Introduction to Deep Learning

Ismini Lourentzou

11-30-2017

Outline

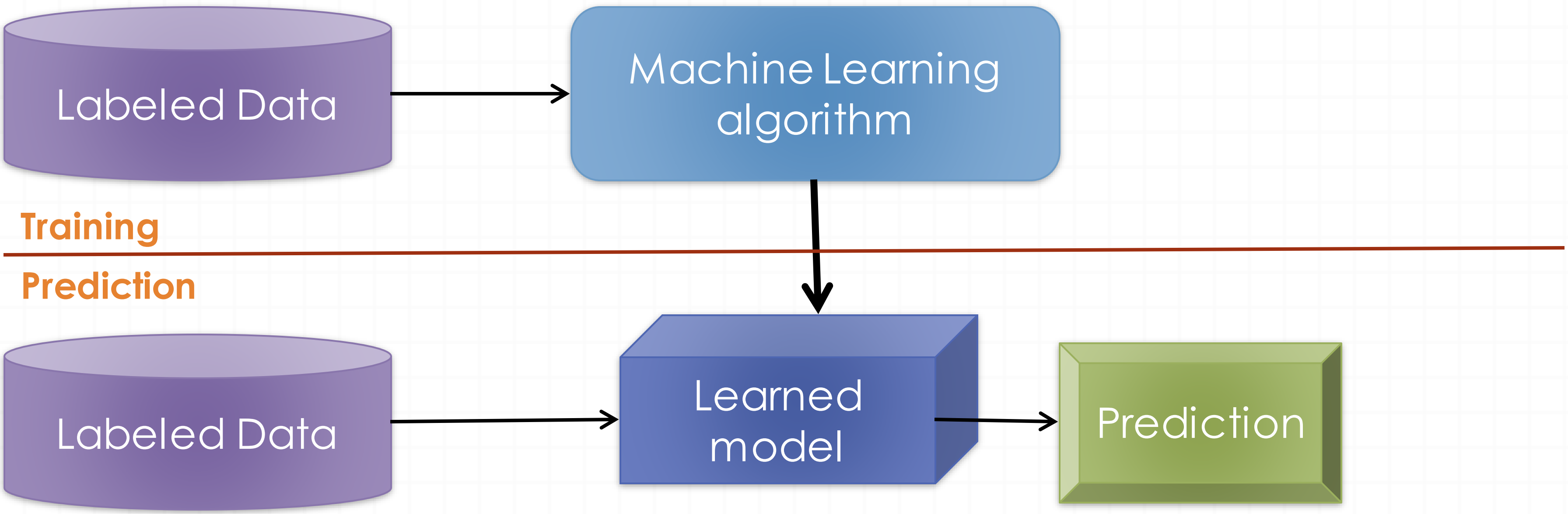
- ❑ Machine Learning basics
- ❑ Introduction to Deep Learning
  - what is Deep Learning
  - why is it useful
- ❑ Main components/hyper-parameters:
  - activation functions
  - optimizers, cost functions and training
  - regularization methods
  - tuning
  - classification vs. regression tasks
- ❑ DNN basic architectures:
  - convolutional
  - recurrent
  - attention mechanism
- ❑ Application example: Relation Extraction

~~Backpropagation~~
~~GANs & Adversarial training~~
~~Bayesian Deep Learning~~
~~Generative models~~
~~Unsupervised / Pretraining~~

Most material from [CS224 NLP with DL course at Stanford]

Machine Learning Basics

Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**



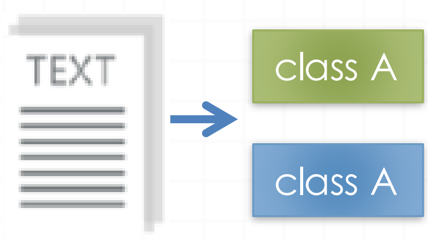
Methods that can learn from and make predictions on data

Types of Learning

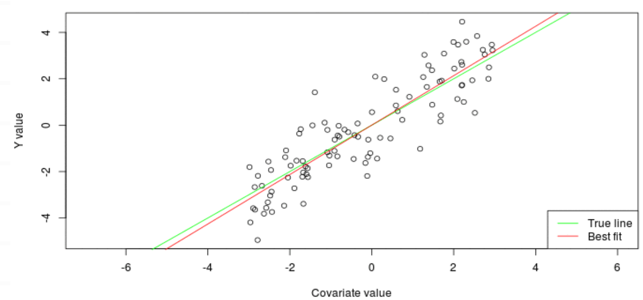
Supervised: Learning with a **labeled training** set
Example: email *classification* with already labeled emails

Unsupervised: Discover **patterns** in **unlabeled** data
Example: *cluster* similar documents based on text

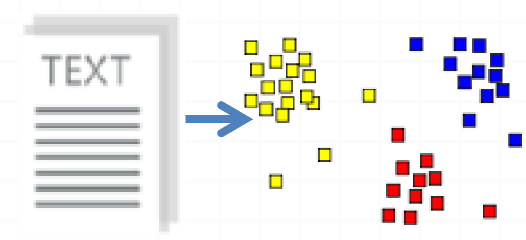
Reinforcement learning: learn to **act** based on **feedback/reward**
Example: learn to play Go, reward: *win or lose*



Classification



Regression



Clustering

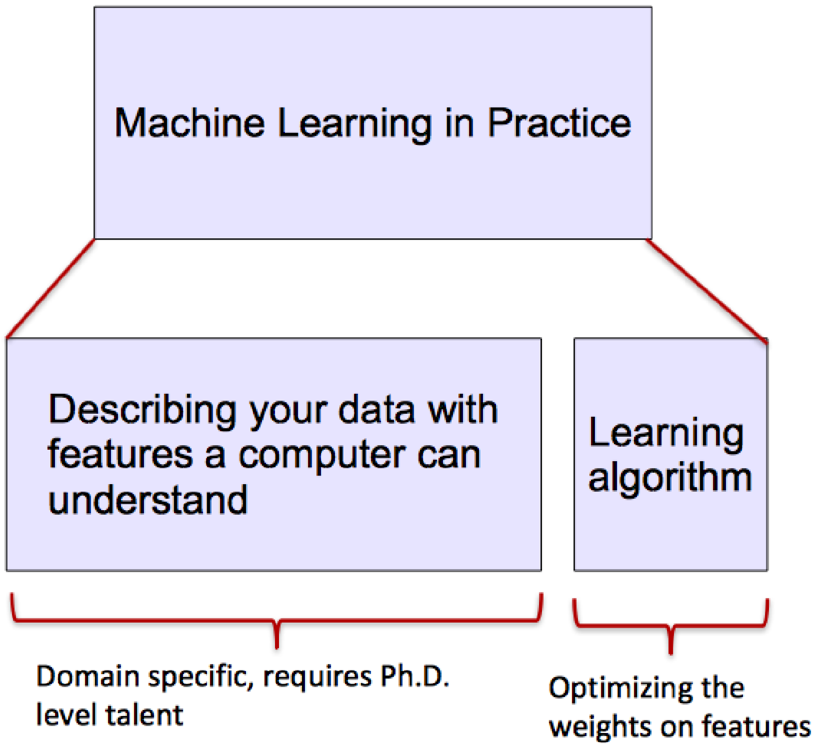
Anomaly Detection
Sequence labeling
…

http://mbjoseph.github.io/2013/11/27/measure.html

ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**

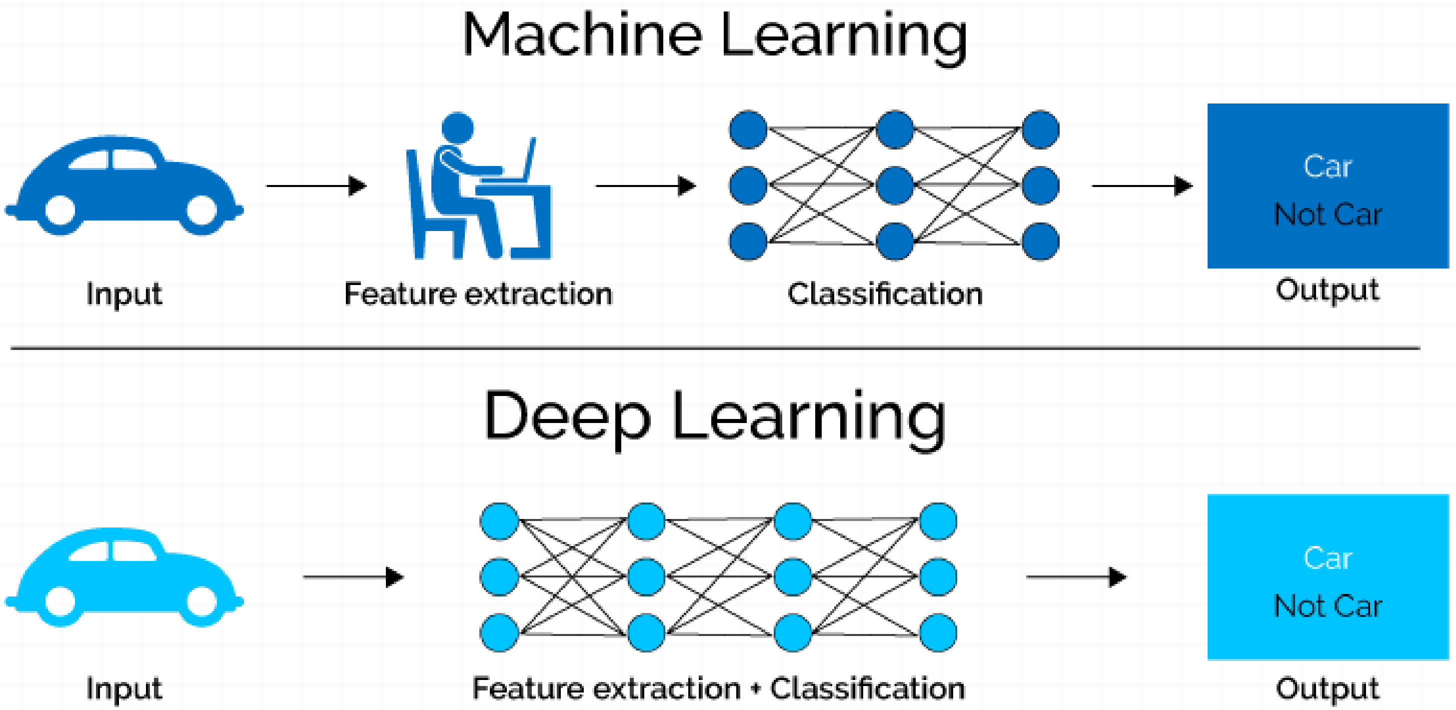
ML becomes just **optimizing weights** to best make a final prediction



Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

What is Deep Learning (DL) ?

A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.
Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**
If you provide the system **tons of information**, it begins to understand it and respond in useful ways.

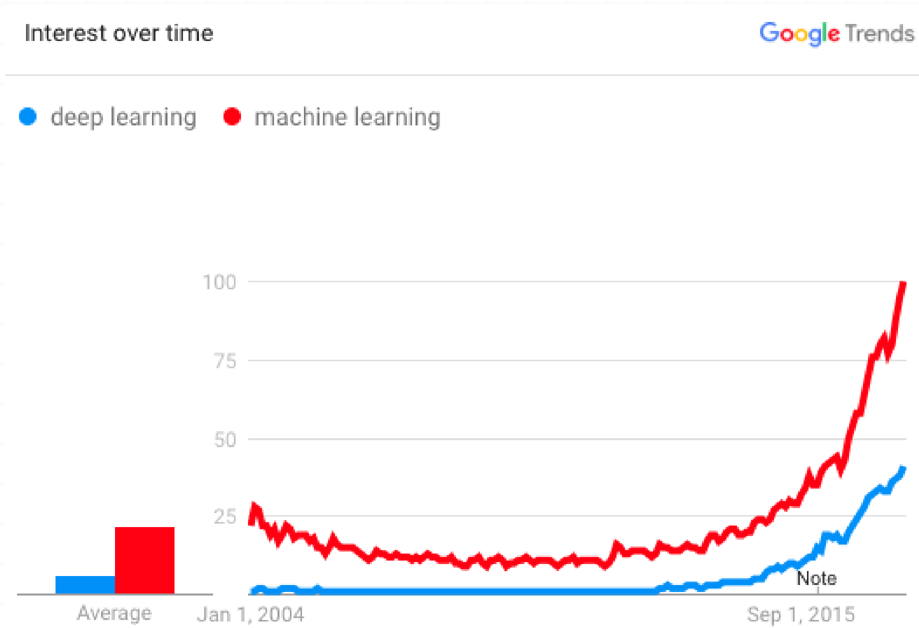


https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

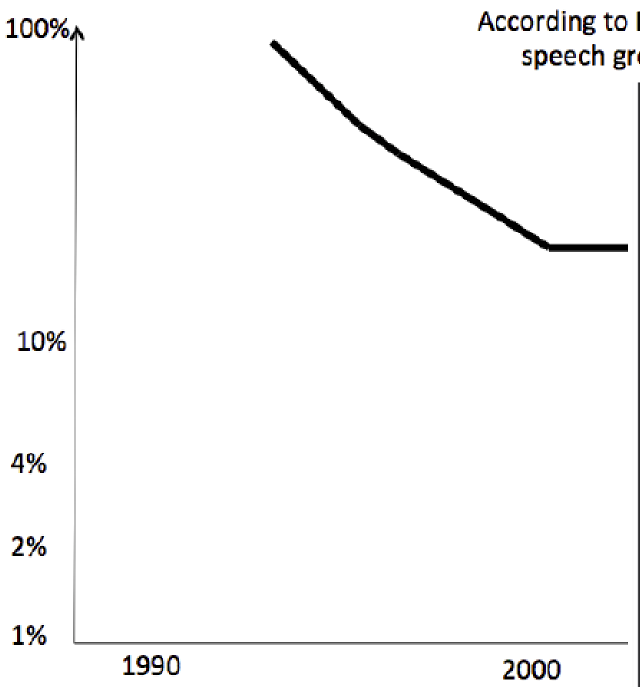
Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

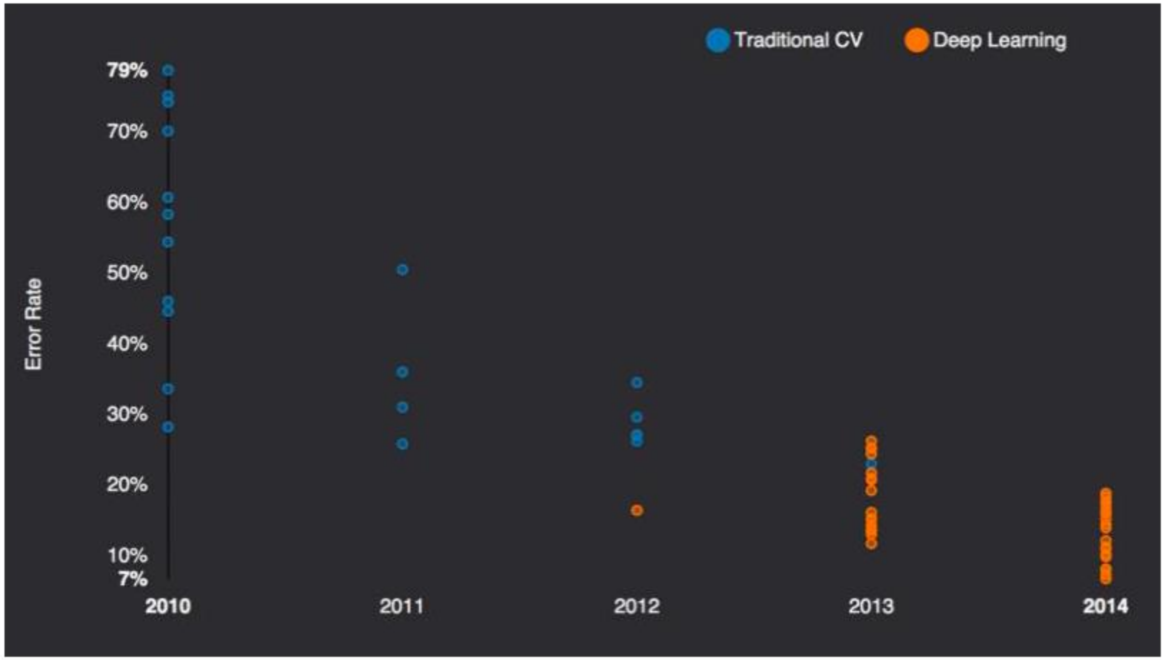
In ~2010 DL started outperforming other ML techniques
first in speech and vision, then NLP



State of the art in …



Deep Learning in Speech Recognition

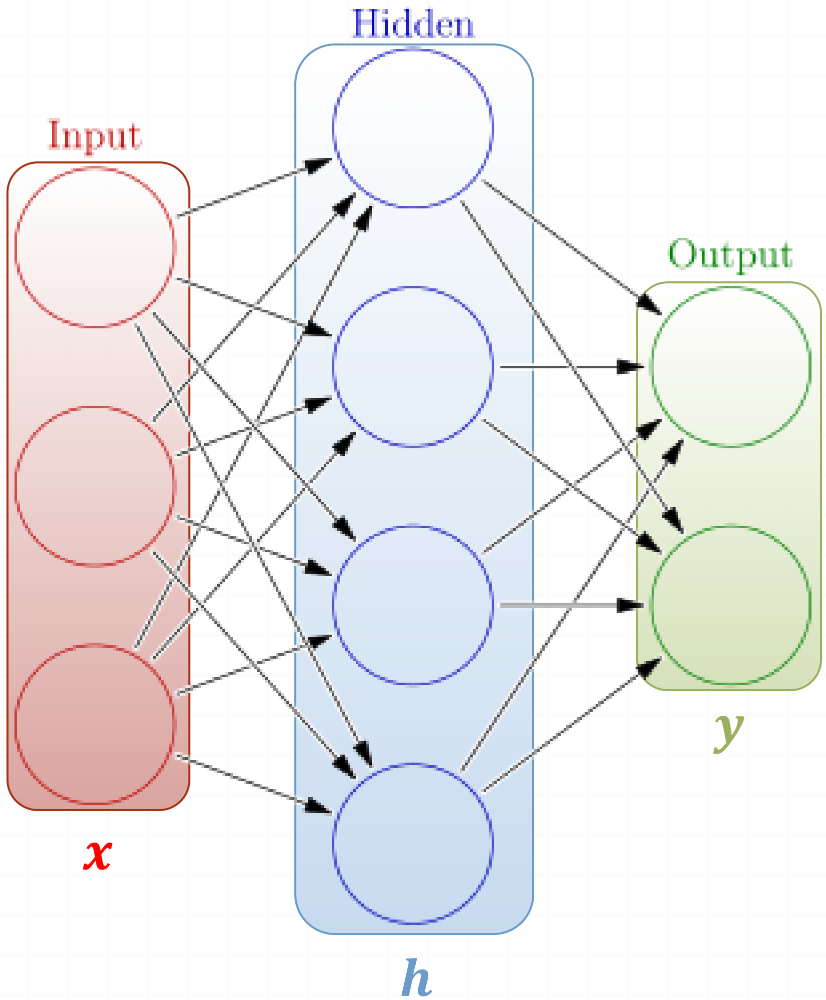


ImageNet: The "computer vision World Cup"

Several big improvements in recent years in NLP
- ✓ Machine Translation
- ✓ Sentiment Analysis
- ✓ Dialogue Agents
- ✓ Question Answering
- ✓ Text Classification …

Leverage different levels of representation
- o words & characters
- o syntax & semantics

Neural Network Intro



Weights

$$h = \sigma(W_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

How do we train?

4 + 2 = 6 neurons (not counting inputs)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

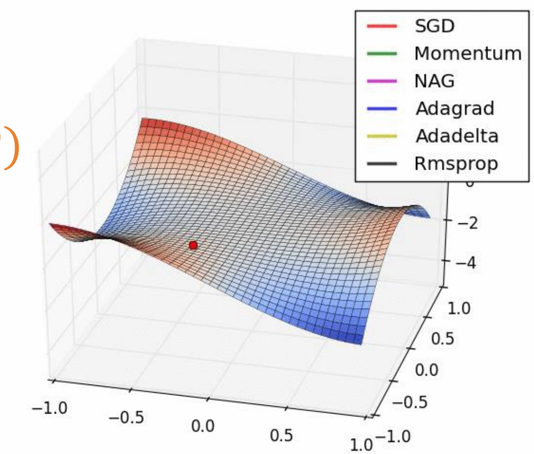
26 learnable **parameters**

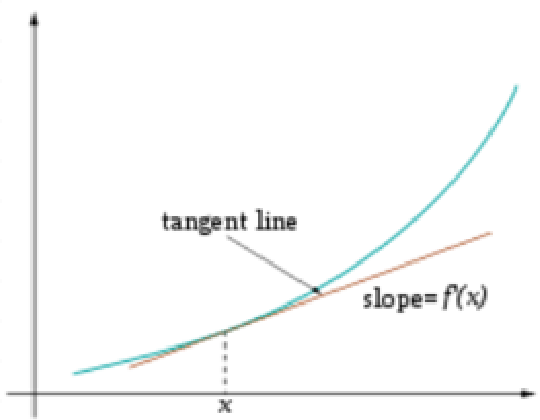
Demo

Training

Sample labeled data (**batch**) → **Forward** it through the network, get predictions → **Back-propagate** the errors → **Update** the network weights

Optimize (min. or max.) **objective/cost function $J(\theta)$**
Generate **error signal** that measures difference between predictions and target values





Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39

Gradient Descent

objective/cost function $J(\theta)$

Review of backpropagation

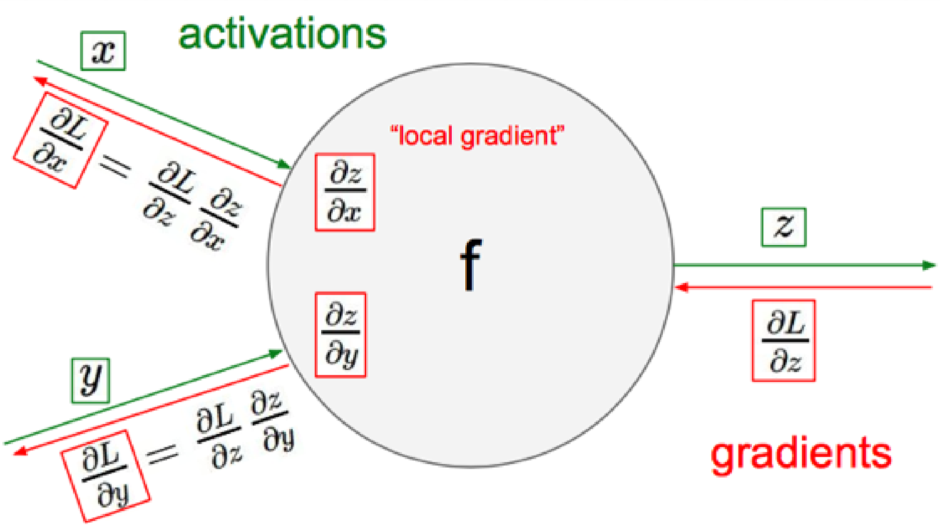
$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{d}{d\theta_j^{old}} J(\theta)$$

Update each element of θ

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

Matrix notation for all parameters

learning rate



Recursively apply **chain rule** though each node

One forward pass

Text (input) representation
TFIDF
Word embeddings
....

0.2	-0.5	0.1
2.0	1.5	1.3
0.5	0.0	0.25
-0.3	2.0	0.0

$\mathbf{W}$

0.1
0.2
0.3

$\boldsymbol{x_i}$

+

1.0
3.0
0.025
0.0

$\boldsymbol{b}$

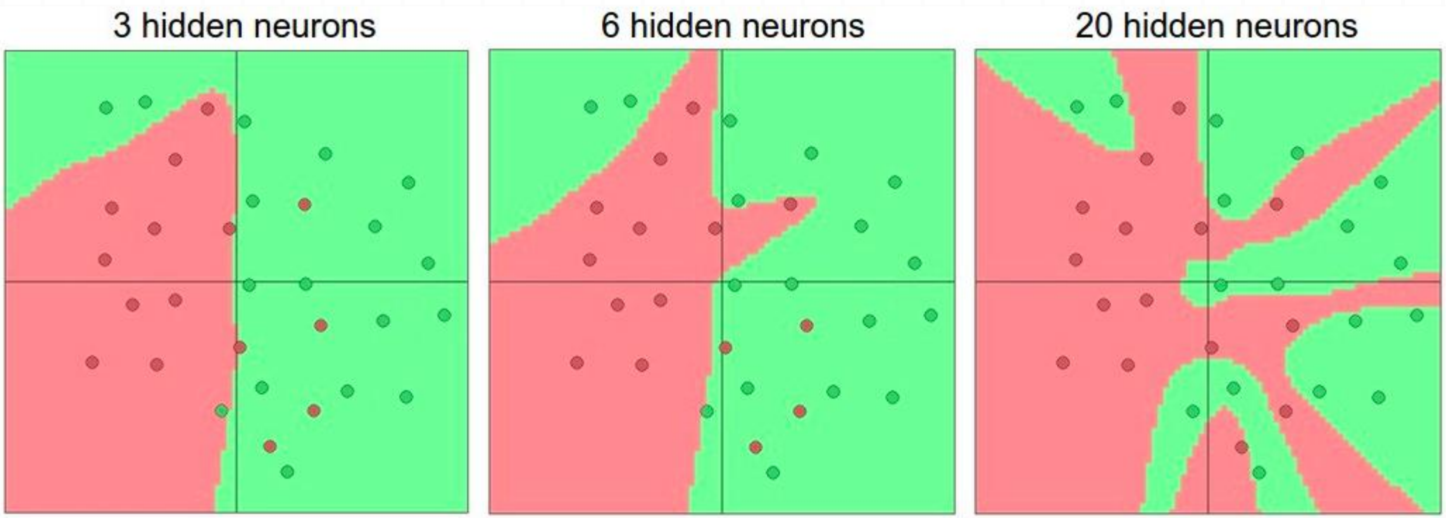
=

0.95	very positive
3.89	$\boxed{\text{positive}}$
0.15	negative
0.37	very negative

$\boldsymbol{\sigma(x_i; W, b)}$

Activation functions

Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function $W_1 W_2 x = Wx$

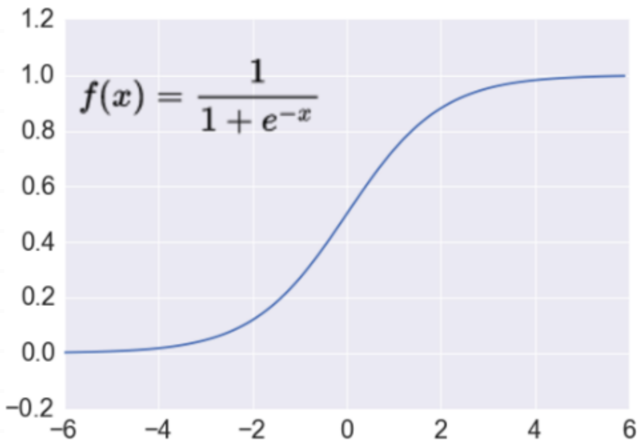


http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

More layers and neurons can approximate more complex functions

Full list: https://en.wikipedia.org/wiki/Activation_function

Activation: Sigmoid



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and "squashes" it into range between 0 and 1.

$$R^n \rightarrow [0,1]$$

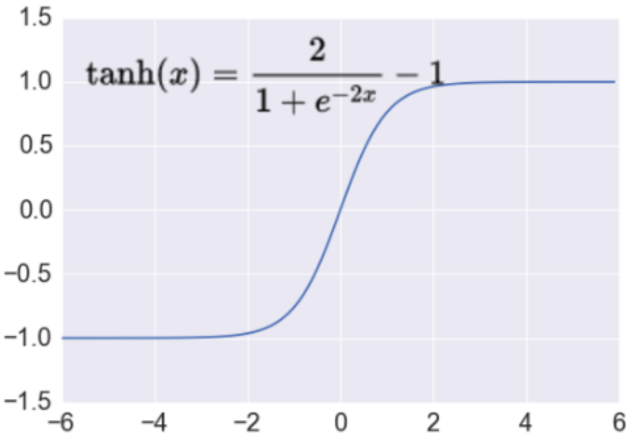
\+ Nice interpretation as the **firing rate** of a neuron

- 0 = not firing at all
- 1 = fully firing

\- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn

- when the neuron's activation are 0 or 1 (saturate)
  - gradient at these regions almost zero
  - almost no signal will flow to its weights
  - if initial weights are too large then most neurons would saturate

Activation: Tanh



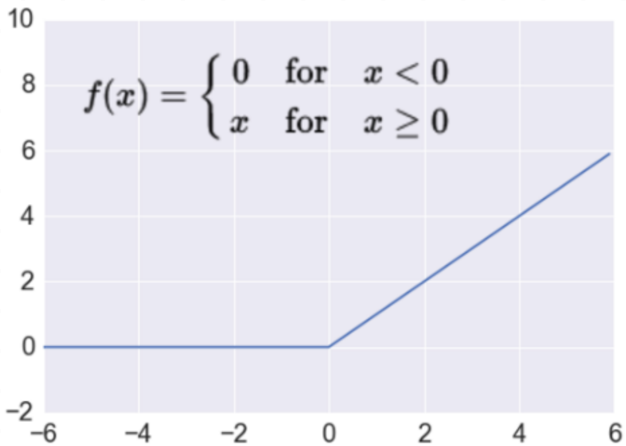
http://adilmoujahid.com/images/activation.png

Takes a real-valued number and "squashes" it into range between -1 and 1.

$$R^n \rightarrow [-1,1]$$

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU



http://adilmoujahid.com/images/activation.png

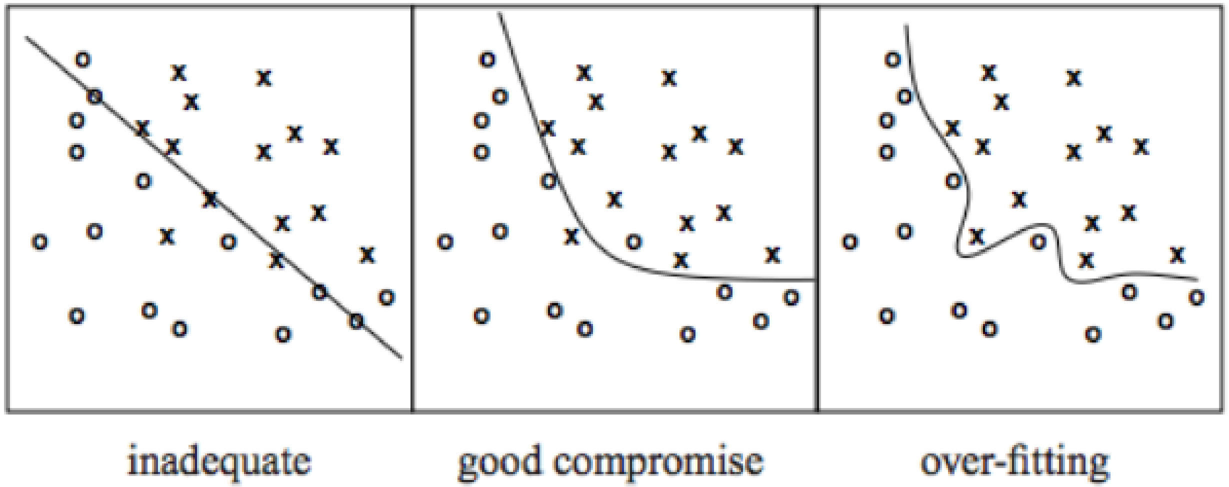
Takes a real-valued number and thresholds it at zero $\mathrm{f}(x) = \max(0, x)$

$$R^n \rightarrow R_+^n$$

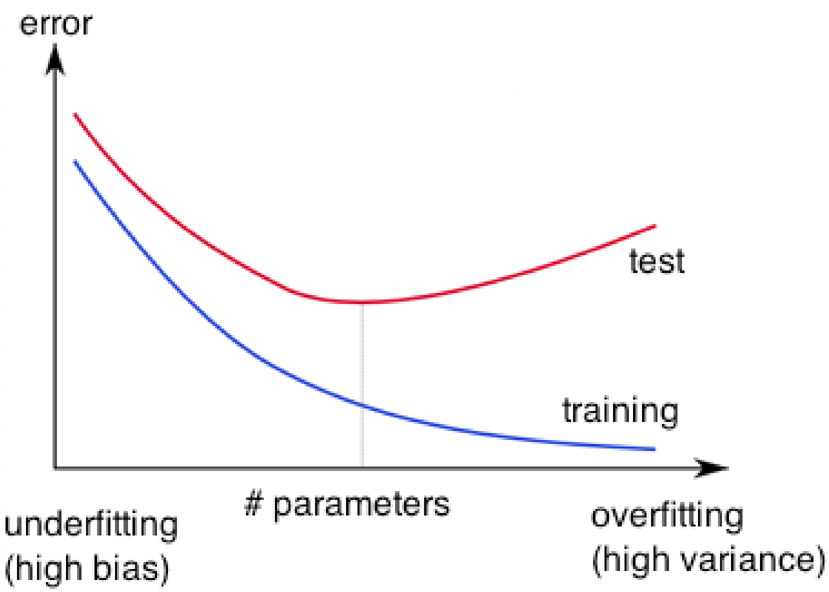
Most Deep Networks use ReLU nowadays

- Trains much **faster**
  - accelerates the convergence of SGD
  - due to linear, non-saturating form
- Less expensive operations
  - compared to sigmoid/tanh (exponentials etc.)
  - implemented by simply thresholding a matrix at zero
- More **expressive**
- Prevents the **gradient vanishing problem**

Overfitting



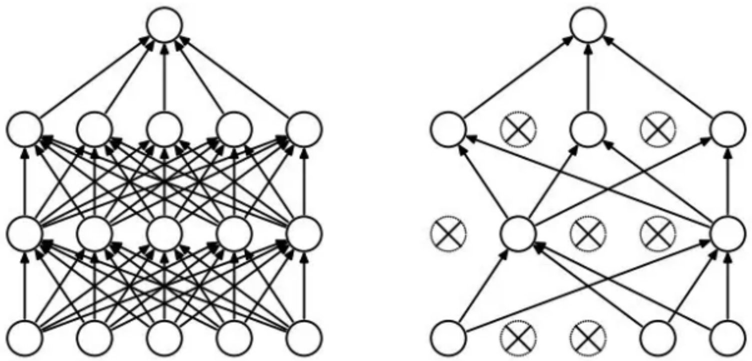
http://wiki.bethanycrane.com /overfitting-of-data



https://www.neuraldesigner.com /images/learning/selection_error.svg

Learned hypothesis may **fit** the training data very well, even outliers (**noise**) but fail to **generalize** to new examples (test data)

Regularization



Dropout

- Randomly drop units (along with their connections) during training
- Each unit retained with fixed probability p, independent of other units
- Hyper-parameter p to be chosen (tuned)

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research (2014)

L2 = weight decay

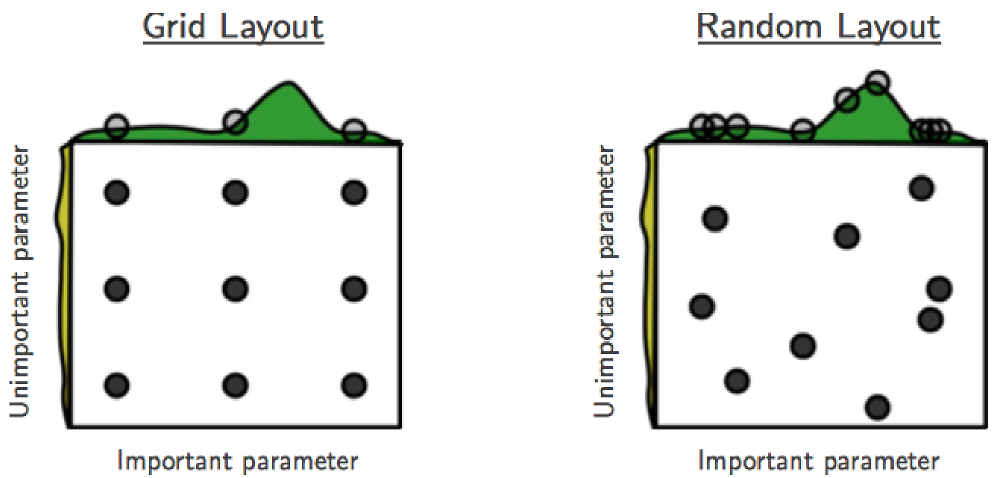
- Regularization term that penalizes big weights, added to the objective
- Weight decay value determines how dominant regularization is during gradient computation
- Big weight decay coefficient → big penalty for big weights

$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

Early-stopping

- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after n subsequent epochs
- n is called patience

Tuning hyper-parameters



g(x) ≈ g(x) + h(y)

g(x) shown in green
h(y) is shown in yellow

Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research, Feb (2012)

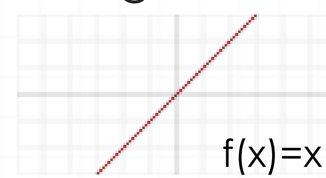
"Grid and random search of 9 trials for optimizing function g(x) ≈ g(x) + h(y)
With grid search, nine trials only test g(x) in three distinct places.
With random search, all nine trials explore distinct values of g. "

Both try configurations randomly and **blindly**
Next trial is independent to all the trials done before

Library available!

Bayesian optimization for hyper-parameter tuning:
Make smarter choice for the next trial, minimize the number of trials
1. Collect the performance at several configurations
2. Make inference and decide what configuration to try next

Loss functions and output

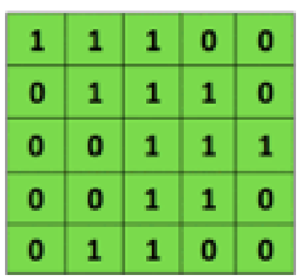
	Classification	Regression
Training examples	R^n x {class_1, …, class_n} (one-hot encoding)	R^n x R^m
Output Layer	Soft-max [map R^n to a probability distribution] $P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$	Linear (Identity) or Sigmoid 
Cost (loss) function	Cross-entropy $J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)}\log\hat{y}_k^{(i)} + \left(1-y_k^{(i)}\right)\log\left(1-\hat{y}_k^{(i)}\right)\right]$	Mean Squared Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$ Mean Absolute Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\lvert y^{(i)} - \hat{y}^{(i)}\right\rvert$

List of loss functions

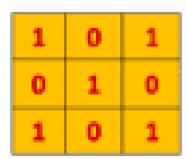
Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and group them afterwards

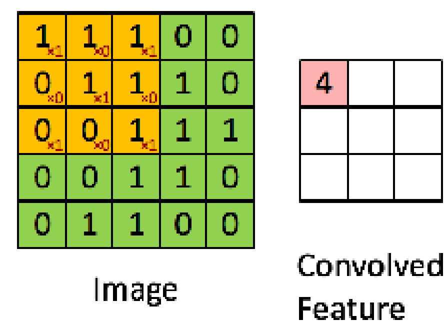
Example: “this takes too long” compute vectors for:
This takes, takes too, too long, this takes too, takes too long, this takes too long



Input matrix



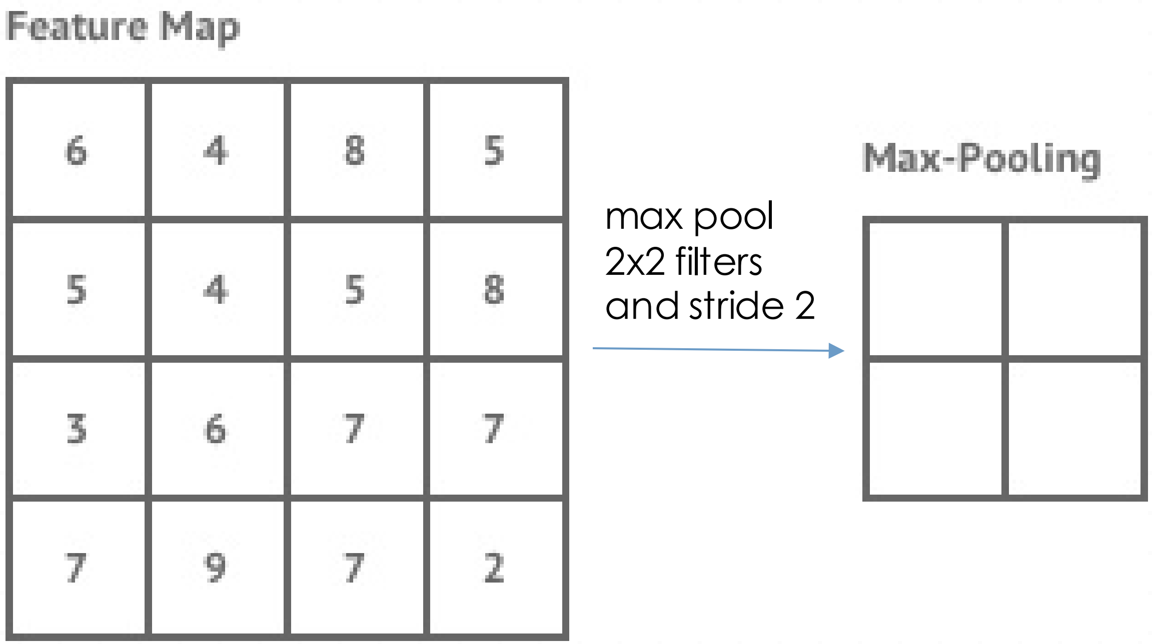
Convolutional 3x3 filter



http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

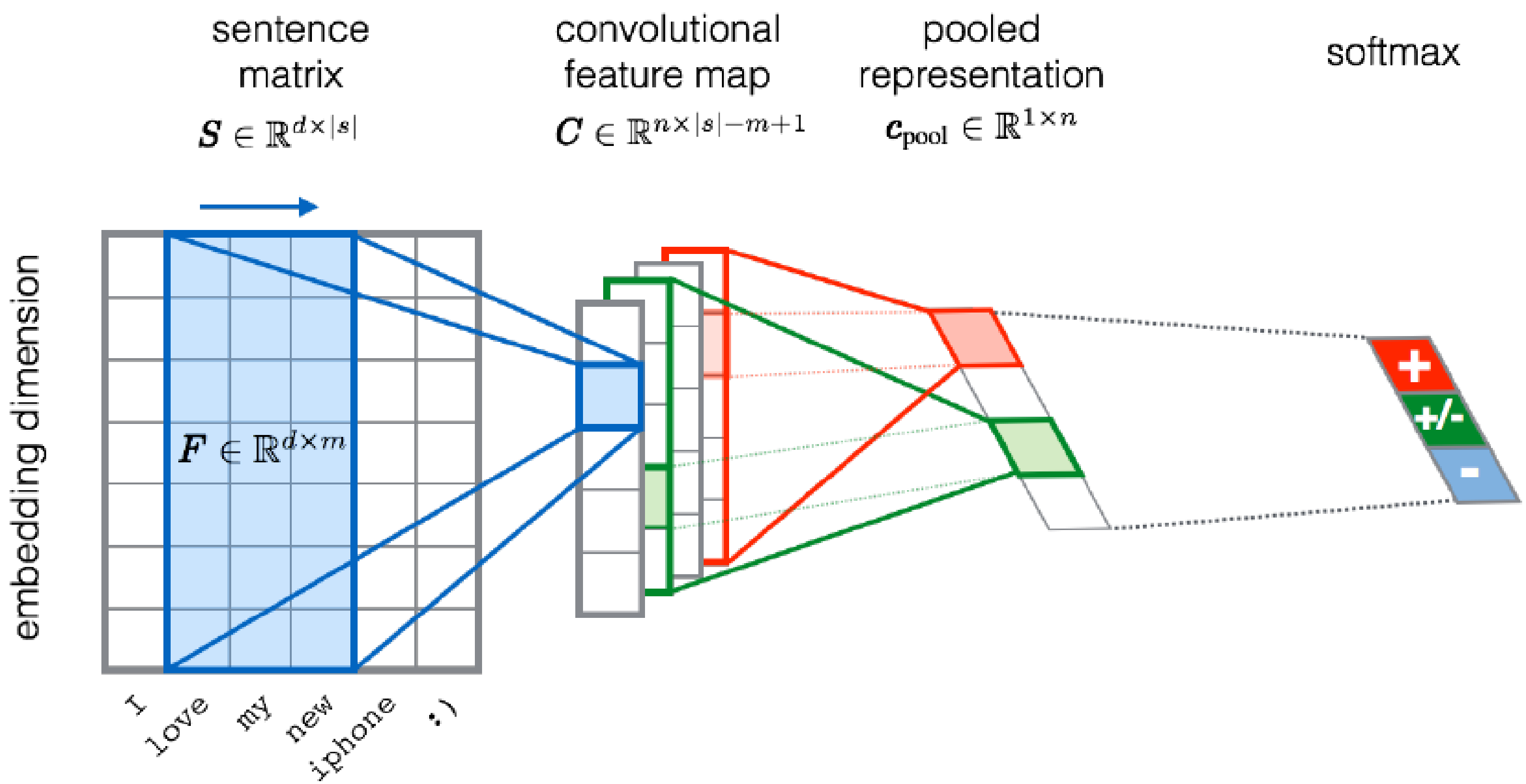
Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and **group them afterwards**



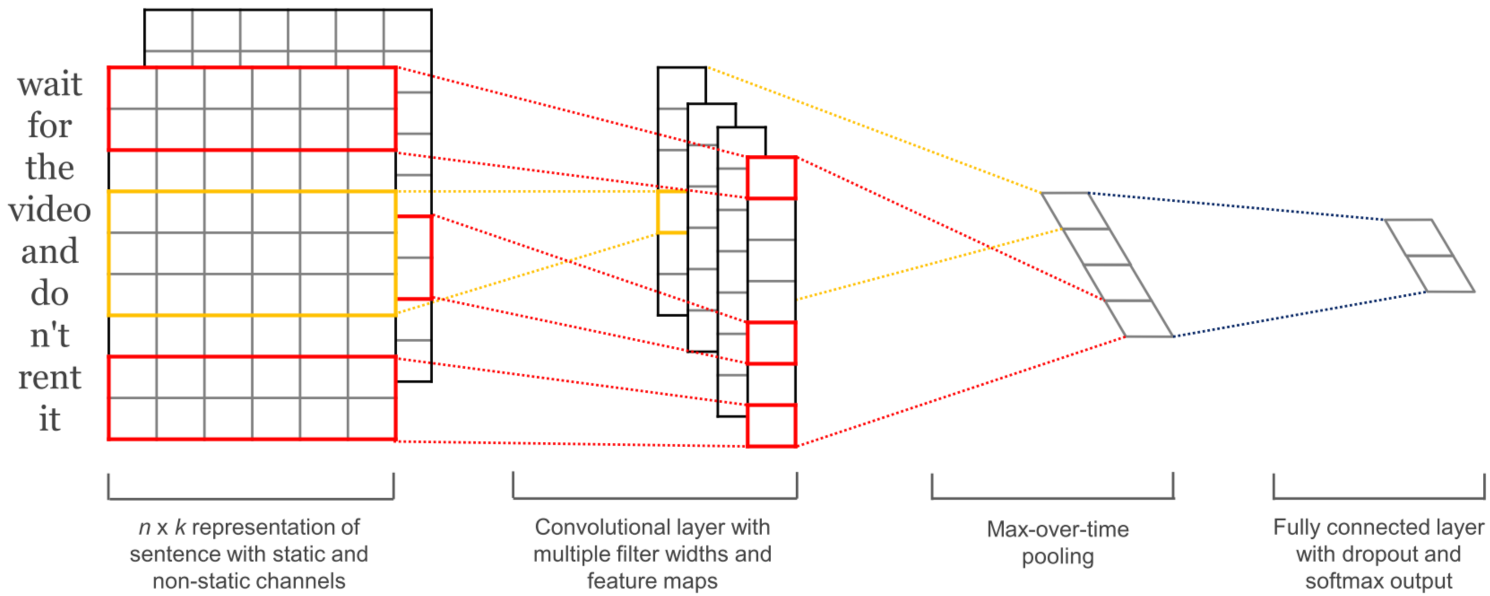
https://shafeentejani.github.io/assets/images/pooling.gif

CNN for text classification



Severyn, Aliaksei, and Alessandro Moschitti. "UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification." *SemEval@ NAACL-HLT*. 2015.

CNN with multiple filters



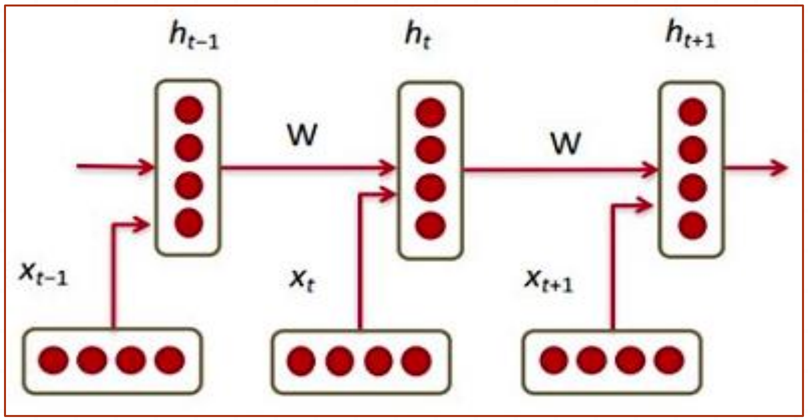
Kim, Y. "Convolutional Neural Networks for Sentence Classification", EMNLP (2014)

sliding over 3, 4 or 5 words at a time

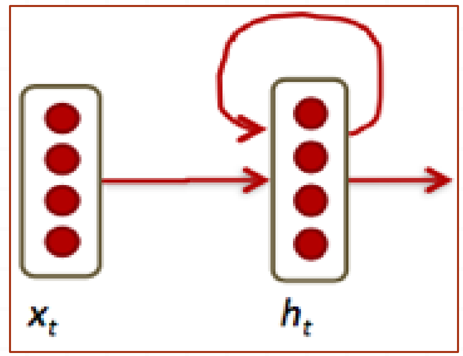
Recurrent Neural Networks (RNNs)

Main RNN idea for text:
Condition on **all previous words**
Use same set of weights at all time steps

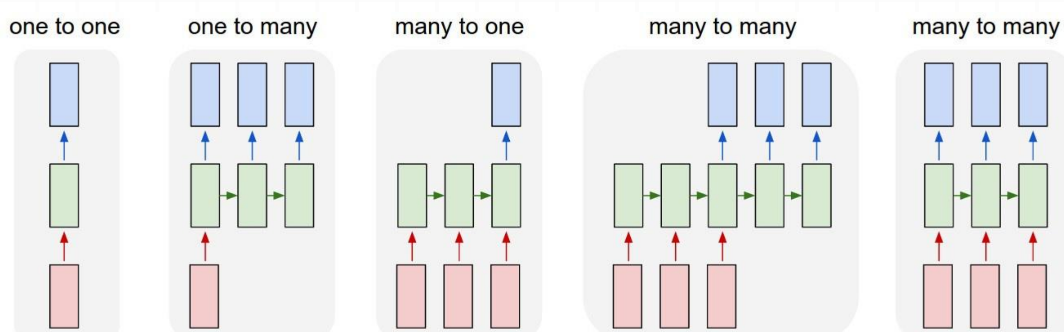
$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$



$\triangleq$



https://pbs.twimg.com/media/C2j-8j5UsAACgEK.jpg



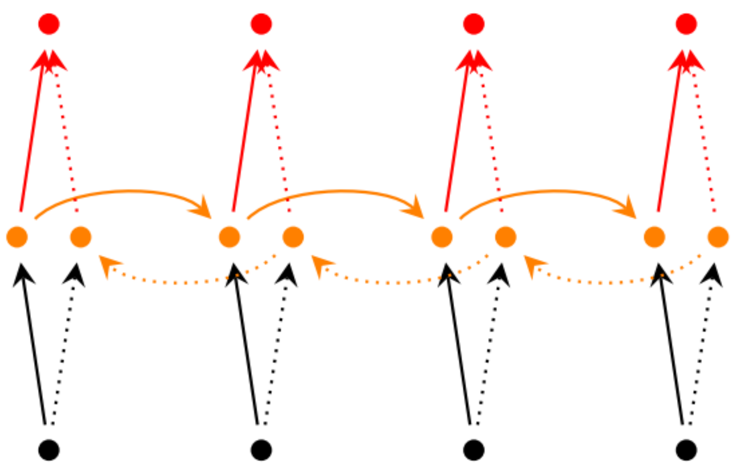
😁Stack them up, Lego fun!

😟[Vanishing gradient problem]

https://discuss.pytorch.org/uploads/default/original/1X/6415da0424dd66f2f5b134709b92baa59e604c55.jpg

Bidirectional RNNs

Main idea: incorporate both left and right context
output may not only depend on the **previous** elements in the sequence, but also **future** elements.



http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

$$\overrightarrow{h}_t = \sigma(\overrightarrow{W}^{(hh)}\overrightarrow{h}_{t-1} + \overrightarrow{W}^{(hx)}x_t)$$

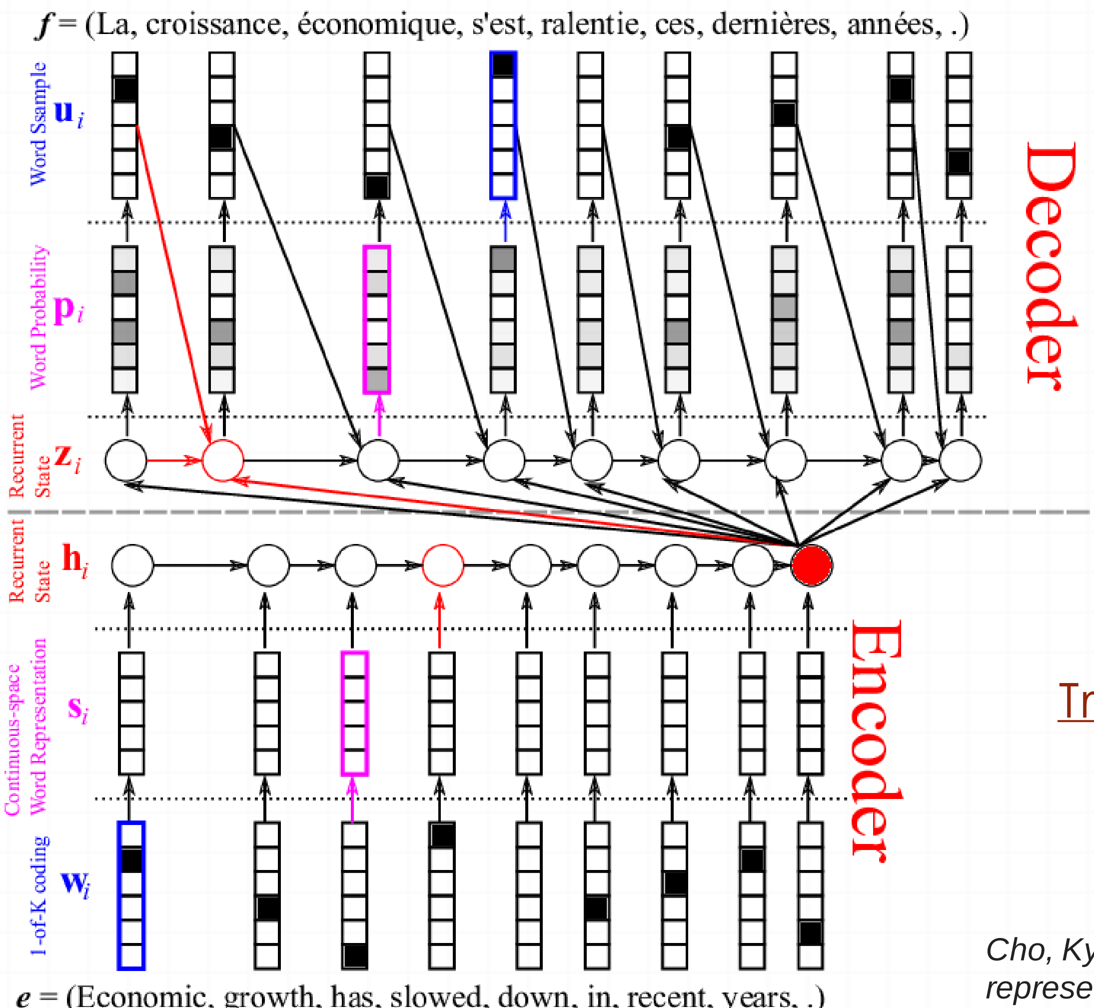
$$\overleftarrow{h}_t = \sigma(\overleftarrow{W}^{(hh)}\overleftarrow{h}_{t+1} + \overleftarrow{W}^{(hx)}x_t)$$

$$y_t = f(\,[\overrightarrow{h}_t;\overleftarrow{h}_t]\,)$$

past and future around a single token

two RNNs stacked on top of each other
output is computed based on the hidden state of both RNNs $[\overrightarrow{h}_t;\overleftarrow{h}_t]$

Sequence 2 Sequence or Encoder Decoder model



[Try many other models for MT]()

Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." EMNLP 2014

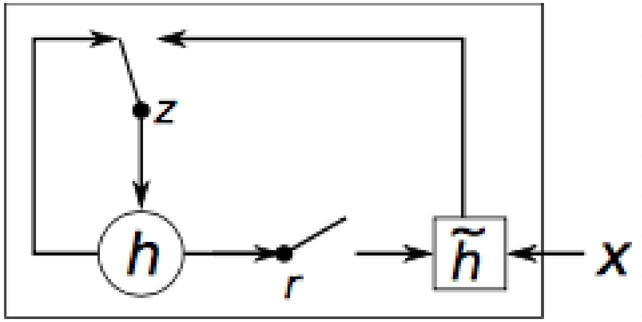
Gated Recurrent Units (GRUs)

Main idea:
keep around memory to capture **long dependencies**
Allow error messages to flow at **different strengths** depending on the inputs

Standard RNN computes hidden layer at next time step directly $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$

Compute an update gate based on current input word vector and hidden state

$$z_t = \sigma(U^{(z)}h_{t-1} + W^{(z)}x_t)$$



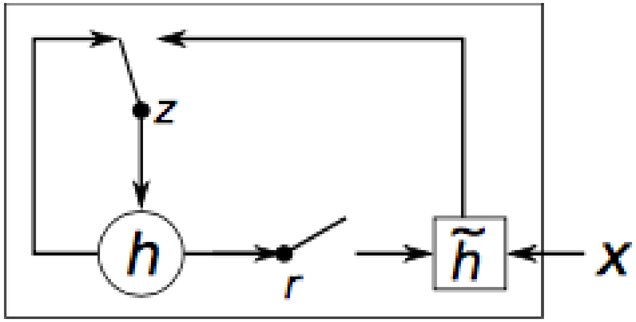
http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

Controls how much of past state should matter now
If z close to 1, then we can copy information in that unit through many steps!

Gated Recurrent Units (GRUs)

Main idea:
keep around memory to capture **long dependencies**
Allow error messages to flow at **different strengths** depending on the inputs

Standard RNN computes hidden layer at next time step directly $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$



http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

Compute an update gate based on current input word vector and hidden state

$$z_t = \sigma(U^{(z)}h_{t-1} + W^{(z)}x_t)$$

Compute a reset gate similarly but with different weights

$$r_t = \sigma\left(U^{(r)}h_{t-1} + W^{(r)}x_t\right)$$

If reset close to 0, ignore previous hidden state (allows model to drop information that is irrelevant in the future)

Units with **short-term** dependencies often have **reset** gates very active
Units with **long-term** dependencies have active **update** gates z

Gated Recurrent Units (GRUs)

Main idea:
keep around memory to capture **long dependencies**
Allow error messages to flow at **different strengths** depending on the inputs

Standard RNN computes hidden layer at next time step directly $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$

Compute an update gate based on current input word vector and hidden state

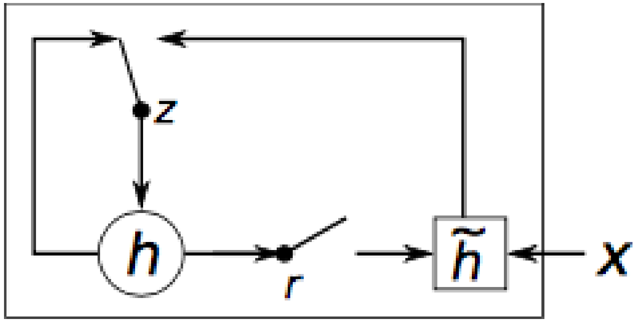
$$z_t = \sigma(U^{(z)}h_{t-1} + W^{(z)}x_t)$$

Compute a reset gate similarly but with different weights

$$r_t = \sigma\left(U^{(r)}h_{t-1} + W^{(r)}x_t\right)$$

New memory $\tilde{h}_t = tanh(r_t \circ Uh_{t-1} + Wx_t)$

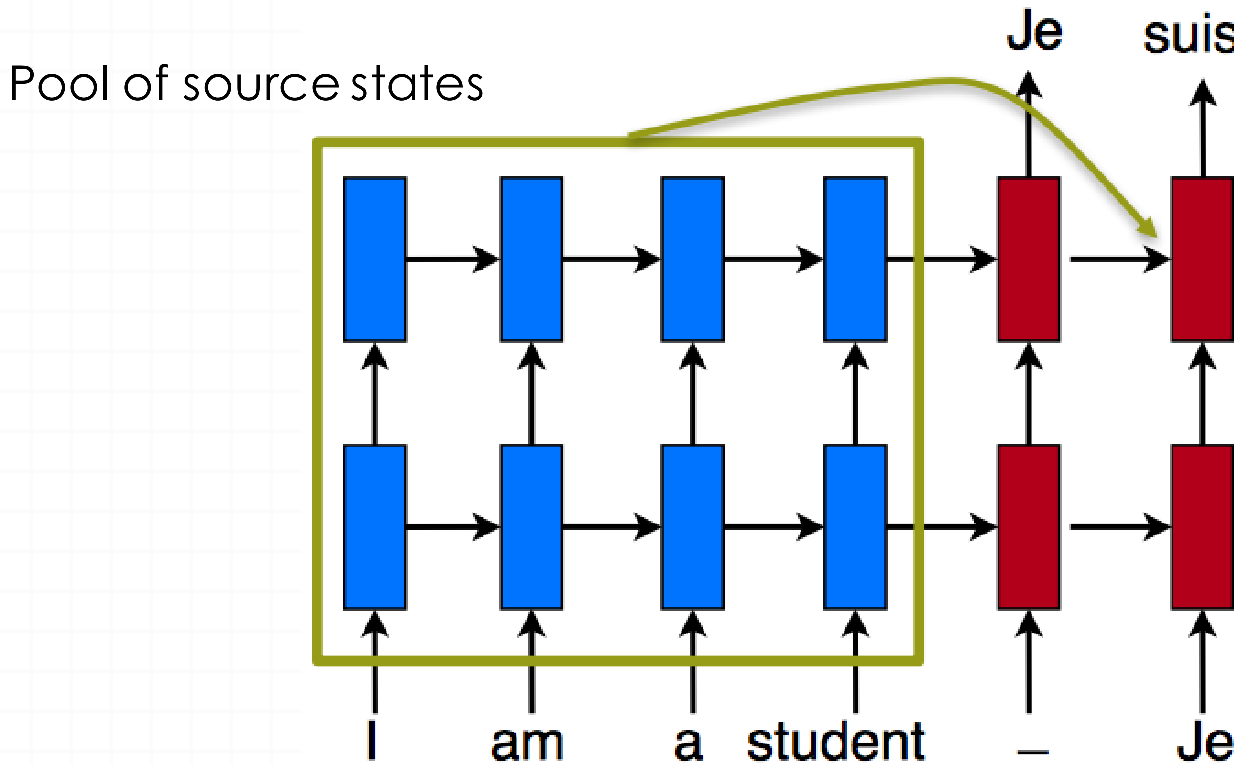
Final memory $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$ combines current & previous time steps



http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

LSTMs are a more complex form, but basically same intuition
GRUs are often more preferred than LSTMs

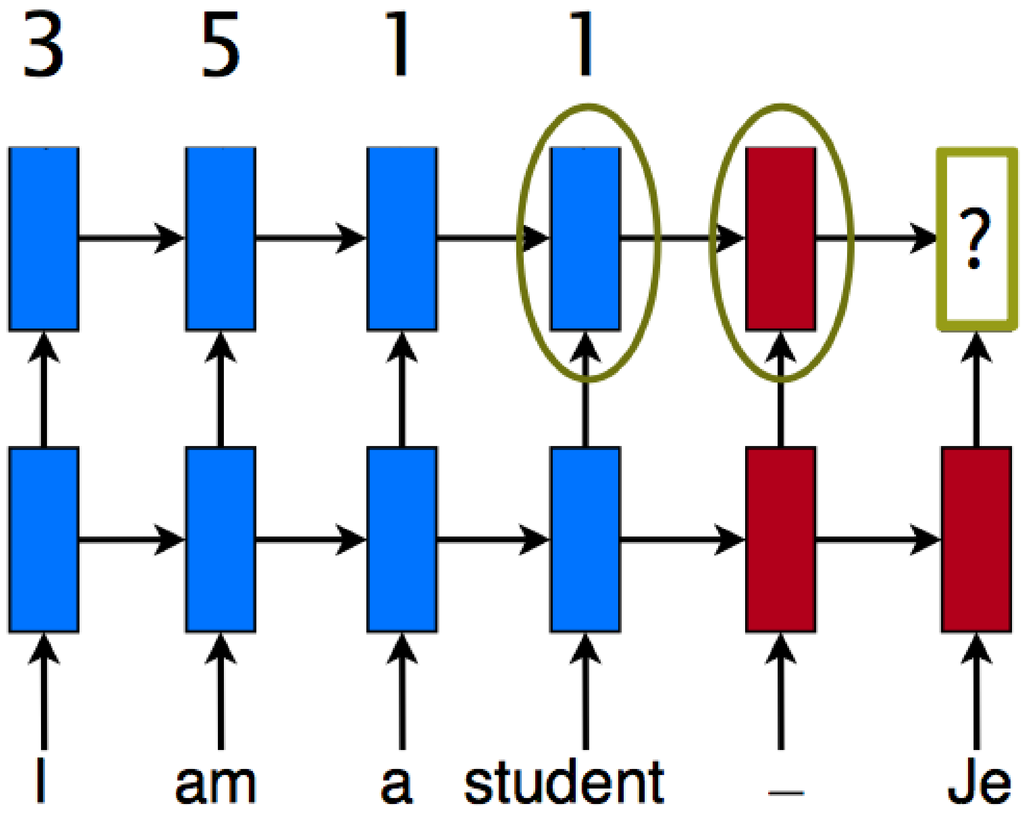
Attention Mechanism



Bahdanau D. et al. "Neural machine translation by jointly learning to align and translate." ICLR (2015)

Main idea: retrieve as needed

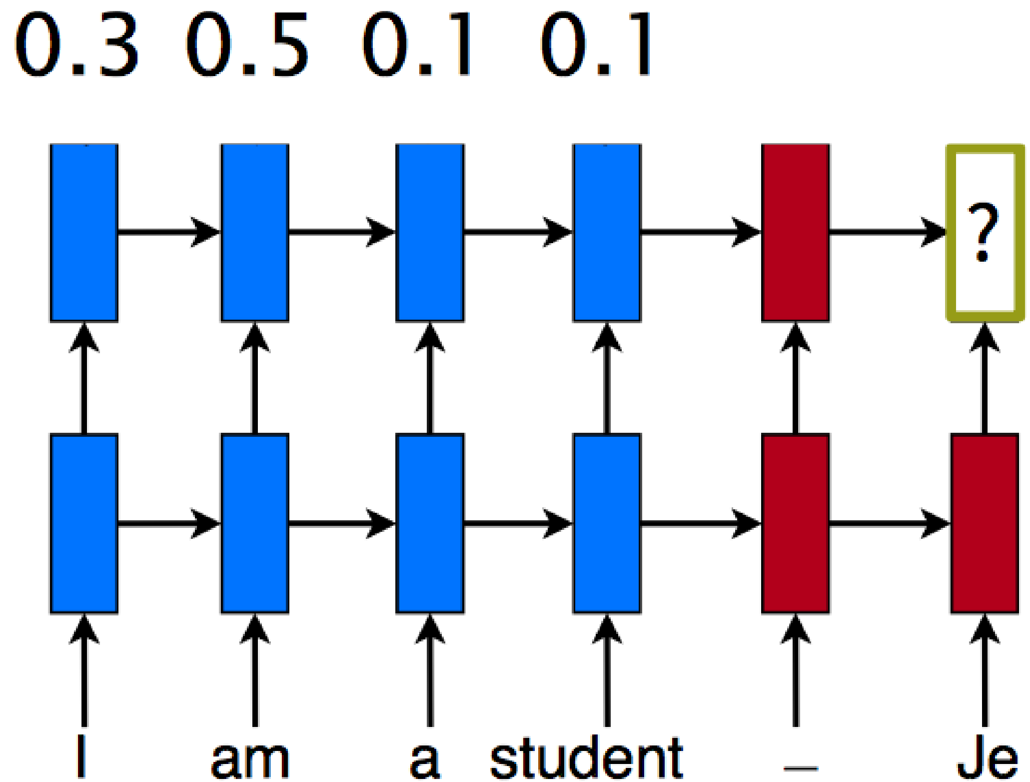
Attention - Scoring



$$score(h_{t-1}, \bar{h}_s) = {h_t}^T \bar{h}_s$$

Compare target and source hidden states

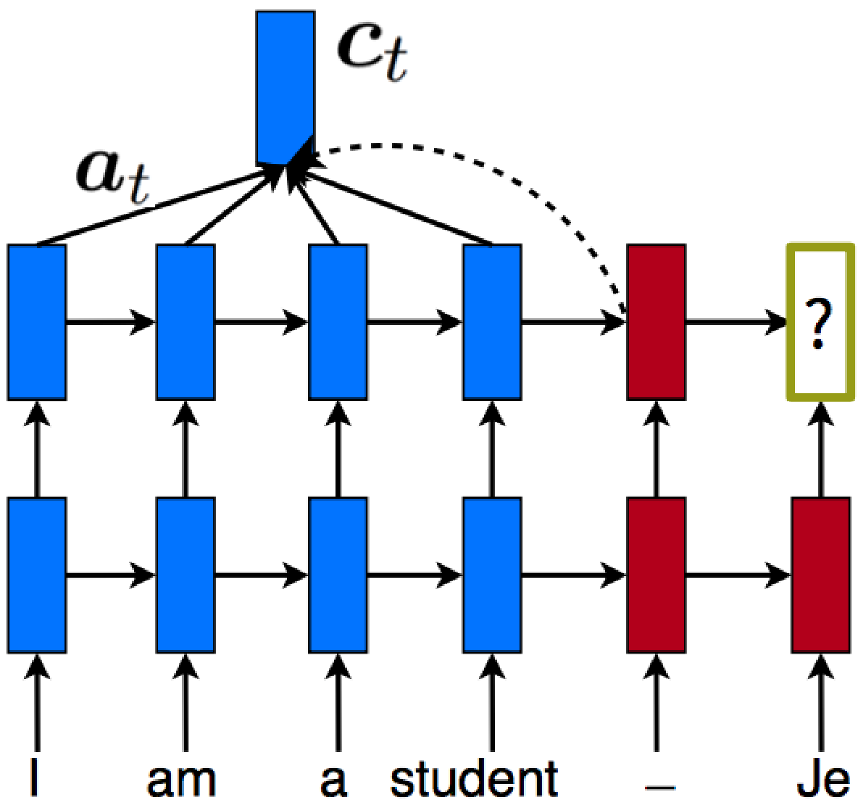
Attention - Normalization



$$a_t(s) = \frac{e^{score(s)}}{\sum_{s'} e^{score(s')}}$$

Convert into alignment weights

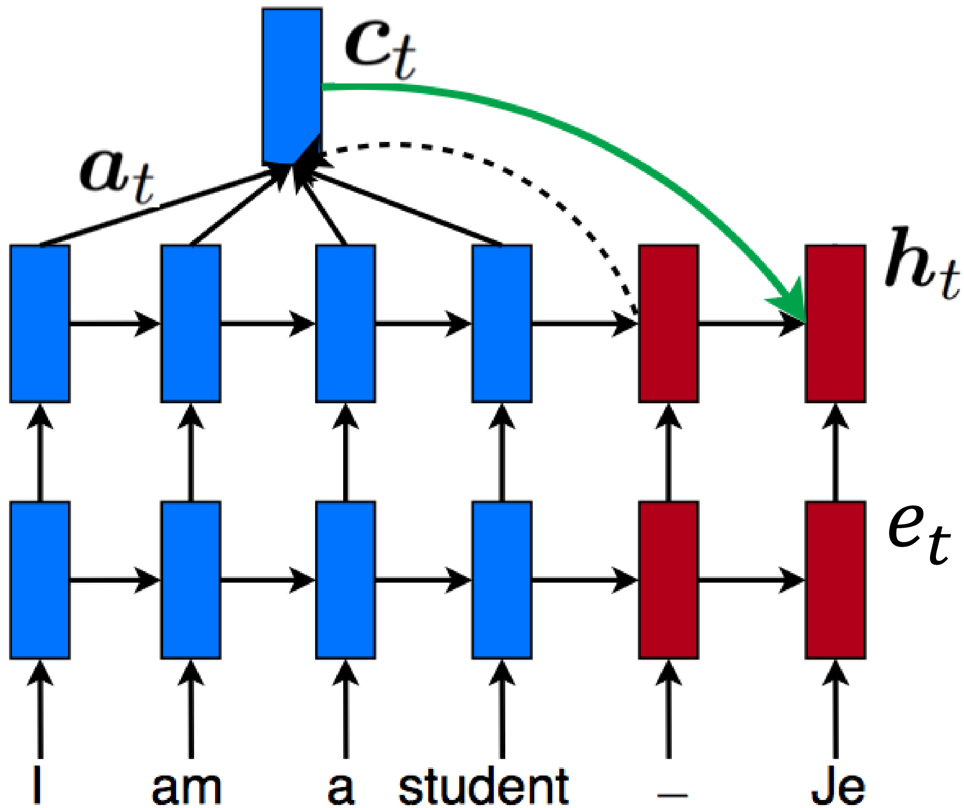
Attention - Context



$$c_t = \sum_s a_t(s)\bar{h}_s$$

Build **context** vector: weighted average

Attention - Context



$$h_t = f(h_{t-1}, c_t, e_t)$$

Compute **next** hidden state

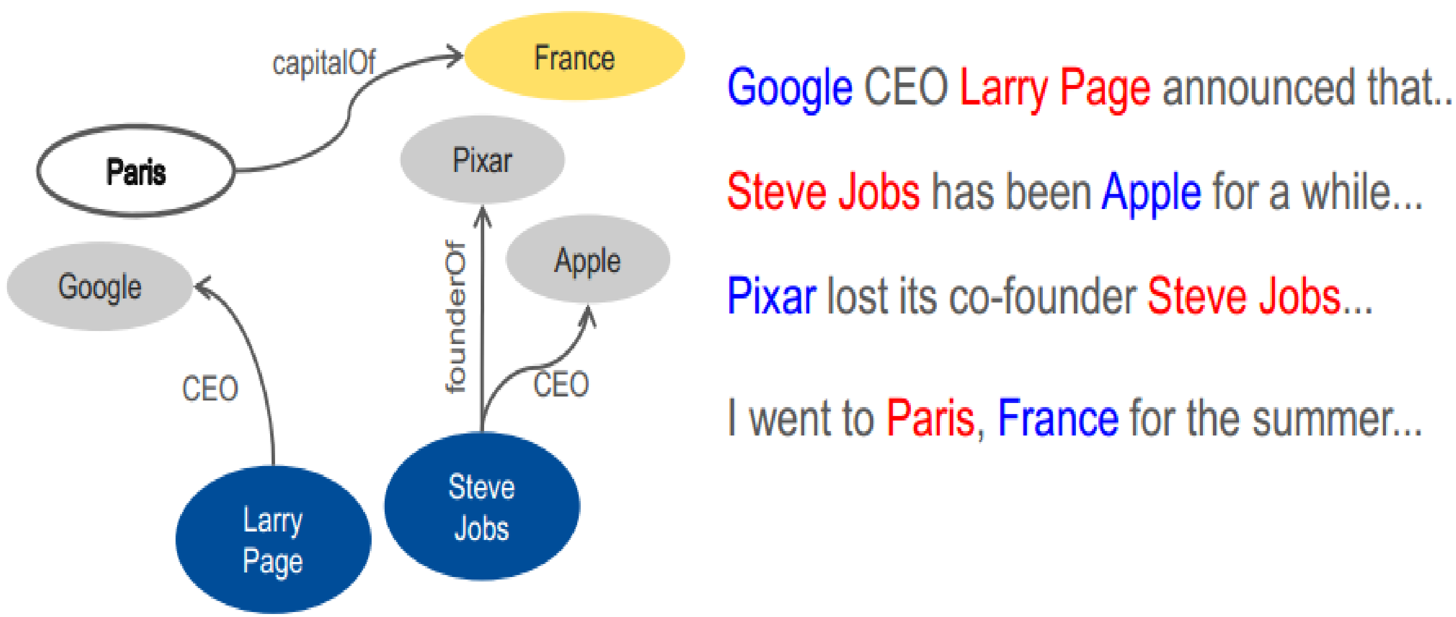
Application Example: IMDB Movie reviews sentiment classification

https://uofi.box.com/v/cs510DL

Binary Classification

Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative)

Application Example: Relation Extraction from text



http://www.mathcs.emory.edu/~dsavenk/slides/relation_extraction/img/distant.png

Useful for:

- knowledge base completion
- social media analysis
- question answering
- ...

Task: binary (or multi-class) classification

sentence S = $w_1\, w_2 .. \mathbf{e_1} .. w_j .. \mathbf{e_2} .. w_n$

$\mathbf{e_1}$ and $\mathbf{e_2}$ entities

"The new **iPhone 7 Plus** includes an improved **camera** to take amazing pictures"

Component-Whole(e_1 , e_2) ?

YES / NO

It is also possible to include more than two entities as well:
"At codons 12, the occurrence of point mutations from G to T were observed" → point mutation(**codon**, **12**, **G**, **T**)

Features / Input representation

1) context-wise split of the sentence

Embeddings Left | Embeddings Middle | Embeddings Right

The new **iPhone 7 Plus** includes an improved **camera** that takes amazing pictures

2) word sequences concatenated with positional features

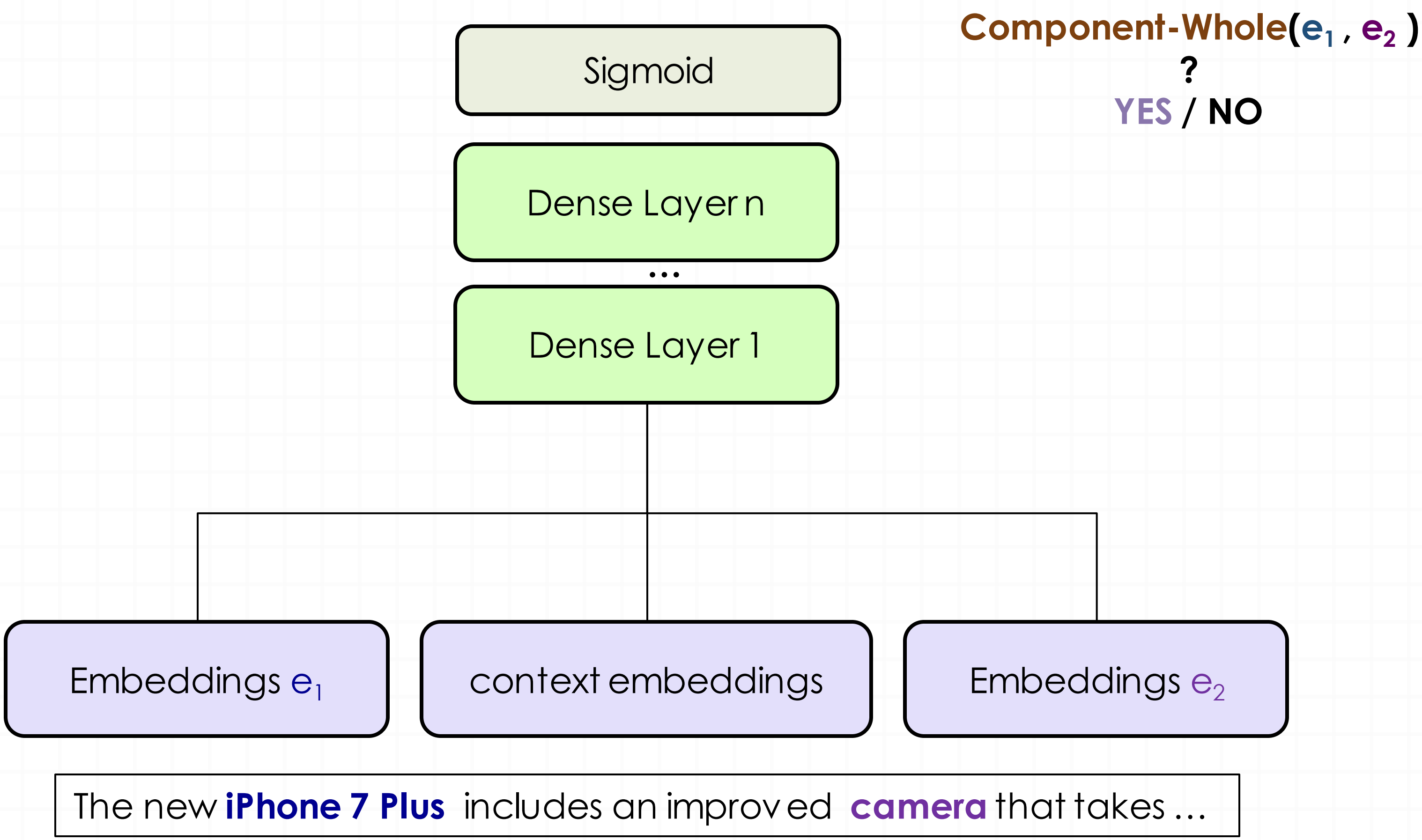
Word indices	Position indices e_1	Position indices $\mathbf{e_2}$
[5, 7, 12, 6, 90 …]	[-1, 0, 1, 2, 3 …]	[-4, -3, -2 -1, 0]
Word Embeddings	Positional emb. e_1	Positional emb. $\mathbf{e_2}$

3) concatenating embeddings of two entities with average of word embeddings for rest of the words

Embeddings e_1 | Embeddings e_2 | context embeddings

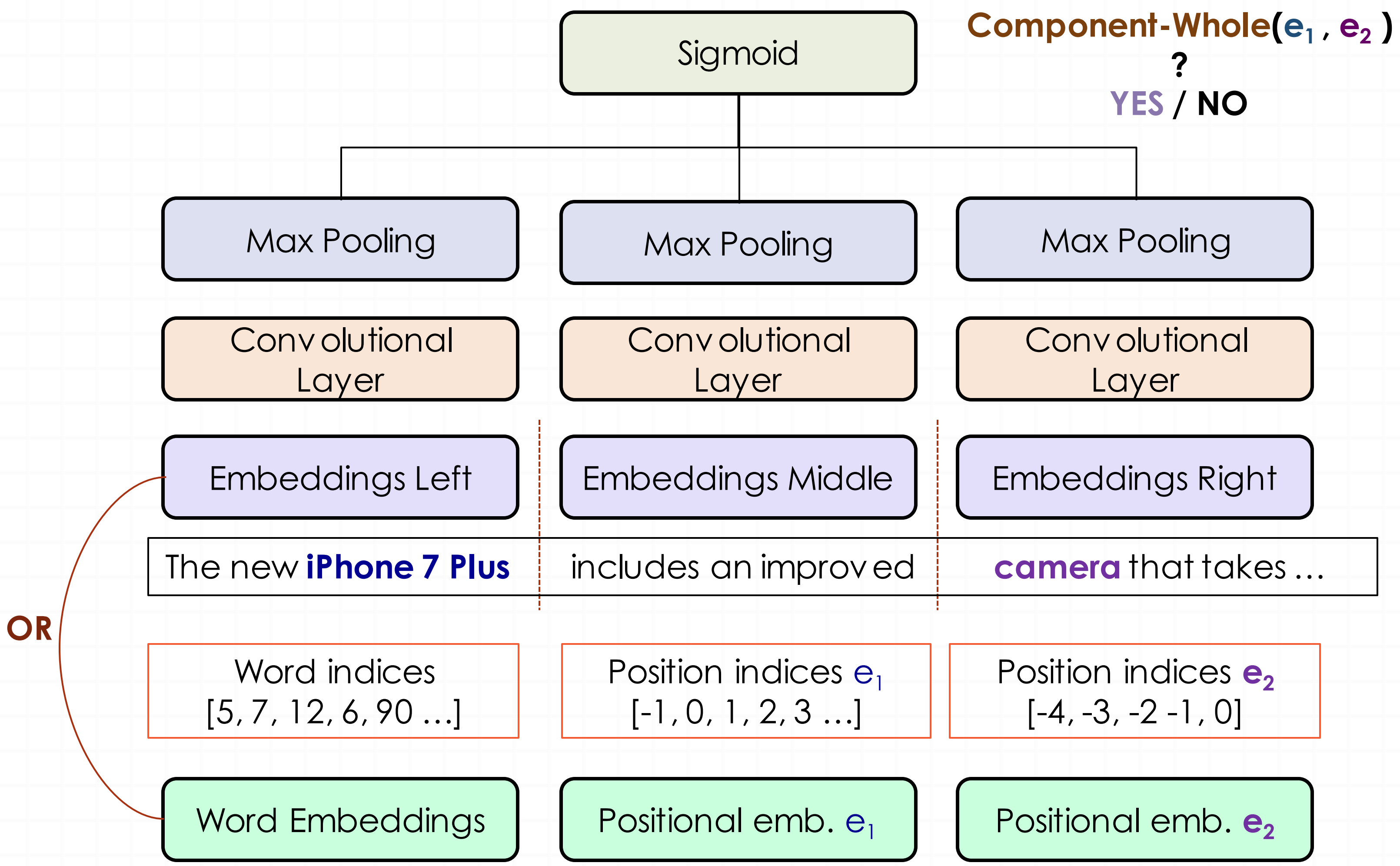
The new **iPhone 7 Plus** includes an improved **camera** that takes amazing pictures

Models: MLP



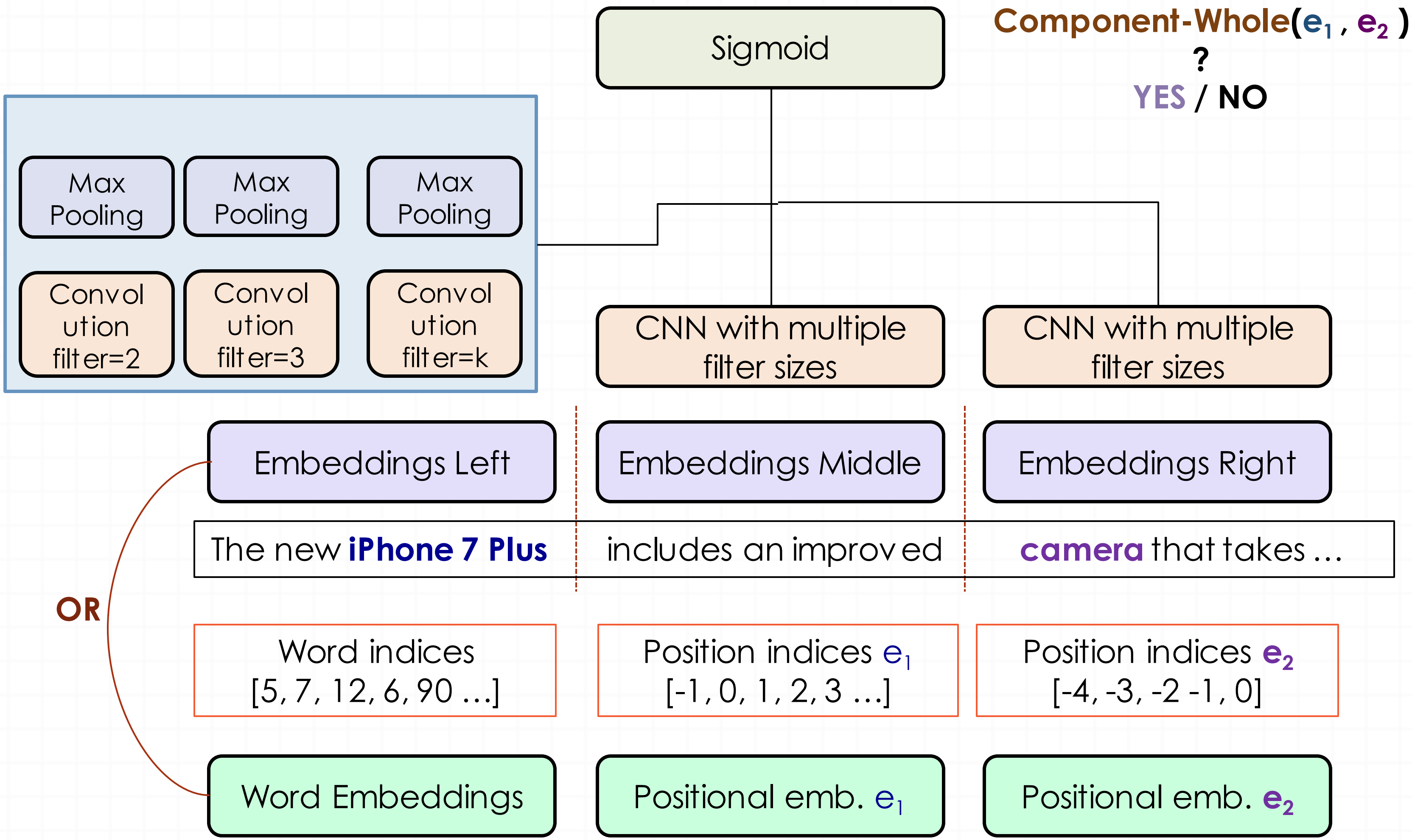
Simple fully-connected multi-layer perceptron

Models: CNN



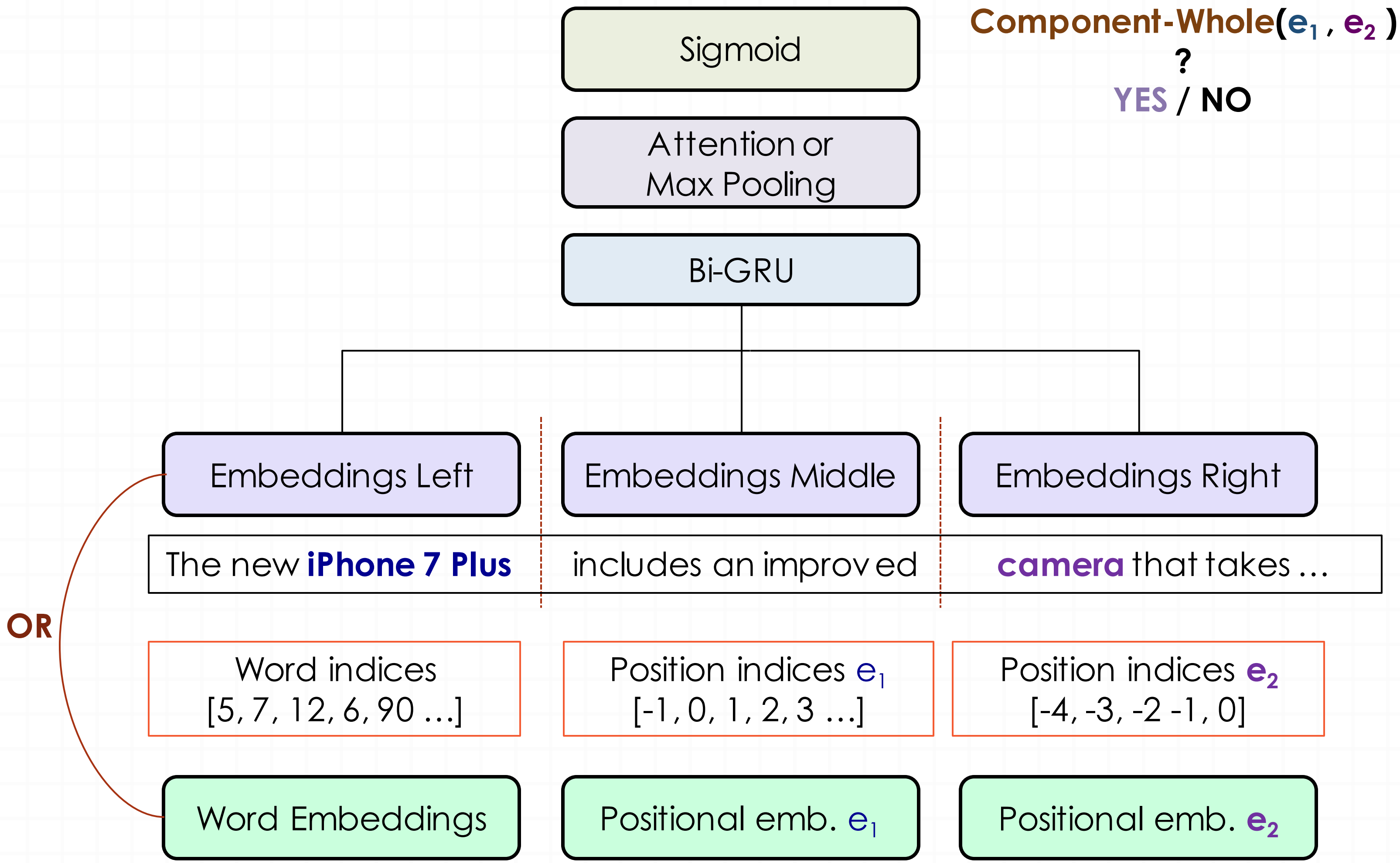
Zeng, D.et al. "Relation classication via convolutional deep neural network".COLING (2014)

Models: CNN (2)



Nguyen, T.H., Grishman, R. "Relation extraction: Perspective from convolutional neural networks." VS@ HLT-NAACL. (2015)

Models: Bi-GRU



Zhang, D., Wang, D. "Relation classication via recurrent neural network." -arXiv preprint arXiv:1508.01006 (2015)
Zhou, P. et al. "Attention-based bidirectional LSTM networks for relation classication. ACL (2016)

Distant Supervision

Circumvent the **annotation** problem – create large dataset

Exploit large knowledge bases to **automatically label** entities and their relations in text

Assumption:

when two entities co-occur in a sentence, a certain relation is expressed

knowledge base

Relation	Entity 1	Entity 2
place of birth	**Michael Jackson**	**Gary**
place of birth	**Barack Obama**	**Hawaii**
…	…	…

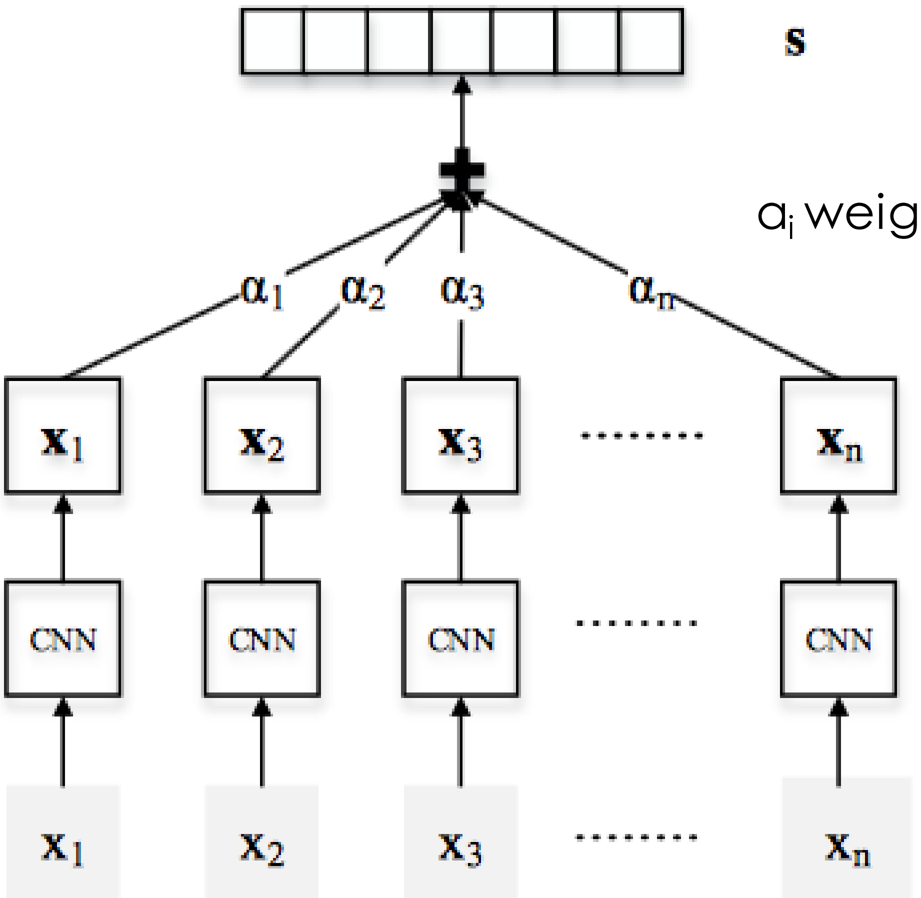
text

Barack Obama moved from **Gary** ….
Michael Jackson met … in **Hawaii**

place of birth

For many ambiguous relations, mere co-occurrence does not guarantee the existence of the relation → Distant supervision produces false positives

Attention over Instances



s representation of the sentence set

α_i weight given by sentence-level attention

$\mathbf{x}_i$ sentence vector representation

x_i sentence for an entity pair (e_1,e_2)
n sentences for relation $r(e_1,e_2)$

Lin et al. "Neural Relation Extraction with Selective Attention over Instances" ACL (2016) [code]

Sentence-level ATT results

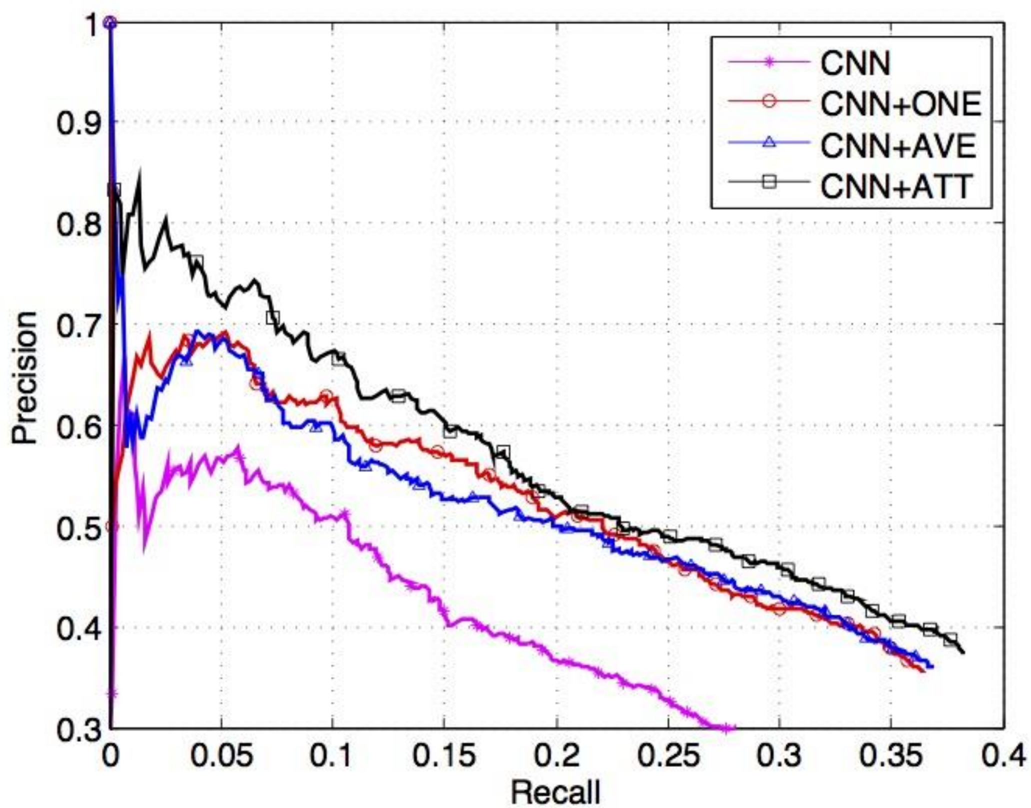
NYT10 Dataset

Align Freebase relations with New York Times corpus (NYT)

53 possible relationships

+NA (no relation between entities)

Data	sentences	entity pairs
Training	522,611	281,270
Test	172,448	96,678



Lin et al. "Neural Relation Extraction with Selective Attention over Instances" ACL (2016) [code]

References

- *Srivastava, Nitish, et al.* "**Dropout: a simple way to prevent neural networks from overfitting.**" *Journal of machine learning research (2014)*
- *Bergstra, James, and Yoshua Bengio.* "**Random search for hyper-parameter optimization.**" *Journal of Machine Learning Research, Feb (2012)*
- *Kim, Y.* "**Convolutional Neural Networks for Sentence Classification**", *EMNLP (2014)*
- Severyn, Aliaksei, and Alessandro Moschitti. "**UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification.**" *SemEval@NAACL-HLT* (2015)
- *Cho, Kyunghyun, et al.* "***Learning phrase representations using RNN encoder-decoder for statistical machine translation***." *EMNLP (2014)*
- Ilya Sutskever et al. "**Sequence to sequence learning with neural networks.**" *NIPS* (2014)
- *Bahdanau et al.* "***Neural machine translation by jointly learning to align and translate.***" *ICLR (2015)*
- *Gal, Y., Islam, R., Ghahramani, Z.* "***Deep Bayesian Active Learning with Image Data***." *ICML (2017)*
- *Nair, V., Hinton, G.E.* "***Rectified linear units improve restricted boltzmann machines***." *ICML (2010)*
- Ronan Collobert, et al. "**Natural language processing (almost) from scratch**." *JMLR* (2011)

- *Kumar, Shantanu. "A Survey of Deep Learning Methods for Relation Extraction." arXiv preprint arXiv:1705.03645 (2017)*
- *Lin et al. "Neural Relation Extraction with Selective Attention over Instances" ACL (2016)* [code]
- *Zeng, D.et al. "Relation classification via convolutional deep neural network". COLING (2014)*
- *Nguyen, T.H., Grishman, R. "Relation extraction: Perspective from CNNs." VS@ HLT-NAACL. (2015)*
- *Zhang, D., Wang, D. "Relation classification via recurrent NN." -arXiv preprint arXiv:1508.01006 (2015)*
- *Zhou, P. et al. "Attention-based bidirectional LSTM networks for relation classification . ACL (2016)*
- Mike Mintz et al. "Distant supervision for relation extraction without labeled data." *ACL- IJCNLP (2009)*

References & Resources

- http://web.stanford.edu/class/cs224n
- https://www.coursera.org/specializations/deep-learning
- https://chrisalbon.com/#Deep-Learning
- http://www.asimovinstitute.org/neural-network-zoo
- *http://cs231n.github.io/optimization-2*
- *https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39*
- https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning
- http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow
- http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp
- https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8
- *http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/*
- *http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/*
- *http://colah.github.io/posts/2015-08-Understanding-LSTMs*
- https://github.com/hyperopt/hyperopt
- https://github.com/tensorflow/nmt

https://giphy.com/gifs/thanks-thank-you-thnx-3o6ozuHcxTtVWJJn32/download