Using Neural Networks for Modelling and Representing Natural Languages

Tomas Mikolov, Facebook AI Research

COLING 2014

Structure of the tutorial

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

Introduction

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

- Machine learning and natural language processing techniques are applied to big datasets to improve search, ranking and many other tasks (spam detection, ads recommendation, email categorization, machine translation, speech recognition, …)

Introduction

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

Basic machine learning applied to NLP

- Before we start talking about neural networks, basic techniques will be briefly mentioned
- As we will see later, neural networks are closely related to the more basic techniques
- To avoid re-discovery of the wheel, it is important to know the prior work
- 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*

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

- 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:

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

- 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 parameter estimation, 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 also 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 (where in our case, N would be the size of the vocabulary)

Bag-of-words representations

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

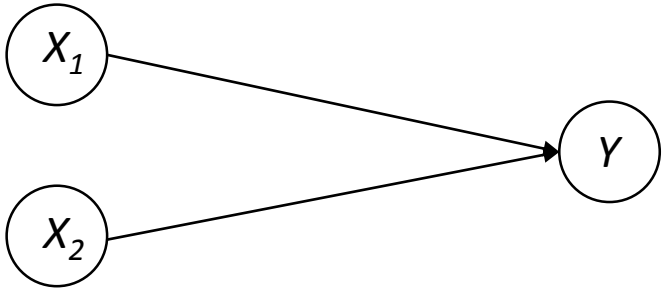
Example:

```
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

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



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

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 problem 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:

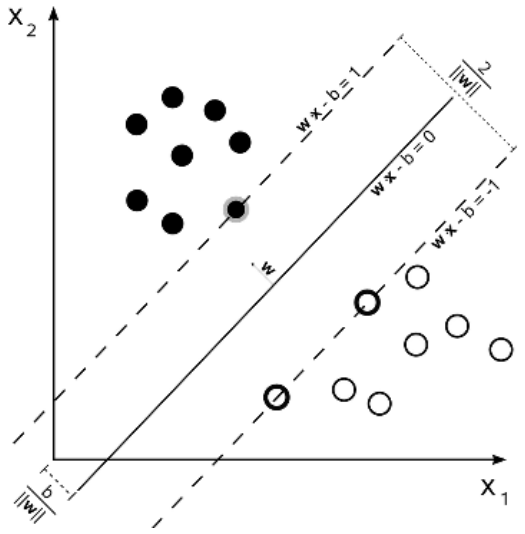


Figure from Wikipedia

- Popular also 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

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”

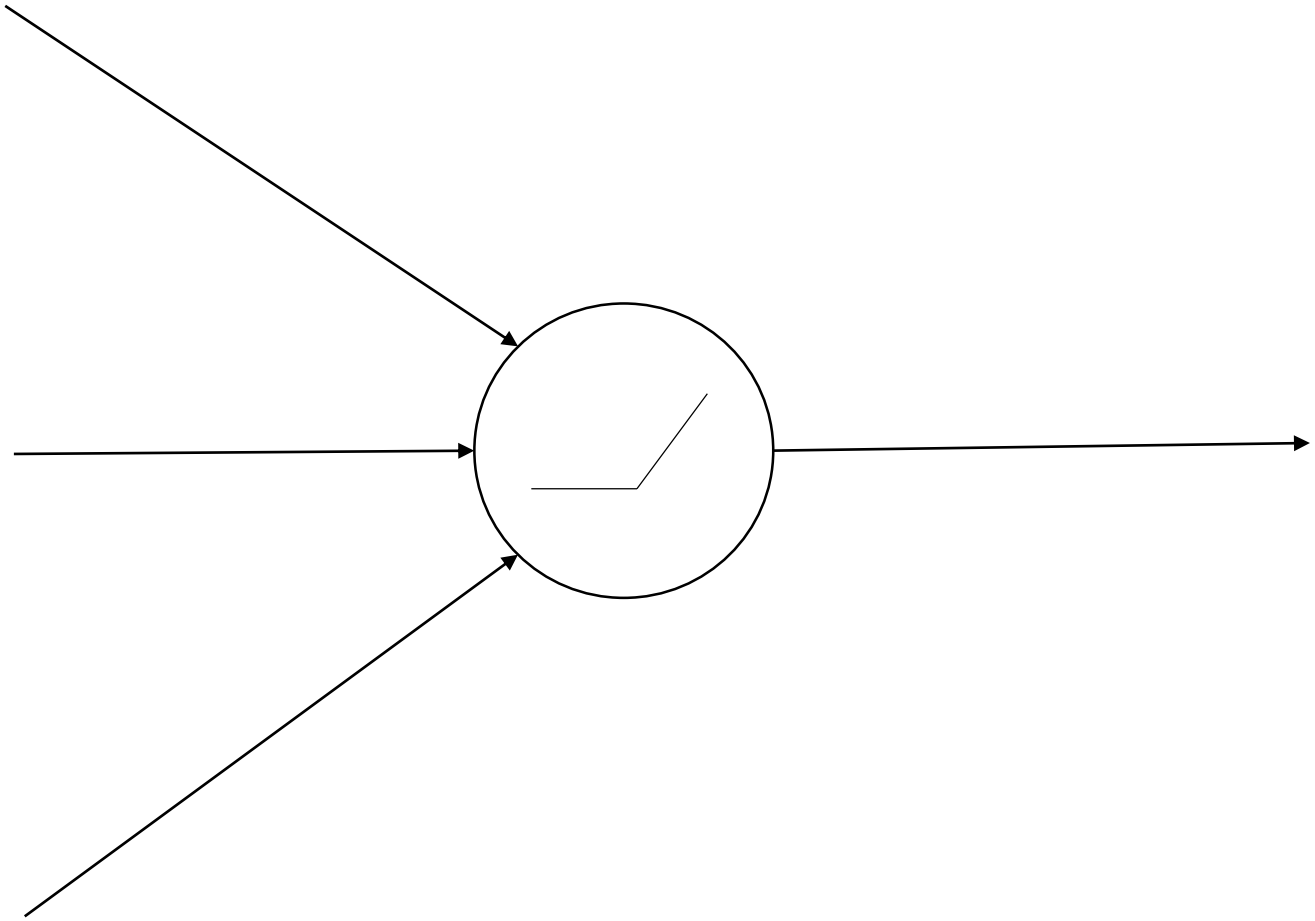
Neural networks: motivation

- The main motivation is to simply come up with more precise way how to represent and model words, documents and language than the basic approaches

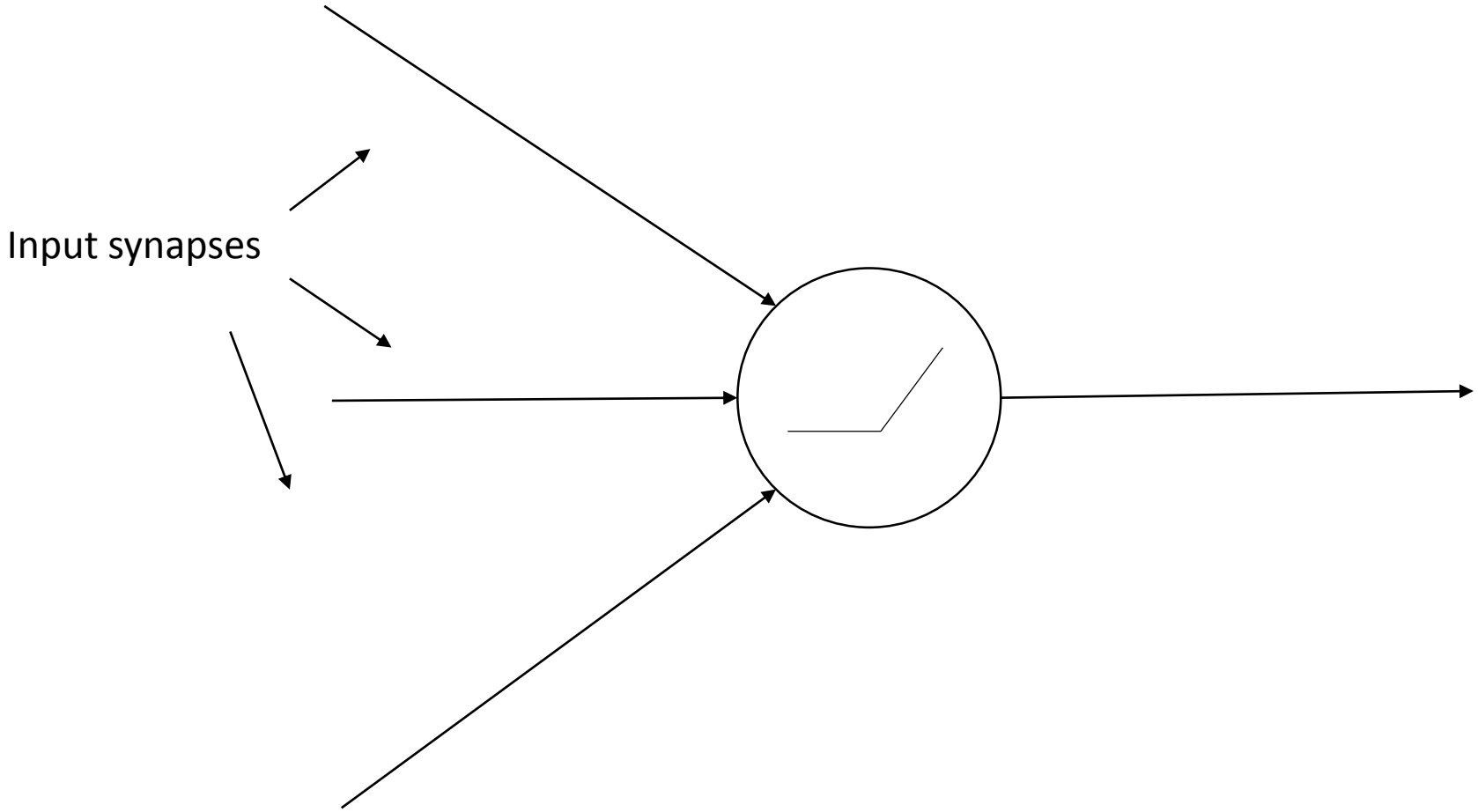
- There is nothing that neural networks can do in NLP that the basic techniques completely fail at

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

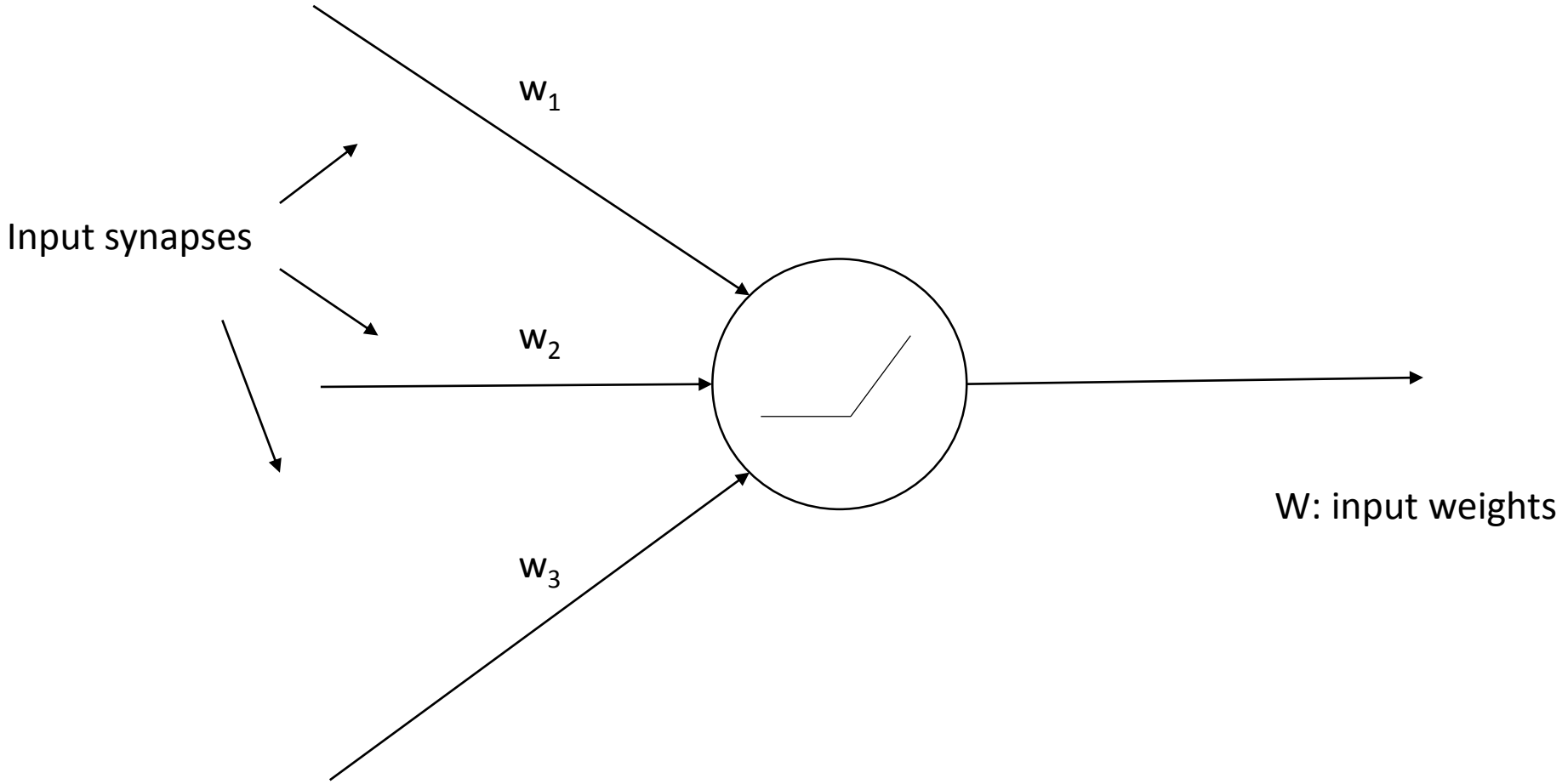
Neuron (perceptron)



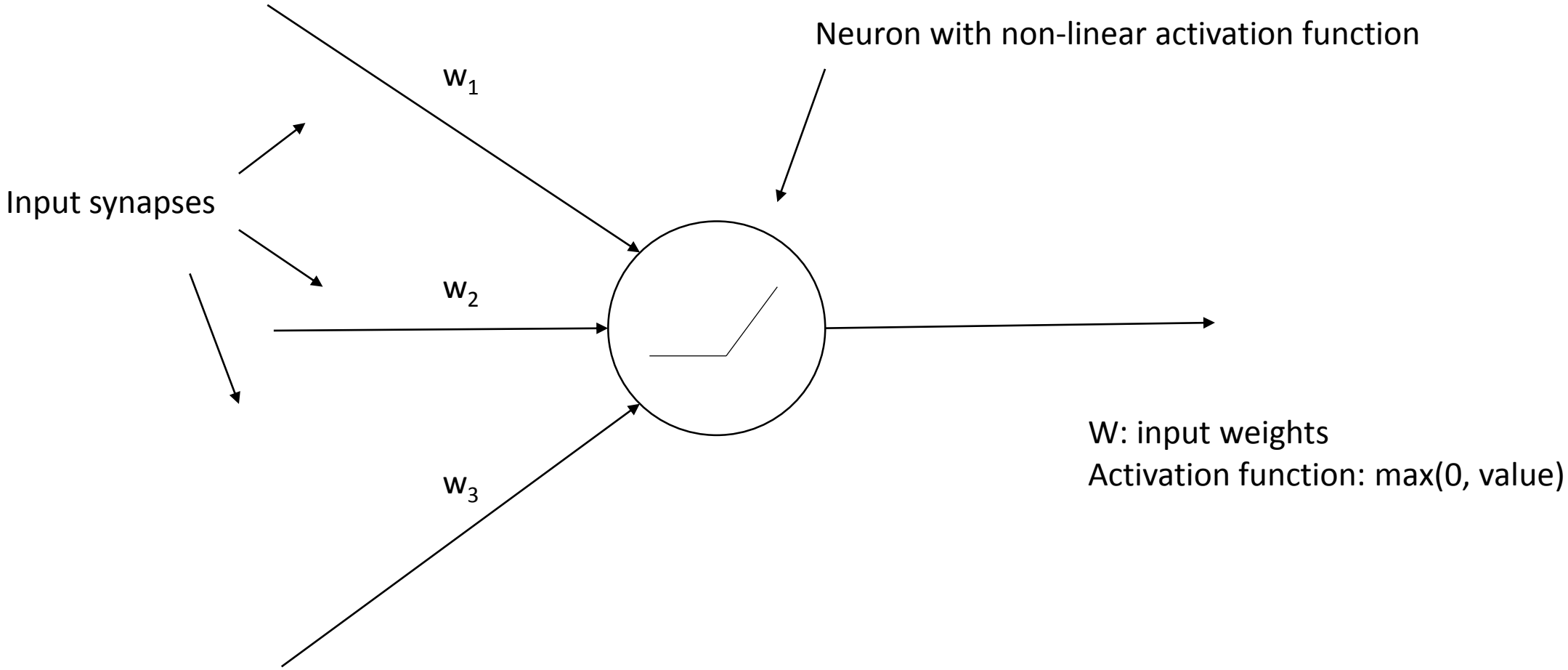
Neuron (perceptron)



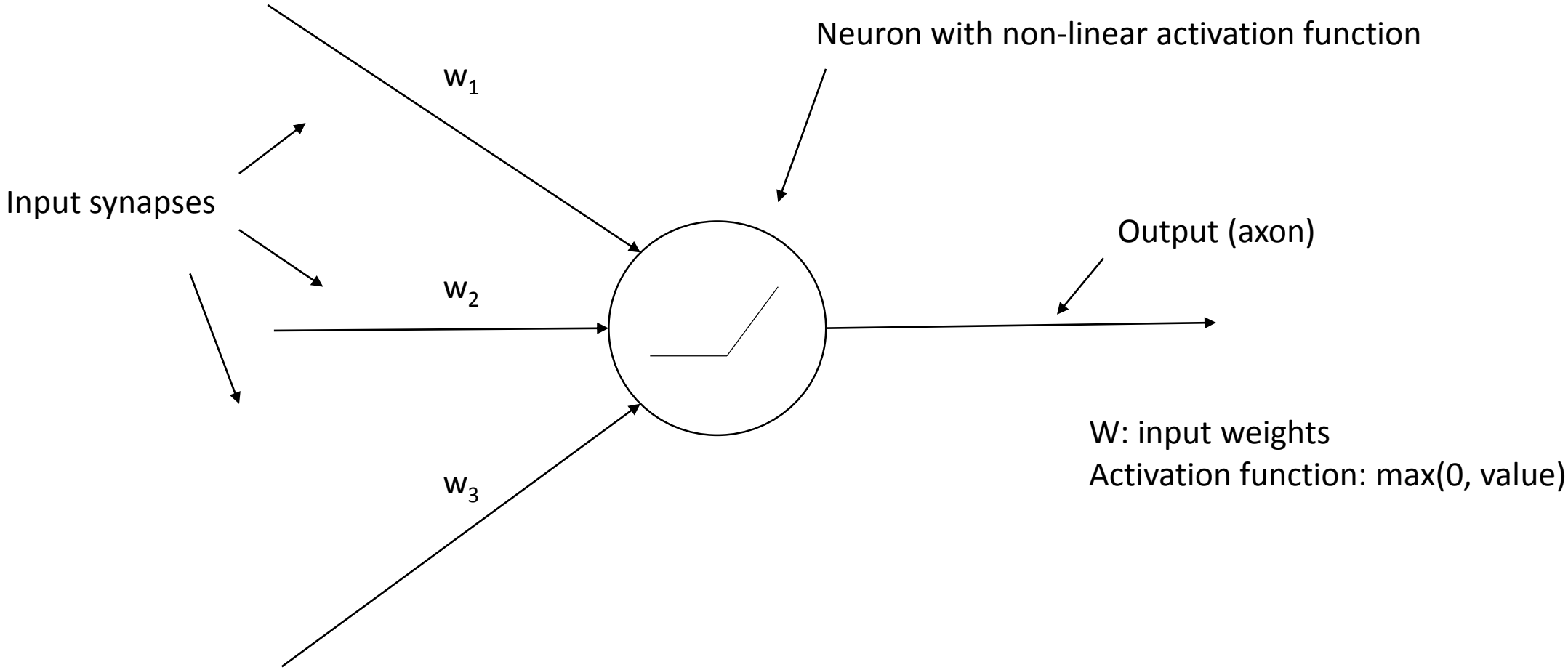
Neuron (perceptron)



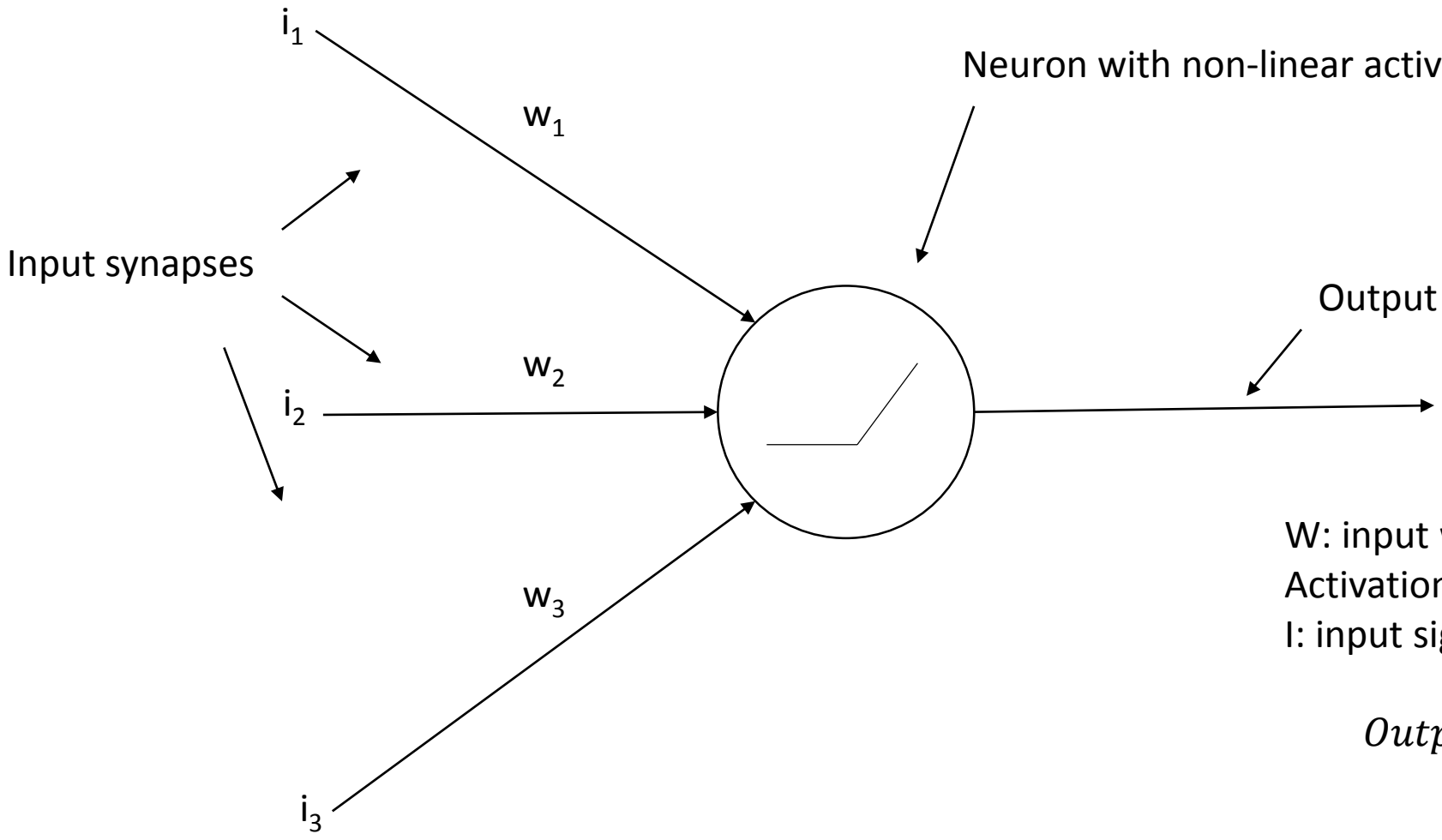
Neuron (perceptron)



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 different frequencies)

- The learning seems 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 usually referred to as “rectified activation function”
- Many other functions can be used
- Other common ones: sigmoid, tanh

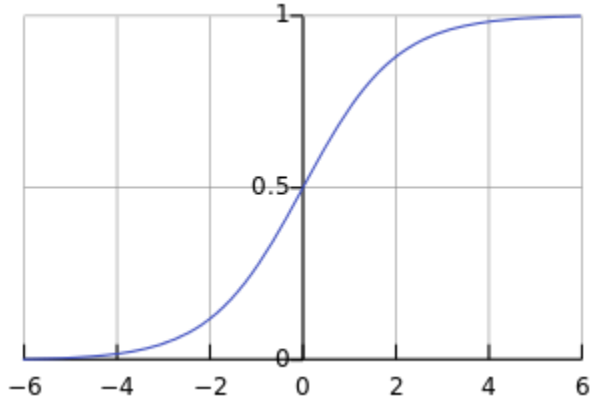
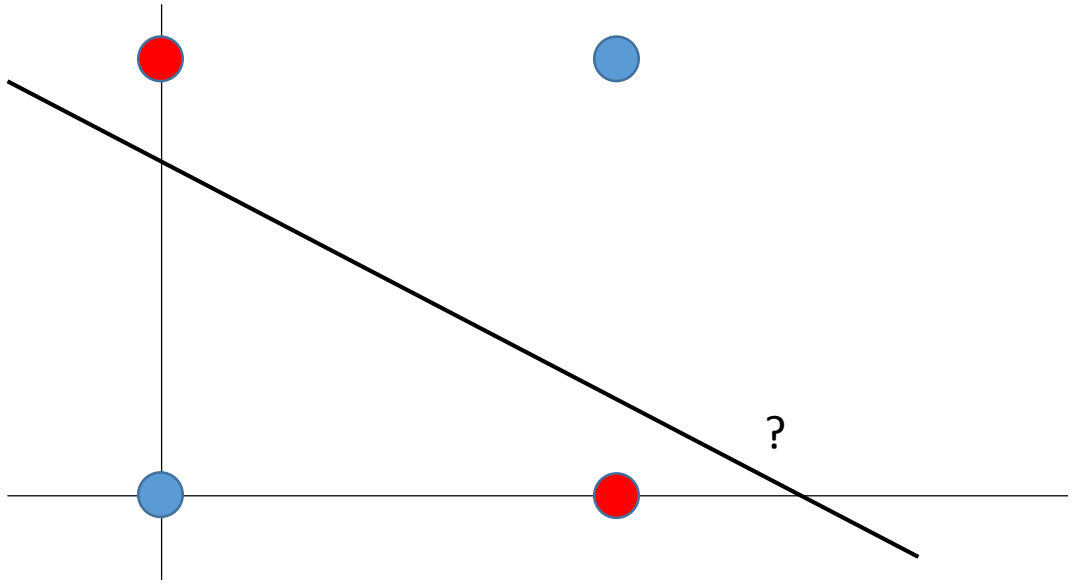


Figure from Wikipedia

Activation function

- The critical part is however the non-linearity
- Example: XOR problem
- There is no linear classifier that can solve this problem:



Non-linearity: example

Intuitive NLP example:

- Input: sequence 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 SVM, 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)

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

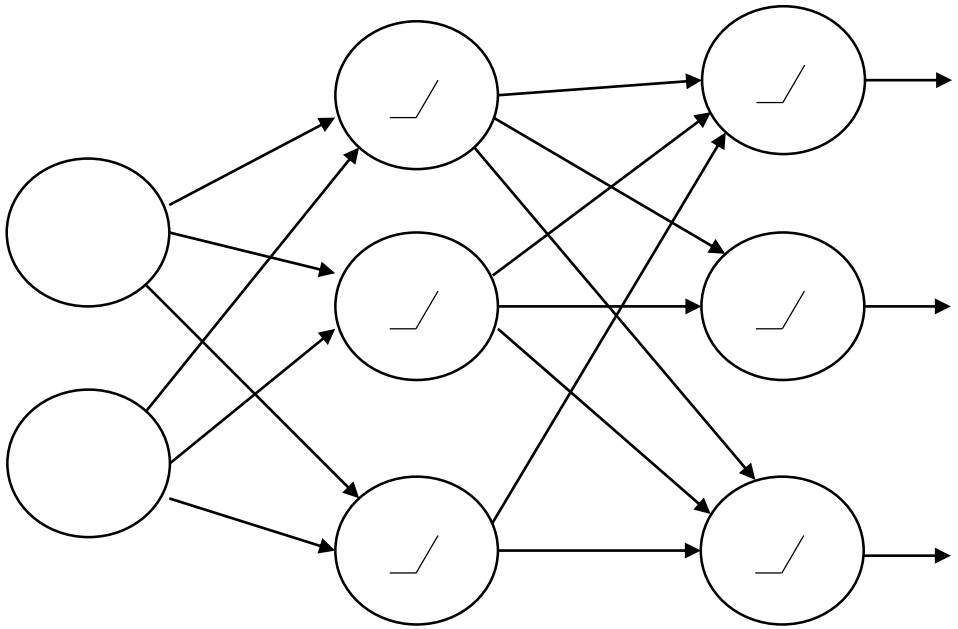
1. some neurons in the hidden layer will activate only for some combination of input features
2. the output layer can represent combination of the activations of the hidden neurons

Hidden layer

- Neural net with one hidden layer is universal approximator: it can represent any function

- 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



INPUT LAYER HIDDEN LAYER OUTPUT LAYER

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 maximized / minimized
- Example: classification accuracy, reconstruction error

Unsupersvied / supervised training

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

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

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

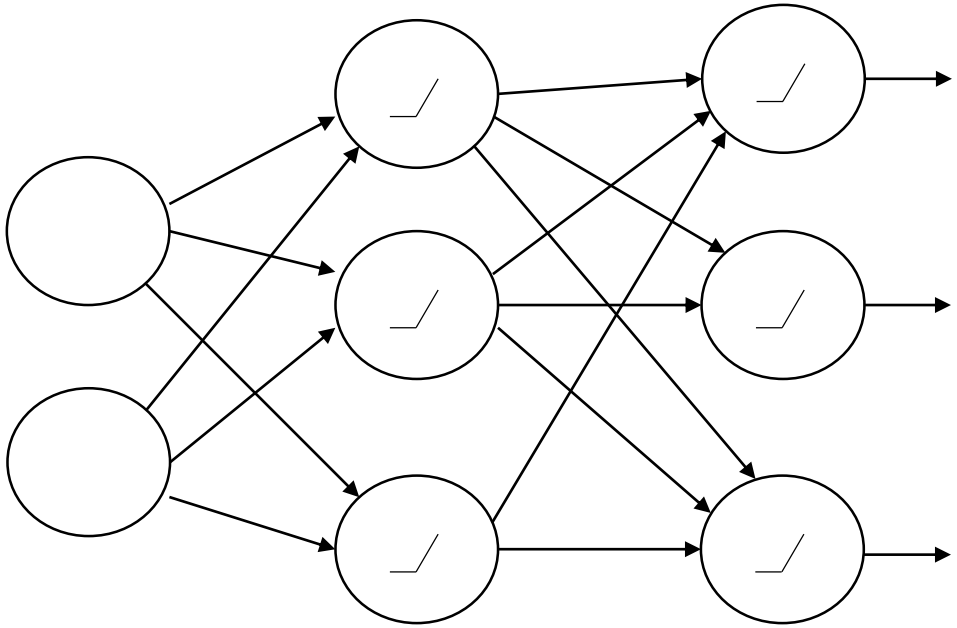
Training of neural networks

- There are many ways how to train neural networks
- The most widely used and successful in practice is stochastic gradient descent
- Many algorithms are introduced as superior to SGD, but when properly compared, the gains 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:

- We update the weights after each training example is presented to the network
- 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

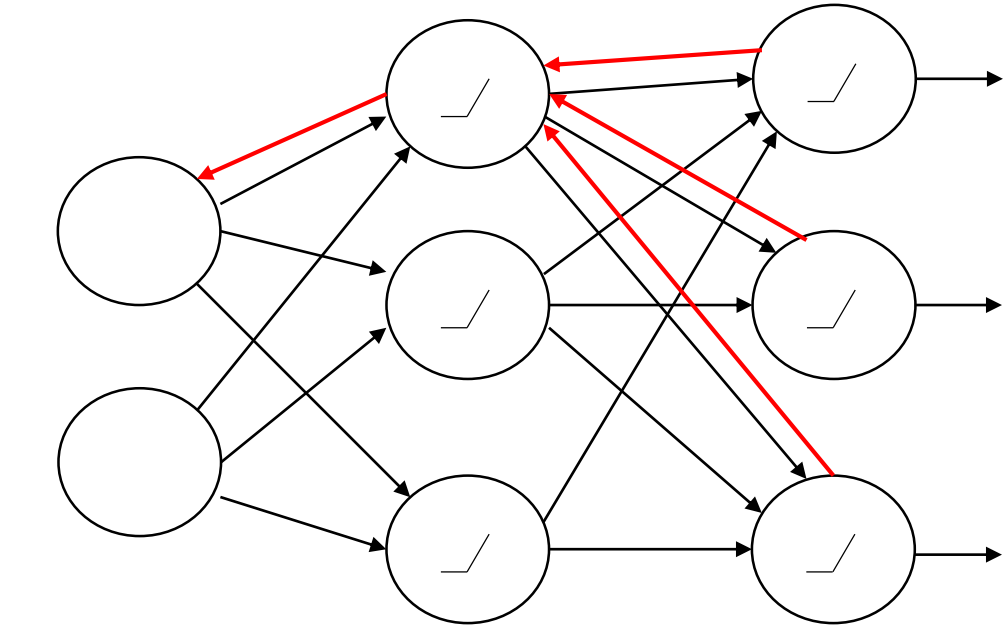
Intuitive explanation of stochastic gradient descent:

- We update the weights after each training example is presented to the network
- 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

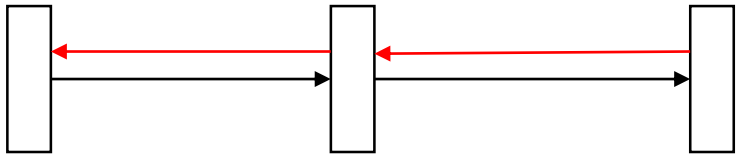
Backpropagation

- 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 much we change the weights: too little value will result in long training time, too high value will erase previously learned patterns

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

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

Regularization

- As the network is trained, it often overfits the training data: it has very good performance during training, but fails to generalize in test

- The network “memorizes” the training data: often, it will contain high weights that are used to model only some small subset of data

- We can try to force the weights to stay small during training to avoid this problem (L1 & L2 regularization)

Training of neural networks: summary

- Stochastic gradient descent and backpropagation are usually good enough

- The 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

Comparison of neural networks to 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 should use both jointly: will be shown later

Deep learning

- Deep model architecture is about having more computational steps (hidden layers) in the model

- Deep learning aims to learn patterns that cannot be learned efficiently with shallow models

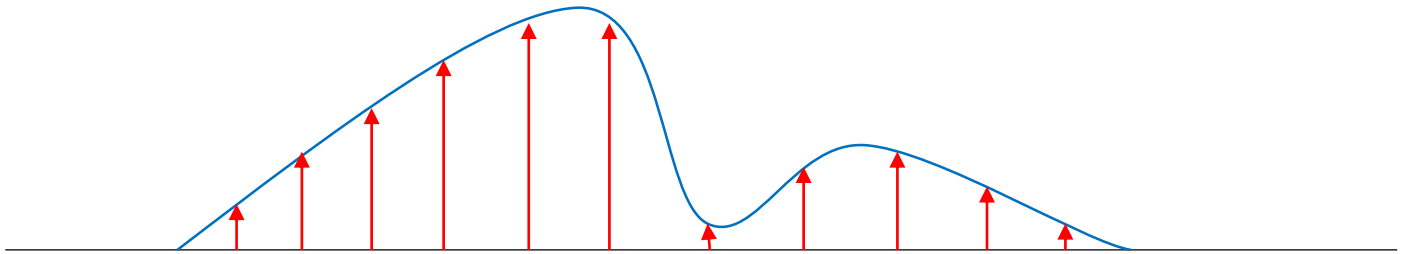
Deep learning

- But: it was previously mentioned that one hidden layer neural net is universal approximator as it can 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

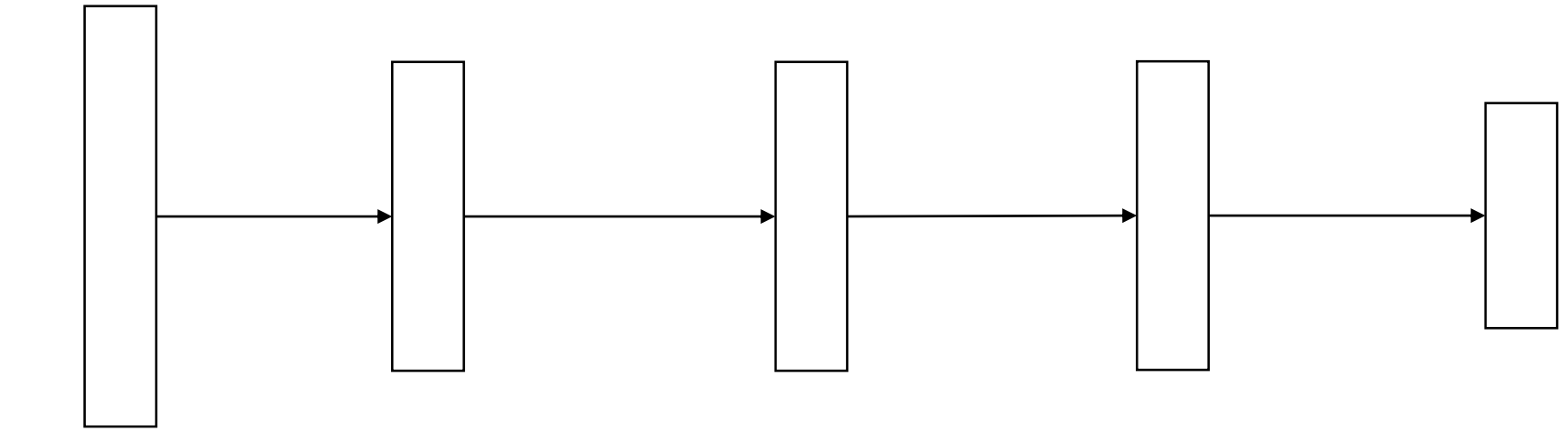
- Example of function that is difficult to represent: parity function (N bits at input, output is 1 if the number of active input bits is odd) (*Perceptrons*, Minsky & Papert 1969)

Deep learning

- Having hidden layers exponentially larger than is necessary is bad
- If we cannot compactly represent patterns, we have to memorize them -> need exponentially more training examples

Deep learning

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



Deep learning

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

- Recently there was a lot of success for 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

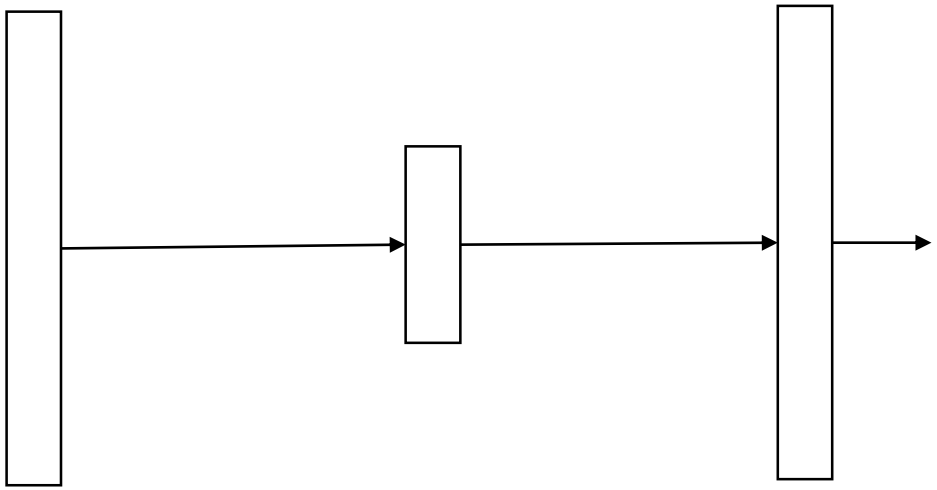
Neural networks and deep learning: summary

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

Distributed representations of words

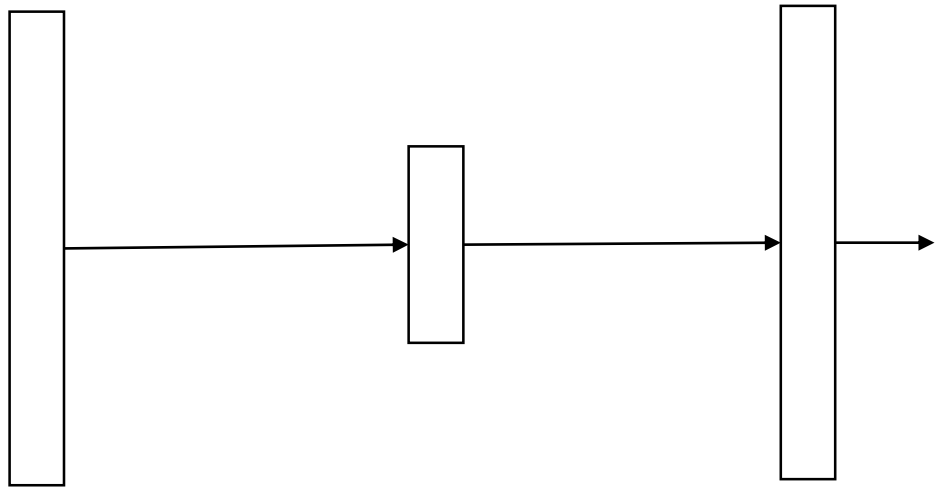
- Vector representation of words computed using neural networks
- Linguistic regularities in the word vector space
- Evaluation of performance
- Application to machine translation

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 representations 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 associated with 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

Word vectors

- These word vectors can be subsequently used as features in many NLP tasks (Collobert et al, 2011)
- As word vectors can be trained on huge text datasets, they provide generalization for systems trained with limited amount of supervised data
- More complex model architectures can be used for obtaining the word vectors (neural net language model with multi-task learning (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 way how to compare word vectors trained using different architectures

- The comparison is tricky: people mostly published just their pre-trained word vectors, everyone was using different datasets both for training and evaluation…

- Conclusions based on experiments with different datasets are misleading

Word vectors - evaluation

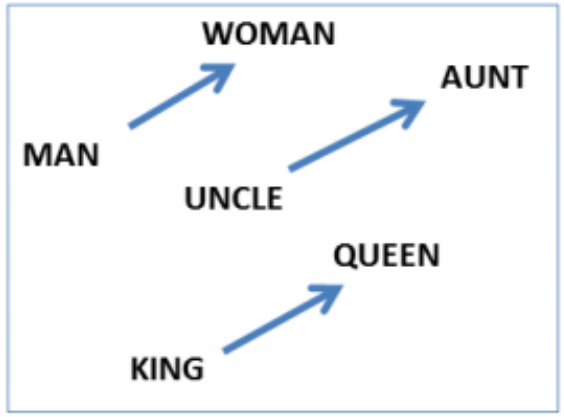
Existing datasets like WS353 (word similarity, 353 word pairs with human judgements of similarity) have several drawbacks:

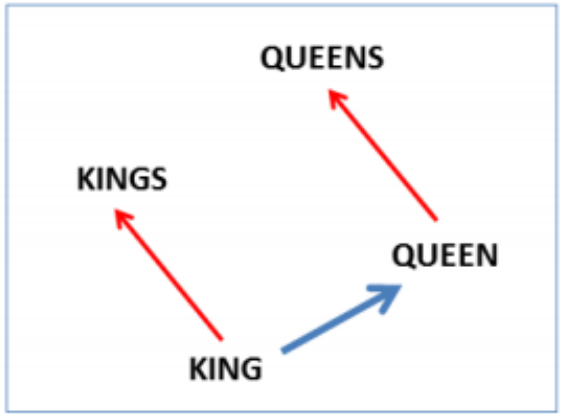
- Tiny size, no heldout / test data split
- Performance is heavily biased by choice of the training data, less so by the architecture of the model itself

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

Word vectors – linguistic regularities

- Recently, 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 regularities in continuous space word representations* (Mikolov et al, 2013))





Word vectors – datasets for evaluation

MSR dataset with 8K questions, mostly focuses on syntax - examples:

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

More in *Linguistic Regularities in Continuous Space Word Representations* (Mikolov, Yih, Zweig, 2013)

Word vectors – datasets for evaluation

Google word-based dataset, almost 20K questions, focuses on both syntax and semantics:

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

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

Word vectors – datasets for evaluation

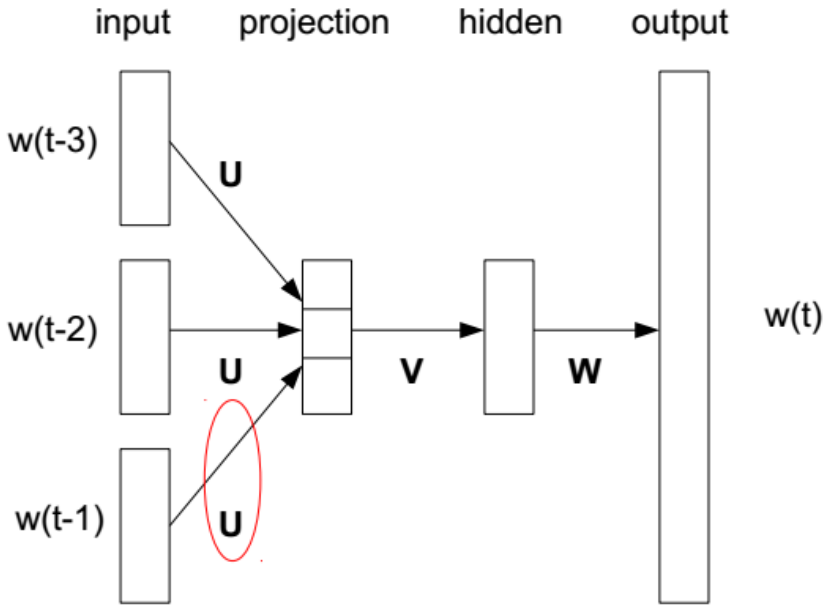
Google phrase-based dataset, focuses on semantics:

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

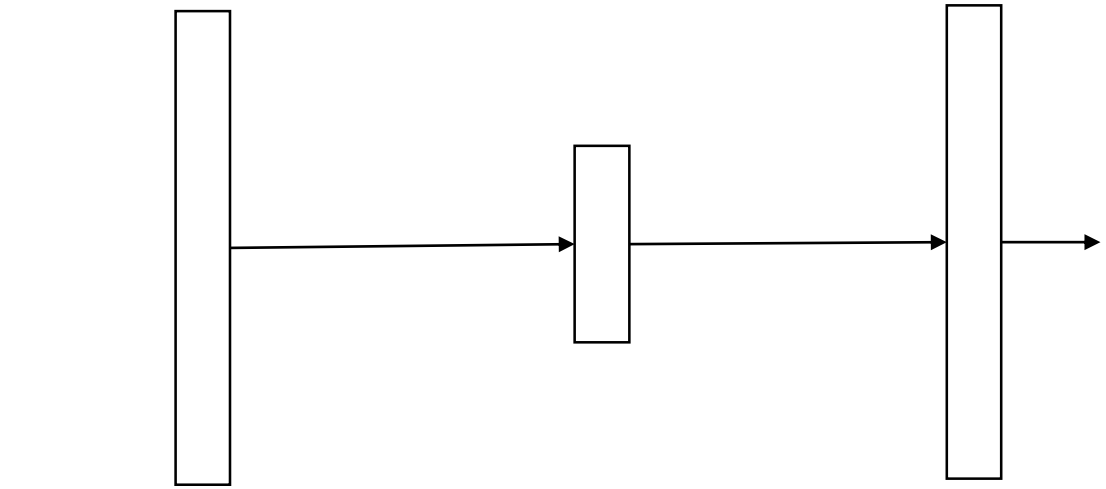
Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al, 2013)

Word vectors – various architectures

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



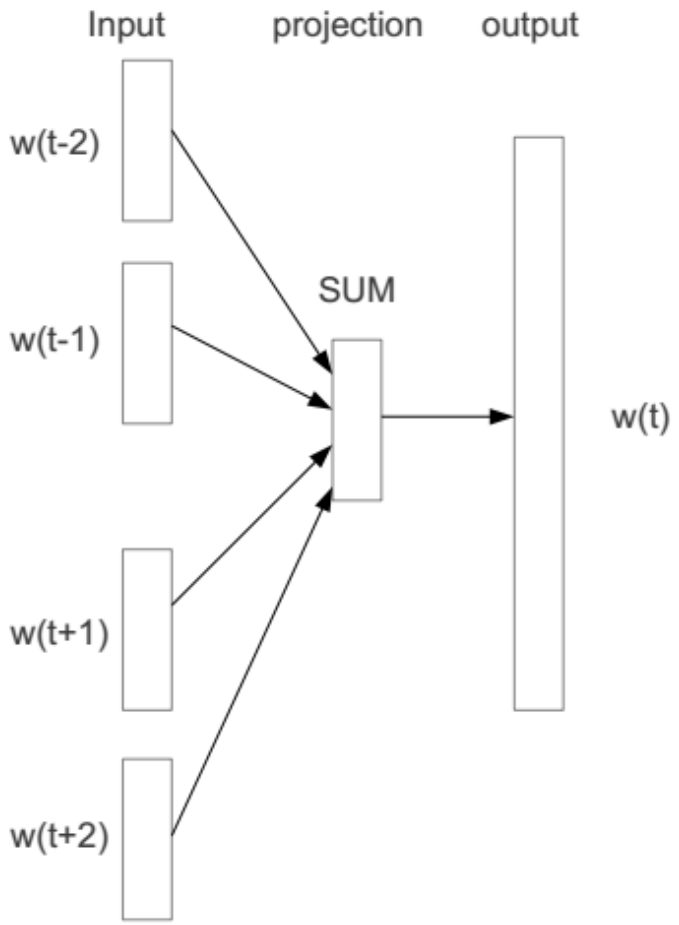
Word vectors – various architectures



- We can extend the bigram NNLM for training the word vectors by adding more context

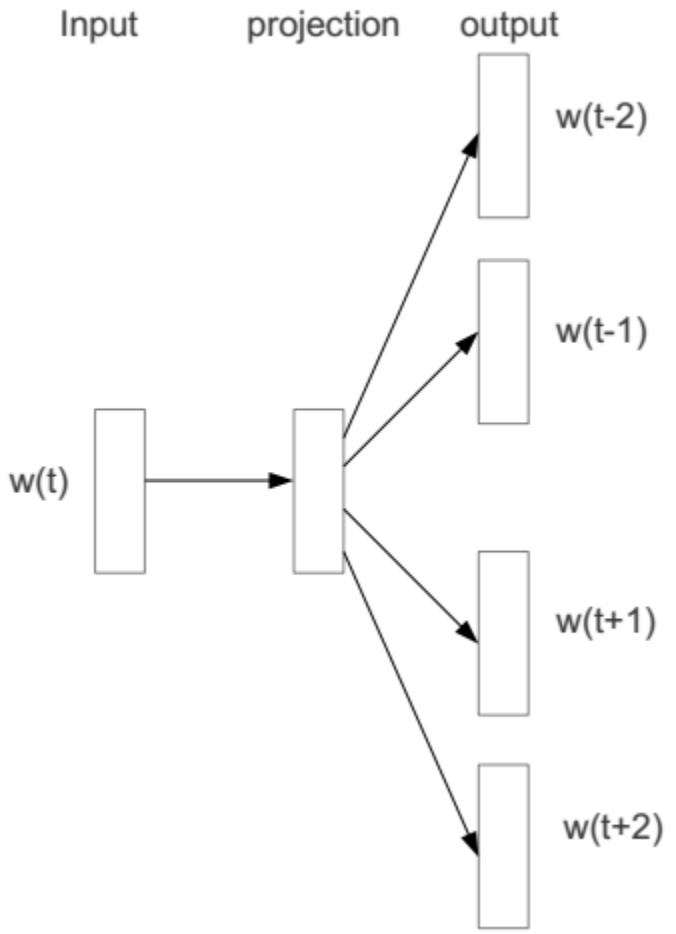
Word vectors – various architectures

- 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
- The hidden layer is just linear



Word vectors – various architectures

- We can reformulate the CBOW model by predicting surrounding words using the current word
- This architectures is called ‘skip-gram NNLM’
- If both are trained for sufficient number of epochs, their performance is similar



Word vectors - training

- SGD + backpropagation

- Solution to very large output layer – size equal to vocabulary size, can easily be in order of millions (too many outputs to evaluate):

1. Hierarchical softmax
2. Negative sampling

Word vectors - training

- It is useful to sub-sample the frequent words (such as ‘the’, ‘is’, ‘a’, …) during training

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

Word vectors – negative sampling

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

- The output neurons are treaded as independent logistic regression classifiers

- This makes the training speed independent on the vocabulary size

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)
- Almost all models are trained on different datasets

Word vectors – scaling up

- The choice of training corpus is usually more important than the choice of the technique itself
- The crucial component of any successful model thus should be low computational complexity

- Optimized code for computing the CBOW and skip-gram models has been published as word2vec project: https://code.google.com/p/word2vec/

Word vectors – nearest neighbors

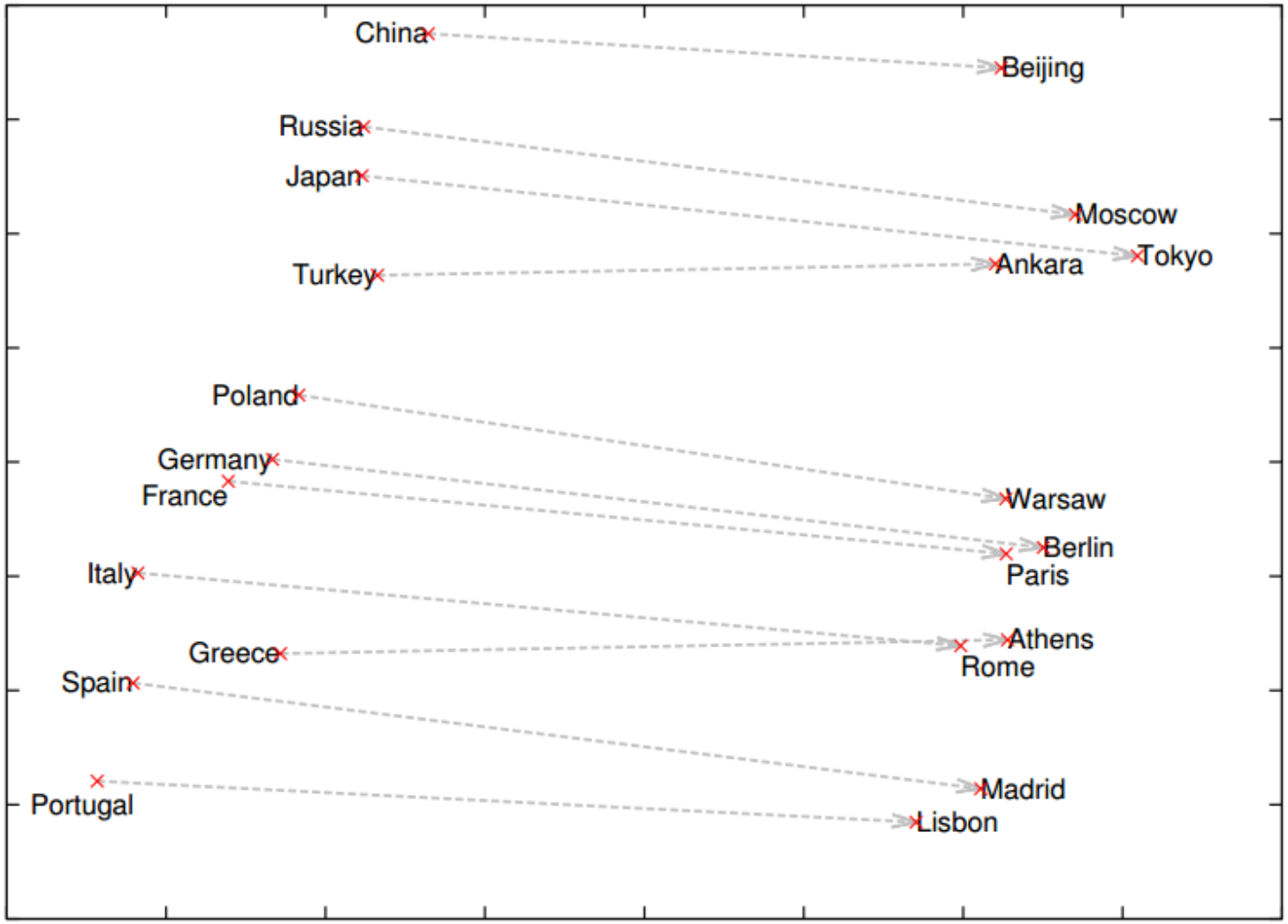
	Redmond	Havel	graffiti	capitulate
Collobert NNLM	conyers lubbock keene	plauen dzerzhinsky osterreich	cheesecake gossip dioramas	abdicate accede rearm
Turian NNLM	McCarthy Alston Cousins	Jewell Arzu Ovitz	gunfire emotion impunity	- - -
Mnih NNLM	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	anaesthetics 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, it does not make sense to treat *New York* or *Air Canada* as separate words; we should be working with phrases (even more obvious for names of people)

- Phrases can be constructed using mutual information criterion: if certain words appear next to each other more than their individual frequency suggests, they likely should form a phrase

- Simple way how to deal with phrases is to 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

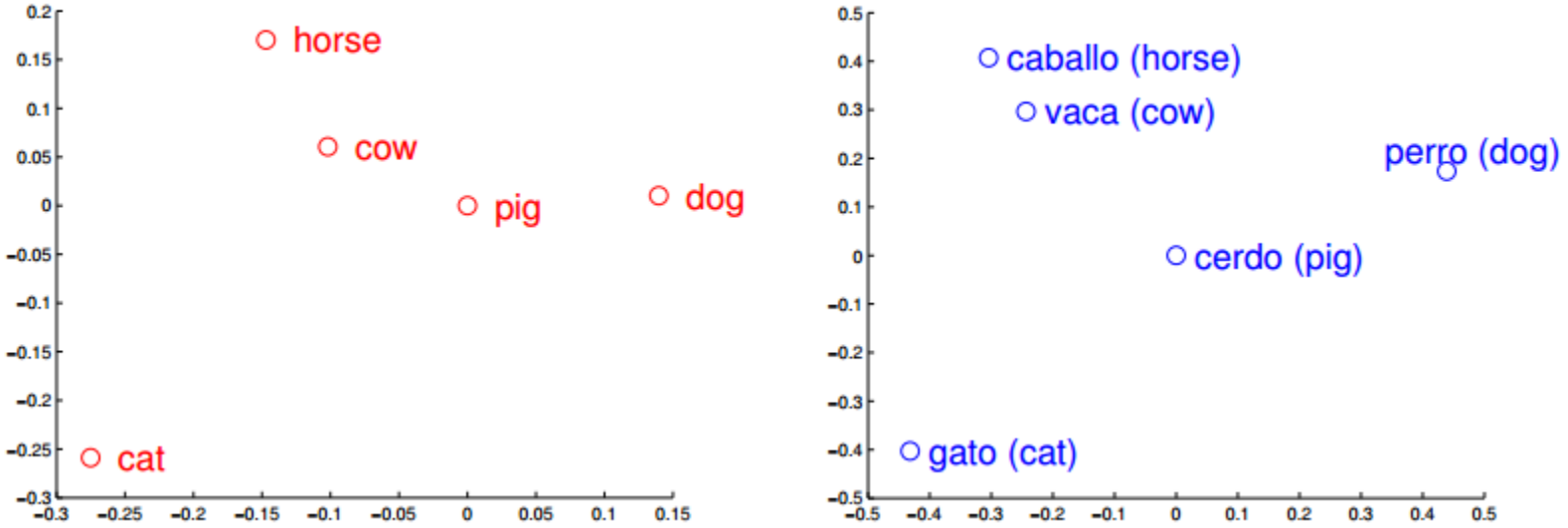
- The sentence representations can be further used in classifiers (logistic regression, SVM, or neural network)

- Needs to be trained for many epochs; seems to achieve state of the art results on sentiment analysis tasks, beating much more intricate approaches such as recursive neural networks

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

Translation of words and phrases using vector spaces

- Idea: there are many languages across the world that describe the same patterns – dogs have four legs, the sky is blue, …
- The words and concepts within languages should be related in a similar way

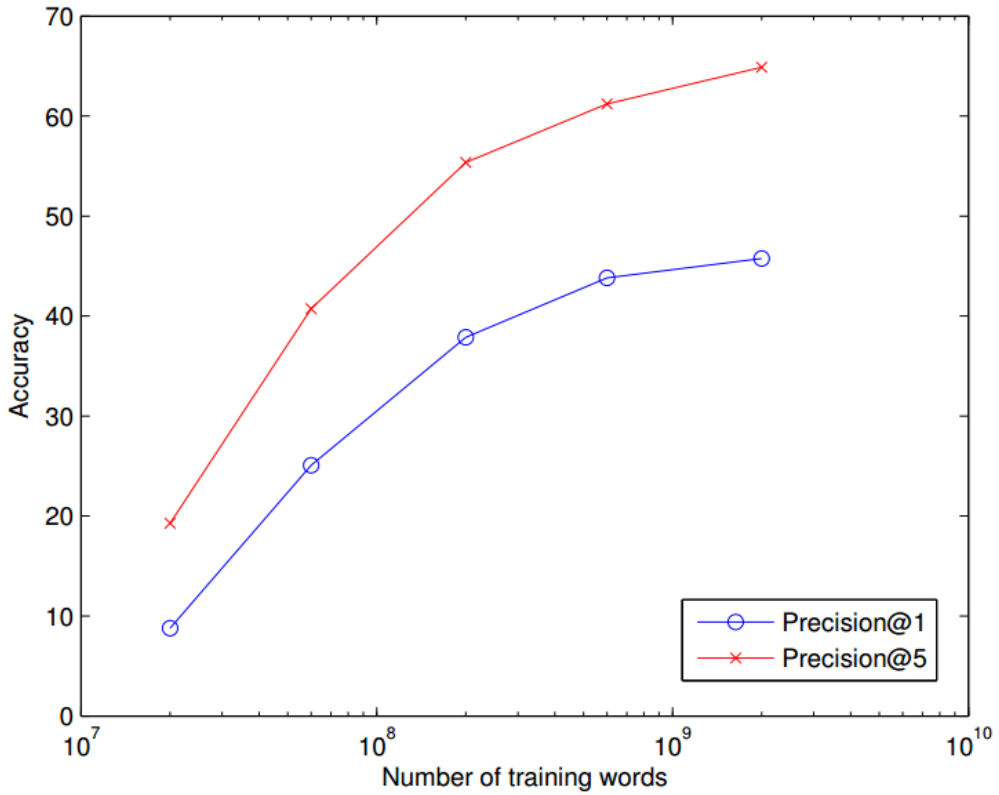


Translation of words and phrases using vector spaces

- Thus, it should be enough to learn mapping between the language 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

- In cases where monolingual data are plentiful and bilingual data are rare (internet slang words, distant language pairs, …), this technique seems promising

More in: *Exploiting similarities among languages for machine translation* (Mikolov et al, 2013)

Comparison to prior state of the art

- The recent discoveries suggest that the whole neural net - word vector idea is very close to the existing distributional semantics models (counts of co-occurrences of words within a window)

- The non-linearities do not seem to be crucial for the unsupervised learning (but would help if one would use the word vectors as features in a classification task)

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

Comparison to prior state of the art

Objective	Representation	MSR	GOOGLE
3COSADD	Embedding	53.98%	62.70%
	Explicit	29.04%	45.05%
3COSMUL	Embedding	**59.09%**	66.72%
	Explicit	56.83%	**68.24%**

Table 3: Comparison of **3COSADD** and **3COSMUL**.

The linguistic regularities are not exclusive property of neural net based representations: *Linguistic Regularities in Sparse and Explicit Word Representations* (Levy & Goldberg, 2014)

Comparison to prior state of the art

	rg	ws	wss	wsr	men	toefl	ap	esslli	battig	up	mcrae	an	ansyn	ansem
best setup on each task														
cnt	74	62	70	59	72	76	66	84	98	41	27	49	43	60
pre	84	75	**80**	**70**	**80**	91	75	86	**99**	41	28	**68**	**71**	**66**
other models														
soa	**86**	**81**	77	62	76	**100**	**79**	**91**	96	**60**	**32**	61	64	61
dm	82	35	60	13	42	77	76	84	94	51	29	NA	NA	NA
cw	48	48	61	38	57	56	58	61	70	28	15	11	12	9

Table 2: Performance of count (cnt), predict (pre), dm and cw models on all tasks.

Comparison to traditional vector space models: *Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors* (Baroni et al, 2014)

Distributed word representations: summary

- Simple models seem to be sufficient
- Parameter tuning is still a bit of an art: context size, number of dimensions, training algorithm, …
- Large text corpora are crucial for good performance (some links will be given in the Resources section)
- Train for more epochs if you have small training sets!

Neural network based language models

- Feedforward and recurrent neural net architectures for language modeling
- Dealing with large number of outputs: class based softmax, hierarchical softmax
- Joint training with maximum entropy model
- Recurrent model with slow features
- Applications in language modeling, speech recognition, machine translation

Language modeling with neural networks

- Statistical language modeling is one of the oldest, well-studied and important NLP tasks
- The language models are core of machine translation, speech recognition and many other applications
- Historically, it was amazingly difficult to convincingly beat N-grams, especially on larger than tiny datasets

N-grams

- Task: compute probability of a sentence *W*

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

- Often simplified to trigrams:

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

Language modeling with neural networks

- Main deficiency of N-grams is the exponential growth of number of parameters with length of the context

- Neural networks address this problem by performing dimensionality reduction and parameter sharing

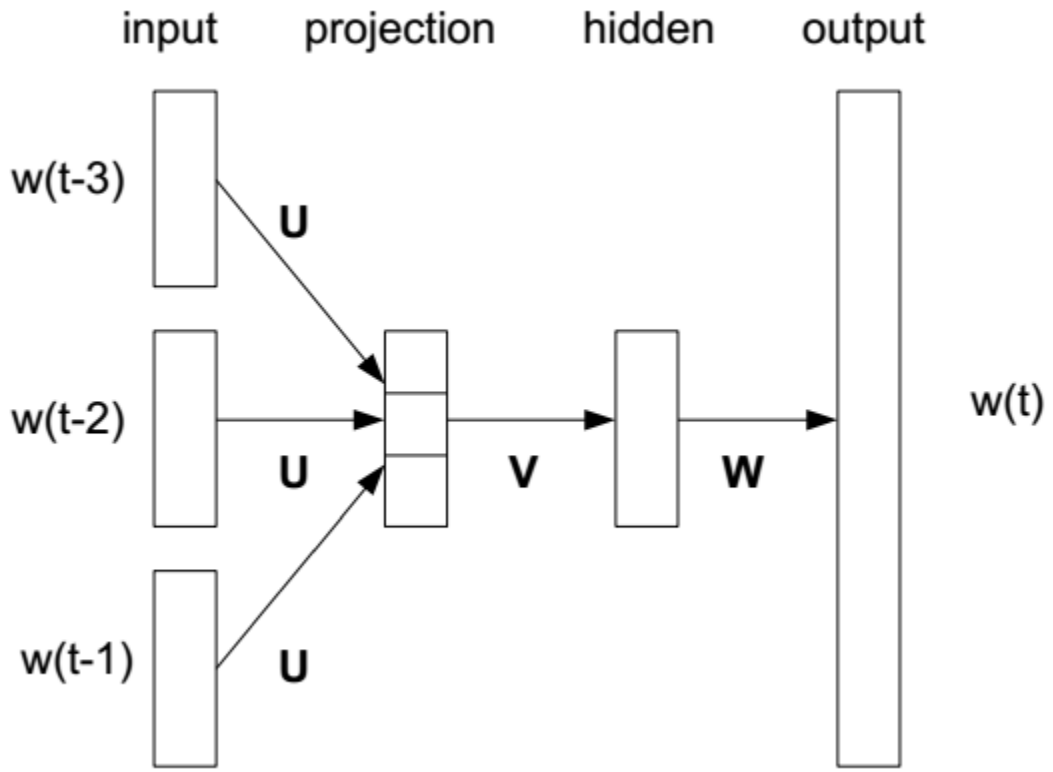
Language modeling with neural networks

- Neural network language models are today state of the art, often applied to systems participating in competitions (ASR, MT)

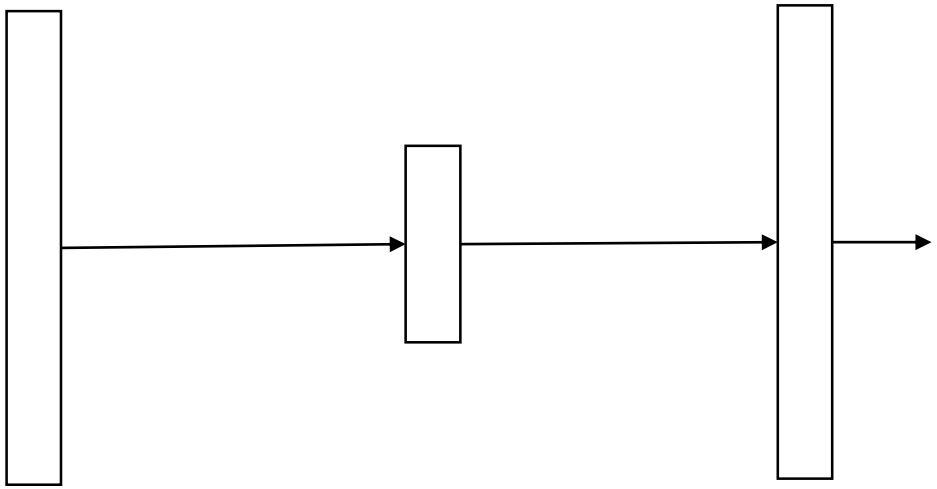
- There are two main types of neural network architectures for language modeling: feedforward and recurrent

Feedforward neural network LM

- Proposed by (Bengio et al, 2003)
- The projection layer is linear
- The hidden layer is non-linear
- Softmax at the output computes probability distribution over the whole vocabulary
- The basic model is computationally very expensive



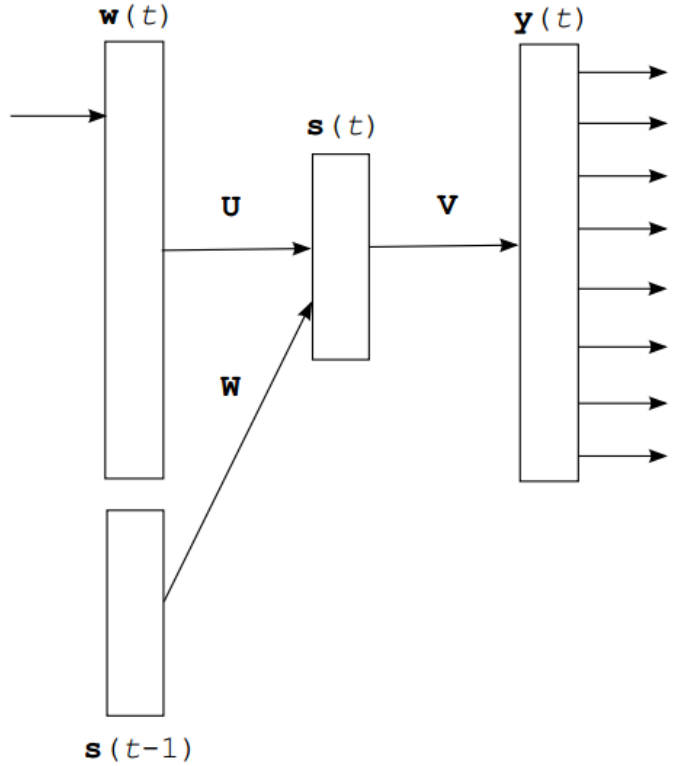
Towards recurrent neural network LM



- Back to the bigram NNLM: is there a better way how to represent time than using N-1 previous words as separate inputs?

Recurrent neural network LM

- Recurrent NNLM is about the same as the bigram NNLM
- Additional weights from the hidden layer in the previous time step
- In theory, the hidden layer can learn to represent unlimited memory

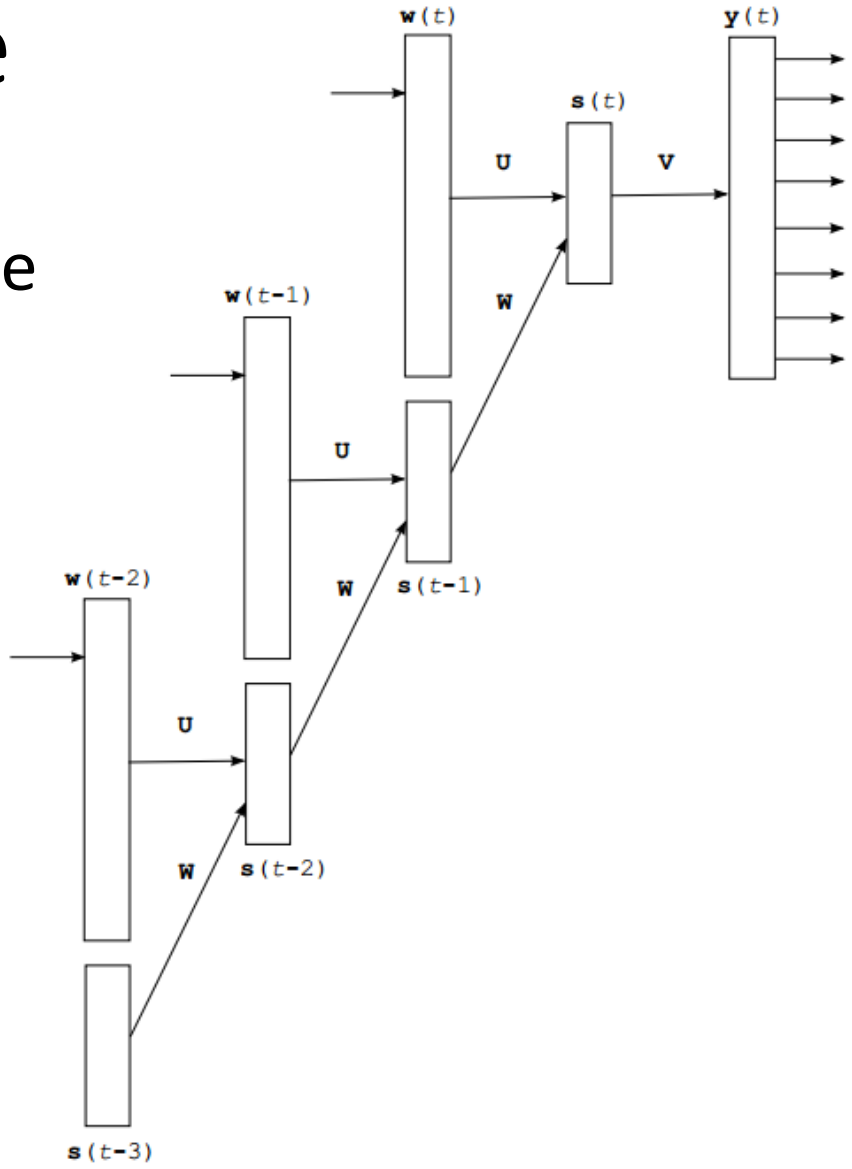


Training of NNLMs

- Feedforward NNLM: the classis SGD + backpropagation
- Recurrent NNLM: the same, but the backpropagation part is more difficult to implement correctly
- The algorithm for computing gradients in RNN is called "Backpropagation through time"

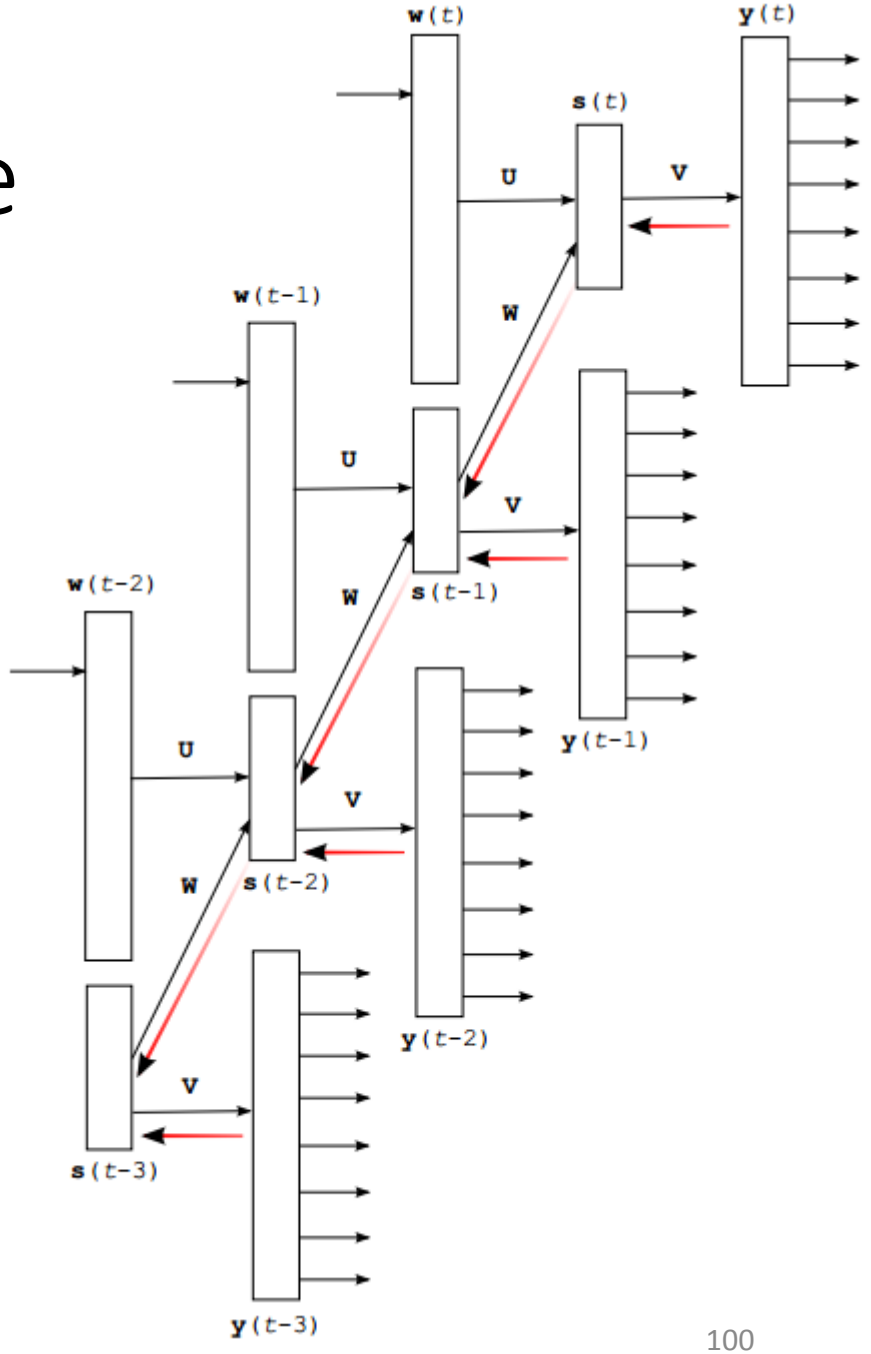
Backpropagation through time

- The intuition is that we unfold the RNN in time
- We obtain deep neural network with shared weights **U** and **W**



Backpropagation through time

- We train the unfolded RNN using normal backpropagation + SGD
- In practice, we limit the number of unfolding steps to 5 – 10
- It is computationally more efficient to propagate gradients after few training examples (batch mode)



Backpropagation through time

Problems of BPTT:

- Vanishing gradients
- Exploding gradients

Vanishing gradients

- As we propagate the gradients back in time, usually their magnitude decreases, and quickly approaches tiny values: this is called vanishing gradient

- In practice this means that learning long term dependencies is difficult

- Special architectures address this problem (*Long Short-term Memory* – LSTM RNN (Hochreiter & Schmidhuber, 1997))

Exploding gradients

- Sometimes, the gradients start to increase exponentially during backpropagation through the recurrent weights: this is the exploding gradient

- While this is a more rare situation, the effect can be catastrophic: huge gradients will lead to big change of weights, and the network will forget almost all it has learned so far

- Simple solution: clip values of the gradients

Comparison of performance on small data: Penn Treebank

- Small, standard dataset, ~1M words
- RNN outperforms FNN by about 10%

Model	Perplexity
Kneser-Ney 5-gram	141
Maxent 5-gram	142
Random forest	132
Feedforward NNLM	140
Recurrent NNLM	125

More results in: *Empirical Evaluation and Combination of Advanced Language Modeling Techniques* (Mikolov et al, 2011)

Scaling up to large datasets

- While neural network LMs have been around for a while, their computational complexity compicated their use in real-world systems

Main computational bottlenecks:

1. Computation of probability distribution in the output layer
2. Dense matrix multiplication (FNN: projection -> hidden layer, RNN: recurrent matrix)

Softmax

- Softmax at the output layer computes probability distribution over the whole vocabulary:

$$g(o) = \frac{e^{o_i}}{\sum_k e^{o_k}}$$

- Sums to 1, all output values are non-negative

Scaling up: short lists

- Instead of computing the whole probability distribution using NNLM, we can evaluate just the most frequent words (2000 – 8000)
- The rest is computed using (fast) N-gram model
- Downside: significantly reduces accuracy (after all, modelling of rare words is what neural nets do better than N-grams)

Continuous space language models (Schwenk, 2007)

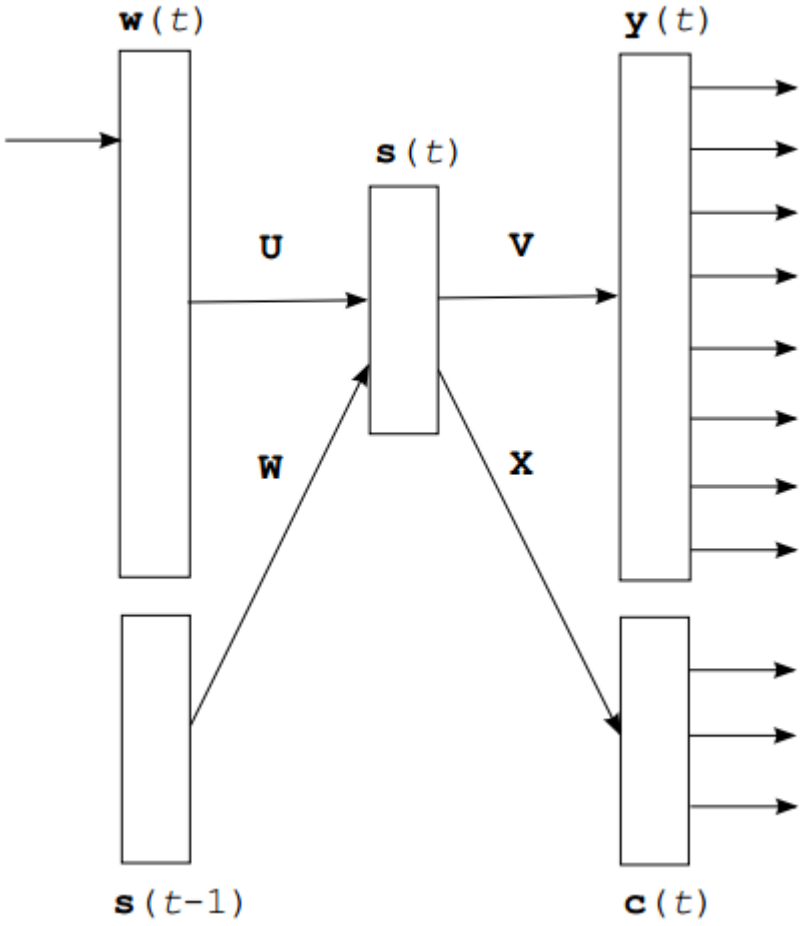
Scaling up: class based softmax

- Instead of normalizing probability over all words, we:

1. Assign each word to a single class
2. Normalize over the class layer
3. Normalize over words from within the current class

- Reduces complexity from |V| to about sqrt(|V|)

Extensions of recurrent neural network language model (Mikolov et al, 2011)



Hierarchical softmax

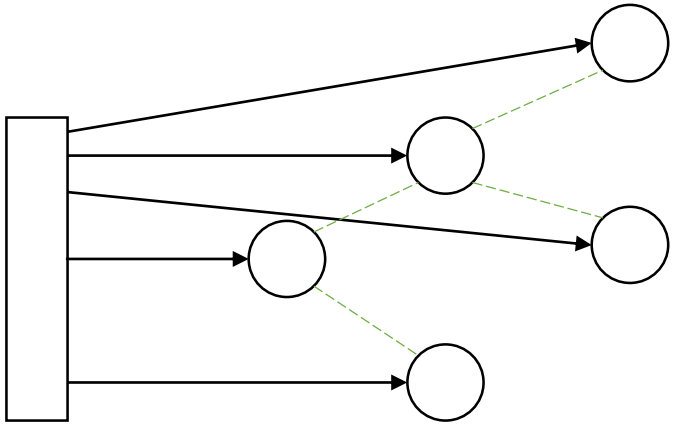
- We can add classes over classes
- Extreme case: binary tree over the whole vocabulary
- Further reduces the complexity of softmax class to about log(|V|)

Hierarchical Probabilistic Neural Network Language Model (Morin & Bengio, 2005)

Hierarchical softmax

- Use of Huffman encoding further increases speed (frequent words have short binary codes)

- Note: the tree nodes are conditioned on the state of the hidden layer (not on the state of other nodes)



HIDDEN LAYER HIERARCHICAL SOFTMAX

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

How to assign words to classes?

- Frequency: simple to implement, no pre-computation, but degrades accuracy:
  - ~10% worse PPL for classes against full softmax
  - ~30% for Huffman binary tree

How to assign words to classes?

Brown classes:

- good accuracy (little to none loss in perplexity)
- has to be pre-computed
- not computationally efficient as the frequency-based classes (need to evaluate more output nodes on average)

Other clustering techniques work too (for example K-means on pre-trained word vectors).

How to assign words to classes?

- Combination of both ideas (similarity of frequency and semantics): good accuracy & speed

- We can learn the classes while penalizing frequent and infrequent words to be within the same class

Speed Regularization and Optimality in Word Classing
(Zweig & Makarychev, 2013)

Further scaling up: joint training of RNN with Maxent model

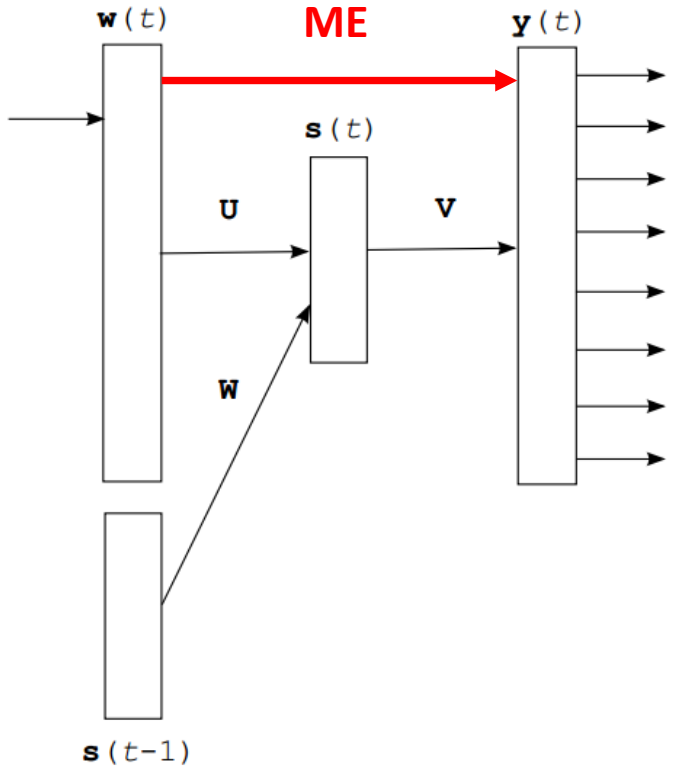
- Logistic regression and neural networks are closely related
- Maximum entropy model is how logistic regression is called in NLP

- Logistic regression: fast, scales to very large datasets
- Neural networks: more compact and robust, generalize better

- Why not combine both?

Joint training of RNN with Maxent model

- Just another matrix of weights in the RNN
- This corresponds to RNN with bigram ME:
- We can use n-gram features for ME



Strategies for training large scale neural network language models (Mikolov et al, 2011)

Joint training of RNN with Maxent model

Joint training allows:

- To use the fast, big sparse model (direct weights between inputs and outputs)
- The slow, dense part (hidden layer) can be much smaller

Joint training of RNN with Maxent model

Q: Why not train models separately and combine their predictions?

A: If neural network is trained jointly with the maxent model, it can learn just the complementary information.

By using the direct connections, we can reduce the hidden layer size greatly and still achieve good performance.

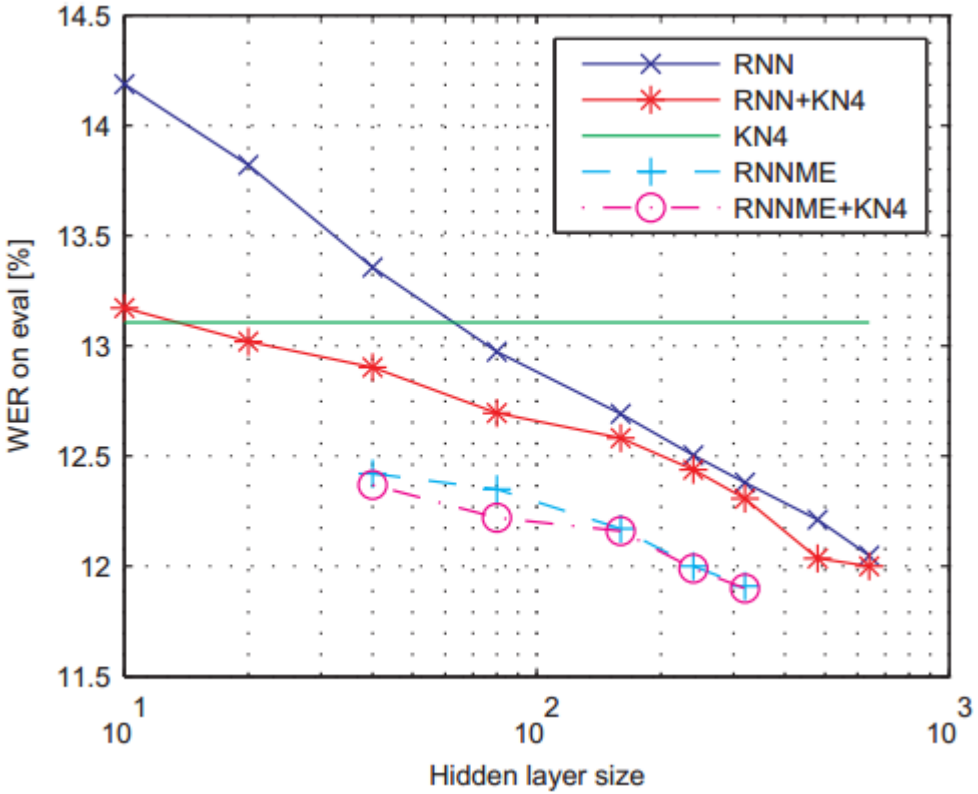
Wall Street Journal ASR task

Model	Perplexity		WER [%]	
	heldout	Eval 92	Eval 92	Eval 93
GT2	167	209	14.6	19.7
GT3	105	147	13.0	17.6
KN5	87	131	12.5	16.6
KN5 (no count cutoffs)	80	122	**12.0**	**16.6**
RNNME-640	59	89	9.6	14.4
combination of RNNME models	-	-	**9.15**	**13.11**

- Performance in simple ASR task: 21% - 24% reduction of Word Error Rate (WER) over good baseline

GT2 = Good-Turing smoothed 2-gram, KN5 = Kneser-Ney smoothed 5-gram

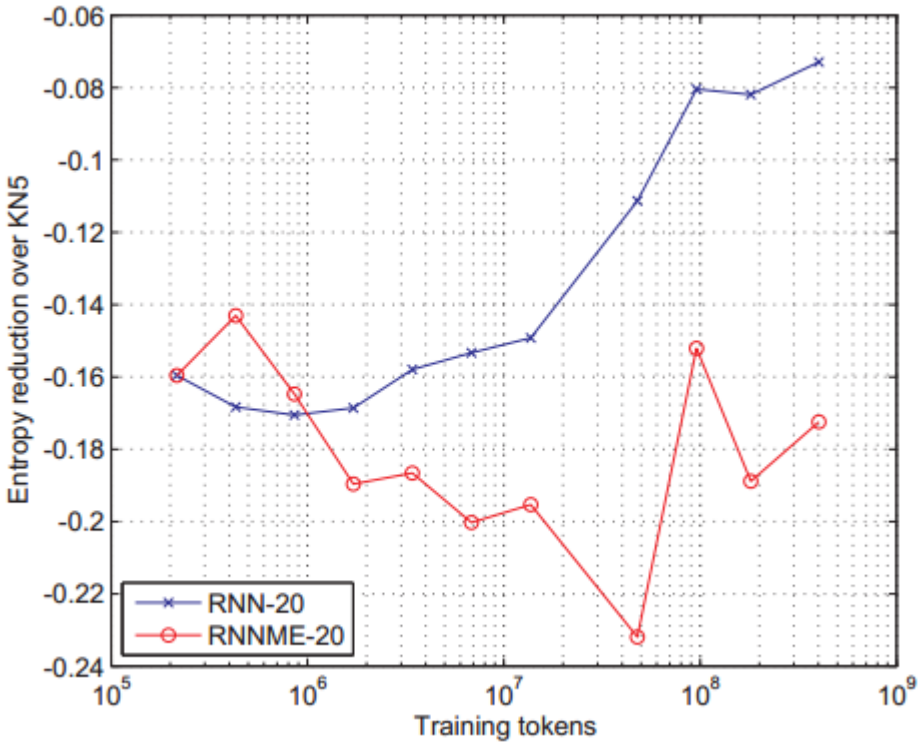
IBM RT04 speech recognition system



Model	WER[%]
KN4 (baseline)	13.11
model M	12.49
3xRNN	**11.70**

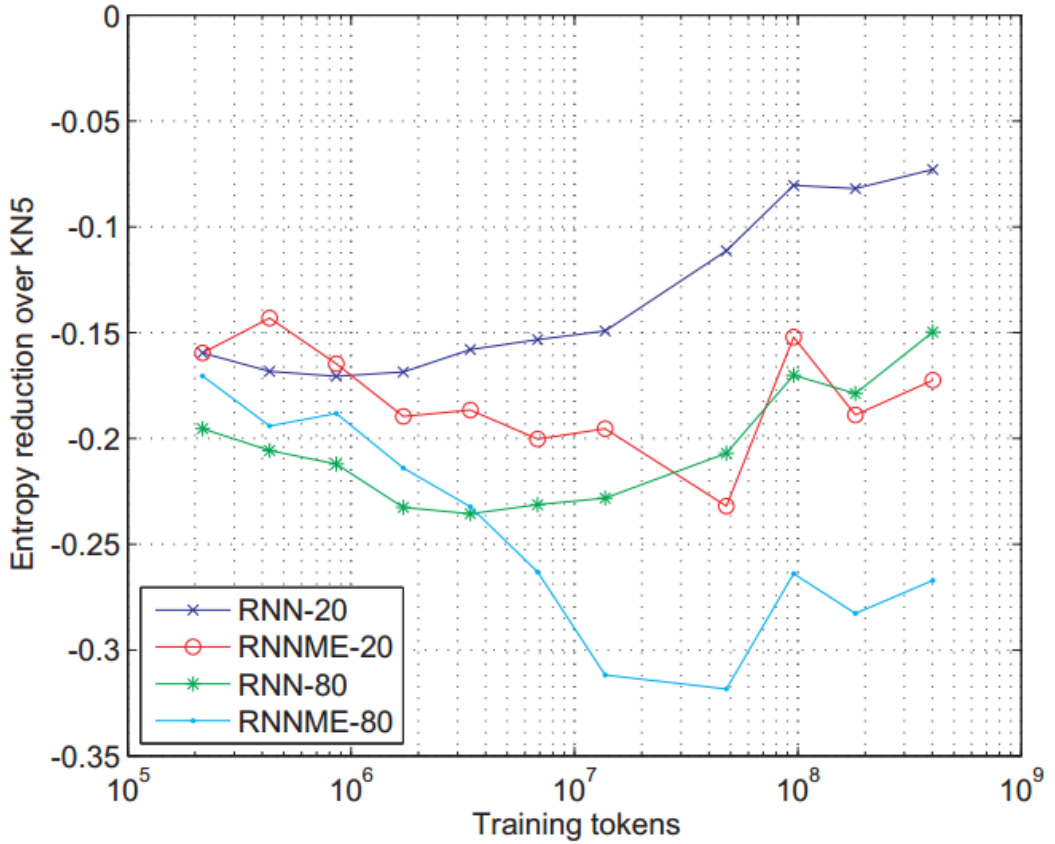
- Strong ASR system from IBM, “model M“ was the previous state-of-art LM
- Bigger models = better results, RNNME works the best

Analysis of results: RNN vs RNNME



The more training data there is, the more useful it is to train the model jointly with ME features (same computational cost)

Analysis of results: RNN vs RNNME



The same conclusion holds even for larger models: gains vanish with more data if the model does not have capacity to store patterns

Multi-threaded training of RNNME

- Another speedup ~10x, scales training to billions of words

Model	Training Time		Perplexity
	[hours]	[CPUs]	
Interpolated KN 5-gram, 1.1B n-grams (KN)	3	100	67.6
Recurrent NN-256 + MaxEnt 9-gram	60	24	58.3
Recurrent NN-512 + MaxEnt 9-gram	120	24	54.5
Recurrent NN-1024 + MaxEnt 9-gram	240	24	51.3

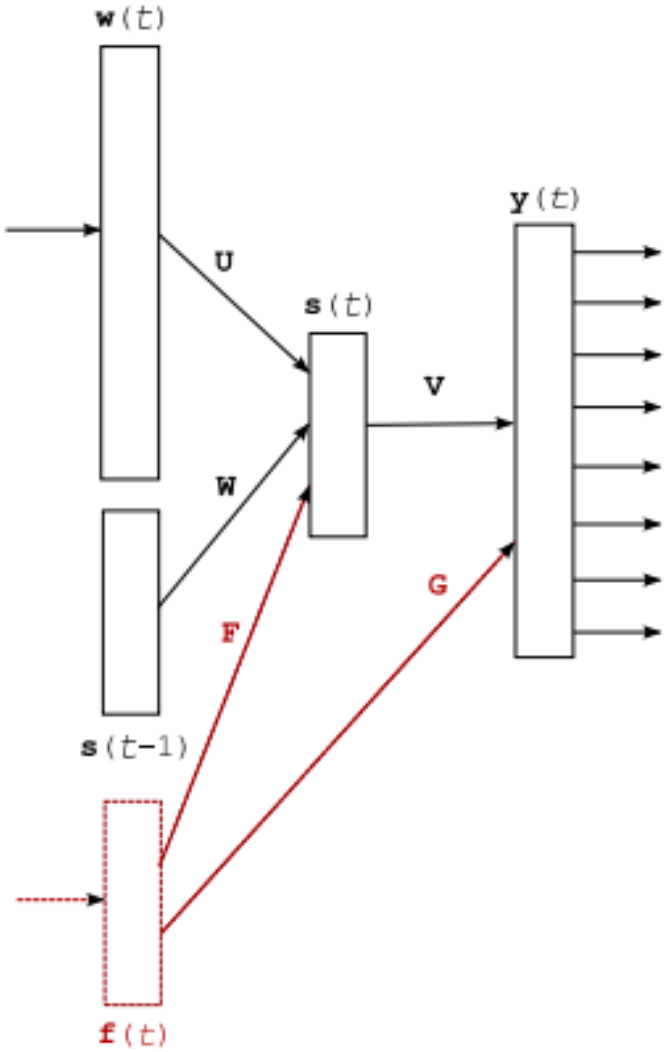
One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling (Chelba et al, 2014)

Summary: NN, RNN, RNNME

- RNN outperforms FNN on language modeling tasks, both are better than n-grams
- The question “are neural nets better than n-grams” is incomplete: the best solution is to use both
- Joint training of RNN and maxent with n-gram features works great on large datasets

Recurrent neural network with additional features

- We can use extra features (POS, external information, long context information, …) represented as additional inputs



Context dependent recurrent neural network language model (Mikolov & Zweig, 2012)

Recurrent neural network with slow features

- We can define the extra features f(t) as exponentially decaying sum of word vectors w(t):

$$f(t) = f(t-1)\gamma + w(t)(1-\gamma)$$

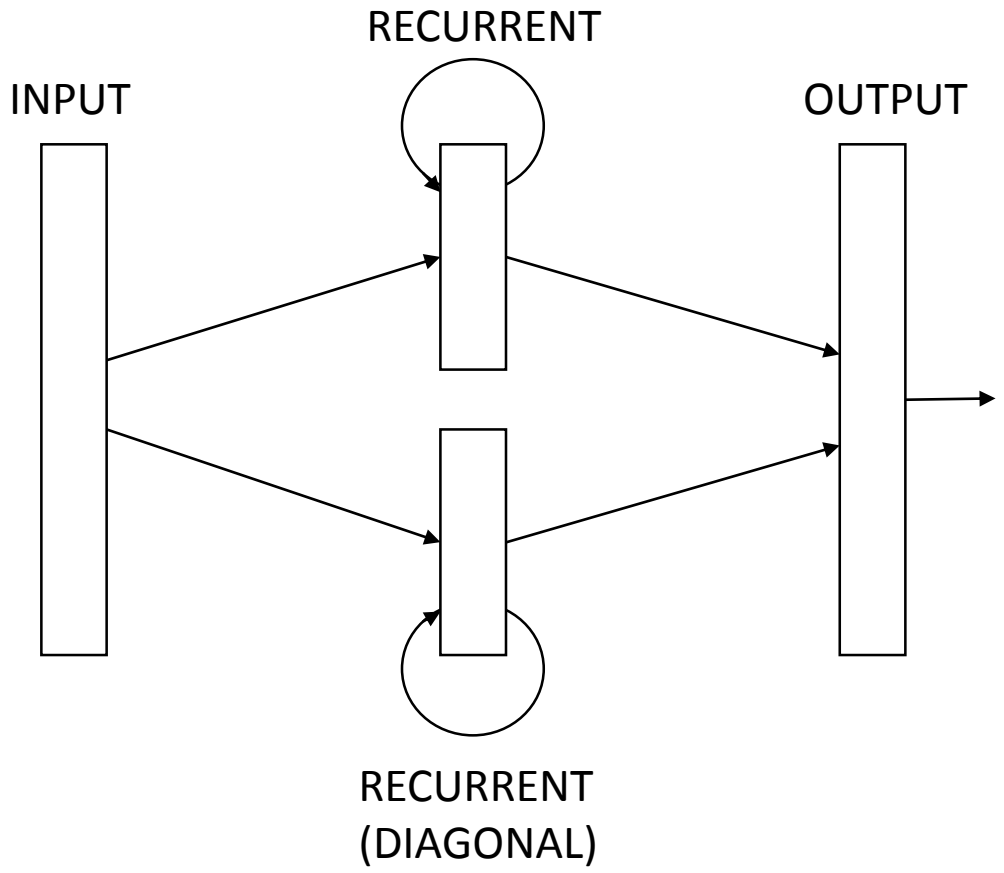
- This will give the model longer term memory

- Performance in language modeling similar to Long Short-Term Memory (LSTM) RNN – about 10% reduction of perplexity over simple RNN

LSTM-based LM: *LSTM Neural Networks for Language Modeling* (Sundermeyer et al, 2012)

Recurrent neural network with slow features

- 2 hidden layers
- One normal
- The other constrained (diagonal)
- The values in diagonal can be fixed (close to 1)



- Learns longer term patterns

Application to machine translation

- Straightforward: N-best list rescoring with NNLMs

- Search space in MT is huge, integration into decoding is a better idea

setting	dev	2004	2005	2006
baseline	38.2	38.4	37.7	34.3
reranking	38.5	38.6	37.8	34.7
decoding	39.1	39.5	38.8	34.9

Decoding with Large-Scale Neural Language Models Improves Translation (Vaswani et al, 2013)

Application to machine translation

- Instead of conditioning prediction of word just on the previous words within target sentence, we can use also the source sentence words

- We can use whole source sentence representation as additional input features

Modeling both translation and language model probability with NNLMs: *Joint Language and Translation Modeling with Recurrent Neural Networks* (Auli et al, 2013)

Application to machine translation

BOLT Test		
	Ar-En	
	BLEU	Gain
"Simple Hier." Baseline	33.8	-
Standard NNJM	40.2	+6.4
Self-Norm NNJM	40.1	+6.3
Pre-Computed NNJM	39.9	+6.1

- Decoding with Joint NN models (probably the biggest improvement in MT recently)
- Additional inputs represent words in the source sentence around position that is currently decoded

Fast and Robust Neural Network Joint Models for Statistical Machine Translation (Devlin et al, 2014)

Summary: Neural net Language Models

- NNLMs are currently the state-of-the-art in language modeling
- Considerable improvements in ASR, MT
- Significant ongoing efforts to scale training to very large datasets

Summary: Neural net Language Models

- Much of the recent success is based on basic principles: big data, big models, algorithmical efficiency, neural nets from the 80's
- Neural network models incrementally extend existing paradigms
- So far there is no big success story of deep learning in NLP: maybe we will have to do something novel to make it work?

Future research

- Current research culture
- How to avoid common mistakes
- Hints how to recognize good papers and ideas
- Promising future directions

Current research culture

- Thousands of NLP papers written each year, publish-or-perish
- Focus on incremental improvements
- The goal for NLP should be to develop machines that understand language: what do we need to change to make the progress faster?

Common problems in current research

Too much focus on state-of-the-art results:

- Not every idea can yield the best results immediately
- Leads to fragmentation of datasets
- Leads to “new names for the old tricks”
- Leads to untrustworthy claims in papers

Solution: start to accept papers with interesting ideas, do not consider high complexity as an advantage, but serious disadvantage

Common problems in current research

- Make the research more open

- It should be required for papers that claim to be state-of-the-art to publish script + code that can reproduce results

How to recognize good papers and ideas

- Reproducible results on standard datasets
- Published code (that actually compiles and works)
- Low complexity of ideas (if it can't be explained simply, probably it can be done better)

- However, it is hard to predict what ideas are going to lead to breakthroughs…

Promising future directions

- We should pick challenging problems that are really important by themselves

Examples:

- Language understanding by machines
- What computational models can compactly represent human languages
- Learning language from increasingly complex examples, and through communication

Future of AI research

Language understanding:

- Should aim to allow machines do what people can do
- Learn language through communication: intelligence is not about knowing answers to everything, but about ability to learn new concepts quickly

We need to revisit basic concepts in machine learning / NLP. Instead of doing just big data statistics, we need to develop new paradigms.

Resources

- Open-source neural-net based NLP software: RNNLM toolkit, word2vec and other tools
- Links to large text corpora, pre-trained models
- Benchmark datasets for advancing the state of the art

RNNLM toolkit

- Available at rnnlm.org
- Allows training of RNN and RNNME models
- Extensions are actively developed, for example multi-threaded version with hierarchical softmax: http://svn.code.sf.net/p/kaldi/code/trunk/tools/rnnlm-hs-0.1b/

Word2vec

- Available at https://code.google.com/p/word2vec/
- Tool for training the word vectors using CBOW and skip-gram architectures, supports both negative sampling and hierarchical softmax
- Optimized for very large datasets (>billions of training words)
- Includes links to models pre-trained on large datasets (100B words)

CSLM: Feedforward NNLM code

- Continuous Space Language Model toolkit: http://www-lium.univ-lemans.fr/cslm/

- Implementation of feedforward neural network language model by Holger Schwenk

Other neural net SW

- List available at http://deeplearning.net/software_links/

- Mostly focuses on general machine learning tools, not necessarily NLP

Large text corpora

Short list available at the word2vec project: https://code.google.com/p/word2vec/#Where_to_obtain_the_training_data

- Sources: Wikipedia dump, statmt.org, UMBC webbase corpus

- Altogether around 8 billion words can be downloaded for free

Benchmark datasets (LMs, word vectors)

- The Penn Treebank setup including the usual text normalization is part of the example archive at rnnlm.org
- WSJ setup (simple ASR experiments, includes N-best lists): http://www.fit.vutbr.cz/~imikolov/rnnlm/kaldi-wsj.tgz
- Datasets for measuring word / phrase similarity available at:

1. http://research.microsoft.com/en-us/um/people/gzweig/Pubs/myz_naacl13_test_set.tgz
2. https://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt
3. https://code.google.com/p/word2vec/source/browse/trunk/questions-phrases.txt

Final summary

- Distributed word representations >= word classes
- Neural nets >= logistic regression

- Neural networks are useful statistical tool, but not the final solution to AI by themselves
- Deep learning is an interesting research direction, but we need more research to understand how to learn complex patterns in language