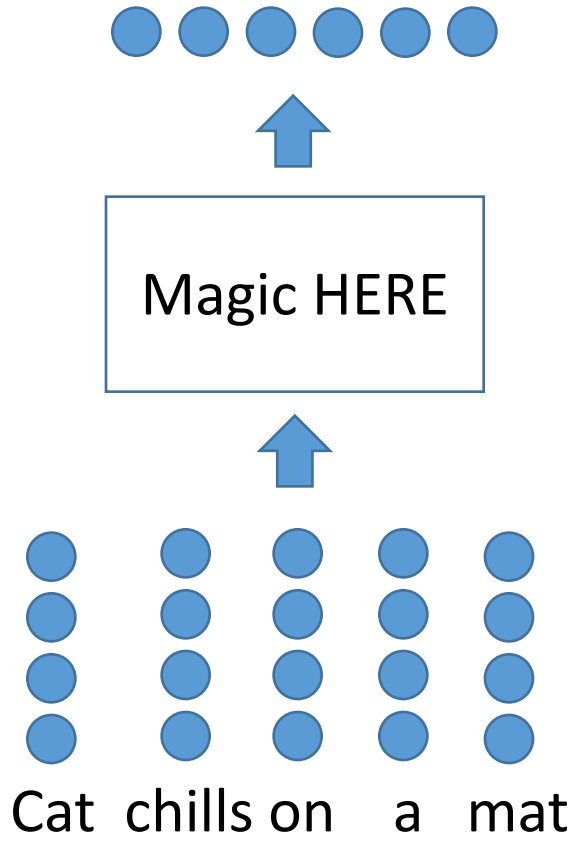


From Words To Sentences

Words can be encoded as vectors.

But what about sentences?

Concatenate words vector



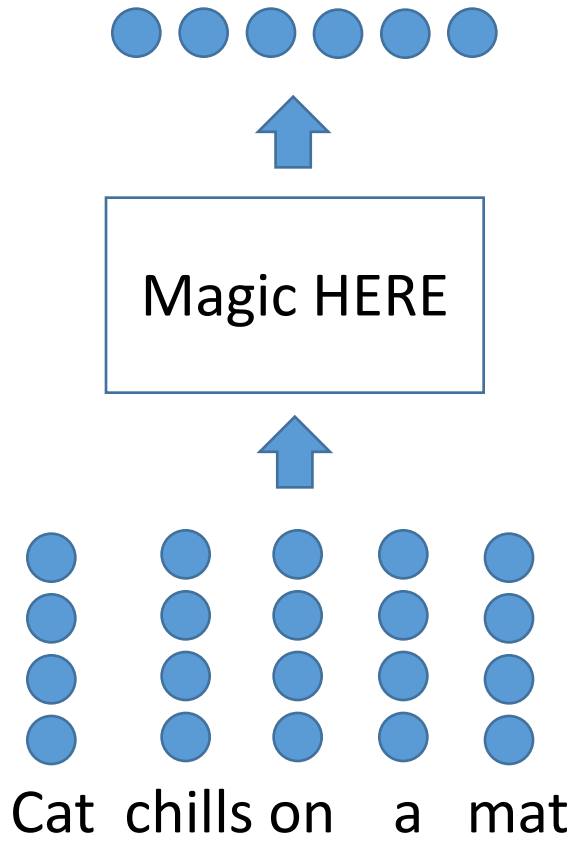
From Words To Sentences

Words can be encoded as vectors.

But what about sentences?

~~**Concatenate** words vector.~~

Sum/Average them.



From Words To Sentences

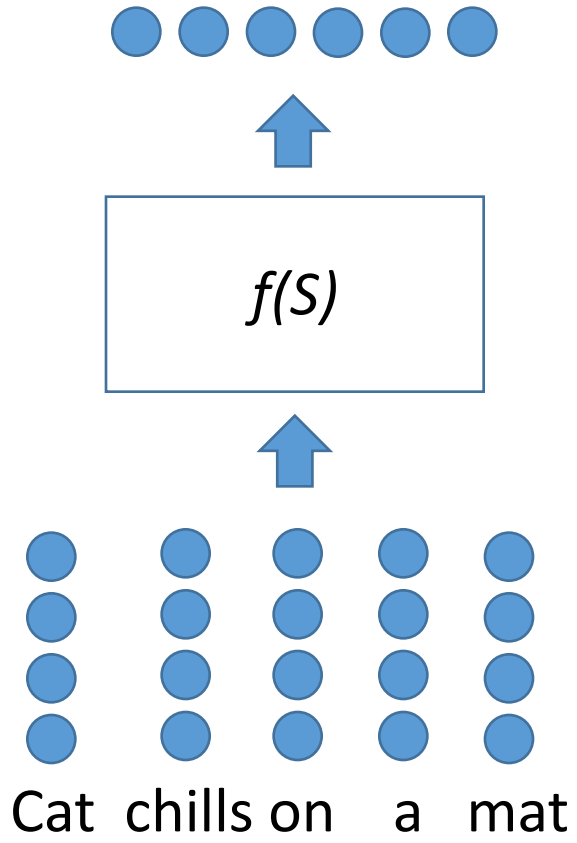
Words can be encoded as vectors.

But what about sentences?

~~**Concatenate** words vector.~~

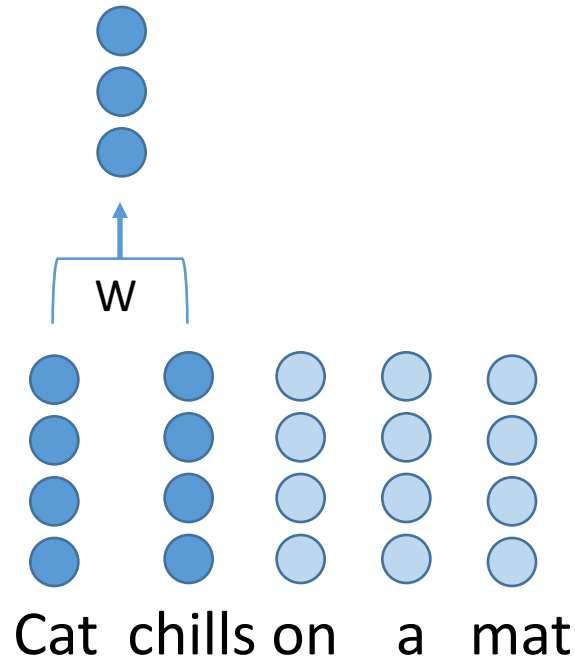
~~**Sum/Average** them.~~

Neural Sentence Model.



Convolutional Neural Networks

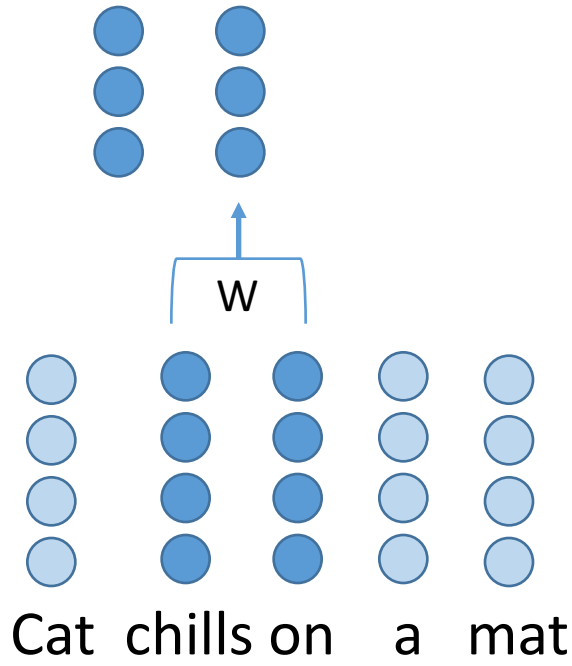
Maps groups of words (context) into vectors. Using a standard Neural Network Layer.



Convolutional Neural Networks

Maps groups of words (context) into a new vector. Using a standard Neural Network Layer.

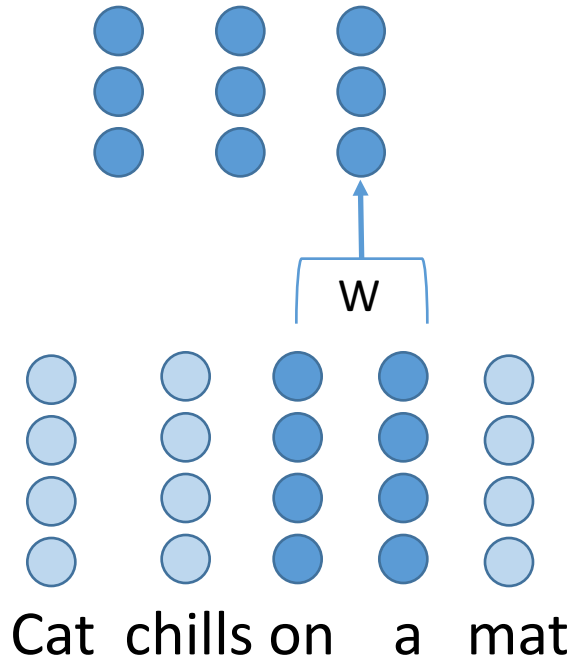
Iterate (convolve) over the whole sentence applying the same layer for each context.



Convolutional Neural Networks

Maps groups of words (context) into a new vector. Using a standard Neural Network Layer.

Iterate (convolve) over the whole sentence applying the same layer for each context.

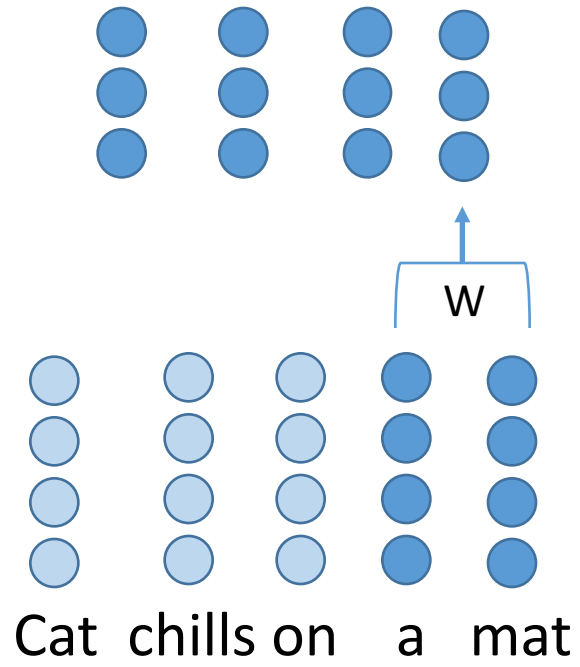


Convolutional Neural Networks

Maps groups of words (context) into a new vector. Using a standard Neural Network Layer.

Iterate (convolve) over the whole sentence applying the same layer for each context.

Obtaining a matrix of word-context vectors

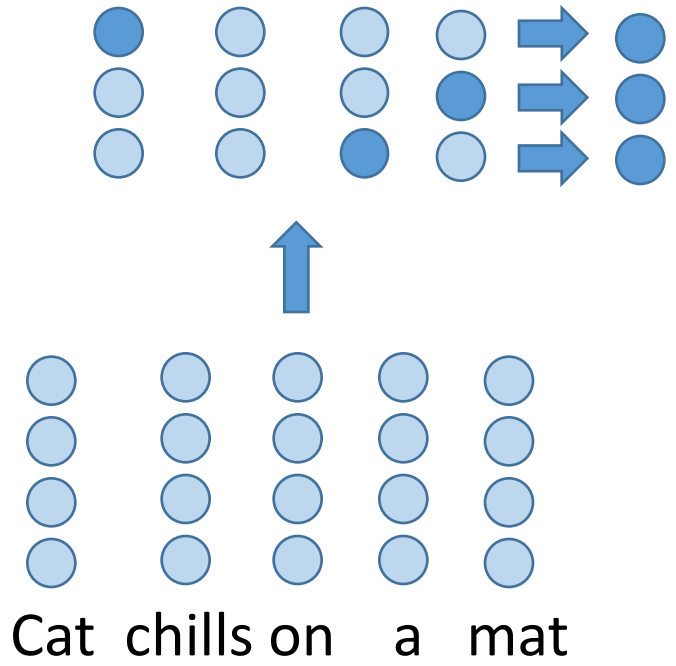


Pooling

We need to map the matrix as output of the convolution into a fixed size vector.

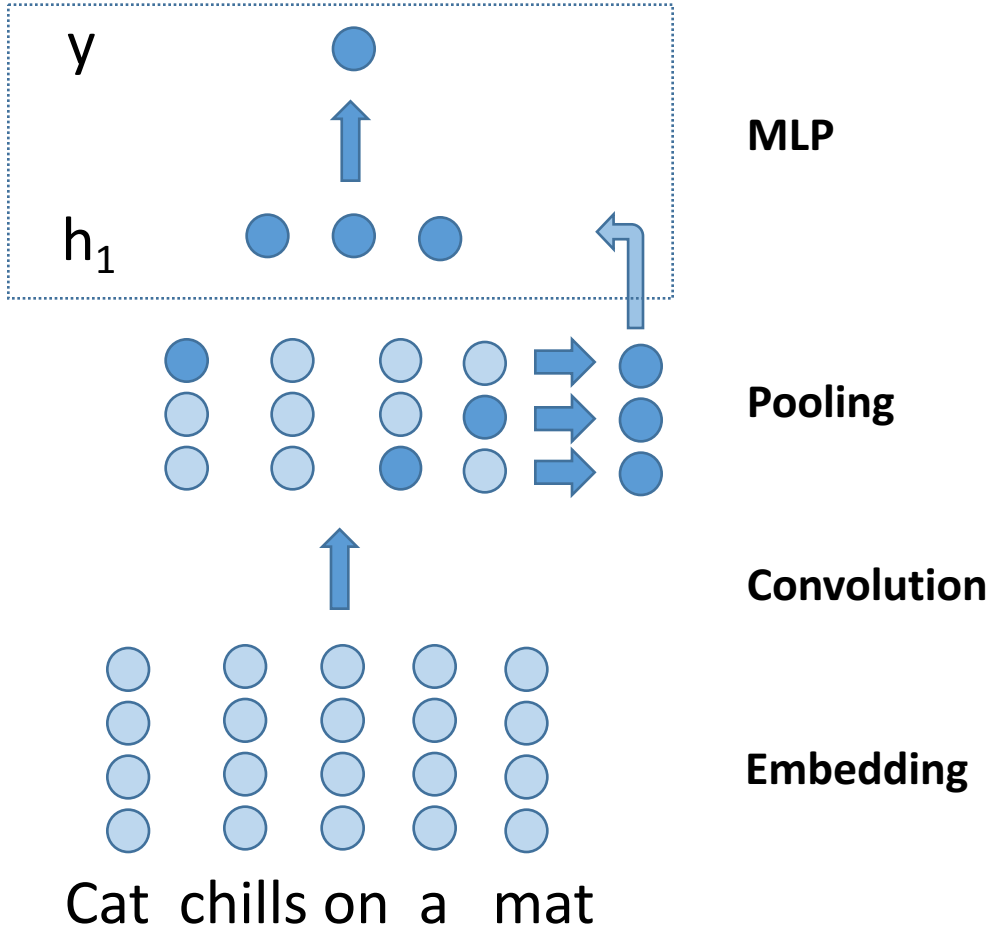
- Average Over Time
- Max Over Time

Max Over Time – Max Pooling

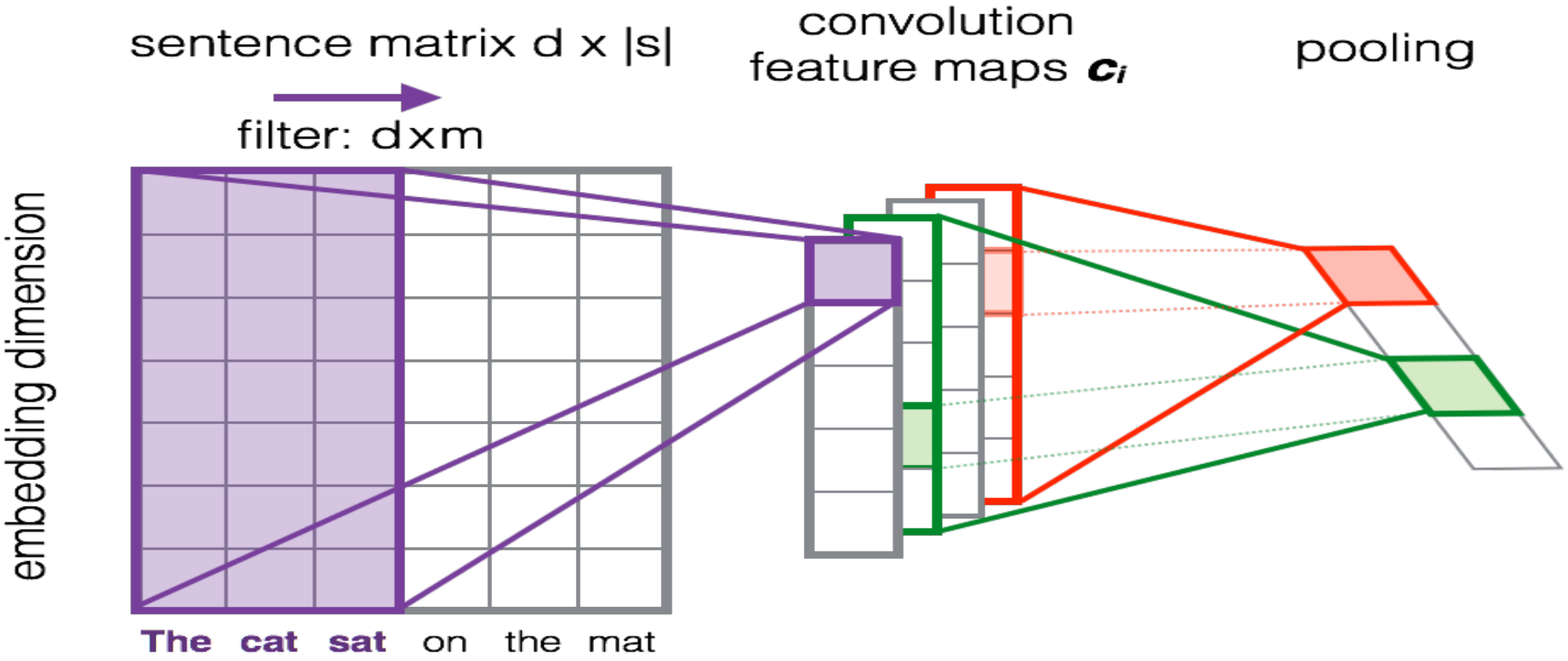


Sentence Classification using CNNs

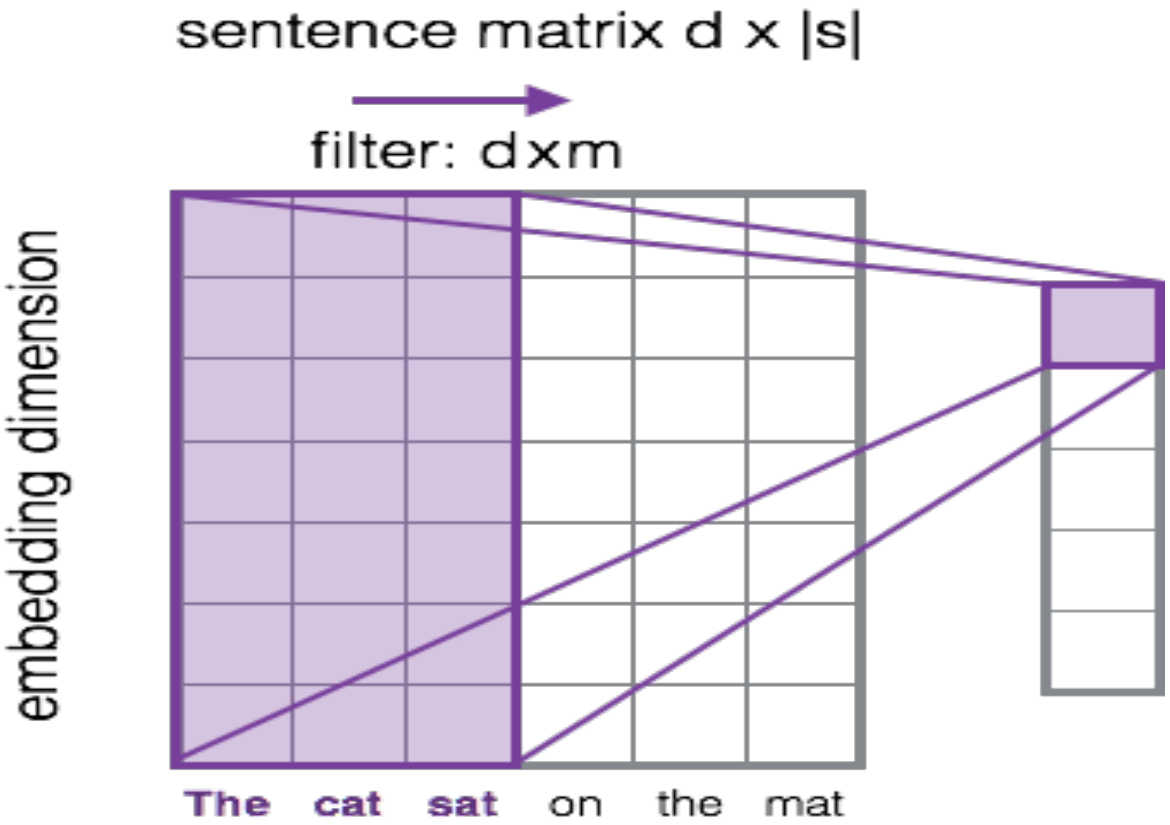
The sentence encoded into fixed size vectors can be fed to standard Neural Network Layer (MLP) for the final classification.



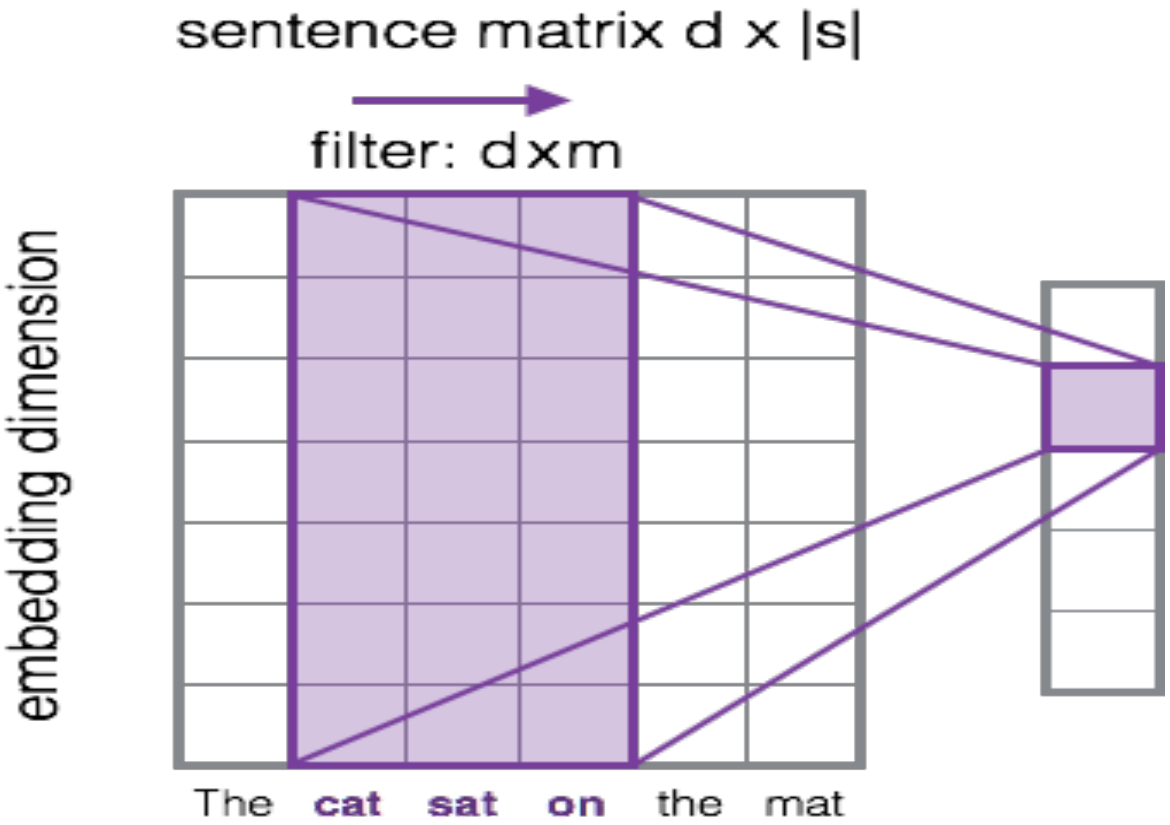
ConvNet Sentence Model



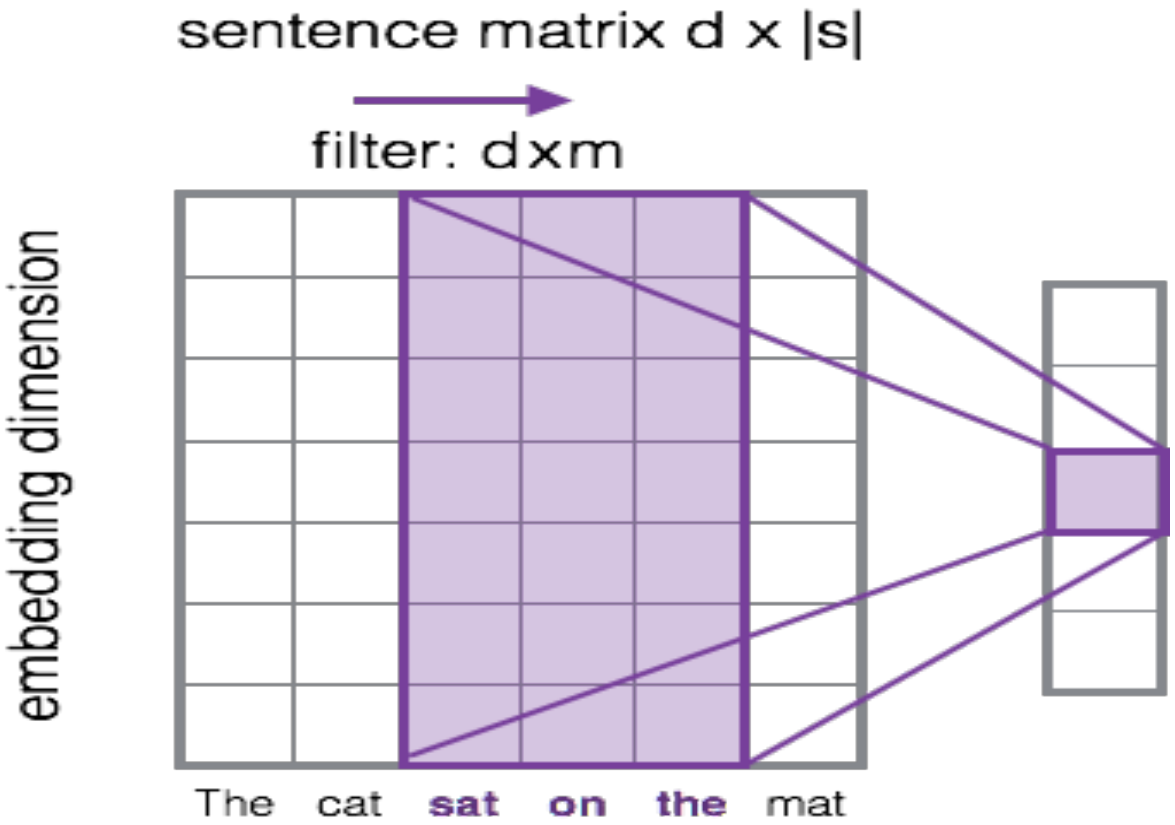
Convolution Op



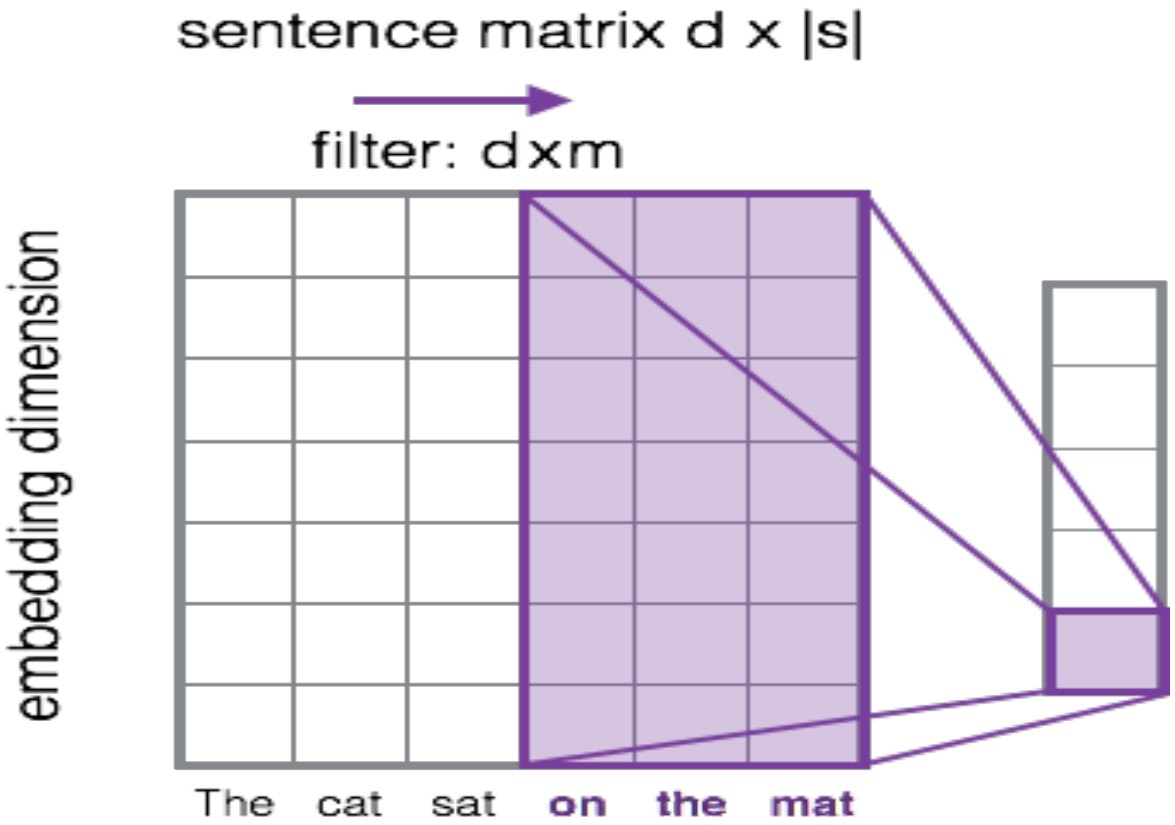
Convolution Op



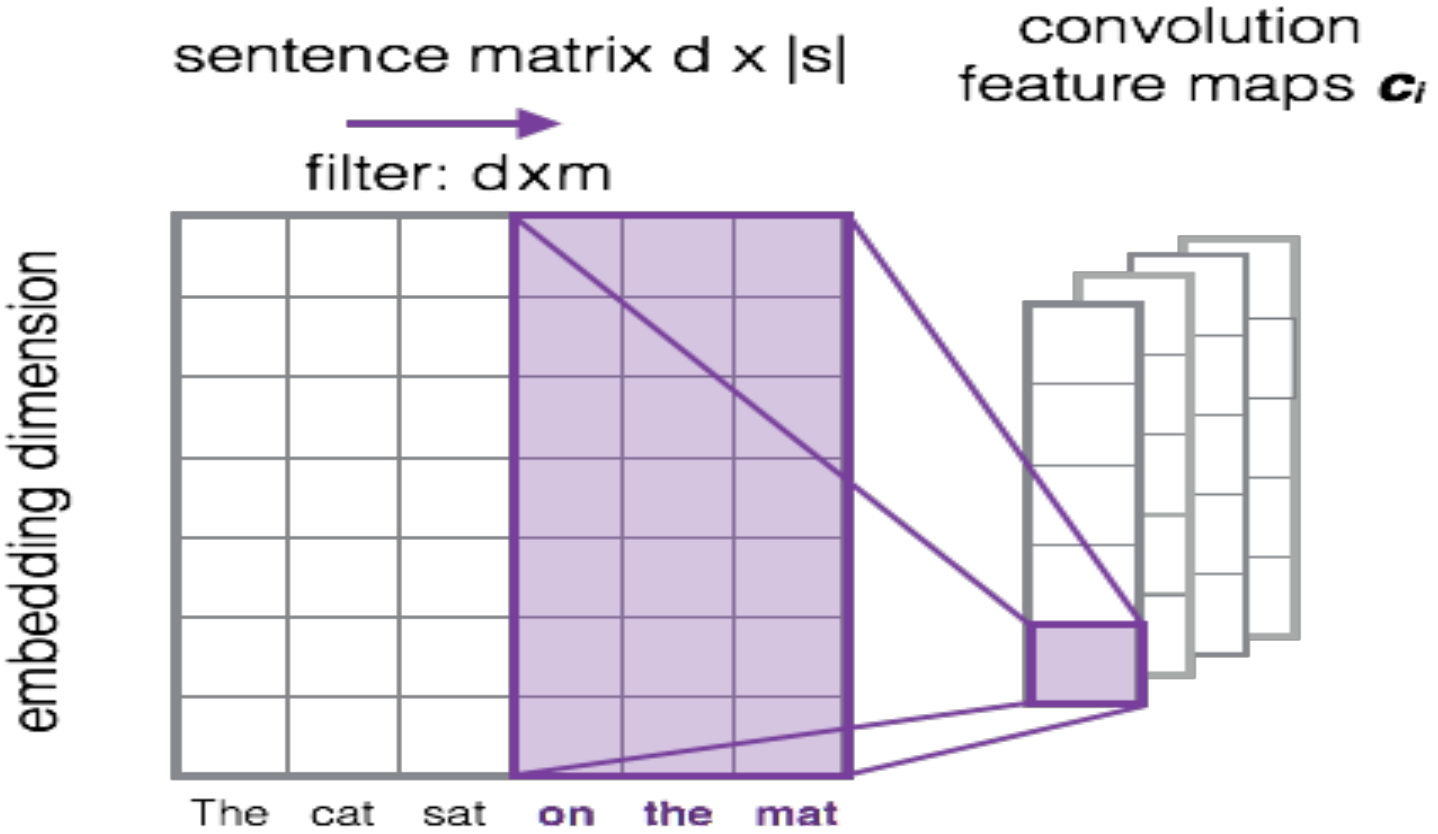
Convolution Op



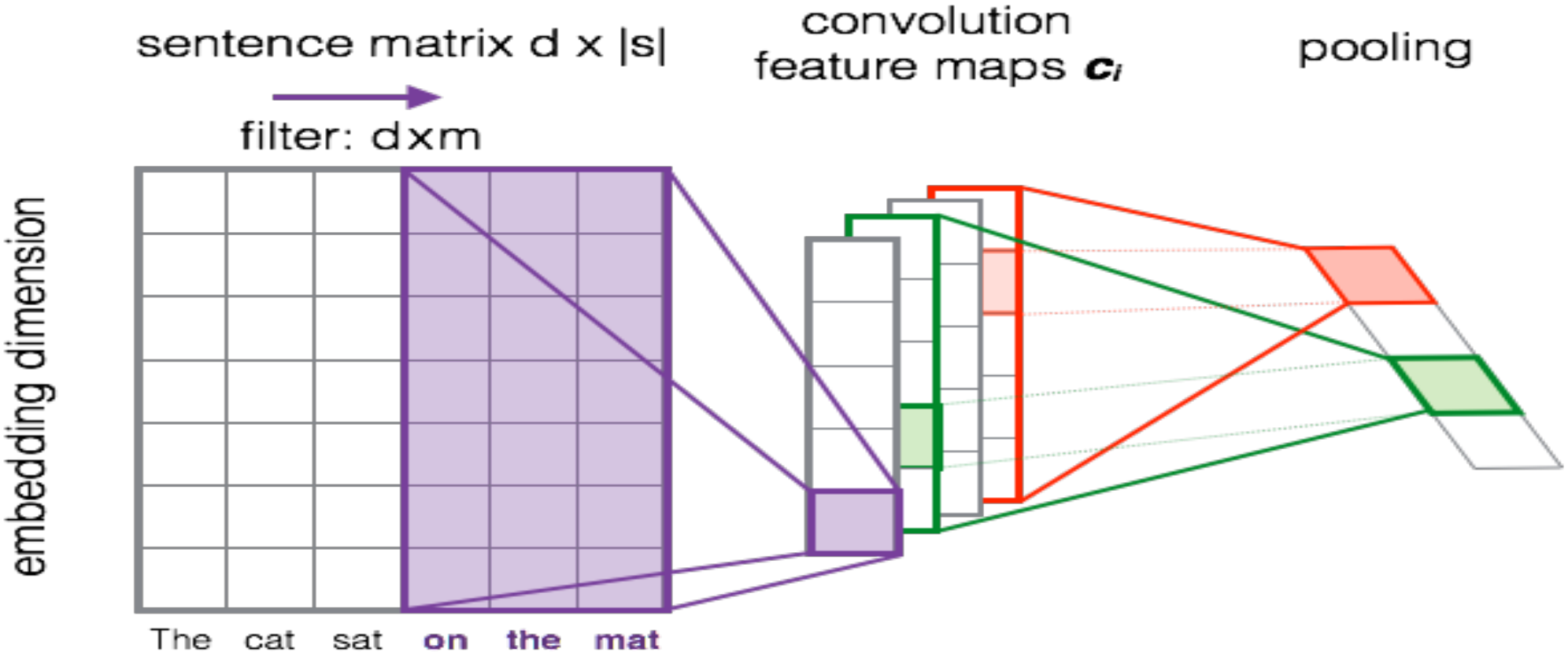
Convolution Op



Multiple Convolution feature maps



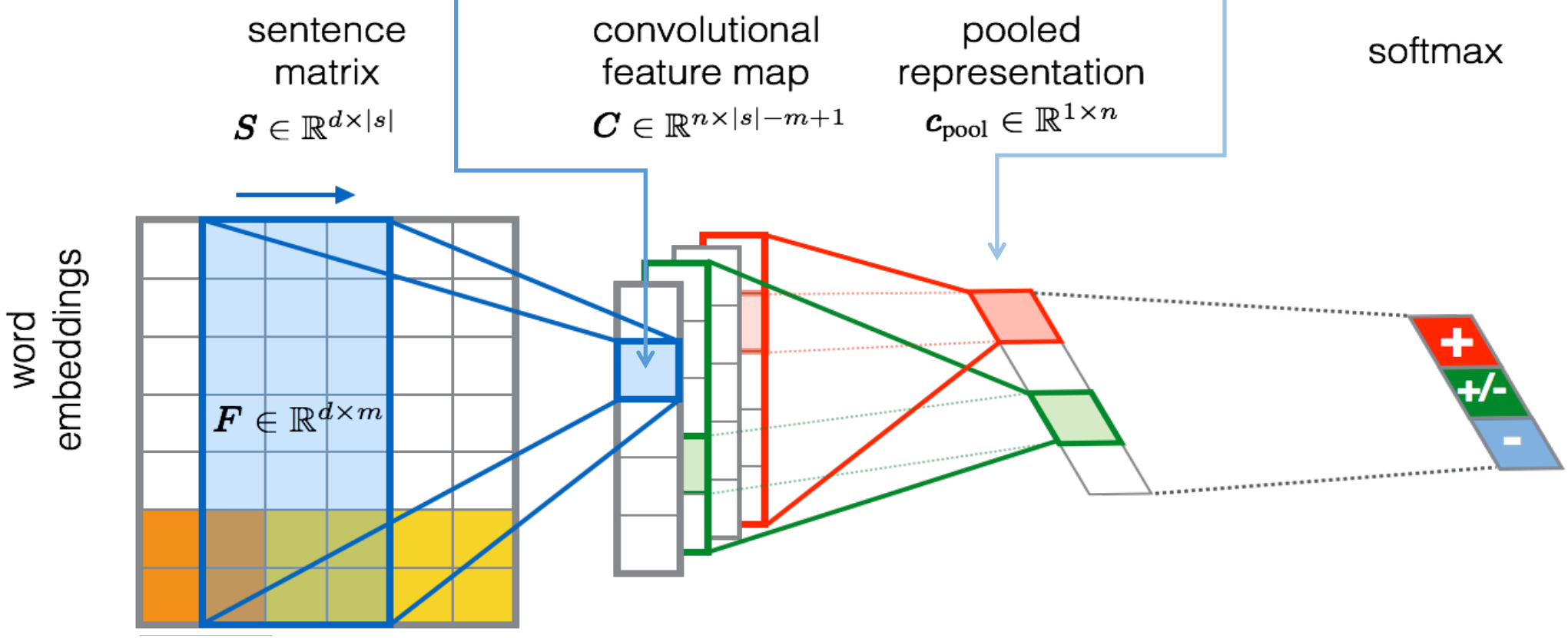
Pooling


sentence matrix d x |s|
convolution
feature maps c_i
pooling
filter: dxm
embedding dimension
The cat sat on the mat

More technically

$$\mathbf{c}_i = (\mathbf{S} * \mathbf{F})_i = \sum_{k,j} (\mathbf{S}_{[:,i-m+1:i]} \otimes \mathbf{F})_{kj}$$

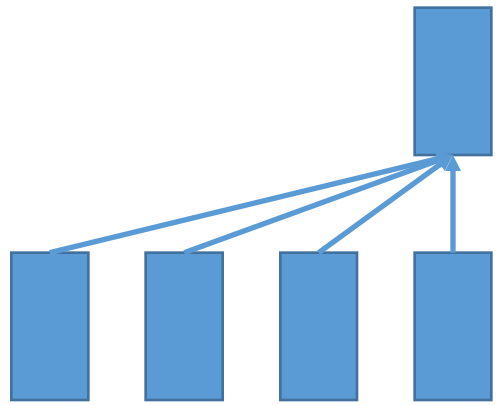
$$\text{pool}(\mathbf{c}_i) : \mathbb{R}^{|s|+m-1} \rightarrow \mathbb{R}$$



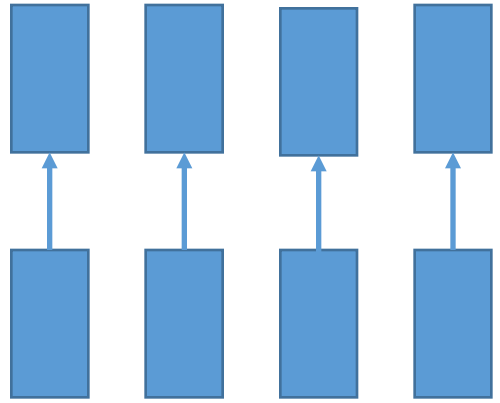
Sequence Labeling

Sequence labeling is the most common task in NLP:

- **Many To One:** assign a label to a sequence (Sentence Classification)
- **Many To Many:** assign a label to each word in the sentence (POS, NER) – *Sequence Labeling*



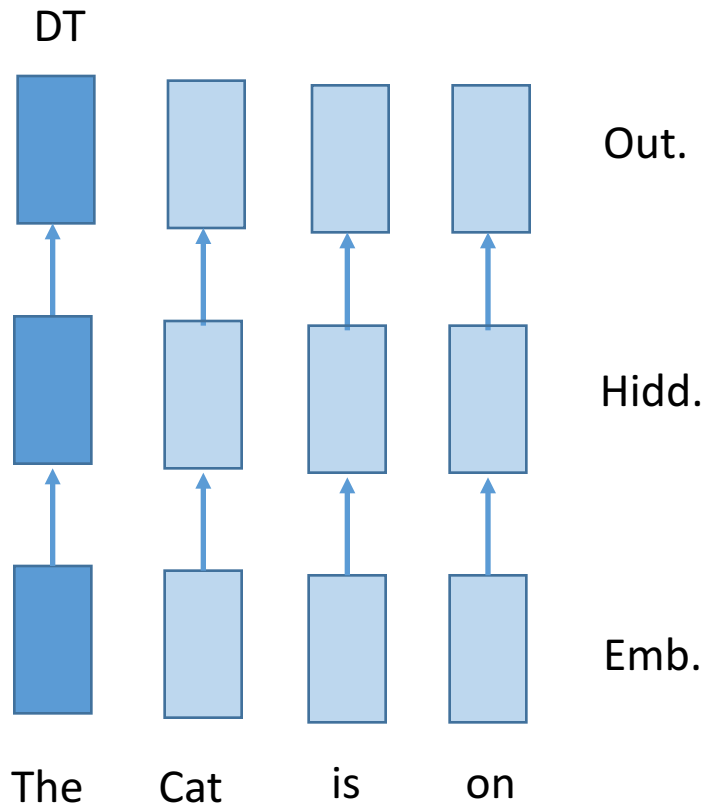
Many to One



Many to Many

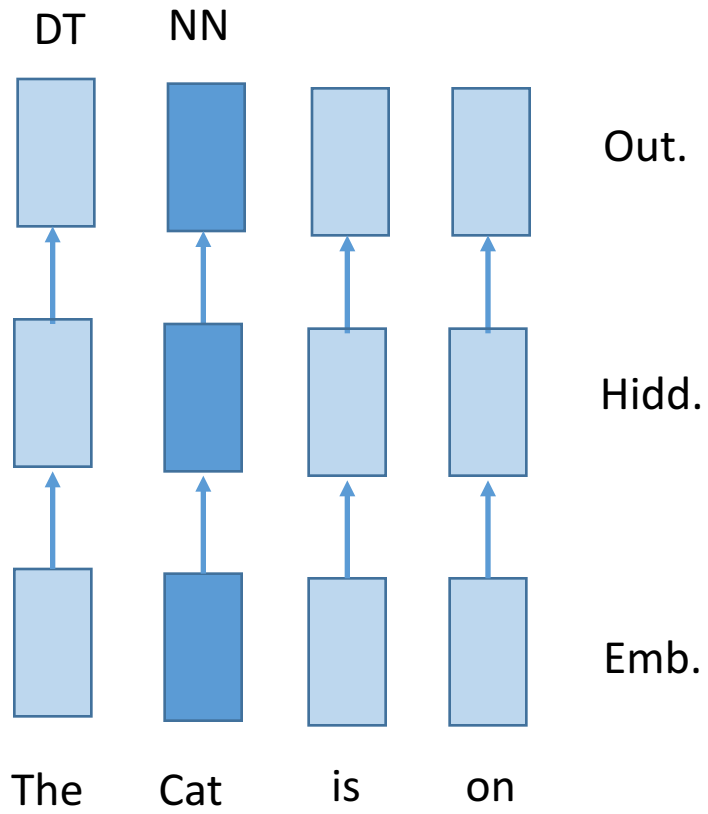
Neural Networks for Sequence Labeling

1. Words are mapped to words embeddings.
2. Each word embedding is fed as input to a MLP
3. The MLP predicts the probability of a label (e.g. POS tag).



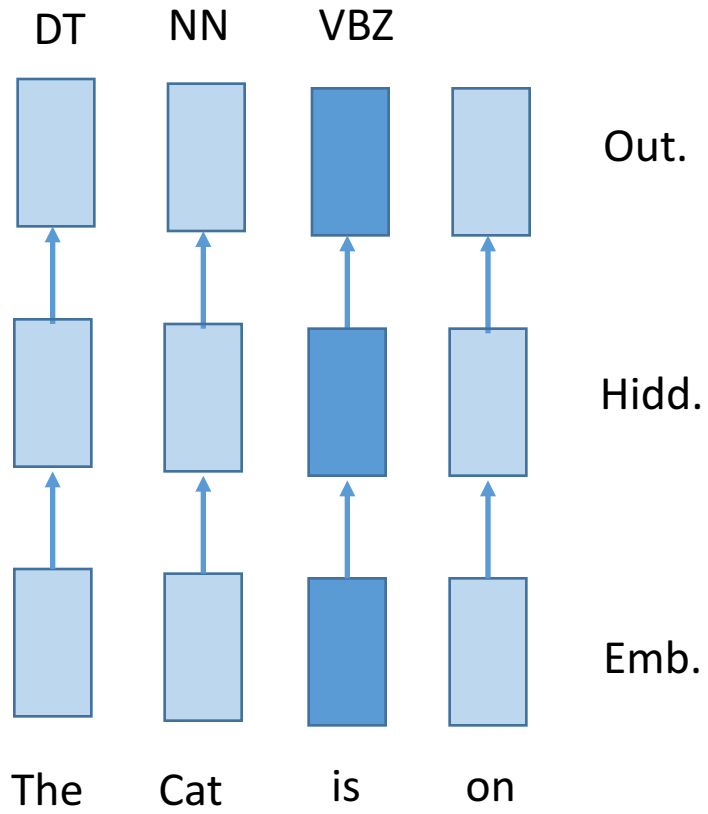
Neural Networks for Sequence Labeling

1. Words are mapped to words embeddings.
2. Each word embedding is fed as input to a MLP
3. The MLP predicts the probability of a label (e.g. POS tag).



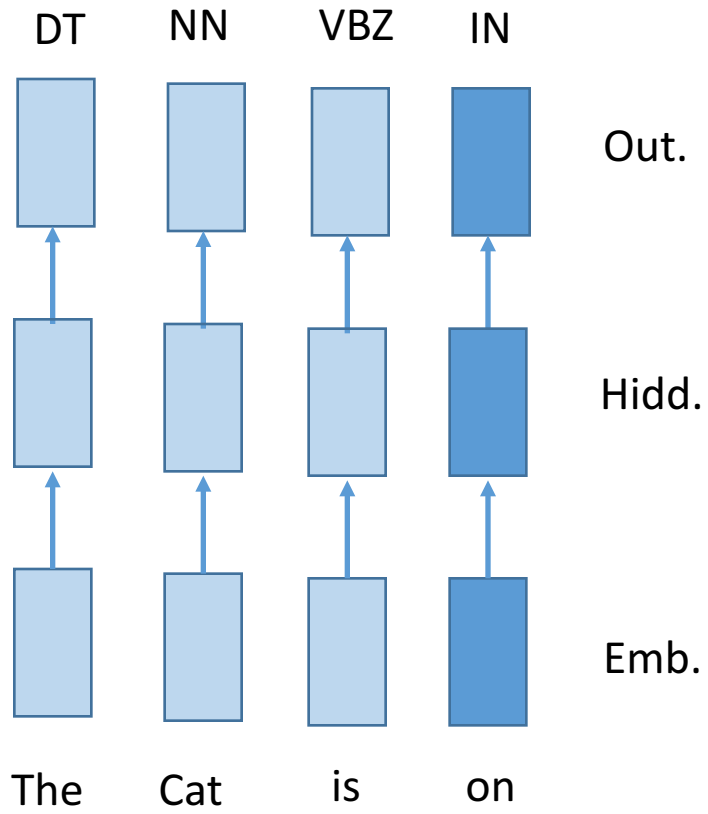
Neural Networks for Sequence Labeling

1. Words are mapped to words embeddings.
2. Each word embedding is fed as input to a MLP
3. The MLP predicts the probability of a label (e.g. POS tag).



Neural Networks for Sequence Labeling

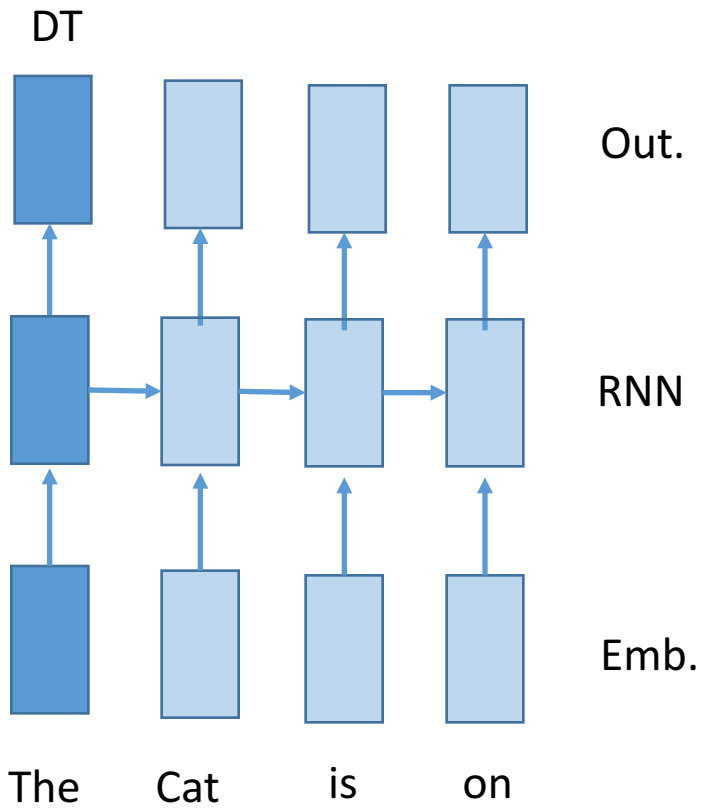
1. Words are mapped to words embeddings.
2. Each word embedding is fed as input to a MLP
3. The MLP predicts the probability of a label (e.g. POS tag).



Recurrent Neural Networks

A Recurrent Neural Networks layer gives the output of the hidden layer as input of the hidden layer itself.

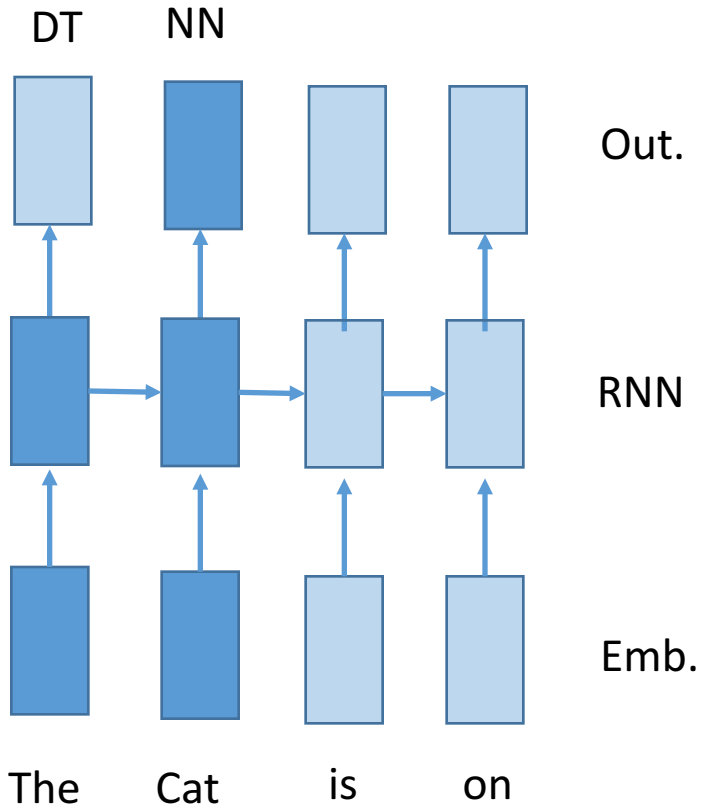
This allows to model dependencies between the words in input



Recurrent Neural Networks

A Recurrent Neural Networks layer gives the output of the hidden layer as input of the hidden layer itself.

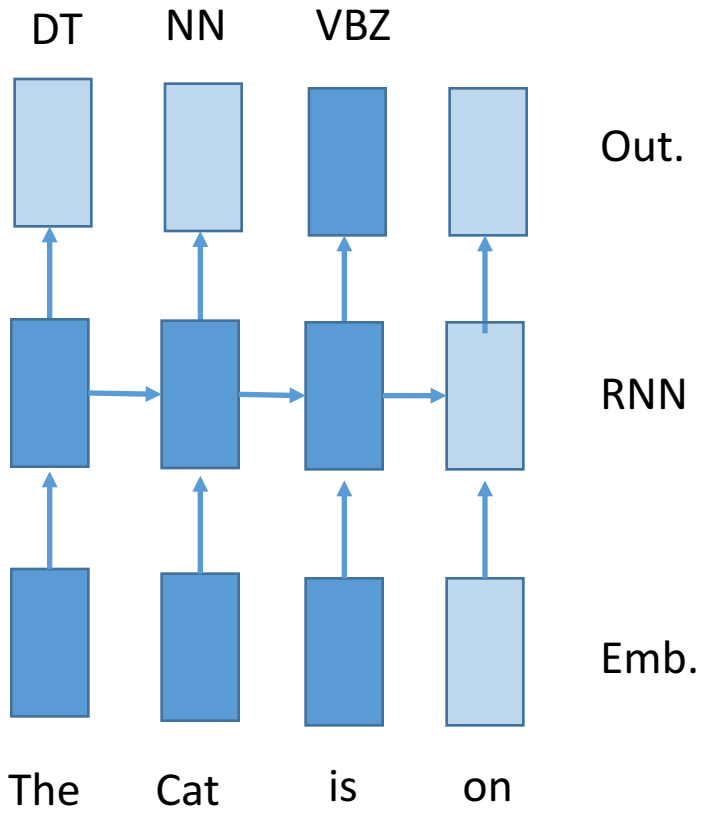
This allows to model dependencies between the words in input



Recurrent Neural Networks

A Recurrent Neural Networks layer gives the output of the hidden layer as input of the hidden layer itself.

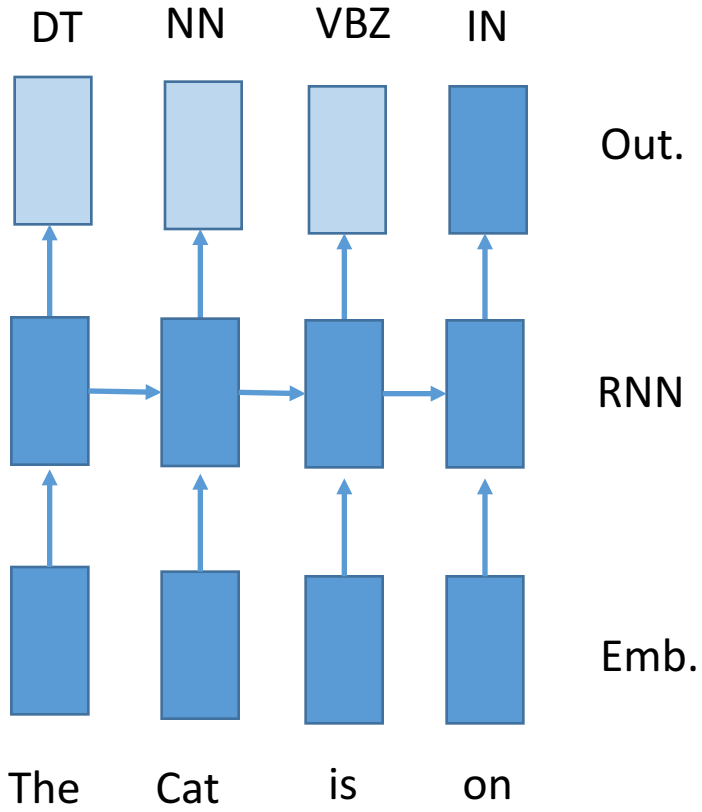
This allows to model dependencies between the words in input



Recurrent Neural Networks

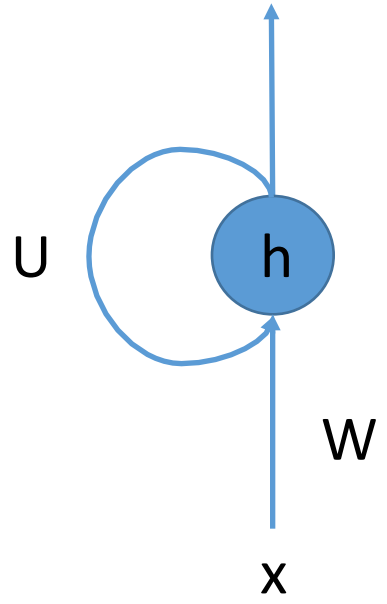
A Recurrent Neural Networks layer gives the output of the hidden layer as input of the hidden layer itself.

This allows to model dependencies between the words in input



RNNs maths

A Recurrent Neural Networks layer gives the output of the hidden layer as input of the hidden layer itself.



$$h := W^T x + U^T h + b$$

More Advanced RNNs

Simple RNN are not used anymore in the literature, because:

- Hard to train (Vanishing/Exploding Gradients)
- They have a single memory (hidden state) that gets corrupted for longer sequences. This because it is overwritten at each iteration.

More advanced RNNs are used nowadays:

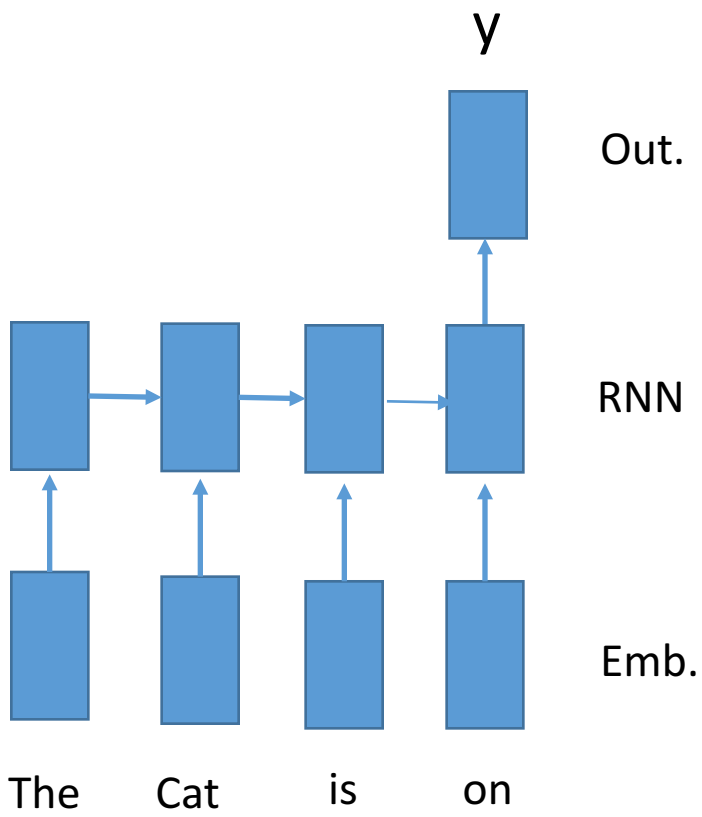
- Long-short term memory (LSTM)
- Gated Recurrent Units (GRU).

RNN for Sentence Modeling

Sentence classification is a special case of Sequence Labeling (assign a label to a sequence).

We can use RNNs to map sentences to fixed size vectors!!!!

The sentence representation will be the last value of the hidden state



Advanced Neural Models

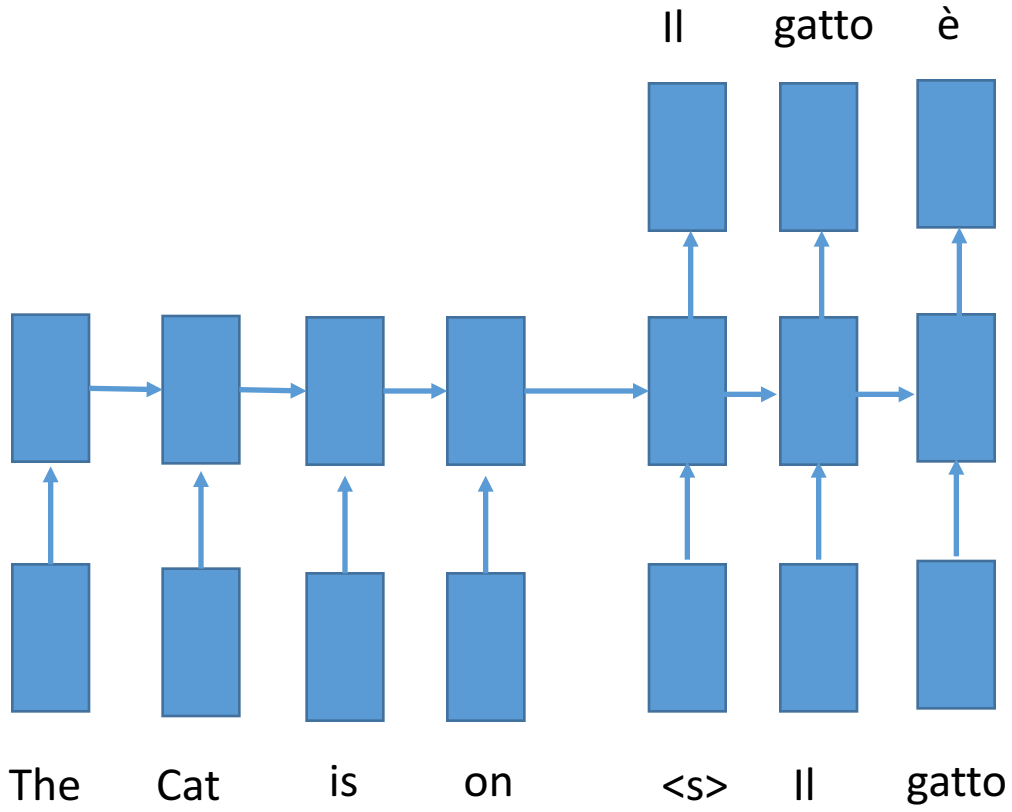
The modularity of Neural Networks make it possible to build more advanced algorithms that could not be done with standard ML techniques.

- Sequence to sequence (Seq2Seq).
- Similarity networks.

Sequence 2 Sequence

These models can be used to solve complex sequence labeling tasks:

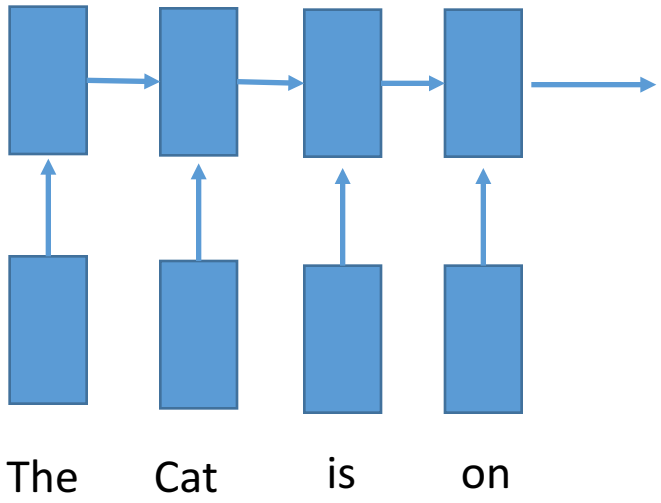
- the number of output label is different from the number of input words.
- E.g. **Machine Translation**



Encoder

The encoder maps the raw input to a fixed size vector.

- the sentence model previously explained

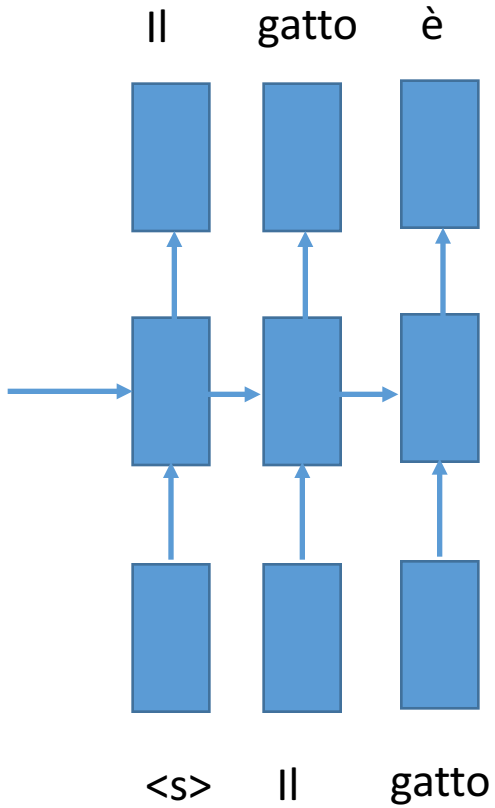


Decoder

Recurrent neural network having the hidden state initialized with the output of the encoder.

The input of the network is the output of the network itself.

At each step it predicts the next word in the sequence.



Summary on Seq2Seq Applications

Machine Translation (English Sentence -> Italian Sentence)

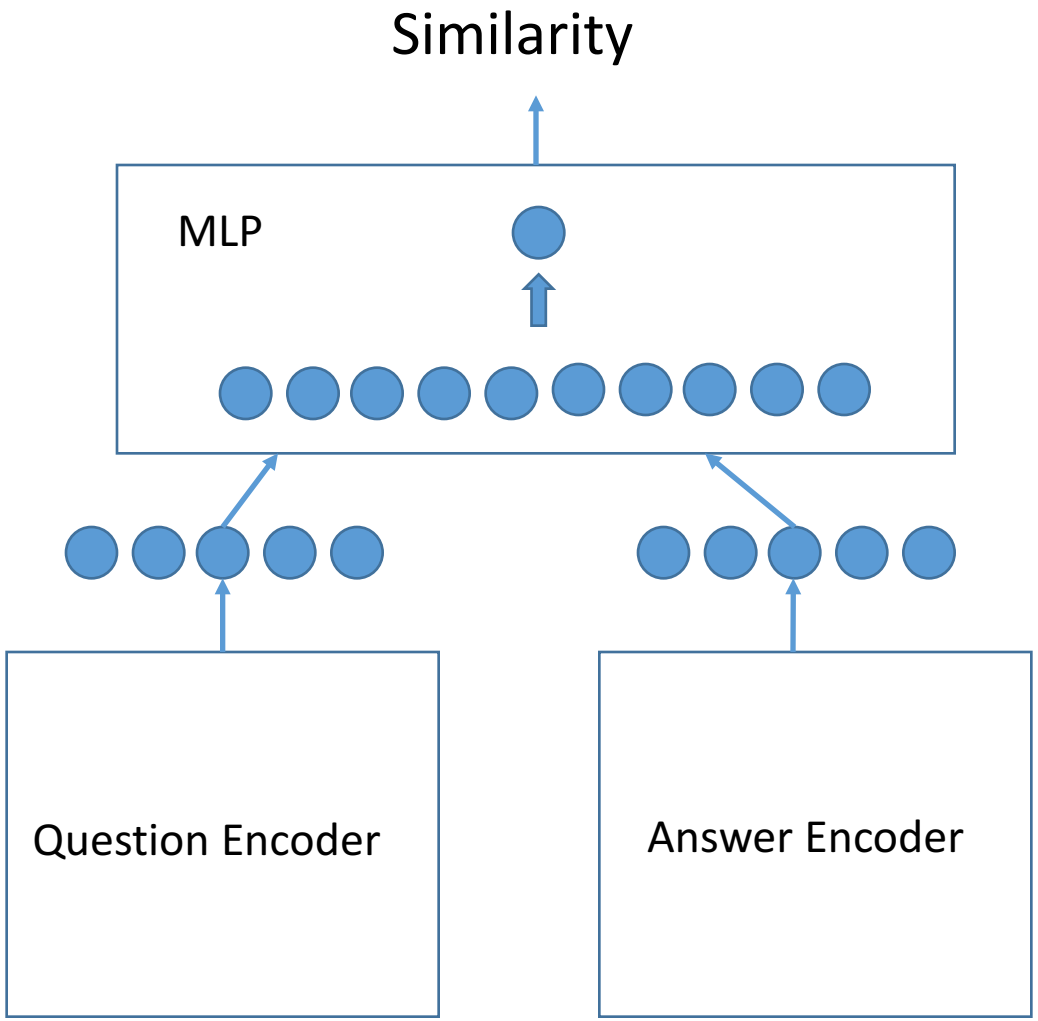
Dependency Parsing (Sentence -> Tree)

Image Captioning (Image -> Caption)

- Basically an **Encoder-Decoder** Model allows to generate sequences given an input **encoded** into a fixed size vector

Rank Text Pairs (QA)

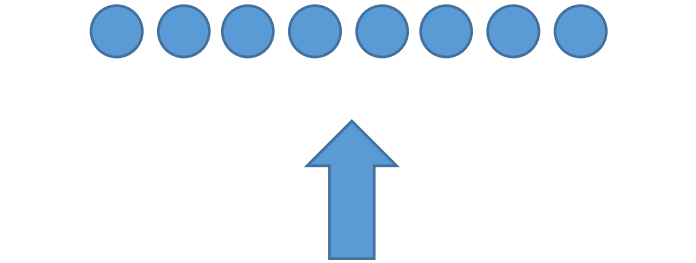
1. Encode Question
2. Encode Answer
3. Concatenate the two representations
4. Feed them to a MLP
5. Train the network to classify Good vs Bad answers to the question



Representing Documents

The idea of using neural networks for representing: words, sentences and documents into **fixed-size vectors** has led to mayor breakthroughs in NLP.

Deep Learning offers powerful tools for doing this.



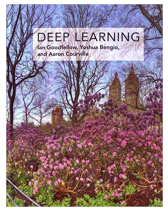
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie fermentum enim, sed ultricies urna sodales in.

Combining NNs with SVMs

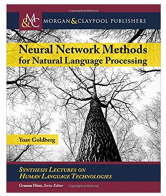
Having good representation of the input leads to considerable improvement even to other kinds of algorithms.

For instance (*Tymoshenko et al., NAACL, CIKM 2016)* obtained state-of-the-art results by combining Question and Answer Representations from a CNN with Tree Kernels for Factoid Question Answering.

RESOURCES



Deep Learning
http://www.deeplearningbook.org/



https://arxiv.org/pdf/1510.00726.pdf

Neural Network Methods in Natural Language Processing

https://explosion.ai/blog/deep-learning-formula-nlp

Embed, encode, attend, predict: The new deep learning formula for state-of-the-art NLP models

Thanks