facebook

Artificial Intelligence Research

Using Neural Networks for Modeling and Representing Natural Languages :

Introduction to Neural Networks

Tomas Mikolov, Piotr Bojanowski, Facebook AI Research

Monday, July 22nd 2019

Structure of the tutorial

- Motivation
- Basic machine learning applied to natural language
- Introduction to neural networks
- Distributed representations of words
- Efficient text classification
- Neural network based language models
- Future research
- Resources

Introduction

- Text processing is the core business of internet companies today (Facebook, Google, Twitter, Baidu, Yahoo, ...)

- Machine learning and natural language processing techniques are applied to big datasets to improve many tasks:
  - Search
  - Ranking
  - Spam detection, fake news detection, ads recommendation, email categorization, machine translation, speech recognition, …

Introduction

- This tutorial introduces artificial neural networks applied to text problems
- The focus is on understanding the core ideas: how do artificial neural networks work, what they can and cannot do, what is deep learning
- Overview of some interesting results that have been already achieved

Basic Machine Learning for NLP

Basic machine learning applied to NLP

- Before we start talking about neural networks, basic techniques will be briefly mentioned
- Neural networks are closely related to other basic machine learning techniques
- To avoid re-discovery of the wheel, it is important to know the basic concepts first
- Finally: while the basic techniques are often trivial, it is very hard to improve upon them (and many fancy techniques fail to do so!)

Basic machine learning applied to NLP

- N-grams
- Word classes
- Bag-of-words representations

- Logistic regression
- Support vector machines

N-grams

- Standard approach to language modeling

- Task: compute probability of a sentence W

$$P(W) = \prod_i P(w_i | w_1 \dots w_{i-1})$$

- Often simplified to trigrams:

$$P(W) = \prod_i P(w_i | w_{i-2}, w_{i-1})$$

N-grams: example

$$P(\text{"}this\ is\ a\ sentence\text{"}) = P(this) \times P(is|this) \times P(a|this, is) \times P(sentence|is, a)$$

- The probabilities are estimated from counts in some (large) text corpus:

$$P(a|this, is) = \frac{C(this\ is\ a)}{C(this\ is)}$$

- Smoothing is used to redistribute probability to unseen events (this avoids zero probabilities)

A Bit of Progress in Language Modeling (Goodman, 2001)

Word classes

- One of the most successful NLP concepts in practice
- Similar words should share parameters, which leads to generalization
- Example:

$$Class_1 = (yellow, green, blue, red)$$
$$Class_2 = (Italy, Germany, France, Spain)$$

- Usually, each vocabulary word is mapped to a single class (similar words share the same class)

Word classes

- There are many ways how to compute the classes – usually, it is assumed that similar words appear in similar contexts

- Instead of using just counts of words, we can use counts of classes, which leads to generalization (better performance on novel data)

Class-based n-gram models of natural language (Brown, 1992)

One-hot representations

- Simple way how to encode discrete concepts, such as words

Example:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday   = [1 0 0 0 0]
Tuesday  = [0 1 0 0 0]
is       = [0 0 1 0 0]
a        = [0 0 0 1 0]
today    = [0 0 0 0 1]
```

Also known as 1-of-N coding (N would be the size of the vocabulary)

Bag-of-words representations

- Sum of one-hot codes
- Ignores order of words

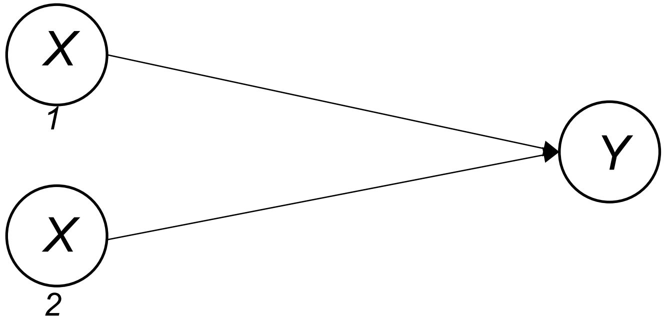
Example:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday Monday         = [2 0 0 0 0]
today is a Monday     = [1 0 1 1 1]
today is a Tuesday    = [0 1 1 1 1]
is a Monday today     = [1 0 1 1 1]
```

Can be extended to bag-of-N-grams to capture local ordering of words

Logistic regression

- Simple machine learning technique to perform classification
- Input is a vector of features, output is usually one (binary classification) or many (multinomial distribution)



- The weight matrix (or vector) directly connects inputs and output(s)

Logistic regression

- Can be trained by stochastic gradient descent, and can be seen as a neural network without any hidden layers (will be described later)

- Also called maximum entropy model in the NLP community

- Example C code for toy problems available at:
  http://ai.stanford.edu/~ajoulin/code/nn.zip
  (joint work with Armand Joulin; includes code for logistic regression, feedforward and recurrent neural networks)

Support vector machines

- Another popular way how to perform classification, very similar to logistic regression
- Tries to maximize margin between the classes:

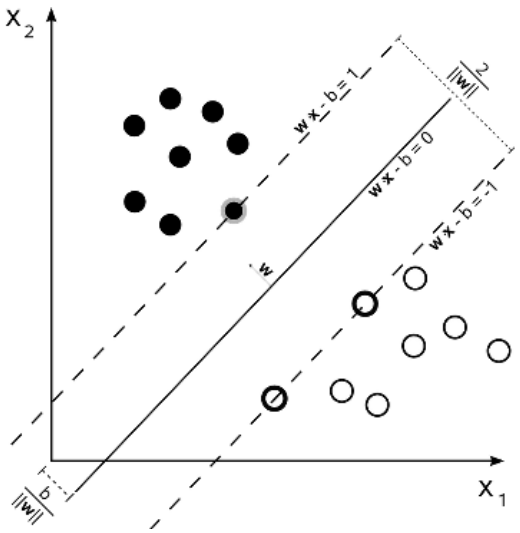


Figure from Wikipedia

- Used to be popular in part because of existence of open-source packages: *libsvm, svmtorch, svmlight*

Basic machine learning: summary

Main statistical tools for NLP:

- Count-based models: N-grams, bag-of-words
- Word classes
- Unsupervised dimensionality reduction: PCA
- Unsupervised clustering: K-means
- Supervised classification: logistic regression, SVMs

Neural Networks

Introduction to neural networks

- Motivation
- Architecture of neural networks: neurons, layers, synapses
- Activation function
- Objective function
- Training: stochastic gradient descent, backpropagation, learning rate, regularization
- Intuitive explanation of deep learning

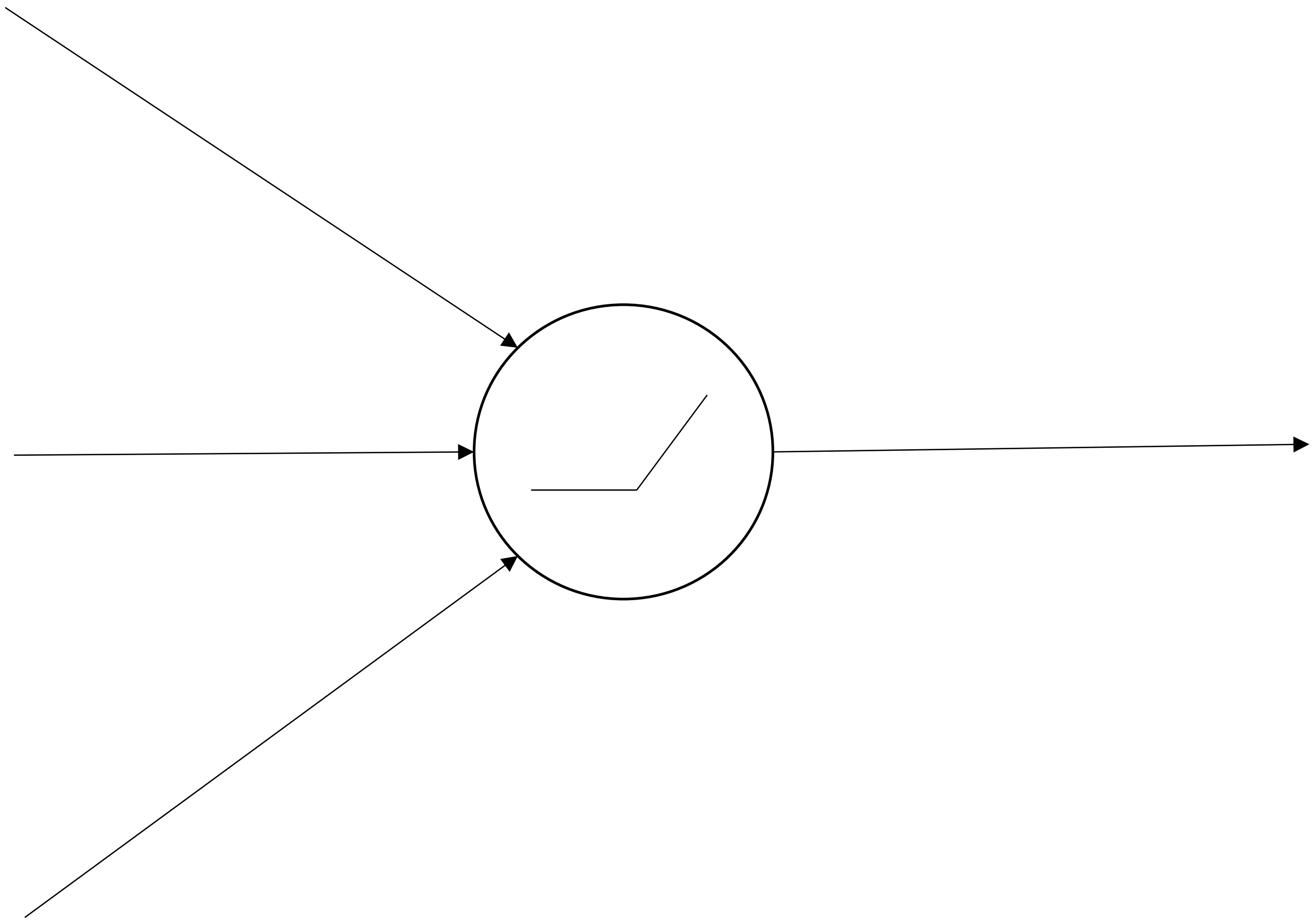
Neural networks: motivation

- The main motivation in NLP is to come up with more precise way how to represent and model words, documents and language

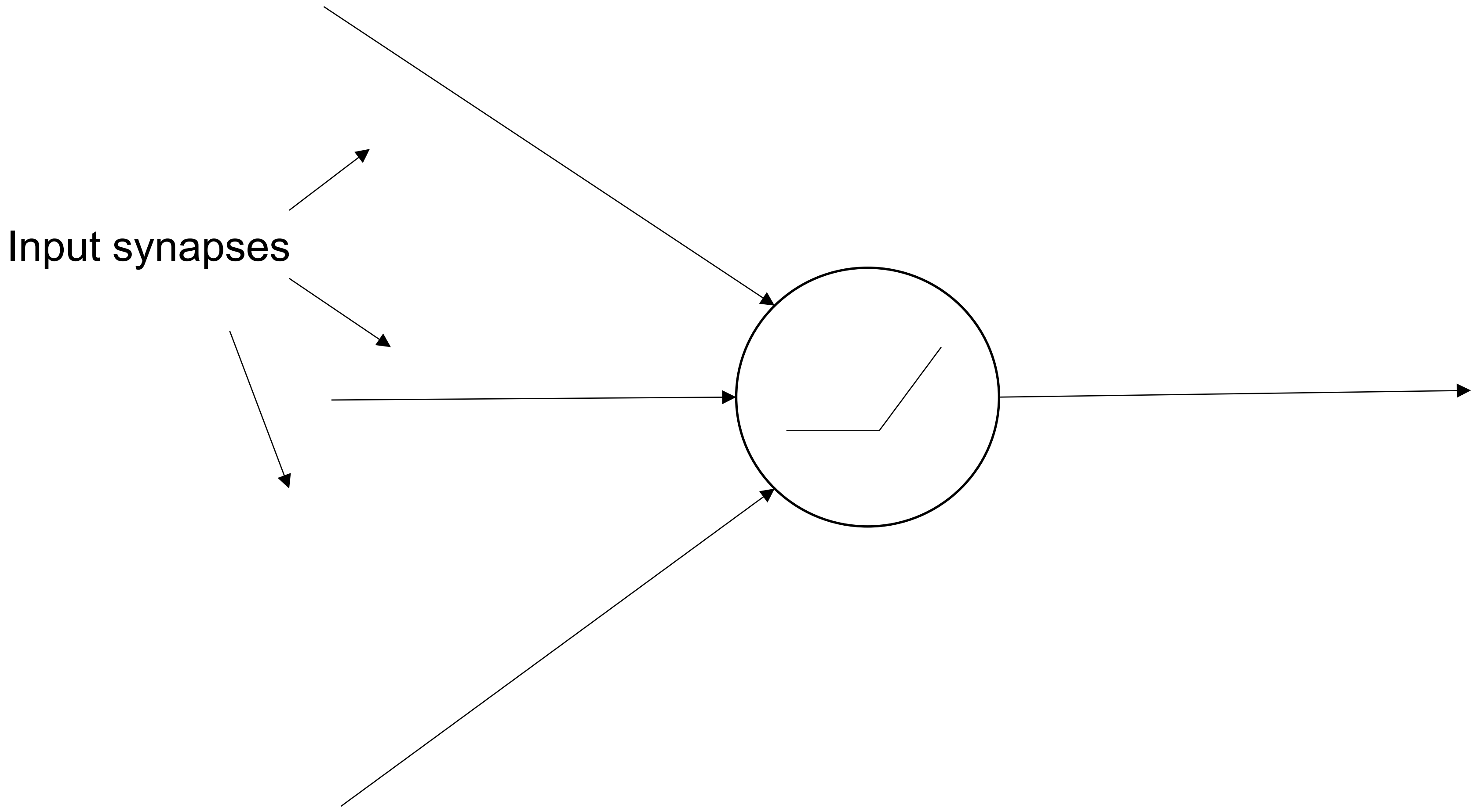
- There is nothing that neural networks can do in NLP that the basic techniques completely fail at

- But: the victory in competitions goes to the best, thus few percent gain in accuracy is important!

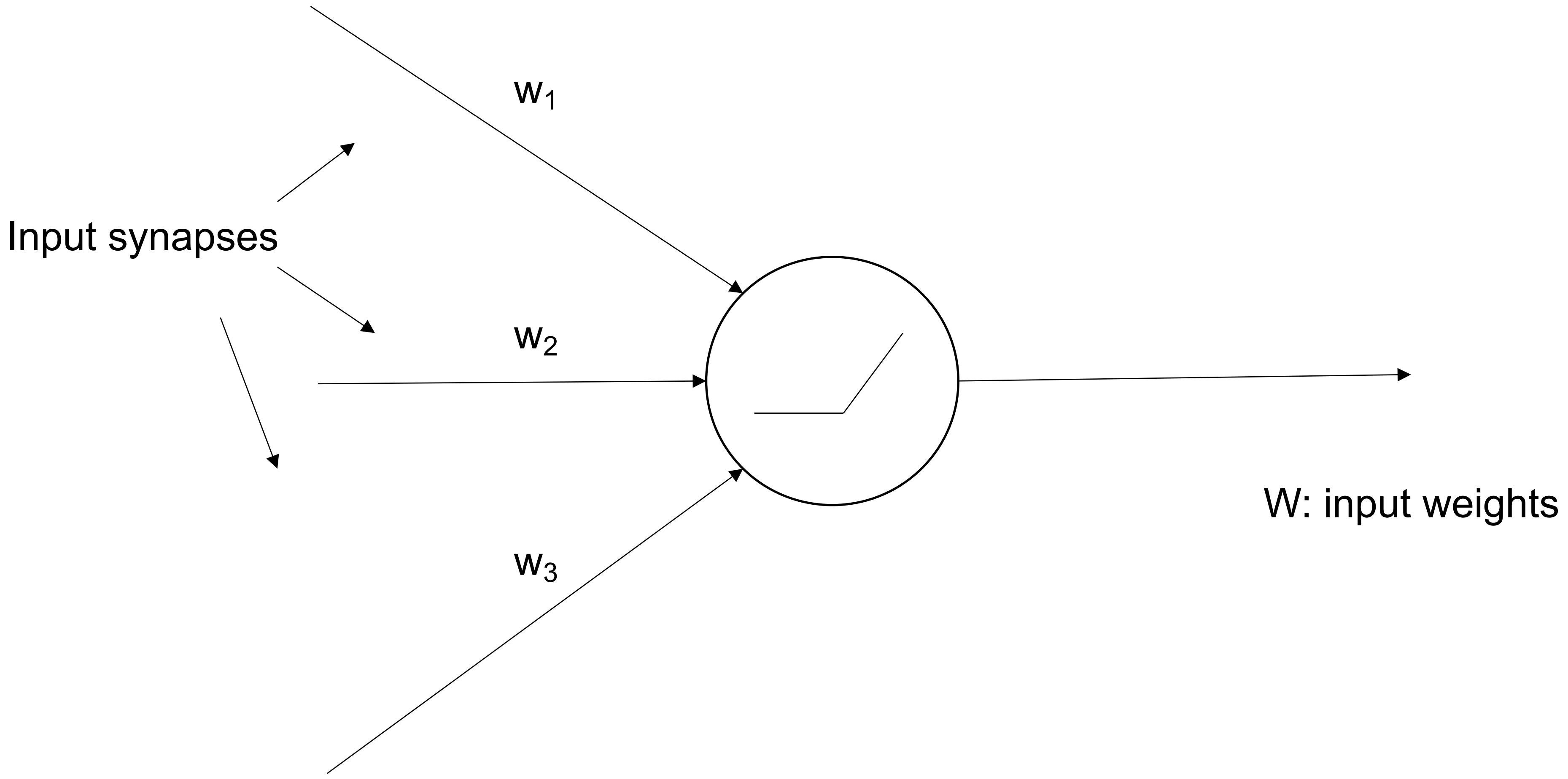
Neuron (perceptron)



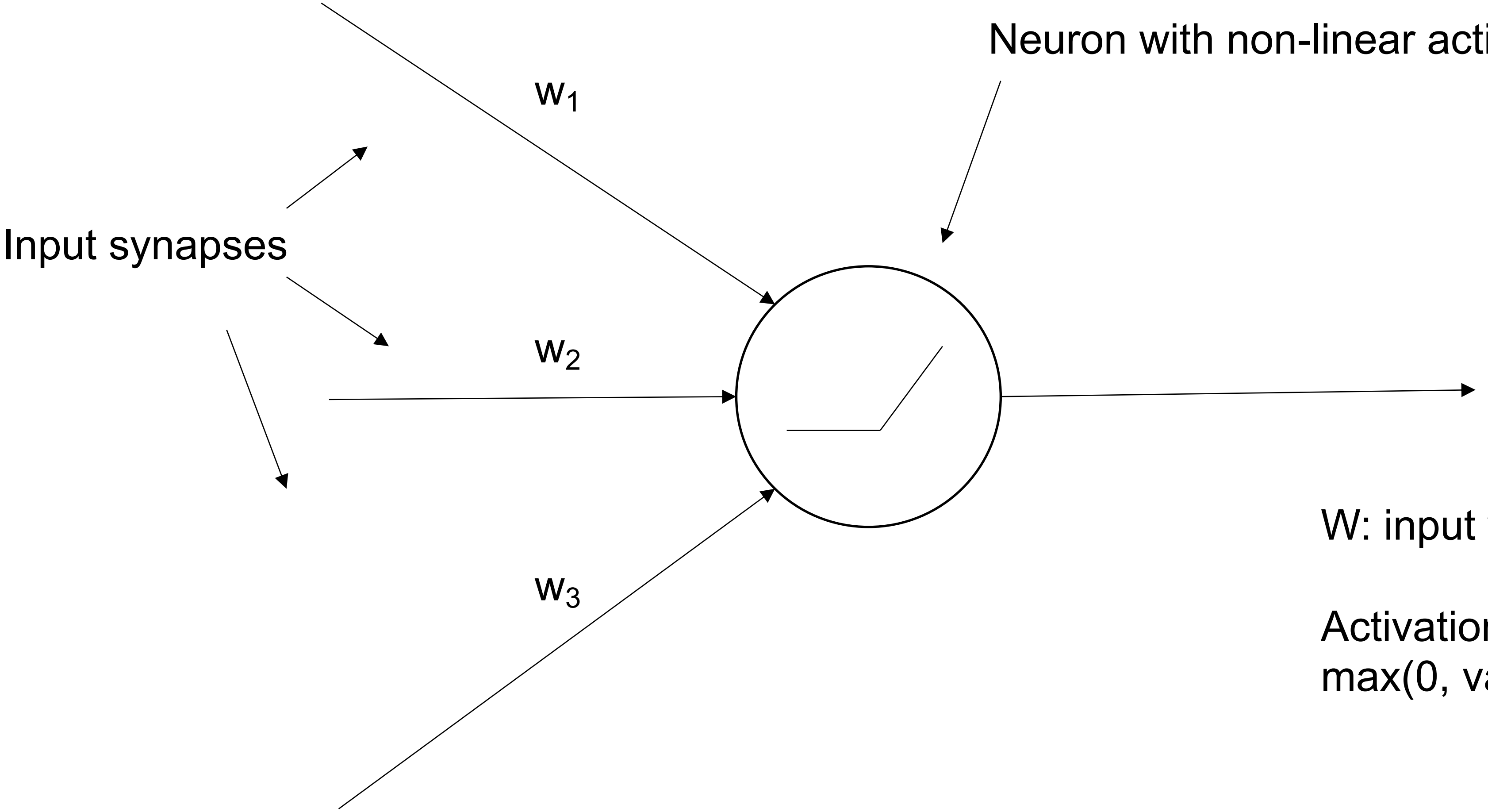
Neuron (perceptron)



Neuron (perceptron)



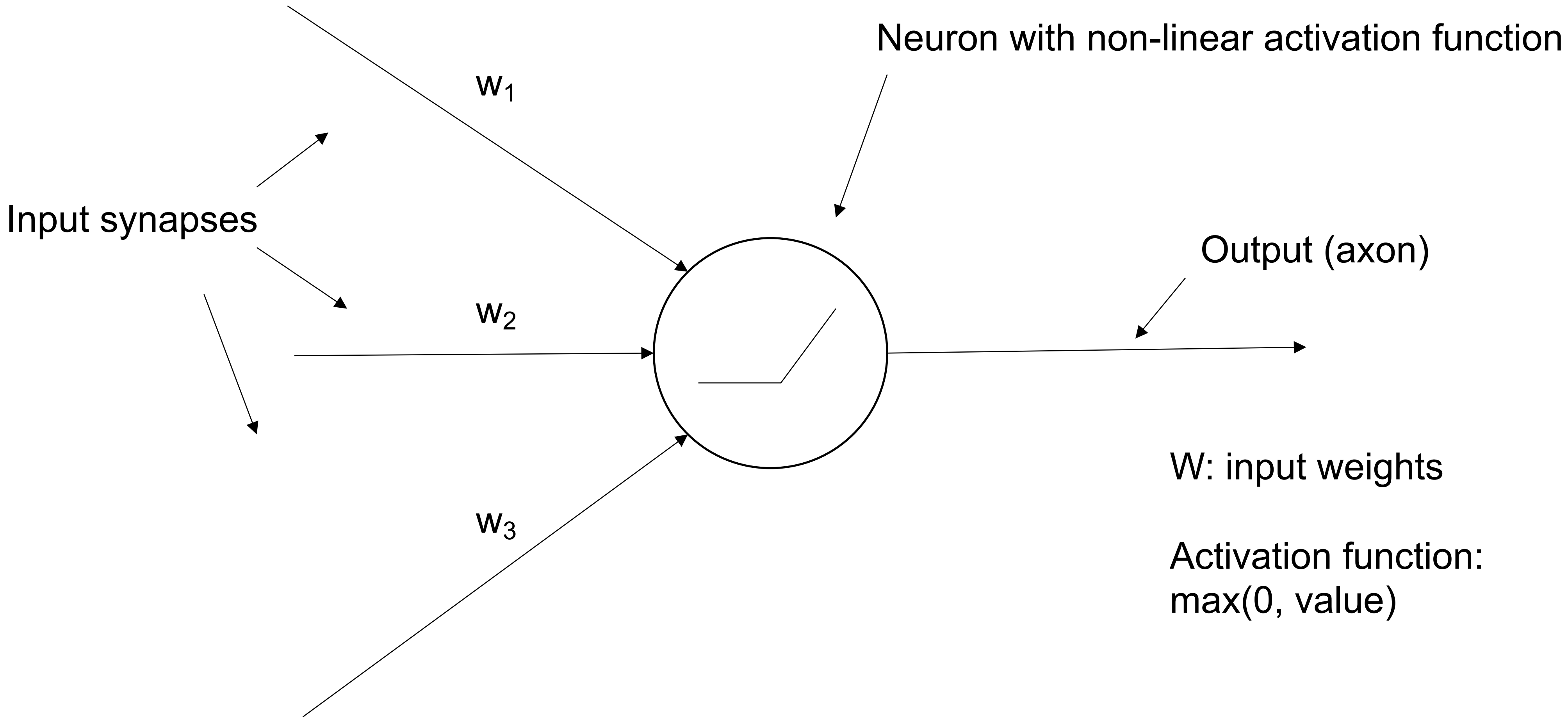
Neuron (perceptron)



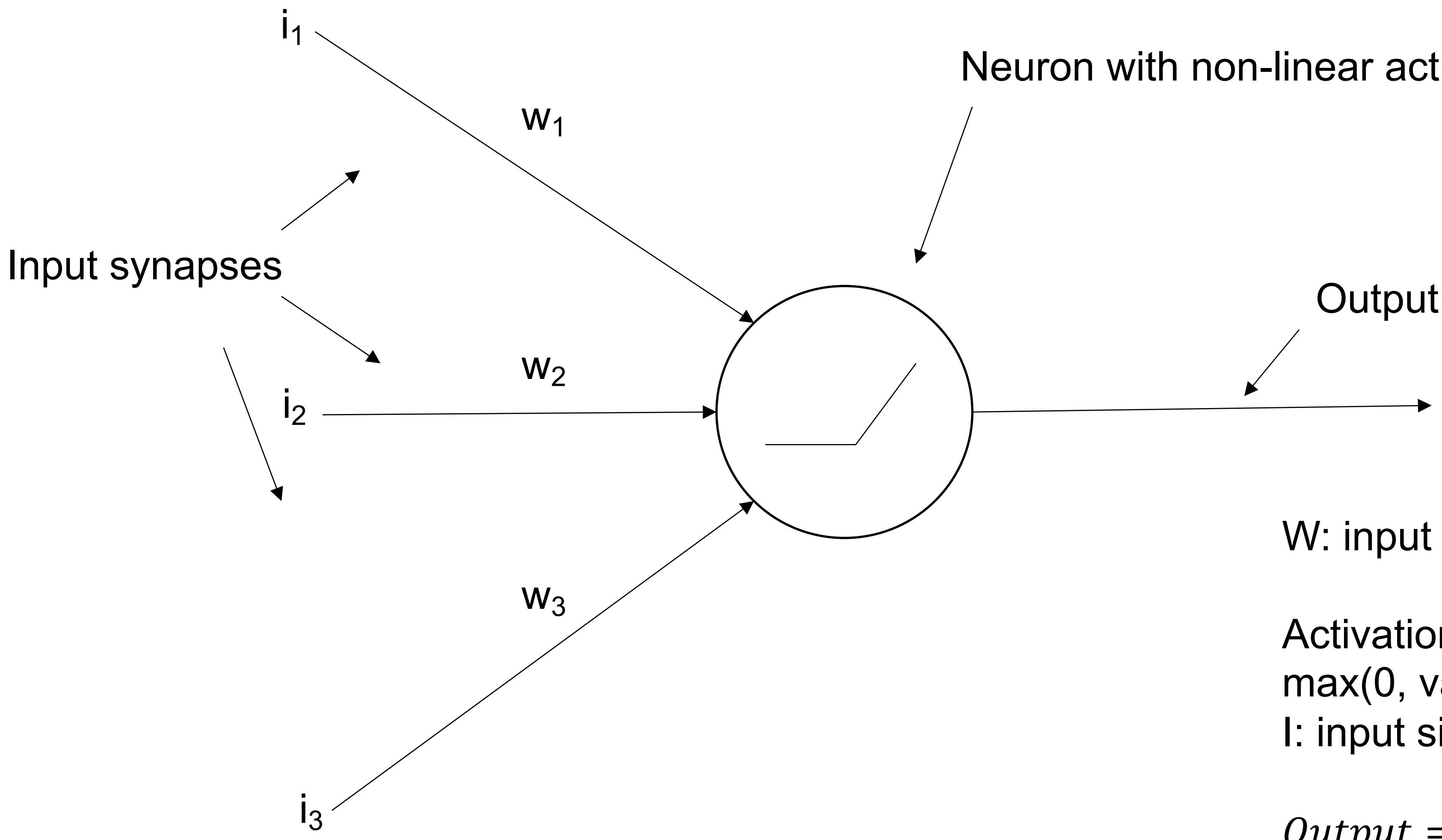
W: input weights

Activation function:
max(0, value)

Neuron (perceptron)



Neuron (perceptron)



W: input weights

Activation function:
max(0, value)
I: input signal

$$Output = \max(0, I \cdot W)$$

Neuron (perceptron)

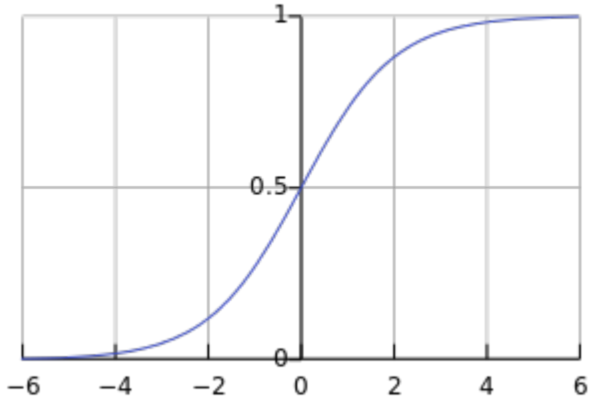
- It should be noted that the perceptron model is quite different from the biological neurons (those communicate by sending spike signals at various frequencies)

- The learning in biological neurons seems to be also quite different

- It would be better to think of artificial neural networks as non-linear projections of data

Activation function

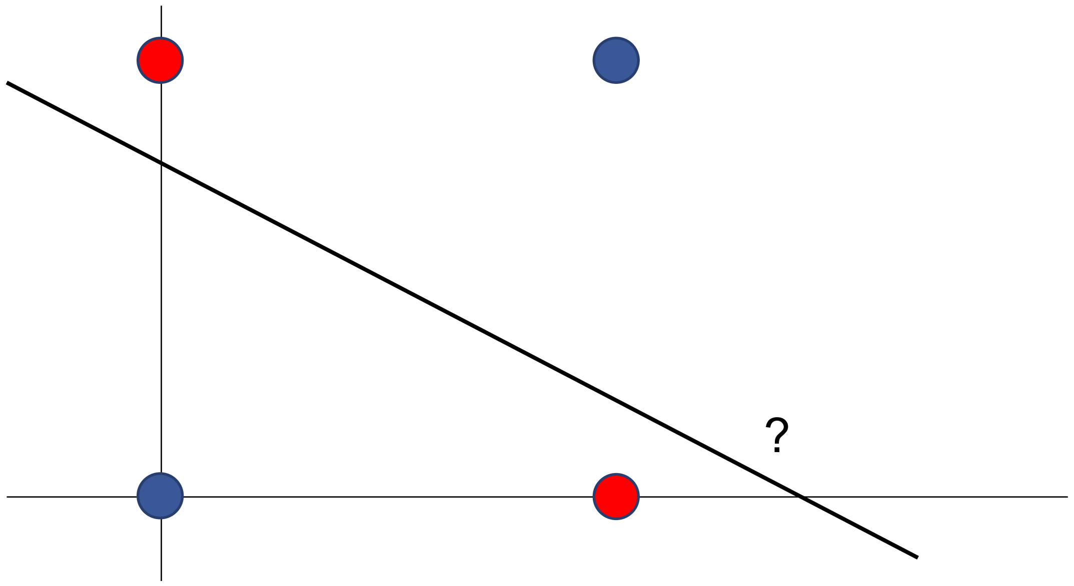
- In the previous example, we used `max(0, value)`: this is nowadays referred to as "rectified linear unit" (ReLU)
- Many other non-linear functions can be used
- Other common ones: sigmoid, tanh



Sigmoid function,
Figure from Wikipedia

Activation function

- The most important property is the non-linearity
- Example: XOR problem
  - There is no linear classifier that can solve this problem:



Non-linearity: example

Intuitive NLP example:

- Input: bag-of-words
- Output: binary classification (for example, positive / negative sentiment)

- Input: *"the idea was not bad"*
- Non-linear classifier can learn that *"not"* and *"bad"* next to each other mean something else than *"not"* or *"bad"* itself
- *"not bad"* !± *"not"* + *"bad"*

Activation function

- The non-linearity is a crucial concept that gives neural networks more representational power compared to some other techniques (linear SVMs, logistic regression)

- Without the non-linearity, it is not possible to model certain combinations of features (like Boolean XOR function), unless we do manual feature engineering

Hidden layer

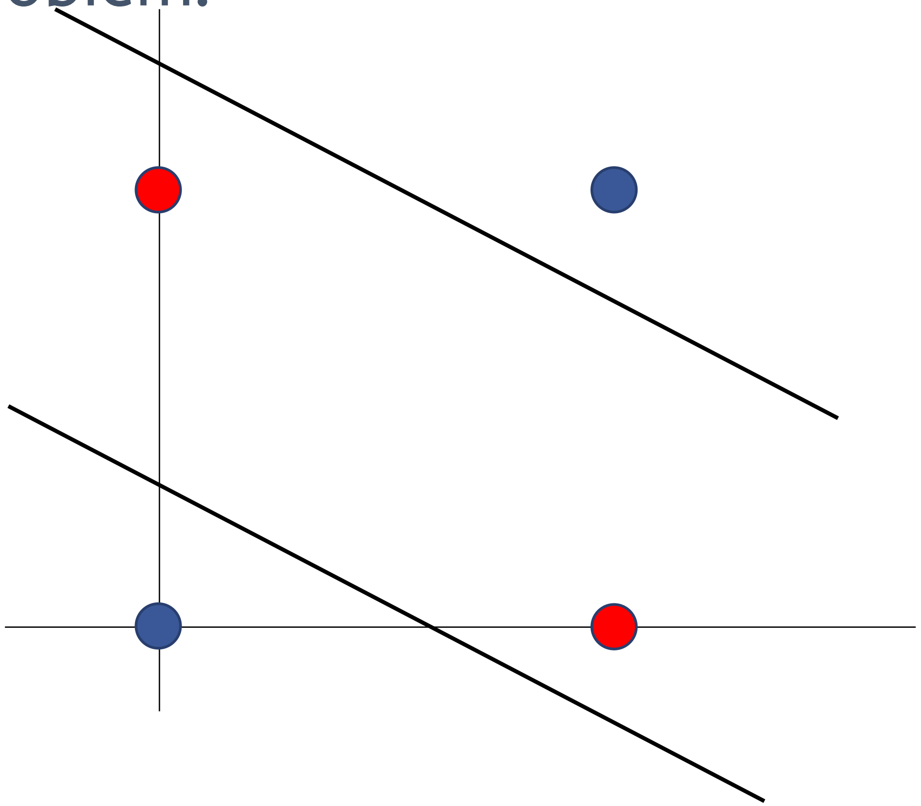
- Hidden layer represents learned non-linear combination of input features (this is different than SVMs with non-linear kernels that are not learned)

- Inputs to a hidden layer can be outputs of a previous hidden layer

Hidden layer

- With a hidden layer, we can solve the XOR problem:

1. some neurons in the hidden layer will activate only for some combination of the input features
2. the output layer can represent a combination of the activations of the hidden neurons

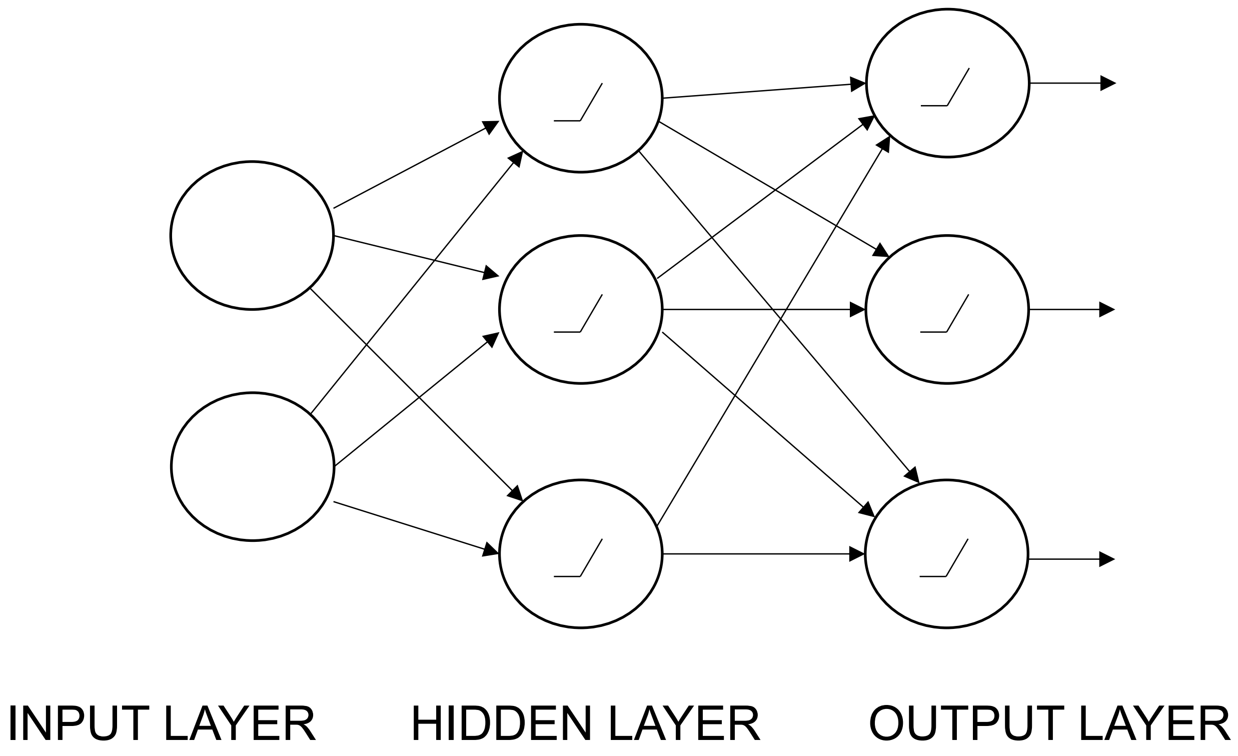


Hidden layer

- Neural net with one hidden layer is a universal approximator: it can represent any function up to arbitrary precision

- However, not all functions can be represented *efficiently* with a single hidden layer – we shall see that in the deep learning section

Neural network layers



Objective function

- Objective function defines how well does the neural network perform some task
- The goal of training is to adapt the weights so that the objective function is optimized (maximized / minimized)
- Example: classification accuracy, reconstruction error

Unsupersvied / supervised training

- When the objective is to model the input data, the training is called unsupervised

- An example is auto-encoder: the objective function is to reconstruct the input features at the output layer (by performing some kind of compression when going through the hidden layers)

- Supervised training means that we have additional labels for the input vectors, and the objective is usually to perform classification

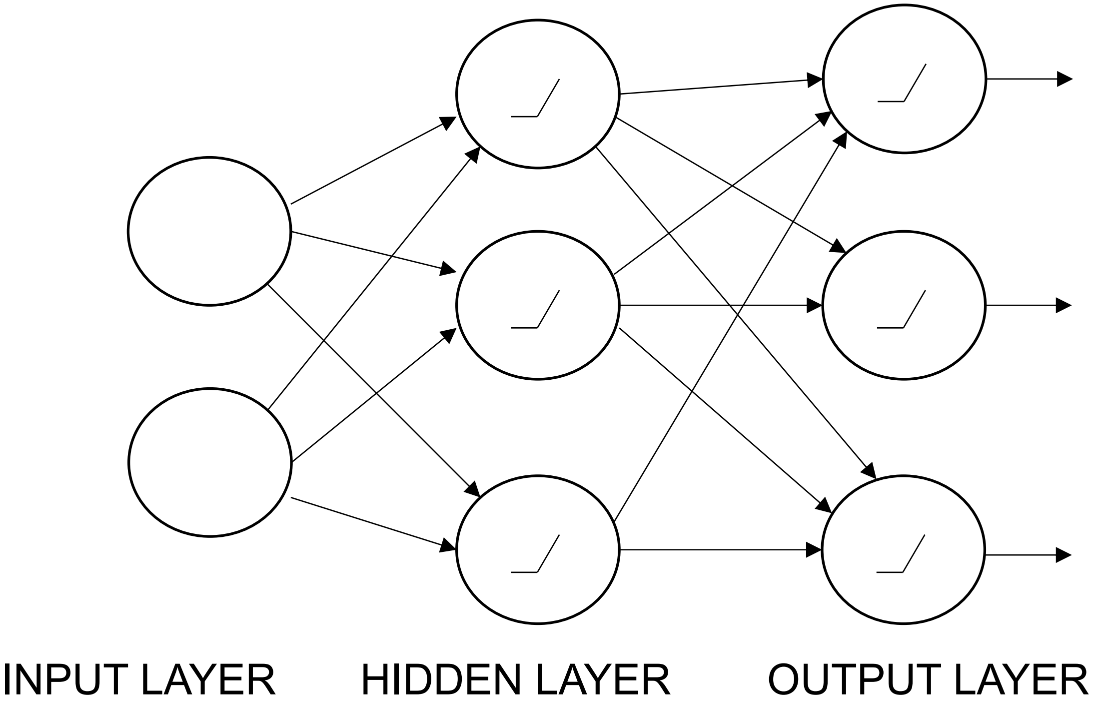
Training of neural networks

- There are many ways how to train neural networks
- The most widely used and successful in practice is stochastic gradient descent (SGD)
- Many algorithms are presented as superior to SGD, but when properly compared, the improvements are not easy to achieve

Training of neural networks

Forward pass:

- Input signal is presented first
- Hidden layer state is computed (vector times matrix operation and non-linear activation)
- Outputs are computed (vectors times matrix operation and usually non-linear activation)



W: input weights
Activation function:
max(0, value)
I: input signal
$Output = \max(0, I \cdot W)$

Training of neural networks - SGD

Intuitive explanation of stochastic gradient descent:

- The input feature vector is used to compute the output vector during the forward pass
- The target vector represents the desired output vector (in case of classification it uses one-hot coding)
- We change the weights a little bit so that next time the same input vector is presented, the output vector will be closer to the target vector

Training of neural networks - SGD

Intuitive explanation of stochastic gradient descent:

- The input feature vector is used to compute the output vector during the forward pass
- The target vector represents the desired output vector (in case of classification it uses one-hot coding)
- We change the weights a little bit so that next time the same input vector is presented, the output vector will be closer to the target vector

Training of neural networks - SGD

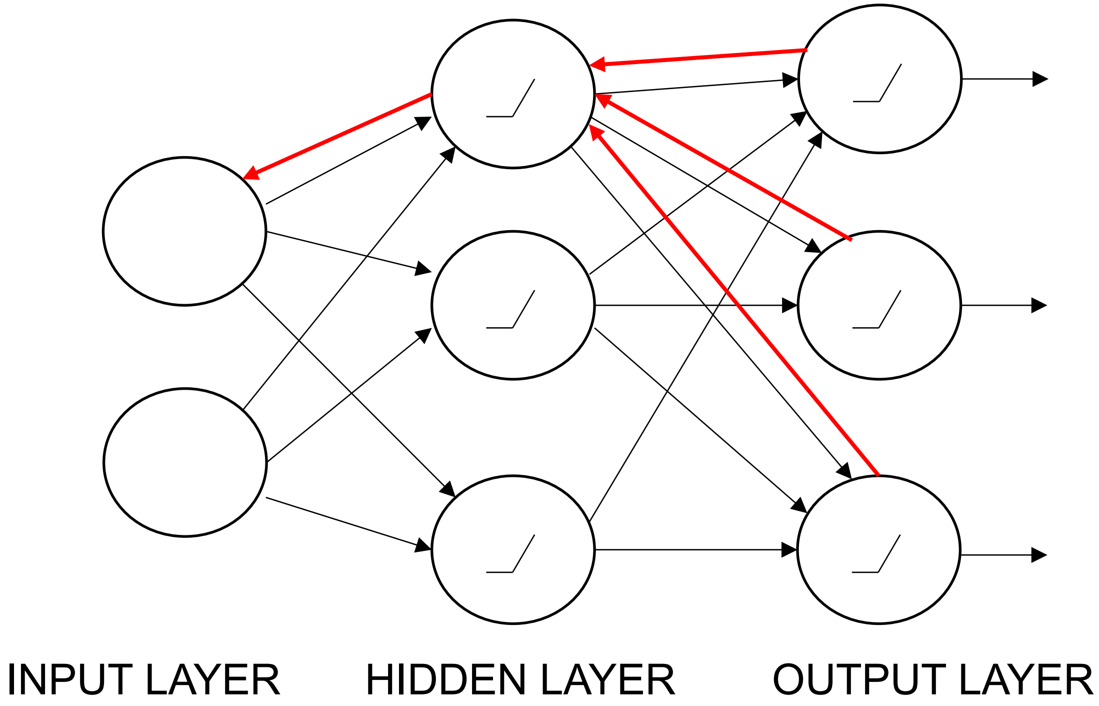
Intuitive explanation of stochastic gradient descent:

- The input feature vector is used to compute the output vector during the forward pass
- The target vector represents the desired output vector (in case of classification it uses one-hot coding)
- We change the weights a little bit so that next time the same input vector is presented, the output vector will be closer to the target vector

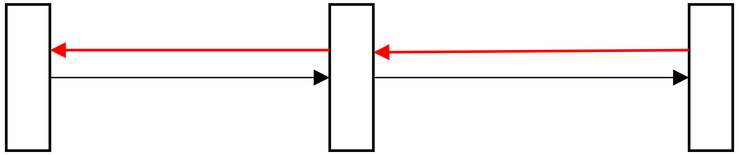
Backpropagation

- To train the network, we need to compute gradient of the error

- The gradients are sent back using the same weights that were used in the forward pass



-
- Simplified graphical representation:



Training of neural networks – learning rate

- Learning rate controls how quickly we change the weights: too little value will result in long training time, too high value will erase what was learned previously

- In practice, we start with a high learning rate and reduce it during the training

- The starting learning rate and how quickly it gets reduced can affect the resulting performance in a great way: you have to tune this!

Training of neural networks – training epochs

- Several training epochs over the training data are often performed
- Usually, the training is finished when performance on held-out (validation) data does not improve
- Held-out data is used only for verification of the performance, the network is not trained on these examples

Regularization

- As the network is trained, it can overfit the training data
  - Overfitting: very good performance on training and bad performance on test data
- The network can "memorize" the training data: often, it will contain high weights that are used to model only some small subset of the data
- We can force the weights to stay small (close to zero) during training to reduce this problem (L1 & L2 regularization)

Training of neural networks: summary

- Stochastic gradient descent and backpropagation are usually good choices for training

- The representational power of neural networks comes from non-linear hidden layer(s)

What training typically does not do

- Choice of the hyper-parameters has to be done manually:
- Type of activation function
- Choice of architecture (how many hidden layers, their sizes)
- Learning rate, number of training epochs
- What features are presented at the input layer
- How to regularize

- It may seem complicated at first - the best way to start is to re-use some existing setup and try your own modifications.

Neural networks and logistic regression

- Neural networks can do everything logistic regression can do (proof: the hidden layer can simply copy inputs)
- Logistic regression is in many cases computationally much more efficient
- Thus, we could use both jointly to be efficient: will be shown later

Deep learning

- Deep model architecture is about having more than one computational step (hidden layer) in the model
- Deep learning aims to learn patterns that cannot be learned efficiently with shallow models (one or zero hidden layers)
  - Should result in better generalization (performance on the test set)

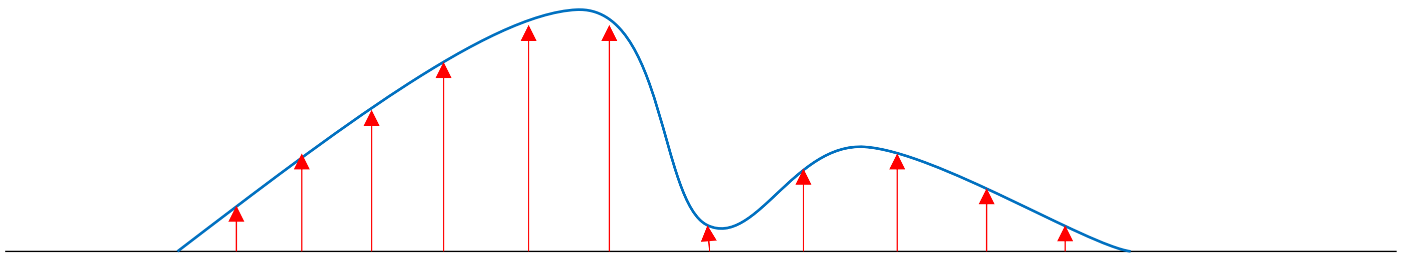
Deep learning

- But: it was previously stated that one hidden layer is enough to represent any function

- Why would we need more hidden layers then?

Deep learning

- The crucial part to understand deep learning is the *efficiency*
- The *"universal approximator"* argument says nothing else than that a neural net with non-linearities can work as a look-up table to represent any function: some neurons can activate only for some specific range of input values



Deep learning

- Look-up table is not efficient: for certain functions, we would need exponentially many hidden units with increasing size of the input layer

- Example of function that is difficult to represent: parity function (N bits at input, output is 1 if the number of active input bits is odd)

Perceptrons, Minsky & Papert 1969

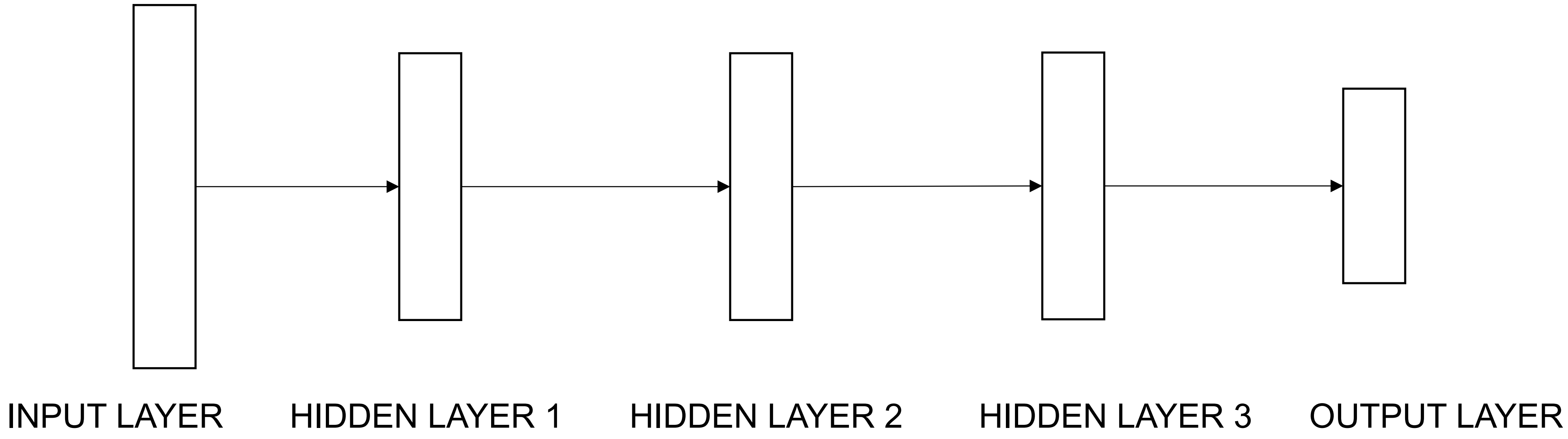
Deep learning

- Having hidden layers exponentially larger than is necessary is bad: too many parameters to learn

- If we cannot compactly represent patterns, we have to memorize them: we need (possibly exponentially) more training examples than is necessary

Deep learning

- Whenever we try to learn a complex function that is a composition of simpler functions, it may be beneficial to use a deep architecture



Deep learning

- Historically, deep learning was assumed to be impossible to achieve by using SGD + backpropagation

- Since ~2010 there was a lot of progress in tasks that contain signals that are very compositional (speech, vision) and where large amount of training data is available

Deep learning

- Deep learning is still an open research problem
- Many deep models have been proposed that do not learn anything else than a shallow (one hidden layer) model can learn: beware the hype!
- Not everything labeled “deep” is a successful example of deep learning
  - Having a deep architecture is not sufficient to learn complex patterns

Neural networks and deep learning: summary

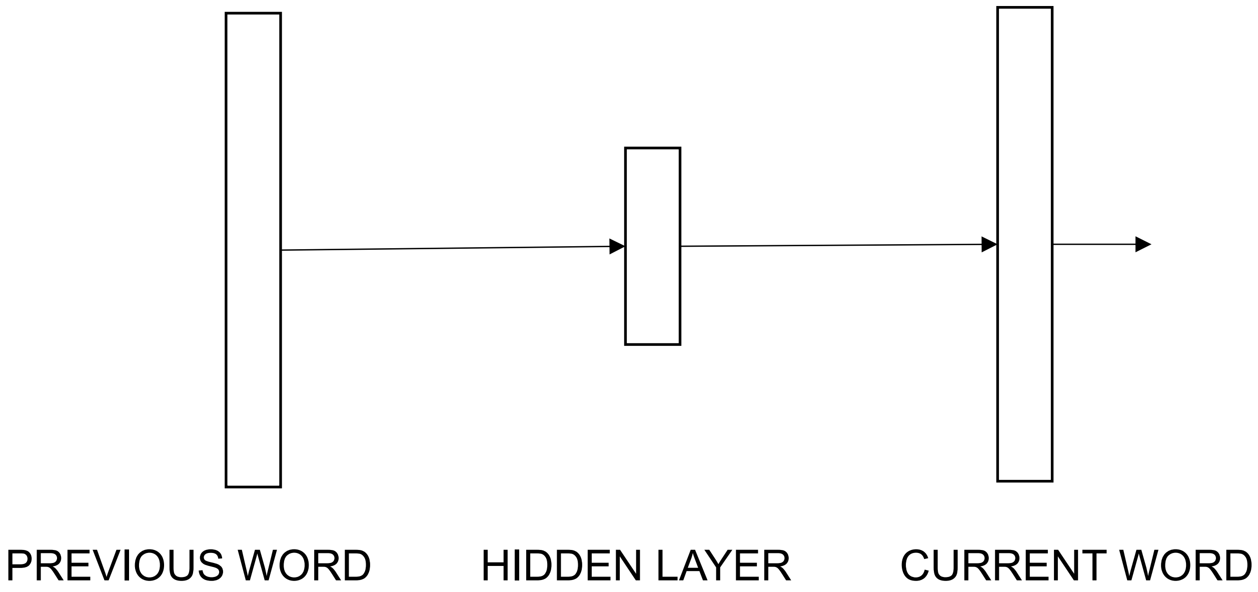
- Neural networks are a basic machine learning technique, can be seen as non-linear projections of the input feature vectors
- Start with SGD and backpropagation for training
- Deep learning can be useful for learning complex patterns in data, especially in vision & speech

Distributed Word Representations

Distributed representations of words

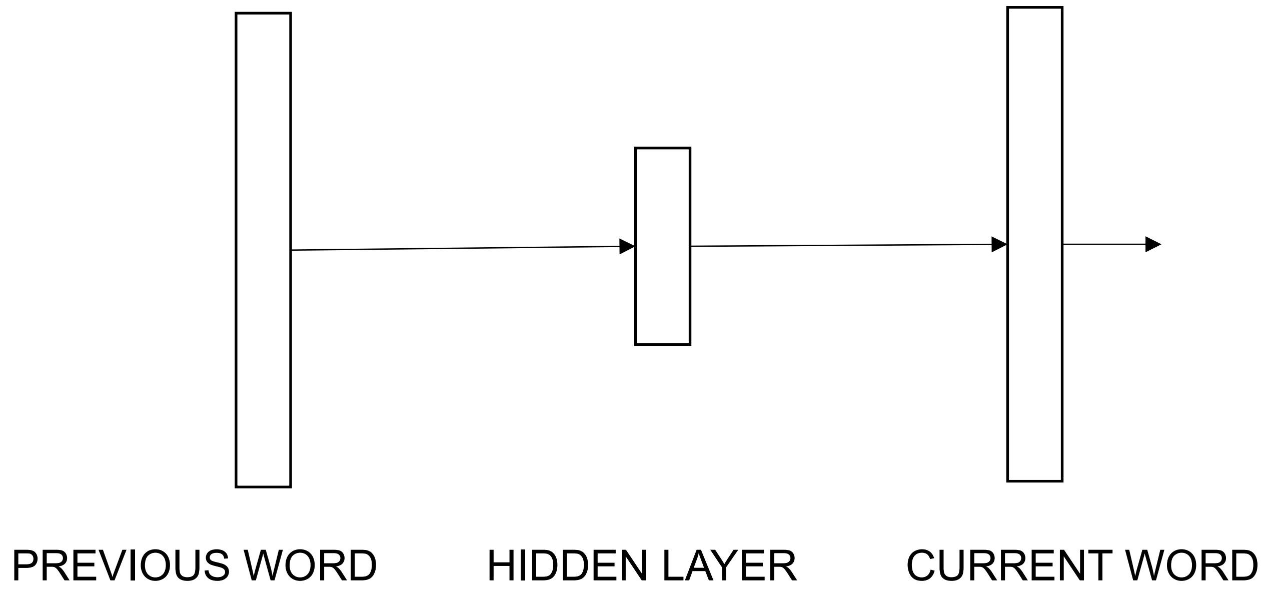
- Vector representation of words computed using neural networks
- Linguistic regularities in the word vector space
- Evaluation of performance
- Application to machine translation
- Recent progress

A very basic neural network applied to NLP



- Bigram neural language model
- Previous word is used to predict the current word by going through hidden layer (classifier with as many outputs as there are words in the vocabulary)

A very basic neural network applied to NLP



- The input is encoded as one-hot
- The model will learn compressed, continuous representation of words (usually the matrix of weights between the input and hidden layer)

Word vectors

- We call the vectors in the matrix between the input and hidden layer *word vectors* (also known as *word embeddings*)

- Each word is represented by a real valued vector in N-dimensional space (usually N = 50 – 1000)

- The word vectors have some similar properties to word classes; however, many degrees of similarity are captured
  - *Paris* is similar to *Berlin*, but also to *France*

Word vectors

- Word vectors can be used as features in many NLP tasks
  - *Natural Language Processing (Almost) from Scratch*, Collobert et al 2011

- Pre-trained word vectors provide generalization for systems trained with limited amount of supervised data

- Complex model architectures can be used to learn the word vectors
  - Neural language model with multi-task learning as in (Collobert & Weston 2008)

Word vectors

- Many architectures were proposed for training the word vectors, some labeled “deep”
- Do we really need deep learning here?
- Do we know how to apply deep learning to this task?

Word vectors

- We need some measurable way how to compare word vectors trained using different architectures

- The comparison is tricky: people mostly publish just their pre-trained word vectors, use different datasets both for training and evaluation…

- Conclusions based on experiments with different datasets are difficult to make

Word vectors - evaluation

Popular datasets like WS353 (word similarity, 353 word pairs with human judgements of similarity) have several drawbacks:

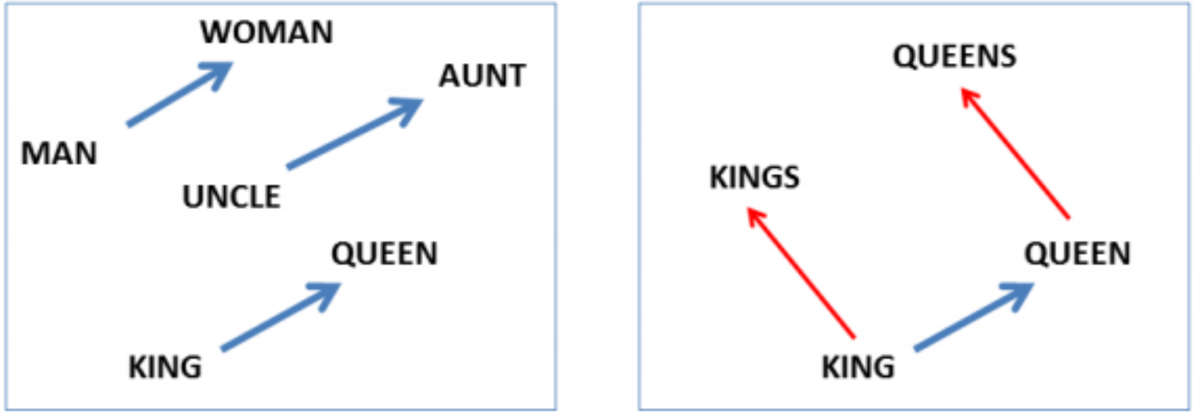
- Tiny size, no heldout / test data split

- Performance is heavily biased by choice of the training data, less so by the architecture of the model itself

Placing Search in Context: The Concept Revisited (Finkelstein et al, 2002)

Word vectors – linguistic regularities

- It was shown that word vectors capture many linguistic properties (gender, tense, plurality, even semantic concepts like “capital city of”)
- We can do nearest neighbor search around result of vector operation “King – man + woman” and obtain “Queen”



Linguistic regula (Mikolov et al, 2013)

Word vectors – datasets for evaluation

Microsoft Research dataset with 8K “analogies” - examples:

- `good:better rough: ___`
- `good:best   rough: ___`
- `better:best rougher: ___`
- `year:years  law: ___`
- `see:saw     return: ___`

Linguistic regularities in continuous space word representations (Mikolov et al, 2013)

Word vectors – datasets for evaluation

Google dataset, almost 20K questions:

- `Athens:Greece` `Oslo: ___`
- `Angola:kwanza` `Iran: ___`
- `brother:sister` `grandson: ___`
- `possibly:impossibly` `ethical: ___`
- `walking:walked` `swimming: ___`

Efficient estimation of word representations in vector space (Mikolov et al, 2013)

Word vectors – datasets for evaluation

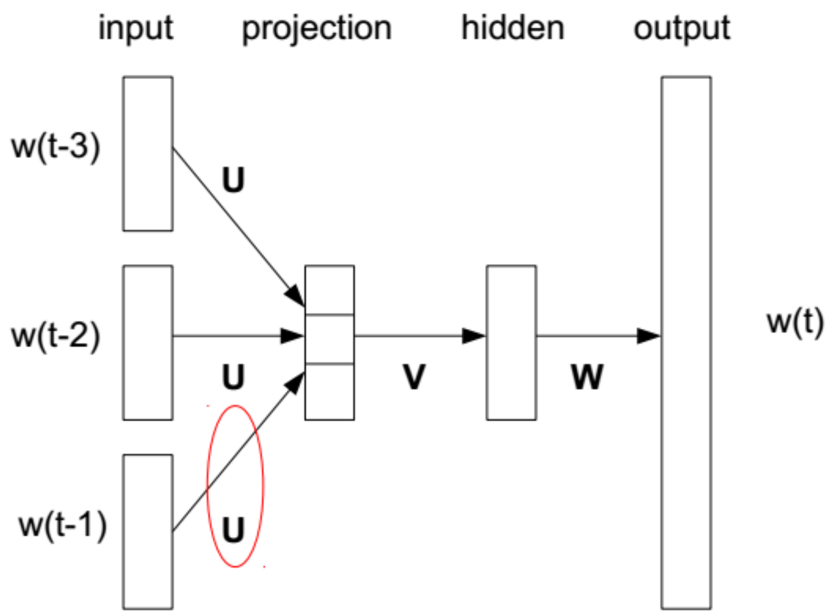
Google phrase-based dataset, focuses on semantics:

- `New York:New York Times` `Baltimore: ___`
- `Boston:Boston Bruins` `Montreal: ___`
- `Detroit:Detroit Pistons` `Toronto: ___`
- `Austria:Austrian Airlines` `Spain: ___`
- `Steve Ballmer:Microsoft` `Larry Page: ___`

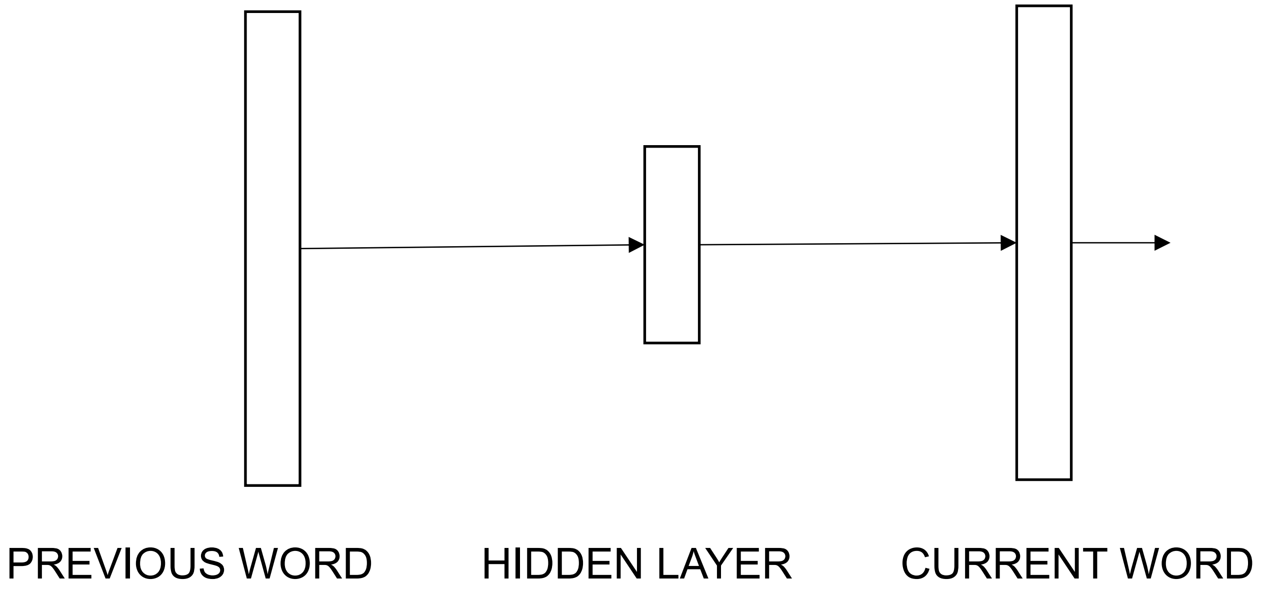
Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al, 2013)

Word vectors – various architectures

- Neural net based word vectors were traditionally trained as part of full neural network language model (Bengio et al, 2003)
- This models consists of an input layer, projection layer, hidden layer and output layer (will be discussed in detail in the next section)



Word vectors – various architectures



- We can extend the bigram NNLM by adding more context

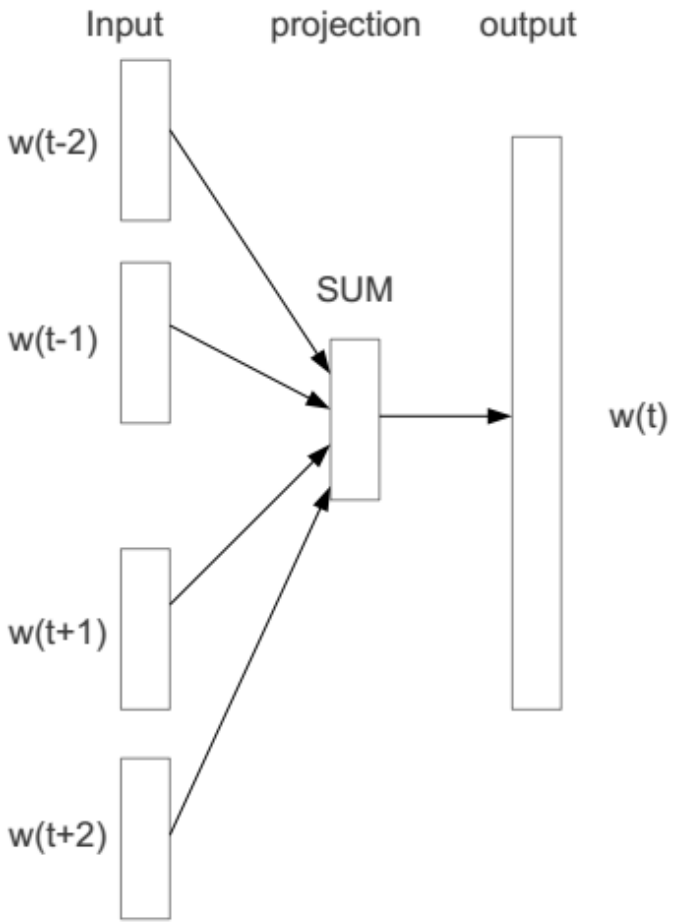
Word vectors – various architectures

- The ‘continuous bag-of-words model’ (CBOW) adds inputs from words within short window to predict the current word

- The weights for different positions are shared

- Computationally much more efficient than normal NNLM (but cannot model n-grams)

- The hidden layer is linear (no activation)

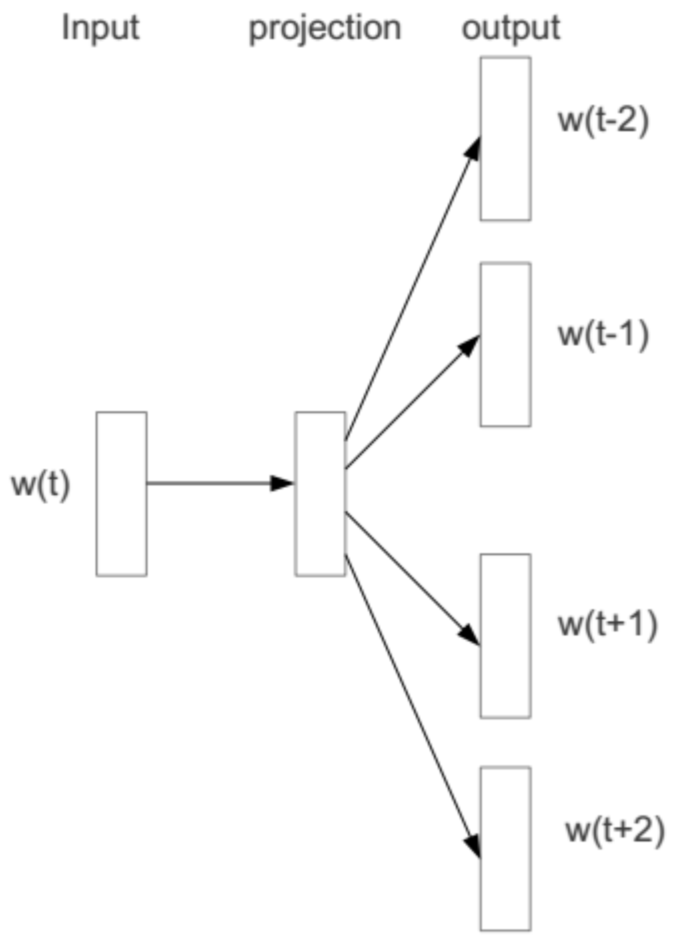


Word vectors – various architectures

- We can reformulate the CBOW model by predicting surrounding words using the current word

- This architecture is called skip-gram

- Similar performance to CBOW after convergence



Word vectors - training

- SGD + backpropagation

- Approximation of very large softmax in the output layer – can easily be in order of millions of outputs (computationally expensive):

1. Hierarchical softmax (will be described in the LM section)
2. Negative sampling

Word vectors – negative sampling

- Instead of propagating signal from the hidden layer to the whole output layer, only the output that represents the positive class + few randomly sampled outputs are evaluated

- These outputs are treated as independent logistic regression classifiers

- This makes the training speed independent on the vocabulary size

Word vectors - subsampling

- It is useful to sub-sample the frequent words (‘the’, ‘is’, ‘a’, ...) during training (discard with probability proportional to the word frequency)

- Non-linearity does not seem to improve performance of these models, thus the hidden layer does not use activation function

Word vectors – comparison of performance

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

- Google 20K questions dataset (word based, both syntax and semantics)
- Going from weeks of training to minutes while improving accuracy!

Word vectors – scaling up

- The choice of training corpus is usually more important than the choice of the technique itself
- Low computational complexity is crucial

- Optimized code published as word2vec project:
  https://code.google.com/p/word2vec/
  https://github.com/tmikolov/word2vec

Word vectors – nearest neighbors

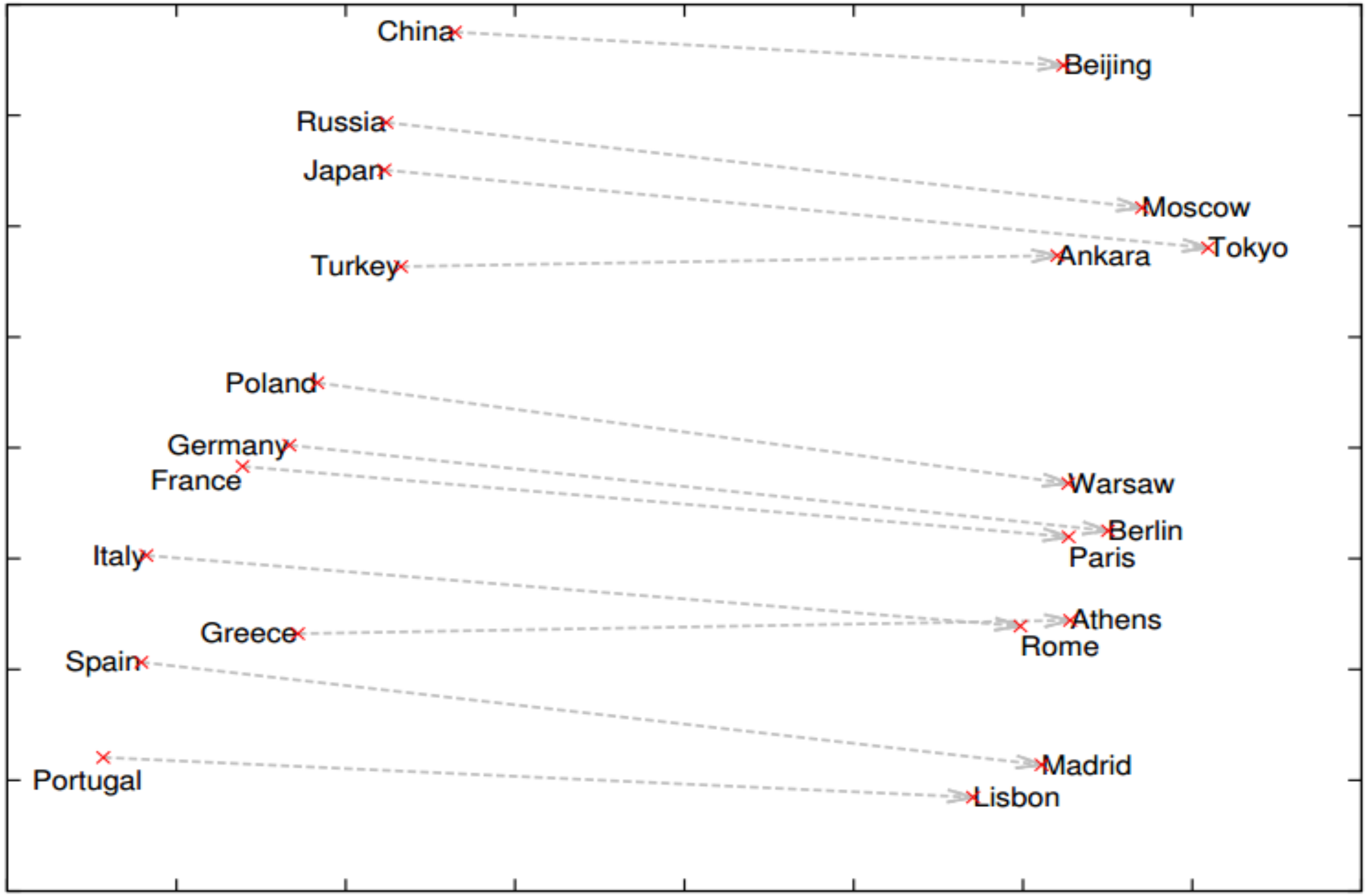
	Redmond	Havel	graffiti	capitulate
Collobert NNLM	conyers lubbock keene	plauen dzerzhinsky osterreich	cheesecake gossip dioramas	abdicate accede rearm
Turian NNLM	McCarthy Alston Cousins	Jewell Arzu Ovitz	gunfire emotion impunity	- - -
Mnih NNLM	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	anaesthetics monkeys Jews	Mavericks planning hesitated
Skip-gram (phrases)	Redmond Wash. Redmond Washington Microsoft	Vaclav Havel president Vaclav Havel Velvet Revolution	spray paint grafitti taggers	capitulation capitulated capitulating

- More training data helps the quality a lot!

Word vectors – more examples

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Word vectors – visualization using PCA



Vectors: from words to phrases

- Linguistically, *New York* or *Air Canada* should be treated as units

- Phrases can be constructed using mutual information criterion between words

- Pre-process the training data and rewrite all phrases as single tokens, such as *New_York* and *Air_Canada*

Sentence-level representations

- To obtain sentence level representations, we can add unique tokens to the data, one for each sentence (or short document)

- These tokens are trained in a similar way like other words in the skip-gram or CBOW models, just using unlimited context window (within the sentence boundaries)

Example:

```
SID__1 We think this was not the best way …
SID__2 Another reason was to …
```

Sentence-level representations

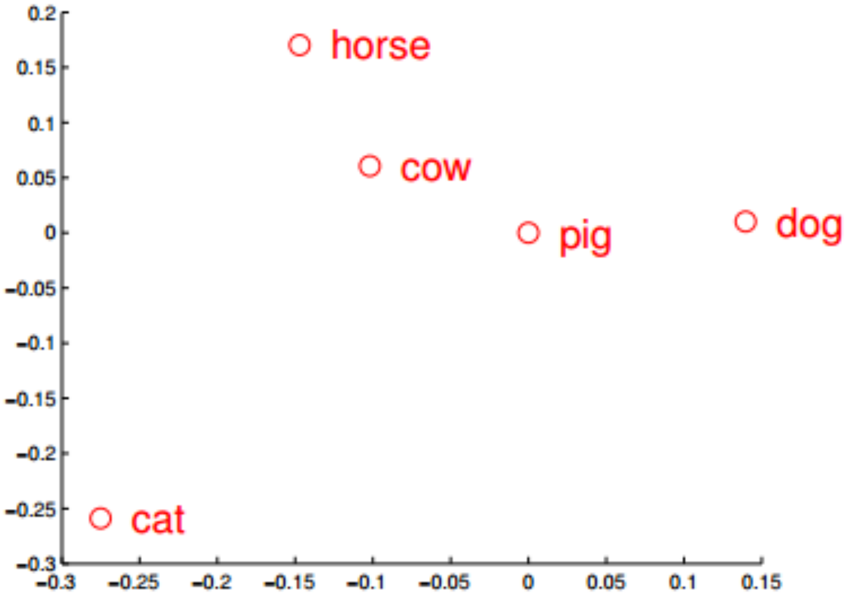
- The sentence representations can be further used in classifiers (logistic regression, SVM, or neural network)

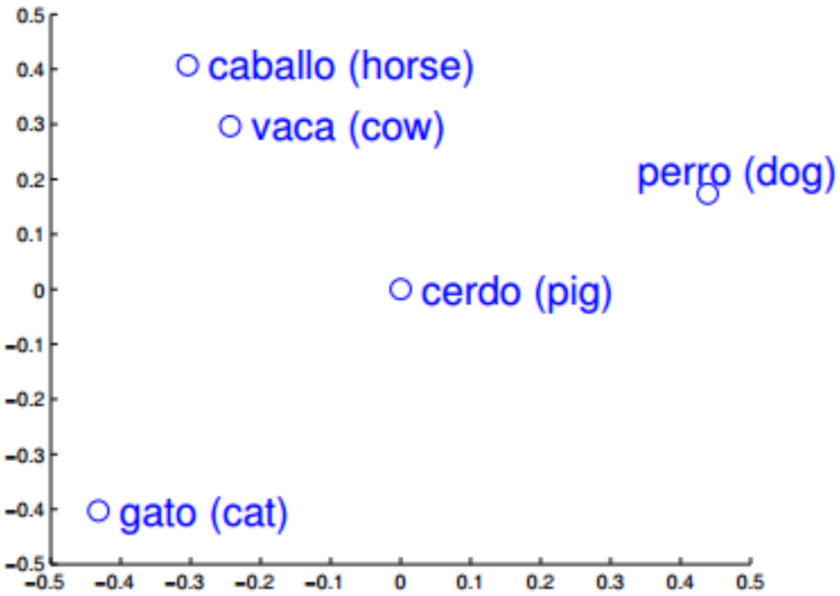
- Needs to be trained for many epochs; good results on sentiment analysis tasks

Distributed Representations of Sentences and Documents (Le et al, 2014)

Translation of words using vector spaces

- Natural languages describe the same concepts – dogs have four legs, the sky is blue everywhere in the world



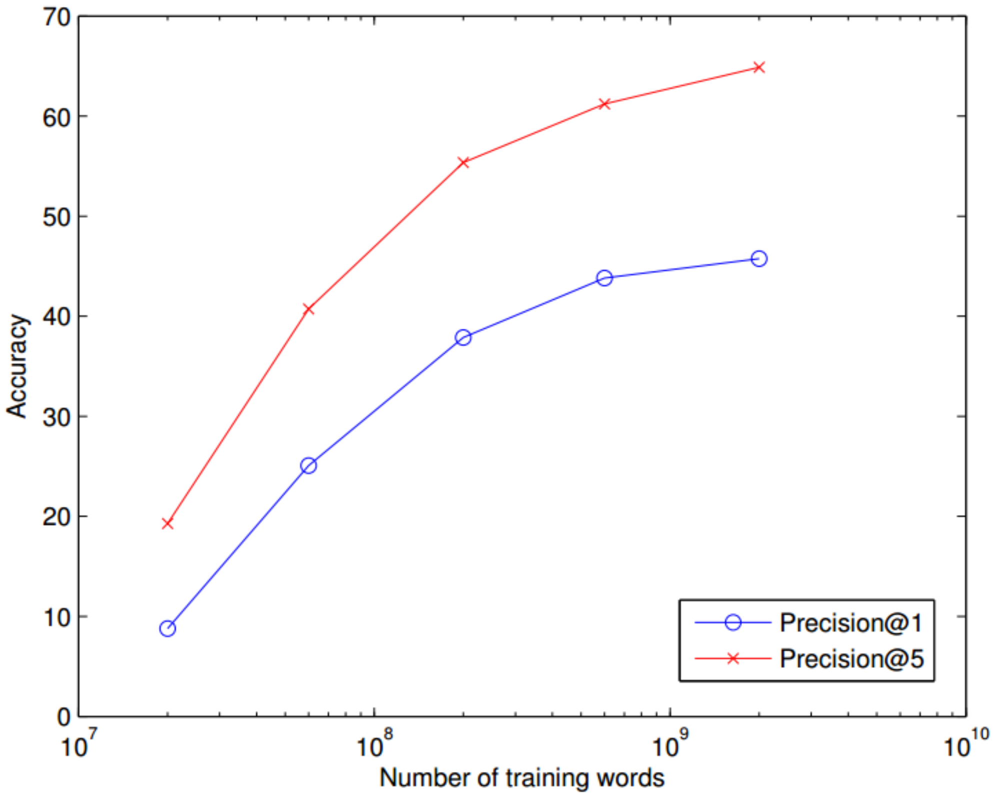


Translation of words using vector spaces

- It could be enough to learn a mapping between the vector spaces to perform basic translation

- We tried to start with small existing dictionary (5K most frequent words), and tried to translate the remaining words

English to Spanish: translation of words



Spanish word	Computed English Translations	Dictionary Entry
emociones	emotions emotion feelings	emotions
protegida	wetland undevelopable protected	protected
imperio	dictatorship imperialism tyranny	empire
destacaron	highlighted emphasized emphasised	highlighted

- The results are surprisingly accurate, especially with models trained on a lot of data

Translation of words and phrases

- We could reach above 90% accuracy for the most confident translations

- This technique is useful when monolingual data is plentiful and bilingual data is rare (internet slang words, distant language pairs, …)

Exploiting similarities among languages for machine translation (Mikolov et al, 2013)

Distributed word representations: summary

- Simple model architectures seem to work the best
- Parameter tuning is still a bit of an art: context size, dimensionality, training algorithm, number of training epochs, ...
- Large text corpora are crucial for good performance (some links will be given in the Resources section)
- Current state of the art for word representations is the fastText project (see next session)