Neural Networks for Natural Language Processing

Tomas Mikolov, Facebook
Brno University of Technology, 2017

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 many tasks:
  - search, ranking
  - spam detection
  - ads recommendation
  - email categorization
  - machine translation
  - speech recognition
  - …and many others

Overview

Artificial neural networks are applied to many language problems:
- Unsupervised learning of word representations: word2vec
- Supervised text classification: fastText
- Language modeling: RNNLM

Beyond artificial neural networks:
- Learning of complex patterns
- Incremental learning
- Virtual environments for building AI

Basic machine learning applied to NLP

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

- Logistic regression

- Neural networks can extend (and improve) the above techniques and representations

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:

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

- For a good model: P("this is a sentence") > P("sentence a is this") > P("dsfdsgdfgdasda")

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 using big text datasets:

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

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

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

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

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 for classification / language modeling tasks, 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)

Basic machine learning overview

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

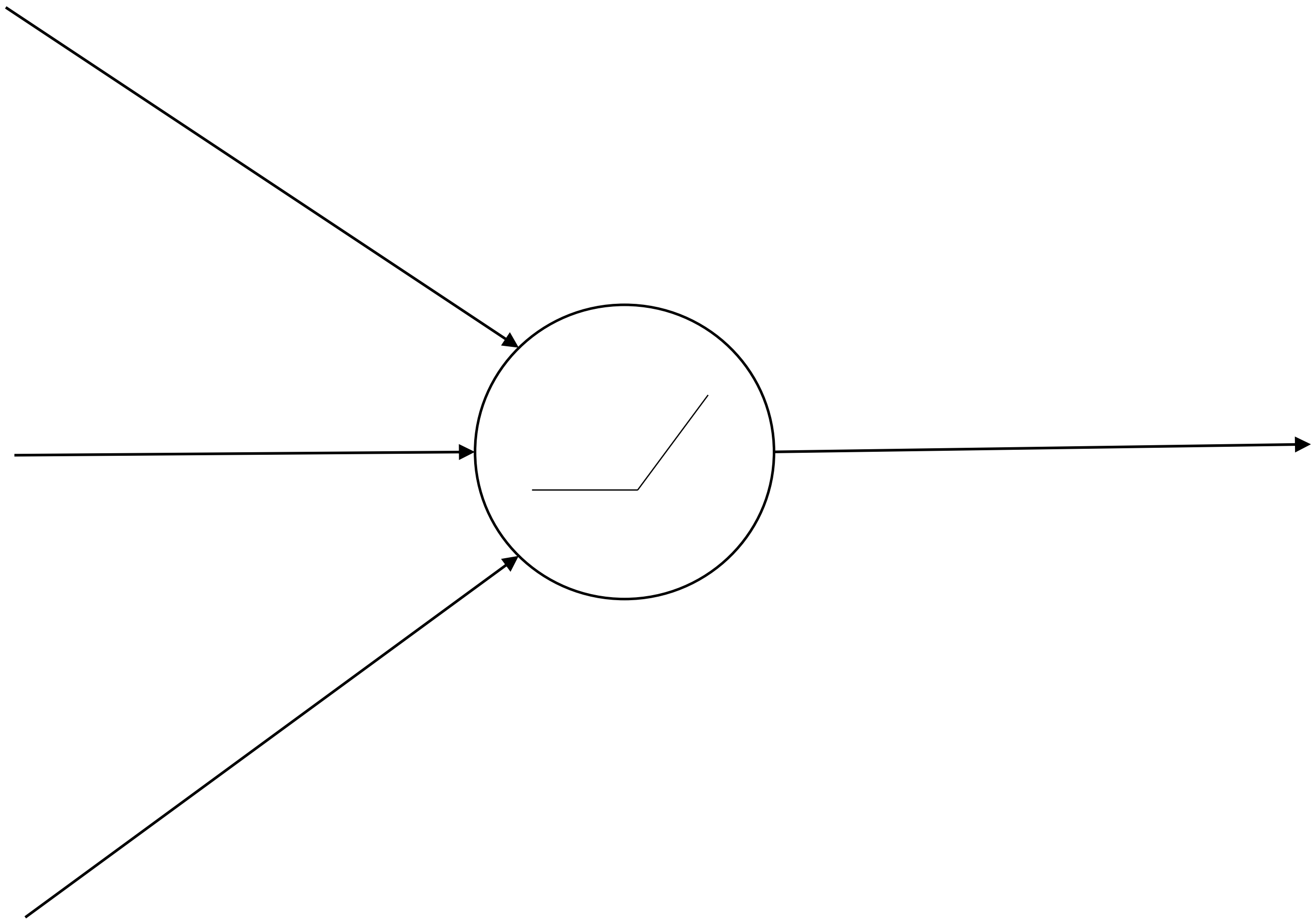
Quick intro to neural networks

- 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”

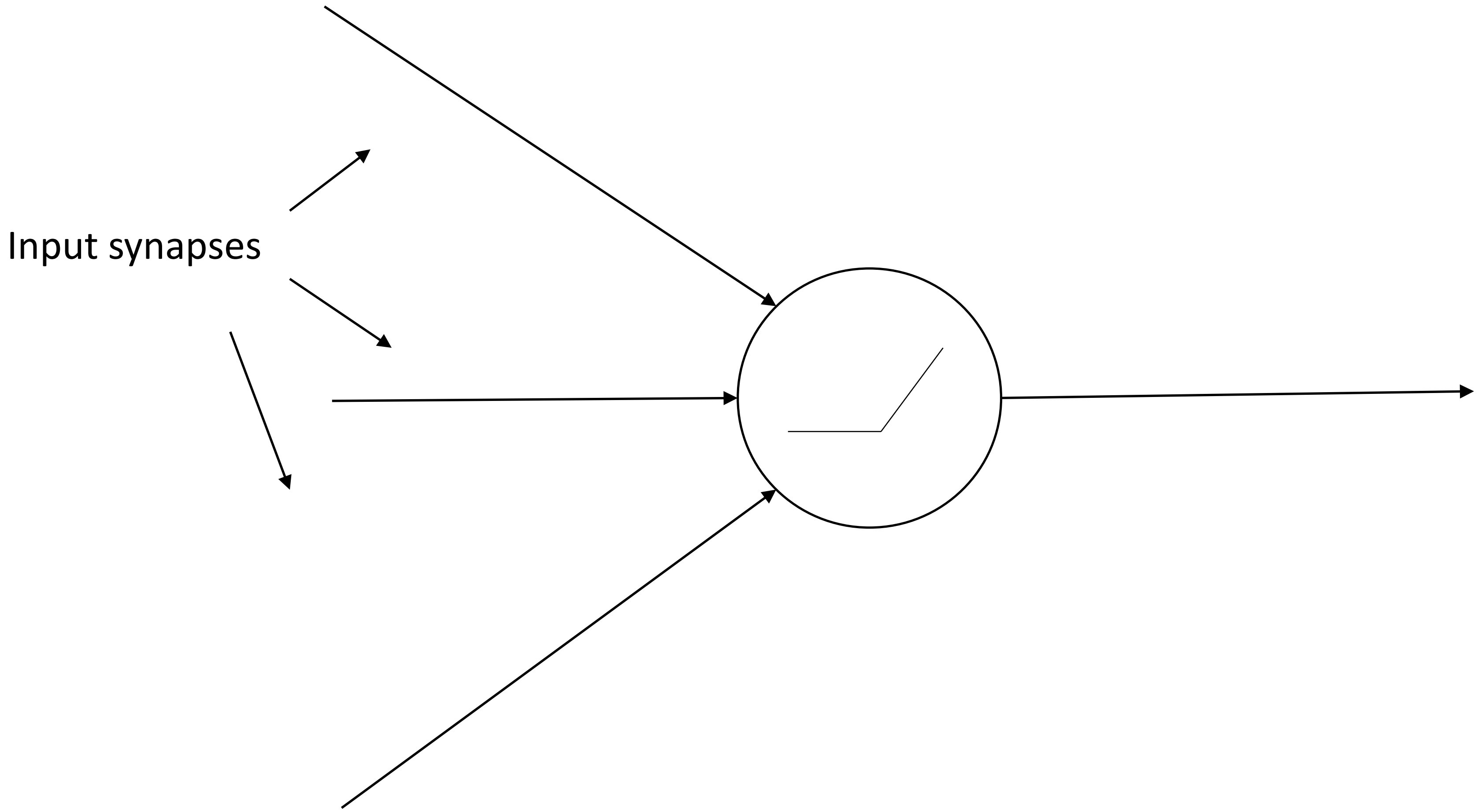
Neural networks in NLP: motivation

- The main motivation is to simply come up with more precise techniques than using plain counting
- 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!

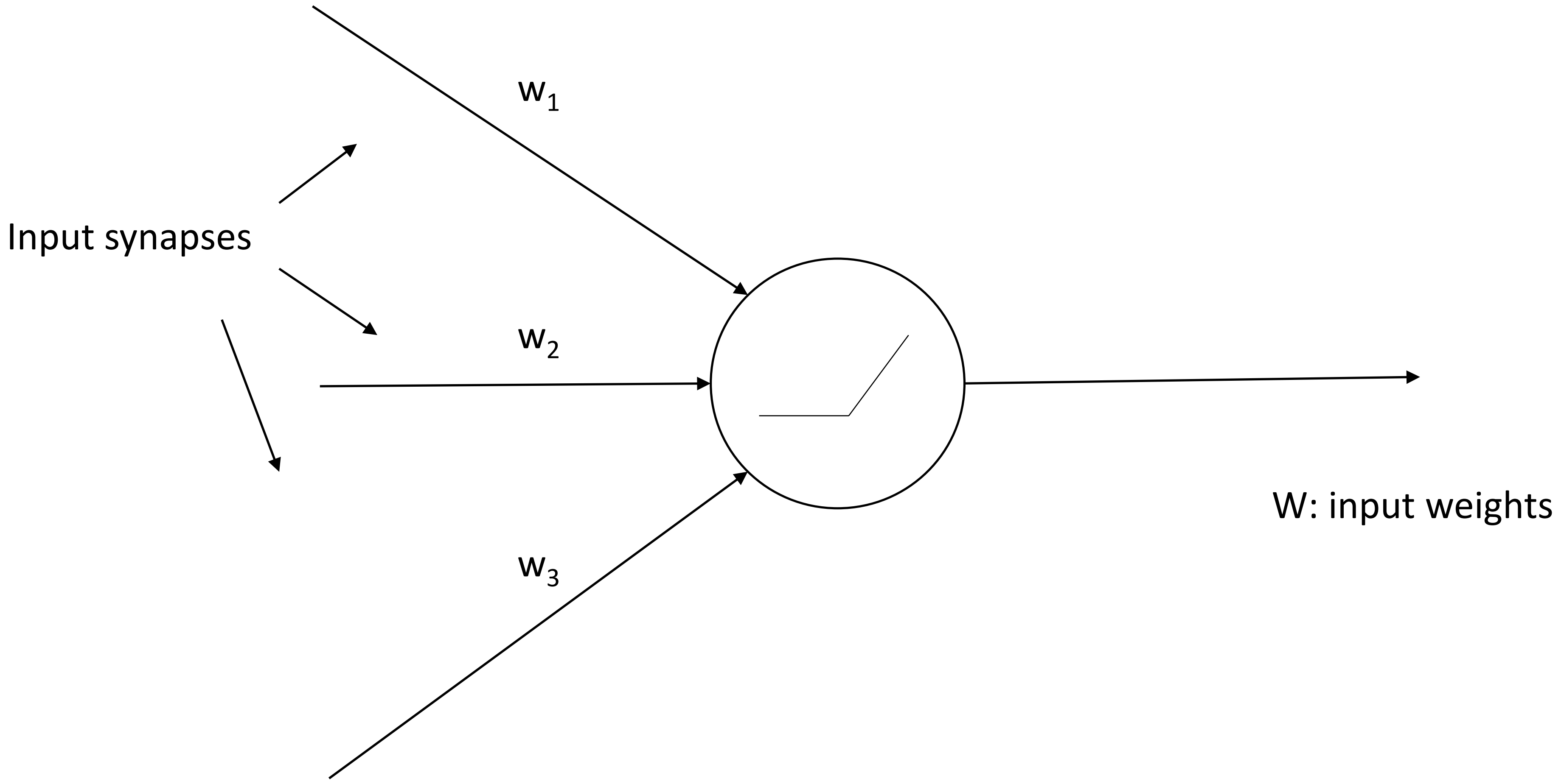
Neuron (perceptron)



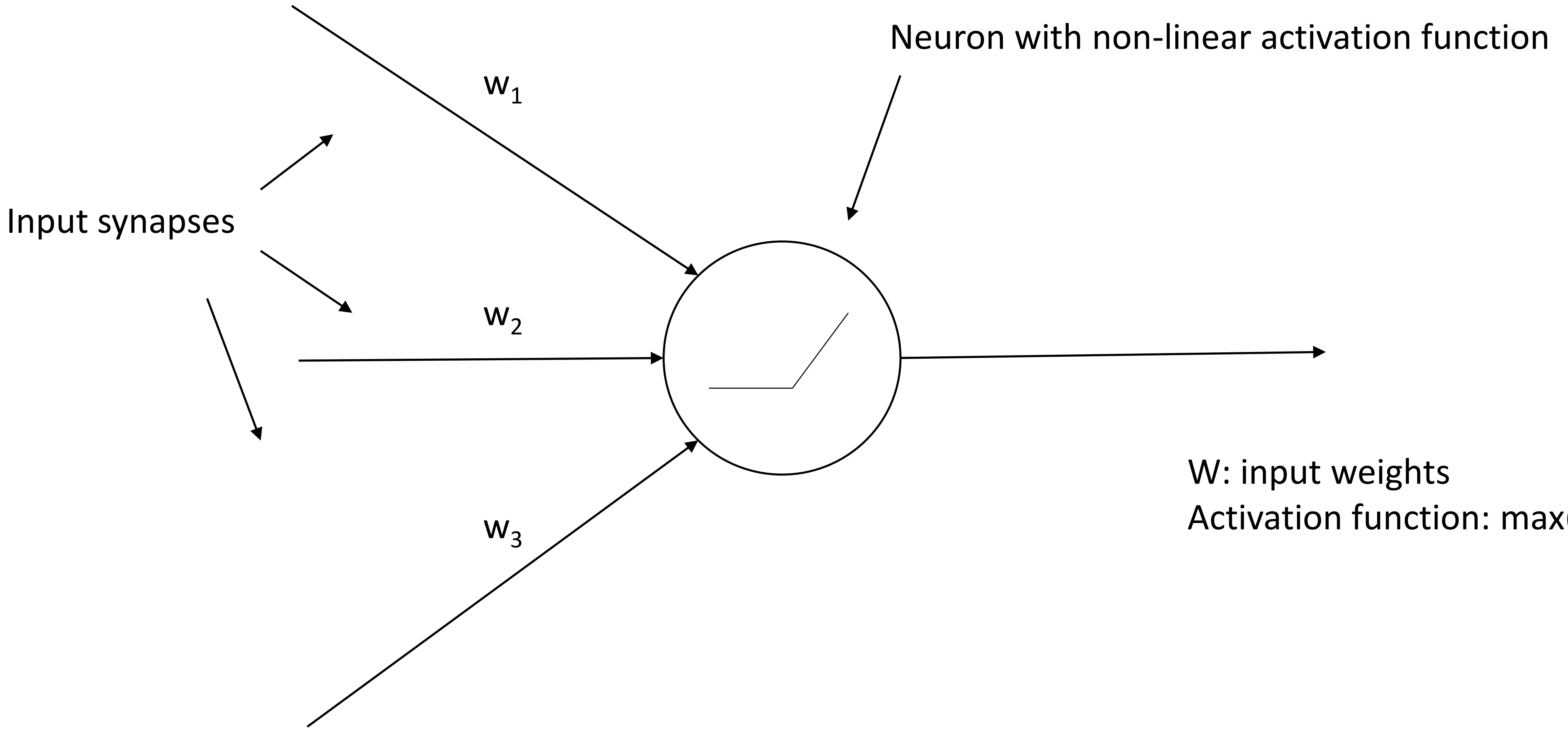
Neuron (perceptron)



Neuron (perceptron)

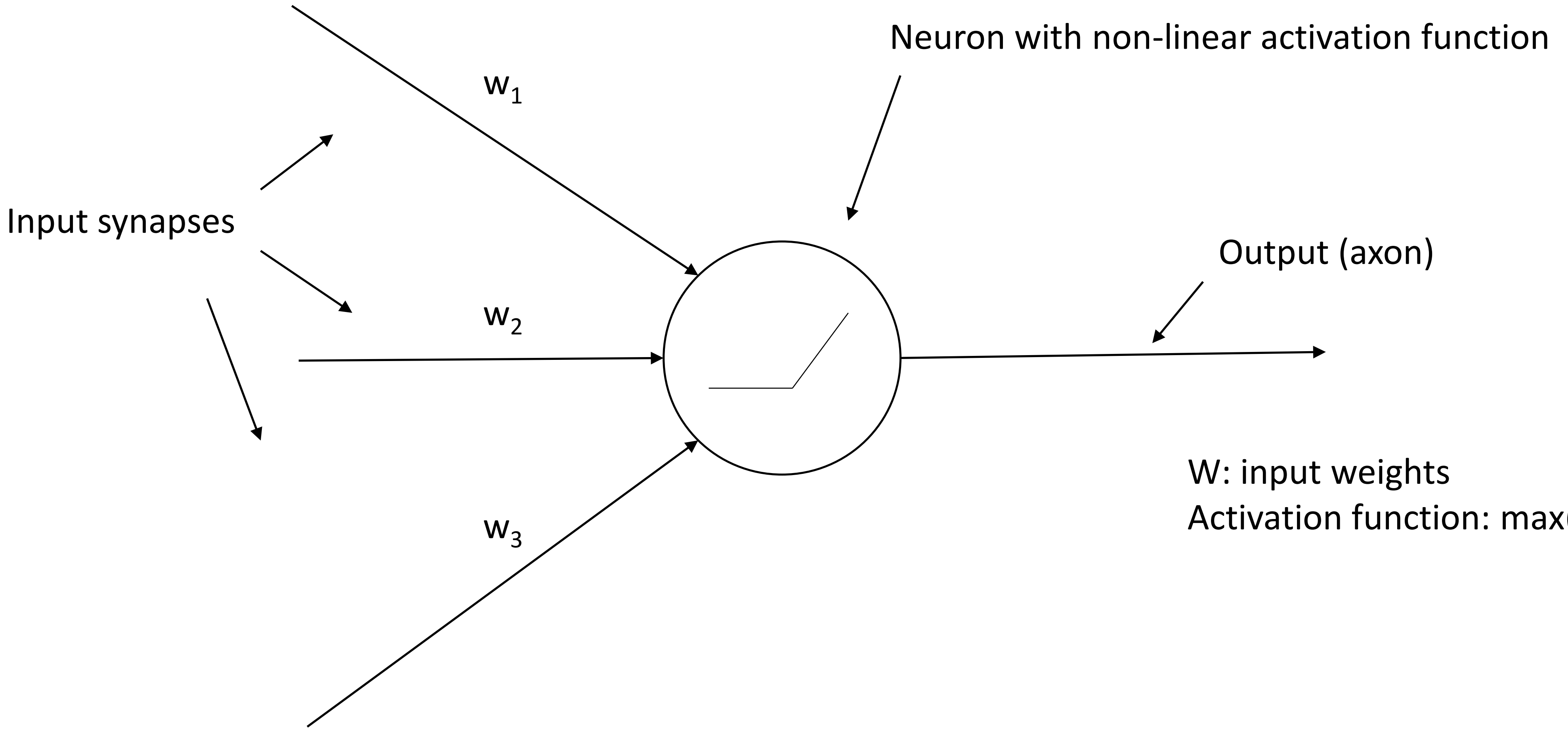


Neuron (perceptron)



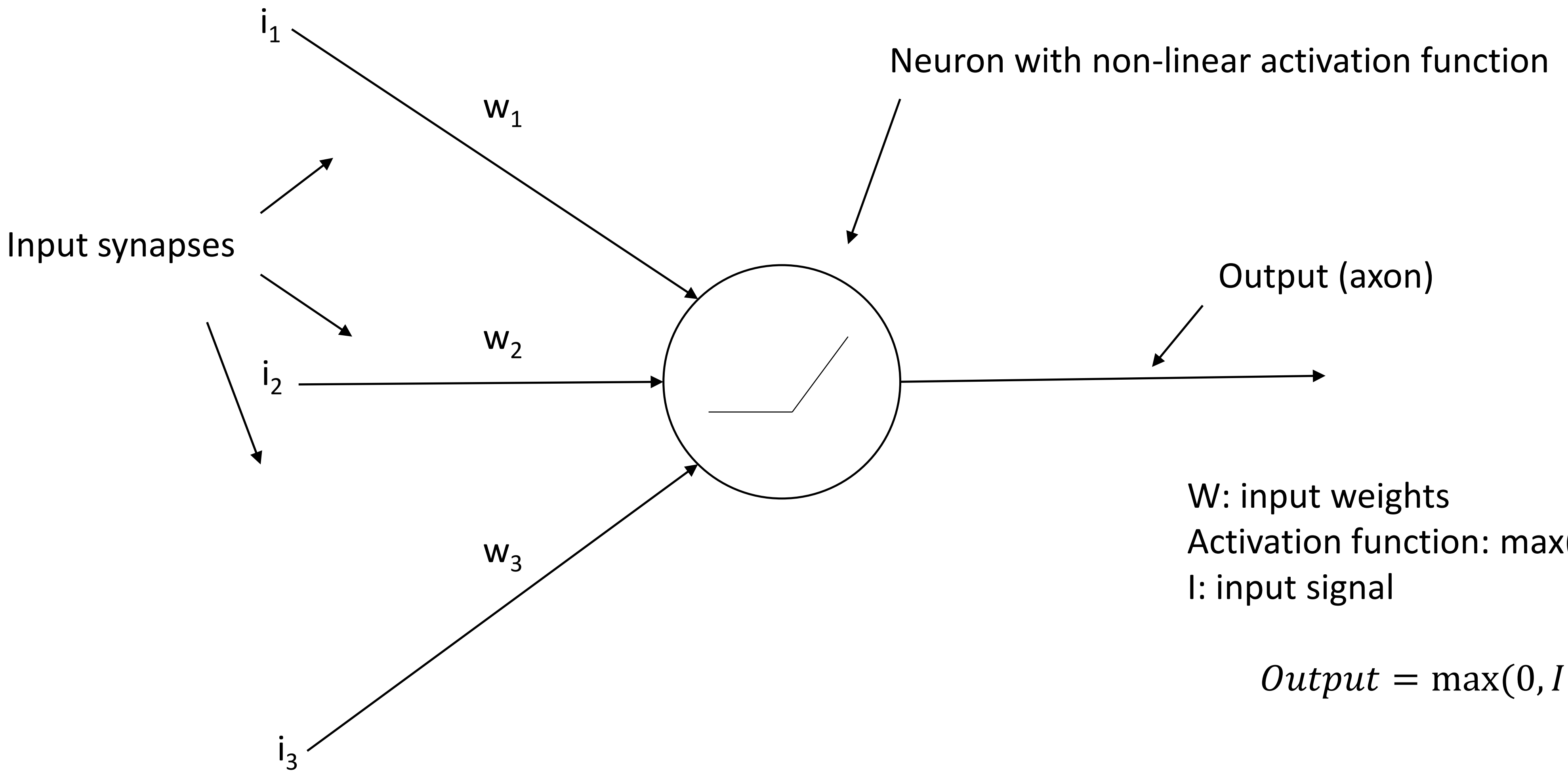
W: input weights
Activation function: max(0, value)

Neuron (perceptron)



W: input weights
Activation function: max(0, value)

Neuron (perceptron)



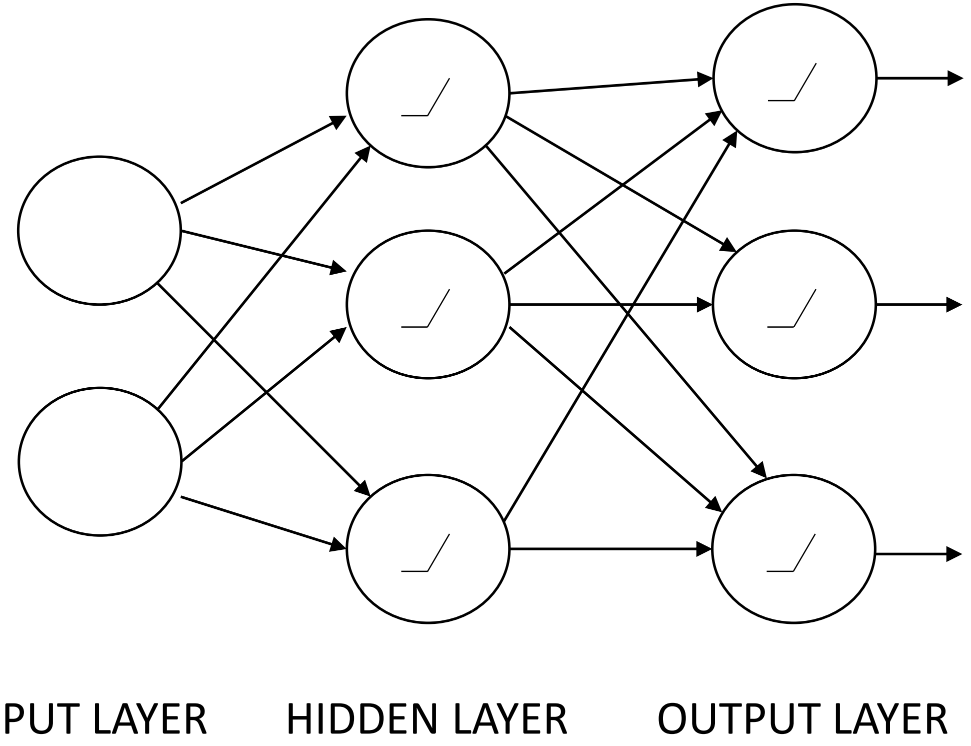
W: input weights
Activation function: max(0, value)
I: input signal

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

Neuron (perceptron)

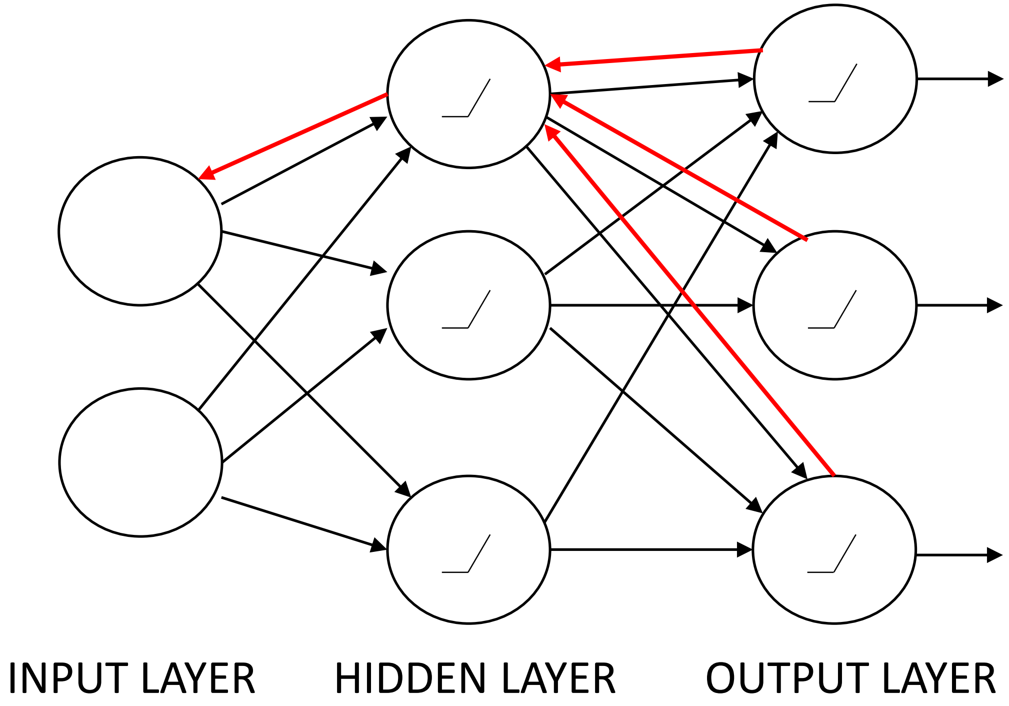
- It should be noted that the perceptron model is quite different from the biological neurons (those communicate by sending spike signals at various frequencies)
- The learning in brains seems also quite different
- It would be better to think of artificial neural networks as non-linear projections of data (and not as a model of brain)

Neural network layers

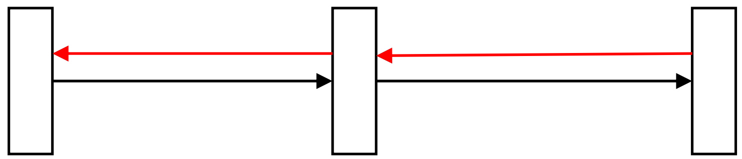


Training: Backpropagation

- 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:



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

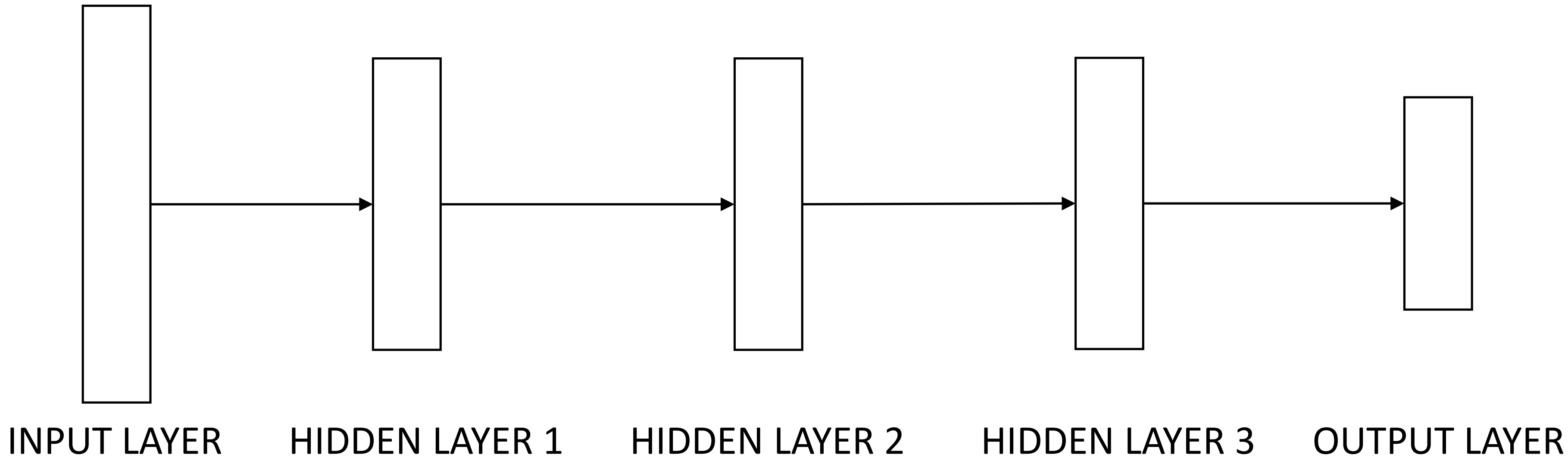
It may seem complicated at first, the best way to start is to re-use some existing setup and try your own modifications.

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
- 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

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



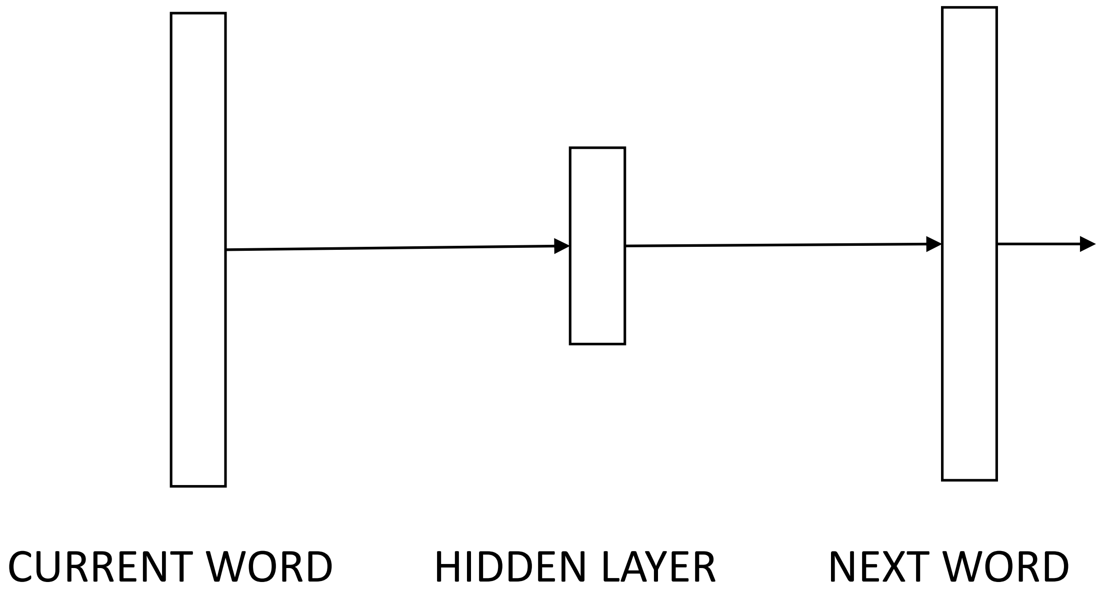
Deep learning

- 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

Distributed representations of words

- Vector representation of words computed using neural networks
- Linguistic regularities in the word vector space
- Word2vec

Basic neural network applied to NLP



- Bigram neural language model: predicts next word
- 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 layers)

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 similar properties to word classes (similar words have similar vector representations)

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

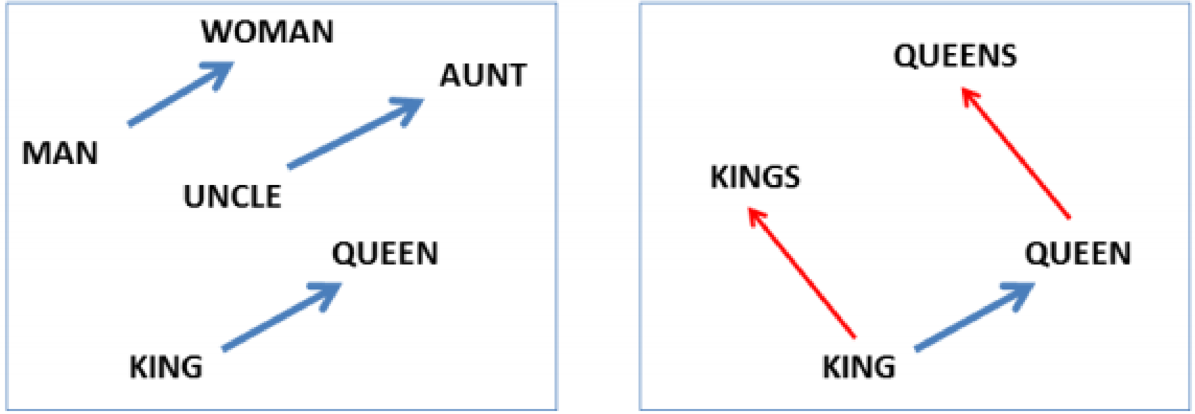
Word vectors

- Many neural architectures were proposed for training the word vectors, usually using several hidden layers

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

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

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: ___`

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

Word vectors – datasets for evaluation

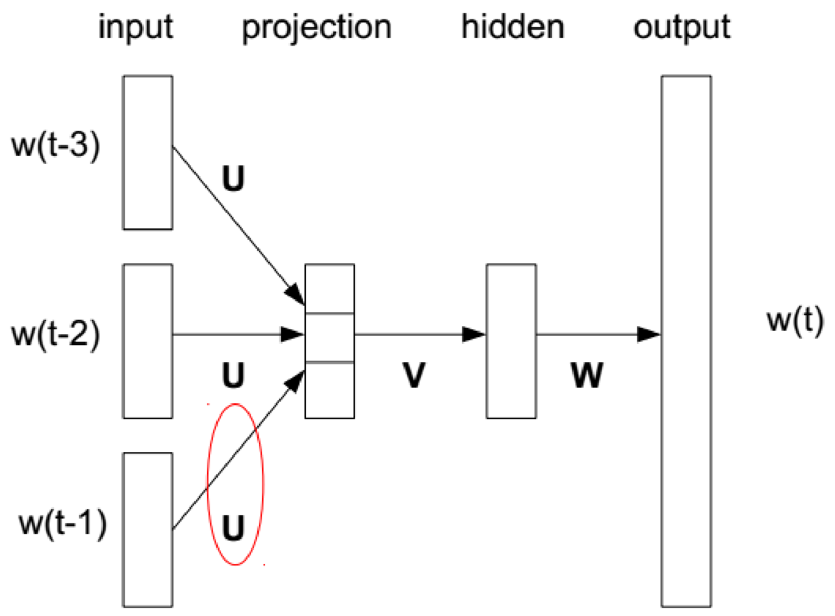
Phrase-based dataset, focuses on semantics:

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

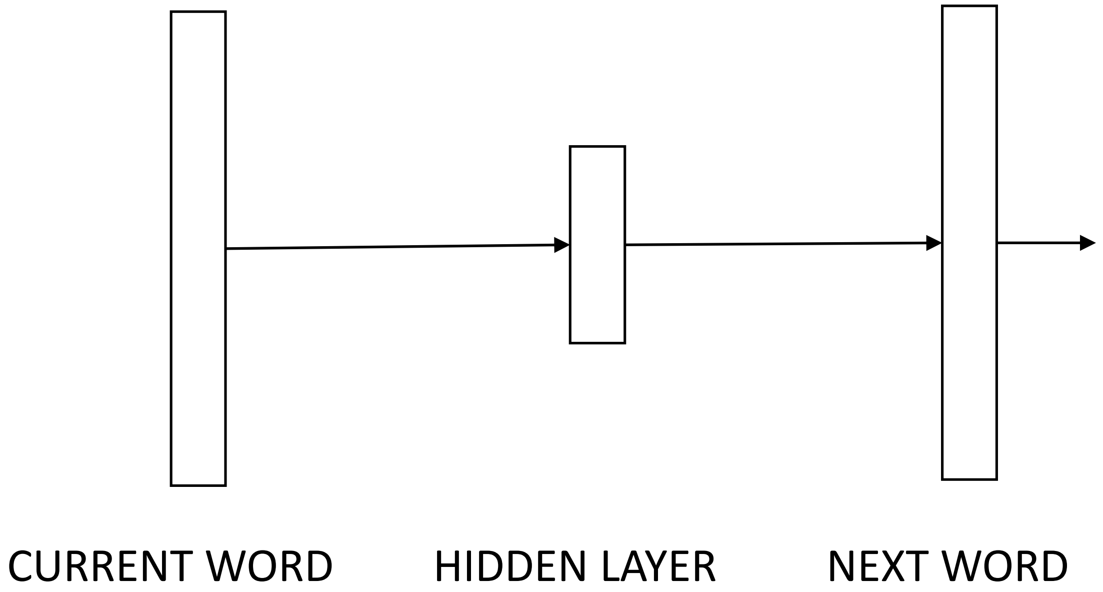
Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al, 2013)

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



Word vectors – various architectures



- We can extend the bigram NNLM for training the word vectors by adding more context without adding the hidden layer!

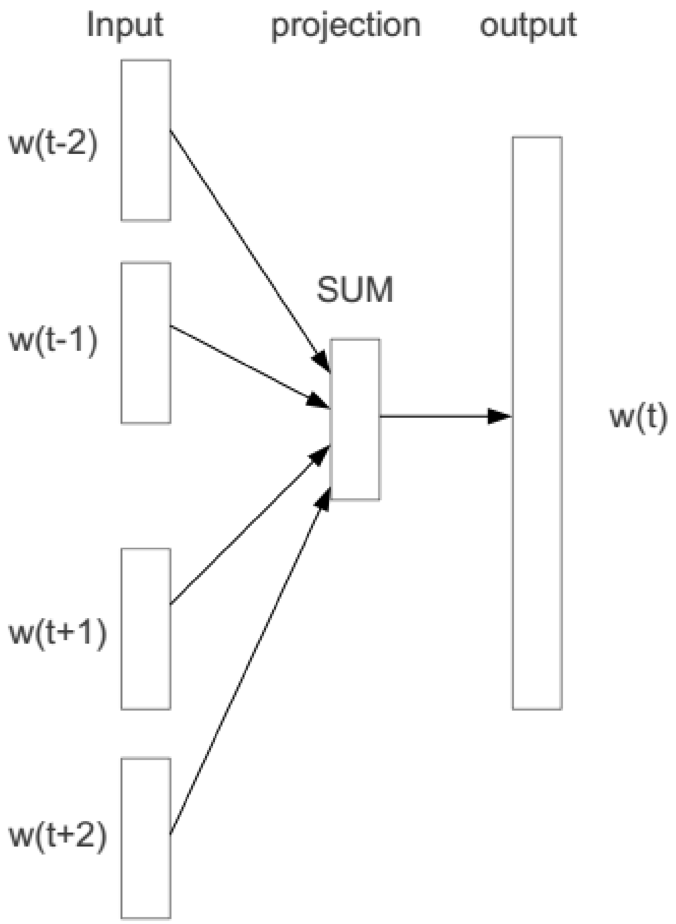
Word vectors – various architectures

- 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

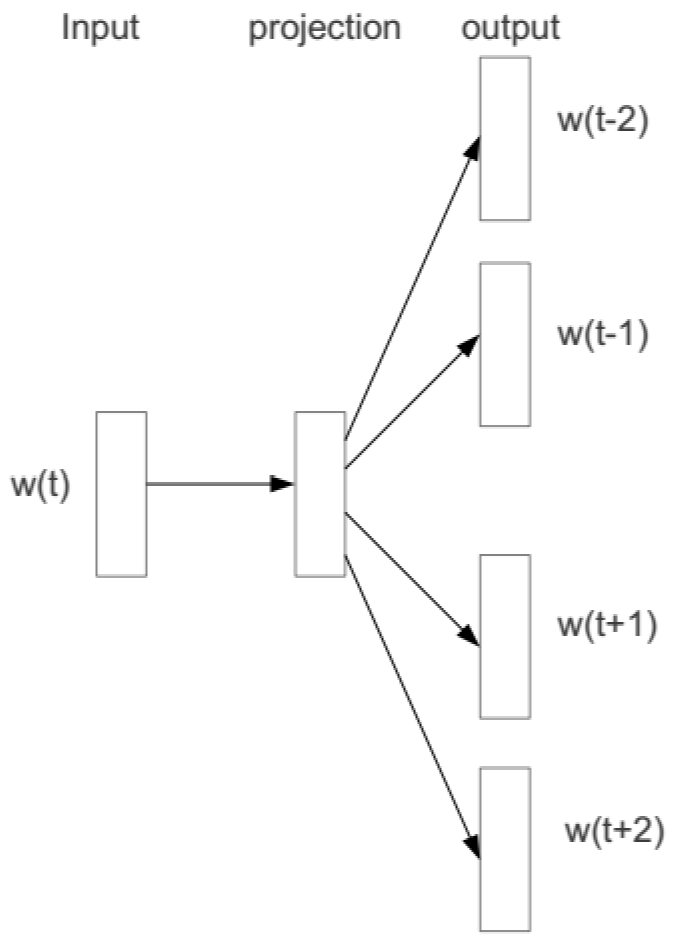
- Computationally much more efficient than n-gram NNLM of (Bengio, 2003)

- The hidden layer is just linear



Word vectors – various architectures

- Predict 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

- Stochastic gradient descent + backpropagation

- Efficient solution to very large softmax – 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 – sub-sampling

- It is useful to sub-sample the frequent words (such as ‘the’, ‘is’, ‘a’, …) during training
- Improves speed and even accuracy for some tasks

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

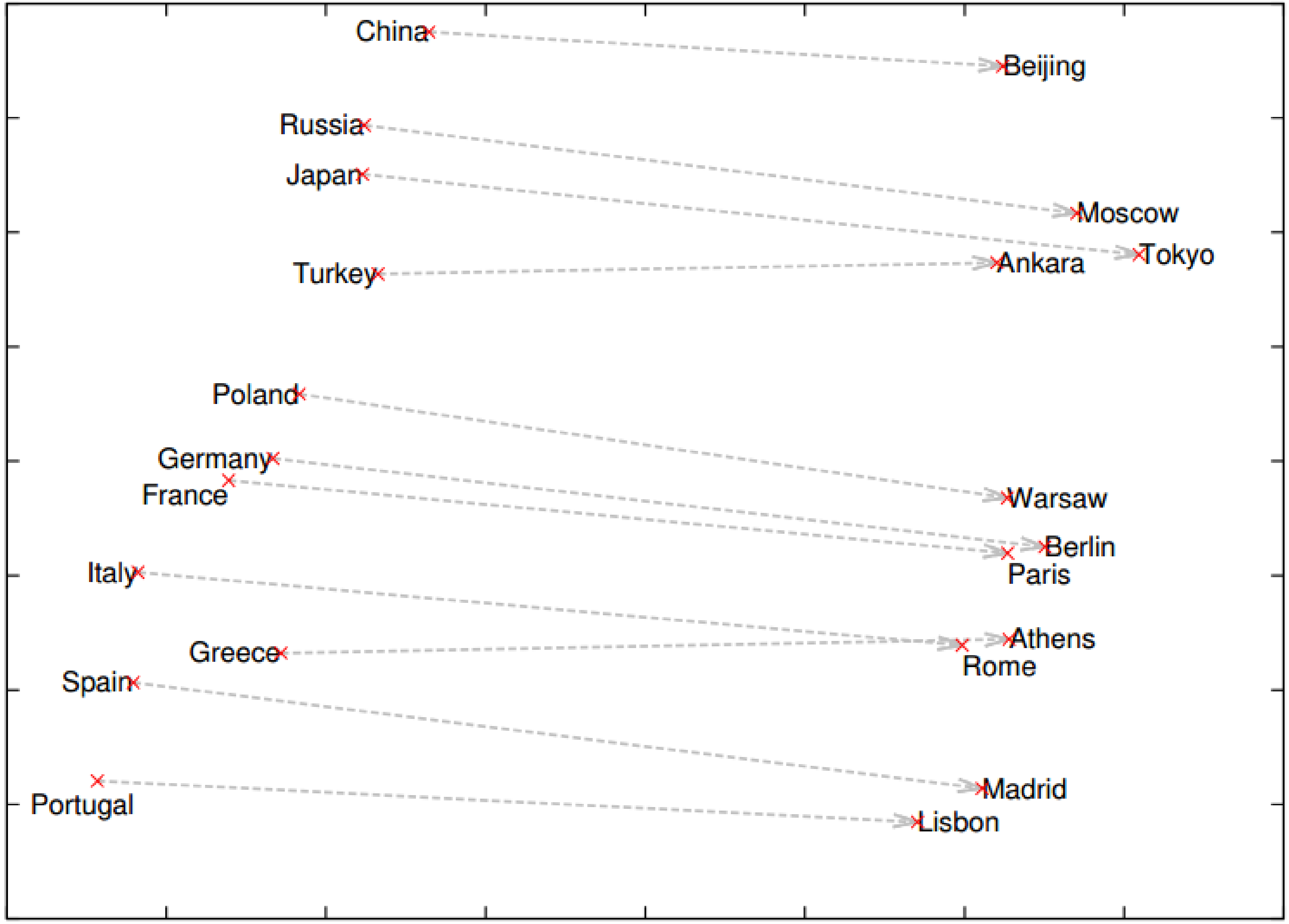
	Redmond	Havel	graffiti	capitulate
Collobert NNLM	conyers lubbock keene	plauen dzerzhinsky osterreich	cheesecake gossip dioramas	abdicate accede rearm
Turian NNLM	McCarthy Alston Cousins	Jewell Arzu Ovitz	gunfire emotion impunity	- - -
Mnih NNLM	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	anaesthetics 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



Distributed word representations: summary

- Simple models seem to be sufficient: no need for every neural net to be deep
- Large text corpora are crucial for good performance
- Adding supervised objective turns word2vec into very fast and scalable text classifier (‘**fastText**’):
  - Often more accurate than deep learning-based classifiers, and 100 000+ times faster to train on large datasets
  - https://github.com/facebookresearch/fastText

Recurrent Networks and Beyond

- Recent success of recurrent networks

- Explore limitations of recurrent networks

- Discuss what needs to be done to build machines that can understand language

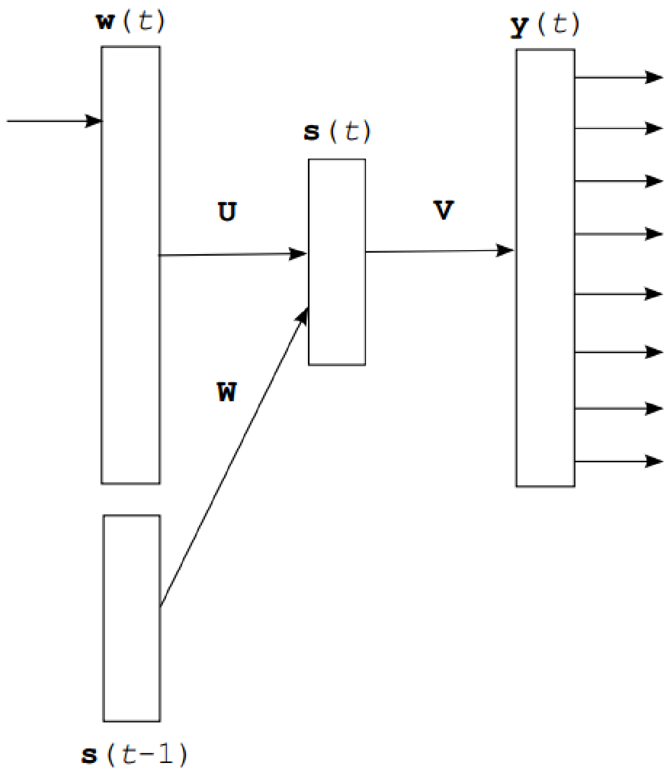
Brief History of Recurrent Nets – 80's & 90's

- Recurrent network architectures were very popular in the 80's and early 90's (Elman, Jordan, Mozer, Hopfield, Parallel Distributed Processing group, …)

- The main idea is very attractive: to re-use parameters and computation (usually over time)

Simple RNN Architecture

- Input layer, hidden layer with recurrent connections, and the output layer
- In theory, the hidden layer can learn to represent unlimited memory
- Also called Elman network (*Finding structure in time*, Elman 1990)



Brief History of Recurrent Nets – 90's - 2010

- After the initial excitement, recurrent nets vanished from the mainstream research

- Despite being theoretically powerful models, RNNs were mostly considered as unstable to be trained

- Some success was achieved at IDSIA with the *Long Short Term Memory RNN* architecture, but this model was too complex for others to reproduce easily

Brief History of Recurrent Nets – 2010 - today

- In 2010, it was shown that RNNs can significantly improve state-of-the-art in language modeling, machine translation, data compression and speech recognition (including strong commercial speech recognizer from IBM)

- RNNLM toolkit was published to allow researchers to reproduce the results and extend the techniques (used at Microsoft Research, Google, IBM, Facebook, Yandex, …)

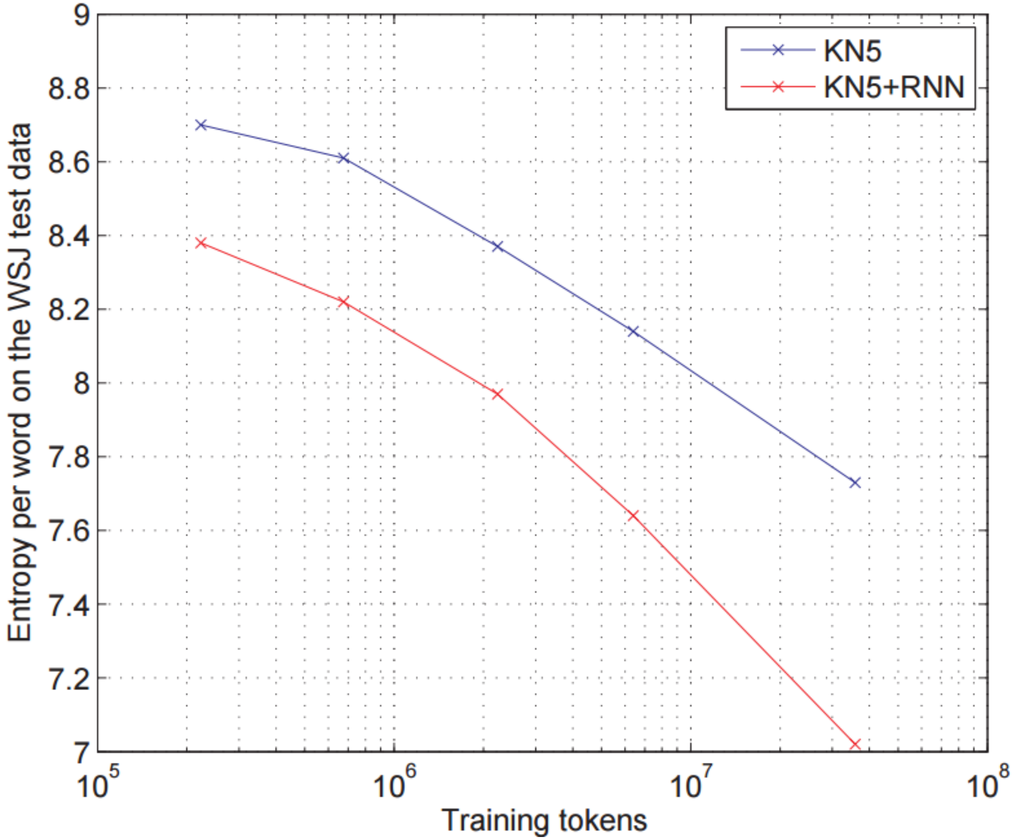
- The key novel trick in RNNLM was trivial: to clip gradients to prevent instability of training

Brief History of RNNLMs – 2010 - today

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-0	90	129	12.4	17.3
RNNME-10	81	116	11.9	16.3
RNNME-80	70	100	10.4	14.9
RNNME-160	65	95	10.2	14.5
RNNME-320	62	93	9.8	14.2
RNNME-480	59	90	10.2	13.7
RNNME-640	59	89	9.6	14.4
combination of RNNME models	-	-	9.24	13.23
+ unsupervised adaptation	-	-	**9.15**	**13.11**

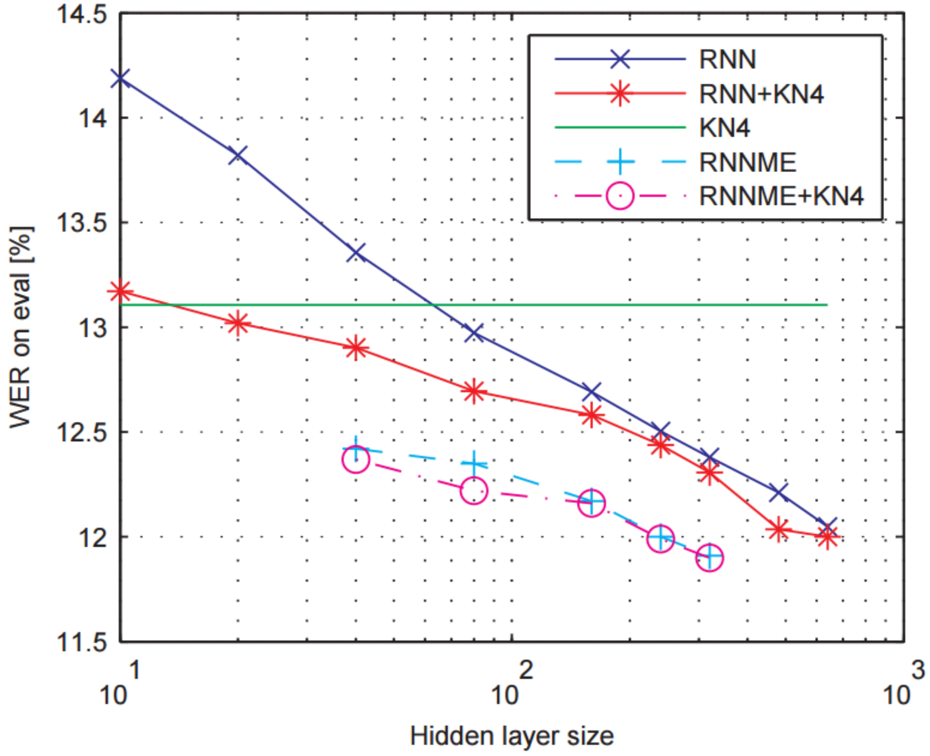
- 21% - 24% reduction of WER on Wall Street Journal setup

Brief History of RNNLMs – 2010 - today



- Improvement from RNNLM over n-gram **increases** with more data!

Brief History of RNNLMs – 2010 - today



- Breakthrough result in 2011: 11% WER reduction over large system from IBM
- Ensemble of big RNNLM models trained on a lot of data

Brief History of RNNLMs – 2010 - today

- RNNs became much more accessible through open-source implementations in general ML toolkits:
  - Theano
  - Torch
  - TensorFlow
  - …

- Training on GPUs allowed further scaling up (billions of words, thousands of hidden neurons)

Recurrent Nets Today

- Widely applied:
  - ASR (both acoustic and language models)
  - MT (language & translation & alignment models, joint models)
  - Many NLP applications
  - Video modeling, handwriting recognition, user intent prediction, …

- Downside: for many problems RNNs are too powerful, models are becoming unnecessarily complex

- Often, complex RNN architectures are preferred because of wrong reasons (easier to get a paper published and attract attention)

Beyond Deep Learning

- Going beyond: what RNNs and deep networks cannot model efficiently?

- Surprisingly simple patterns! For example, memorization of variable-length sequence of symbols

Beyond Deep Learning: Algorithmic Patterns

- Many complex patterns have short, finite description length in natural language (or in any Turing-complete computational system)

- We call such patterns *Algorithmic patterns*

- Examples of algorithmic patterns: $a^n b^n$, sequence memorization, addition of numbers learned from examples

- These patterns often cannot be learned with standard deep learning techniques

Beyond Deep Learning: Algorithmic Patterns

- Among the myriad of complex tasks that are currently not solvable, which ones should we focus on?

- We need to set ambitious end goal, and define a roadmap how to achieve it step-by-step

A Roadmap towards Machine Intelligence

Tomas Mikolov, Armand Joulin and Marco Baroni

Ultimate Goal for Communication-based AI

Can do almost anything:

- Machine that helps students to understand homeworks
- Help researchers to find relevant information
- Write programs
- Help scientists in tasks that are currently too demanding (would require hundreds of years of work to solve)

The Roadmap

- We describe a minimal set of components we think the intelligent machine will consist of
- Then, an approach to construct the machine
- And the requirements for the machine to be scalable

Components of Intelligent machines

- Ability to communicate
- Motivation component
- Learning skills (further requires long-term memory), ie. ability to modify itself to adapt to new problems

Components of Framework

To build and develop intelligent machines, we need:

- An environment that can teach the machine basic communication skills and learning strategies
- Communication channels
- Rewards
- Incremental structure

The need for new tasks: simulated environment

- There is no existing dataset known to us that would allow to teach the machine communication skills

- Careful design of the tasks, including how quickly the complexity is growing, seems essential for success:
  - If we add complexity too quickly, even correctly implemented intelligent machine can fail to learn
  - By adding complexity too slowly, we may miss the final goals

High-level description of the environment

Simulated environment:

- Learner
- Teacher
- Rewards

Scaling up:

- More complex tasks, less examples, less supervision
- Communication with real humans
- Real input signals (internet)

Simulated environment - agents

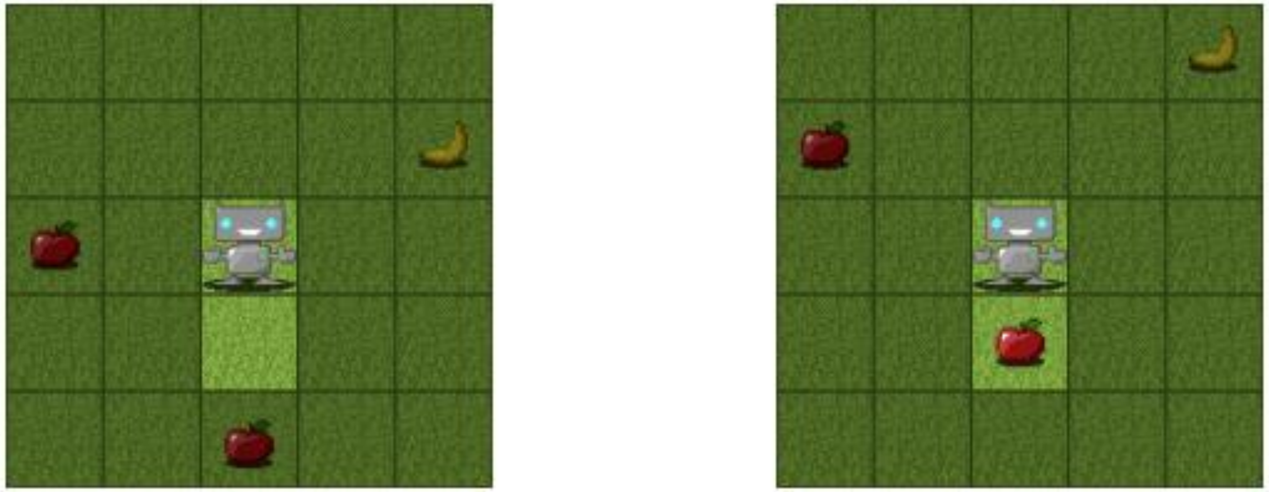
- Environment: simple script-based reactive agent that produces signals for the learner, represents the world
- Learner: the intelligent machine which receives input signal, reward signal and produces output signal to maximize average incoming reward
- Teacher: specifies tasks for Learner, first based on scripts, later to be replaced by human users

Simulated environment - communication

- Both Teacher and Environment write to Learner's input channel
- Learner's output channel influences its behavior in the Environment, and can be used for communication with the Teacher
- Rewards are also part of the IO channels

Visualization for better understanding

- Example of input / output streams and visualization:



Input:	*Output:*
T: move and look.	
	@E: I move.

Input:	*Output:*
E: you moved.	
	@E: I look.
E: there is an apple.	
R: 1.	

How to scale up: fast learners

- It is essential to develop fast learner: we can easily build a machine today that will “solve” simple tasks in the simulated world using a myriad of trials, but this will not scale to complex problems

- In general, showing the Learner new type of behavior and guiding it through few tasks should be enough for it to generalize to similar tasks later

- There should be less and less need for direct supervision through rewards

How to scale up: adding humans

- Learner capable of fast learning can start communicating with human experts (us) who will teach it novel behavior

- Later, a pre-trained Learner with basic communication skills can be used by human non-experts

How to scale up: adding real world

- Learner can gain access to internet through its IO channels
- This can be done by teaching the Learner how to form a query in its output stream

The need for new techniques

Certain trivial patterns are nowadays hard to learn:

- $a^n b^n$ context free language is out-of-scope of standard RNNs
- Sequence memorization breaks LSTM RNNs

- We show this in a recent paper *Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets*

Scalability

To hope the machine can scale to more complex problems, we need:

- Long-term memory
- (Turing-) Complete and efficient computational model
- Incremental, compositional learning
- Fast learning from small number of examples
- Decreasing amount of supervision through rewards

- Further discussed in: *A Roadmap towards Machine Intelligence http://arxiv.org/abs/1511.08130*

Some steps forward: Stack RNNs (Joulin & Mikolov, 2015)

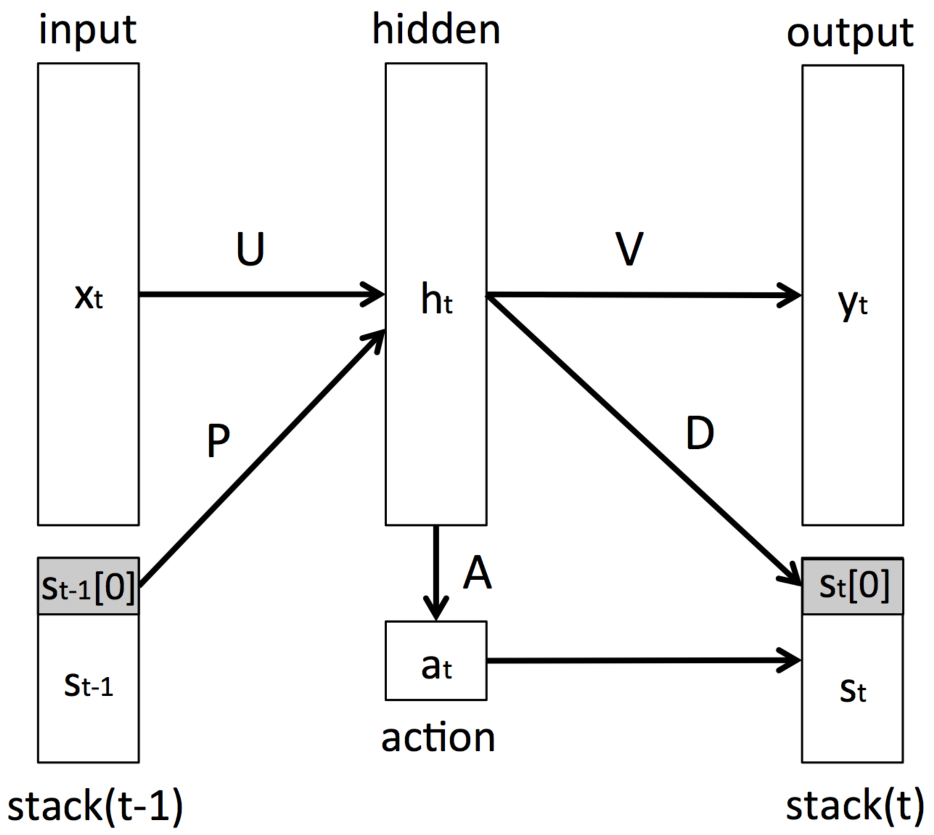
- Simple RNN extended with a long term memory module that the neural net learns to control

- The idea itself is very old (from 80's – 90's)

- Our version is very simple and learns patterns with complexity far exceeding what was shown before (though still very toyish): much less supervision, scales to more complex tasks

Stack RNN

- Learns algorithms from examples
- Add structured memory to RNN:
  - Trainable [read/write]
  - Unbounded
- Actions: PUSH / POP / NO-OP
- Examples of memory structures: stacks, lists, queues, tapes, grids, …



Algorithmic Patterns

Sequence generator	Example
$\{a^n b^n \mid n > 0\}$	aab**ba**aab**bba**b**a**aaaab**bbbb**
$\{a^n b^n c^n \mid n > 0\}$	aaab**bbccca**b**ca**aaaab**bbbbccccc**
$\{a^n b^n c^n d^n \mid n > 0\}$	aab**bccdda**aab**bbcccddda**b**cd**
$\{a^n b^{2n} \mid n > 0\}$	aab**bbba**aab**bbbbba**b**b**
$\{a^n b^m c^{n+m} \mid n, m > 0\}$	aabc**cca**aabbc**cccca**bc**c**
$n \in [1, k],\ X \rightarrow nXn,\ X \rightarrow =$	$(k = 2)$ 12=**21**2122=**2212**11121=**12111**

- Examples of simple algorithmic patterns generated by short programs (grammars)

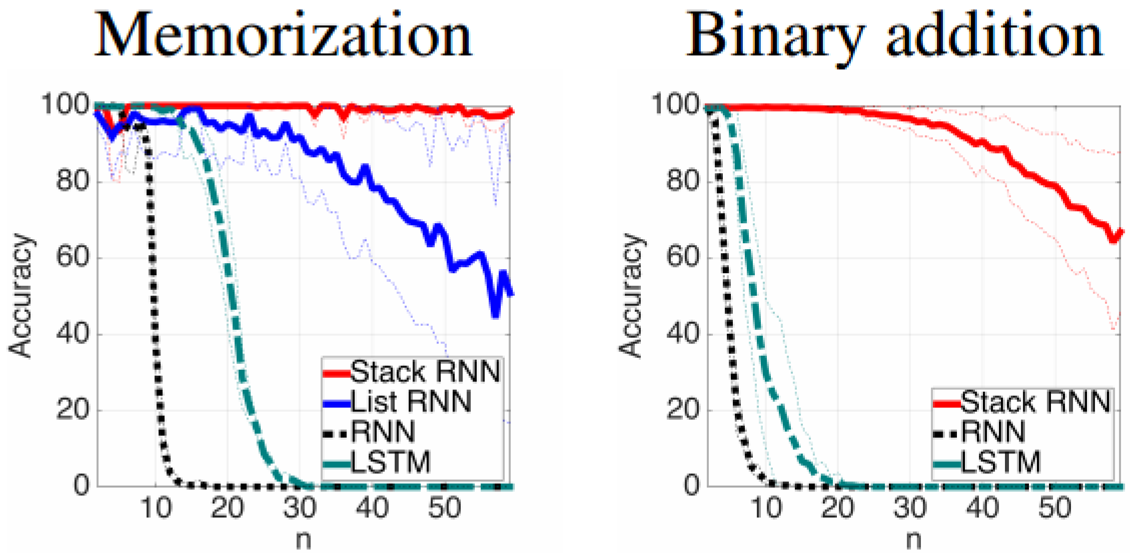
- The goal is to learn these patterns **unsupervisedly** just by observing the example sequences

Algorithmic Patterns - Counting

method	$a^n b^n$	$a^n b^n c^n$	$a^n b^n c^n d^n$	$a^n b^{2n}$	$a^n b^m c^{n+m}$
RNN	25%	23.3%	13.3%	23.3%	33.3%
LSTM	100%	100%	68.3%	75%	100%
List RNN 40+5	100%	33.3%	100%	100%	100%
Stack RNN 40+10	100%	100%	100%	100%	43.3%
Stack RNN 40+10 + rounding	100%	100%	100%	100%	100%

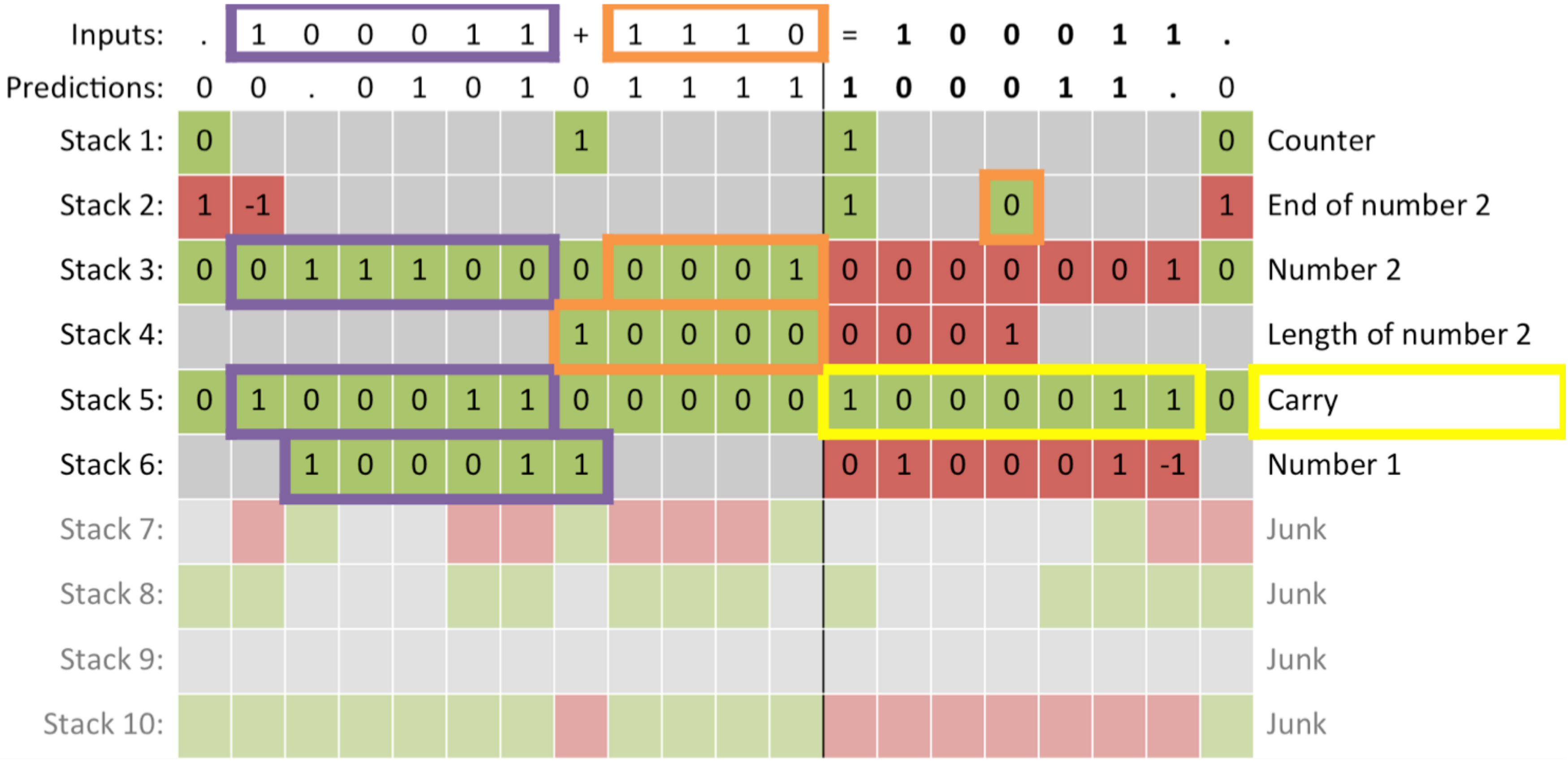
- Performance on simple counting tasks
- RNN with sigmoidal activation function cannot count
- Stack-RNN and LSTM can count

Algorithmic Patterns - Sequences



- Sequence memorization and binary addition are out-of-scope of LSTM
- Expandable memory of stacks allows to learn the solution

Binary Addition



- **No supervision in training, just prediction**
- Learns to: store digits, when to produce output, carry

Stack RNNs: summary

The good:

- Turing-complete model of computation (with >=2 stacks)
- Learns some algorithmic patterns
- Has long term memory
- Simple model that works for some problems that break RNNs and LSTMs
- Reproducible: https://github.com/facebook/Stack-RNN

The bad:

- The long term memory is used only to store partial computation (ie. learned skills are not stored there yet)
- Does not seem to be a good model for incremental learning
- Stacks do not seem to be a very general choice for the topology of the memory

Conclusion

To achieve true artificial intelligence, we need:

- AI-complete goal
- New set of tasks
- Develop new techniques
- Motivate more people to address these problems