Deep Learning for NLP

When Big Data hits Machine Learning

Nils Reimers

Course-Website: www.deeplearning4nlp.com

What is Deep Learning?

- Deep Learning is a subfield of Machine Learning
- Most machine learning methods work well because of human-designed features
  - Recasens et al. (2009) lists >30 features for event coreference resolution
  - Most time spent on engineering features
- Machine learning becomes optimizing weights to make a final prediction given the features

Feature	Definition
PRON_m_1	m_1 is a pronoun
PRON_m_2	m_2 is a pronoun
HEAD_MATCH	Head match
WORDNET_MATCH	EuroWordNet match
NP_m_1	m_1 NP type
NP_m_2	m_2 NP type
NE_m_1	m_1 NE type
NE_m_2	m_2 NE type
NE_MATCH	NE match
SUPERTYPE_MATCH	Supertype match
GENDER_AGR	Gender agreement
NUMBER_AGR	Number agreement
ACRONYM	m_2 is an acronym of m_1
QUOTES	m_2 is in quotes
FUNCTION_m_1	m_1 function
FUNCTION_m_2	m_2 function
COUNT_m_1	m_1 count
COUNT_m_2	m_2 count
SENT_DIST	Sentence distance
MENTION_DIST	Mention distance
WORD_DIST	Mention distance

Machine Learning vs. Deep Learning

Traditional Machine Learning

Feature Engineering: Describe your data with features a computer can understand	Machine Learning: Some hyper-parameter tuning

- Task, language and text-domain specific
- Requires high domain expertise (Ph.D.-level)
- Often requires other tools and resources (POS-tagger, Wordnet …)

Deep Learning Approach

Getting Domain Expertise	Design / select a suitable network architecture	Optimize architecture & fine-tune parameters

Deep Learning

Deep Learning

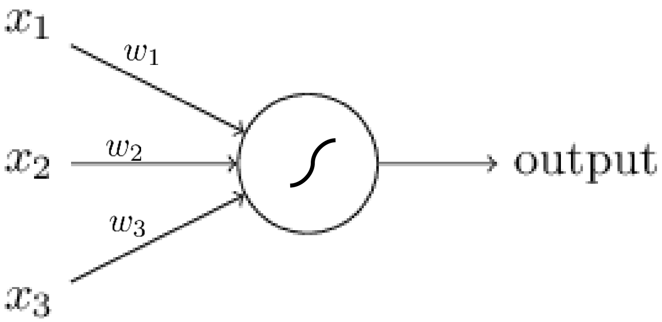
- *Representation Learning* attempts to automatically learn good features or representations
- *Deep Learning* attempts to learn multiple levels of representation from **raw input**

- Instead of modeling the problem by the design of features, deep learning attempts to learn by itself a good representation (i.e. features).
- Large amounts of data are typically needed
  - They must not necessarily be labeled (see autoencoders, word embeddings)
  - For small datasets: Hand-designed features can still be included

Neural Network Basics

Neural Network Basics

- Given several **inputs**: $x_1, x_2, x_3, ... \in \mathbb{R}$
  and several **weights**: $w_1, w_2, w_3, ... \in \mathbb{R}$
  and a **bias** value: $b \in \mathbb{R}$



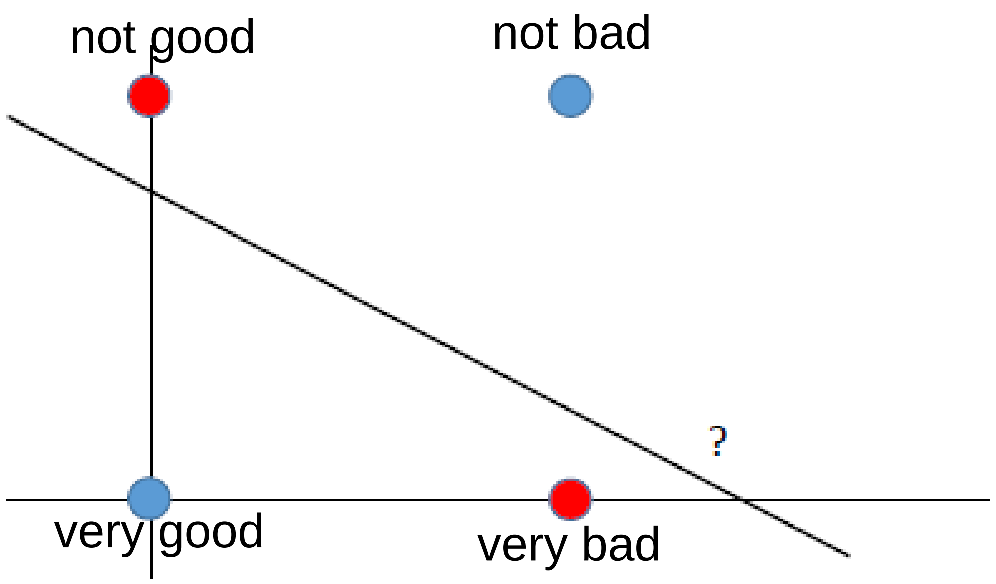
- A neuron produces a single output:

$$o_1 = s(\sum_i w_i x_i + b)$$

- This sum $\sum_i w_i x_i + b$ is called the **activation** of the neuron
- The function *s* is called the **activation function** for the neuron
- The weights and bias values are typically initialized randomly and learned during training

Activation Function & Non-linearity

- The critical part is the non-linearity
- Example: Create a classifier, that learns the XOR function
- No linear classifier can solve this problem



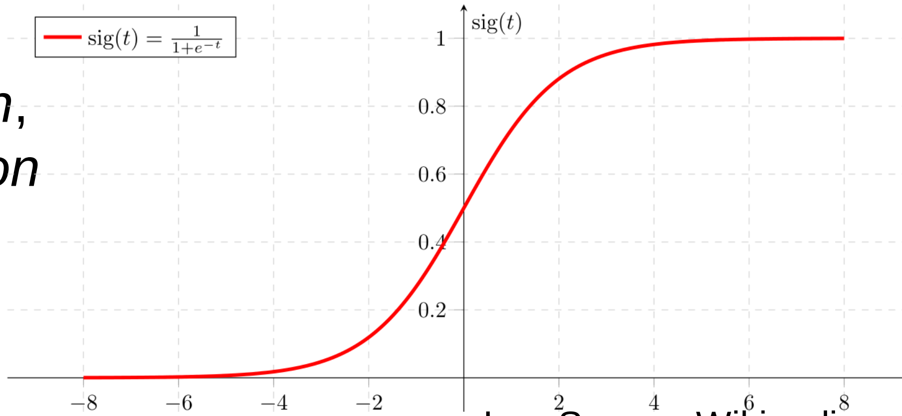
- "not bad" != "not" + "bad"

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 impossible to model certain combinations of features (like Boolean XOR function), unless we do manual feature engineering

- Typical non-linear activation functions for neural nets are the *sigmoid function*, the *hyperbolic tangent* or a *step function*

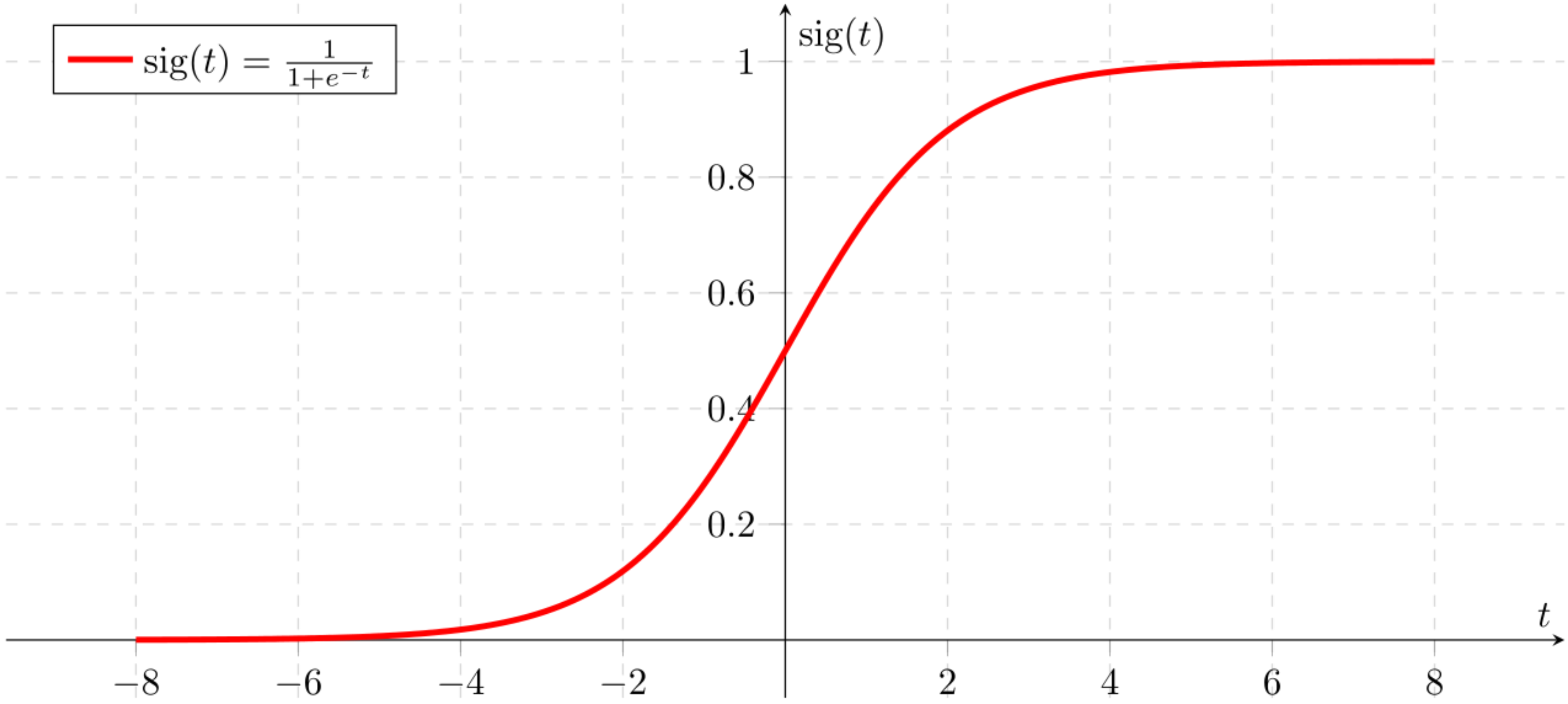


Img-Source: Wikipedia

Typical Activation Function

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sigmoid Function scales between 0 and 1

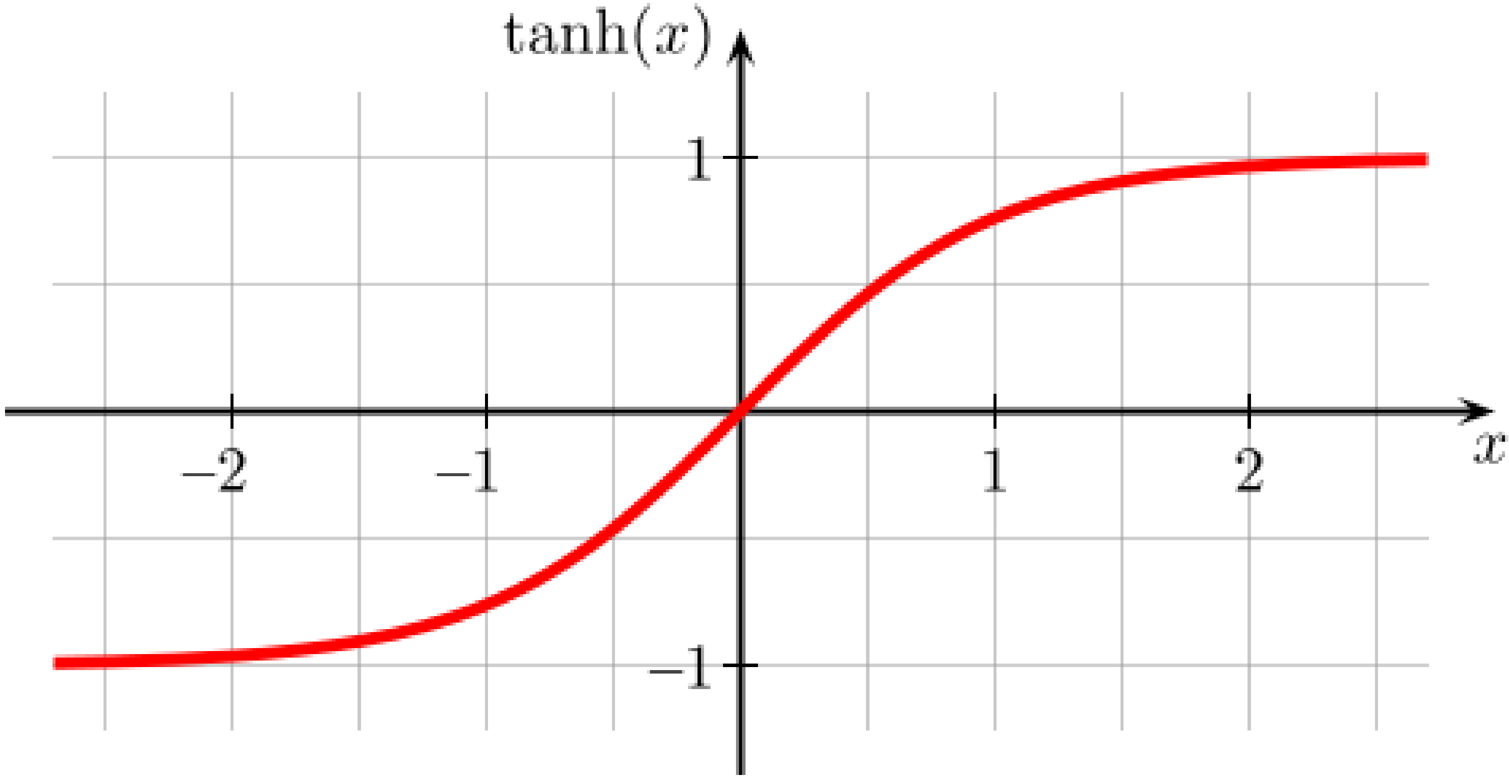


Img-Source: Wikipedia

UKP

Typical Activation Function

- *hyperbolic tangent* scales between -1 and 1



Img-Source: Wikipedia

Typical Activation Function

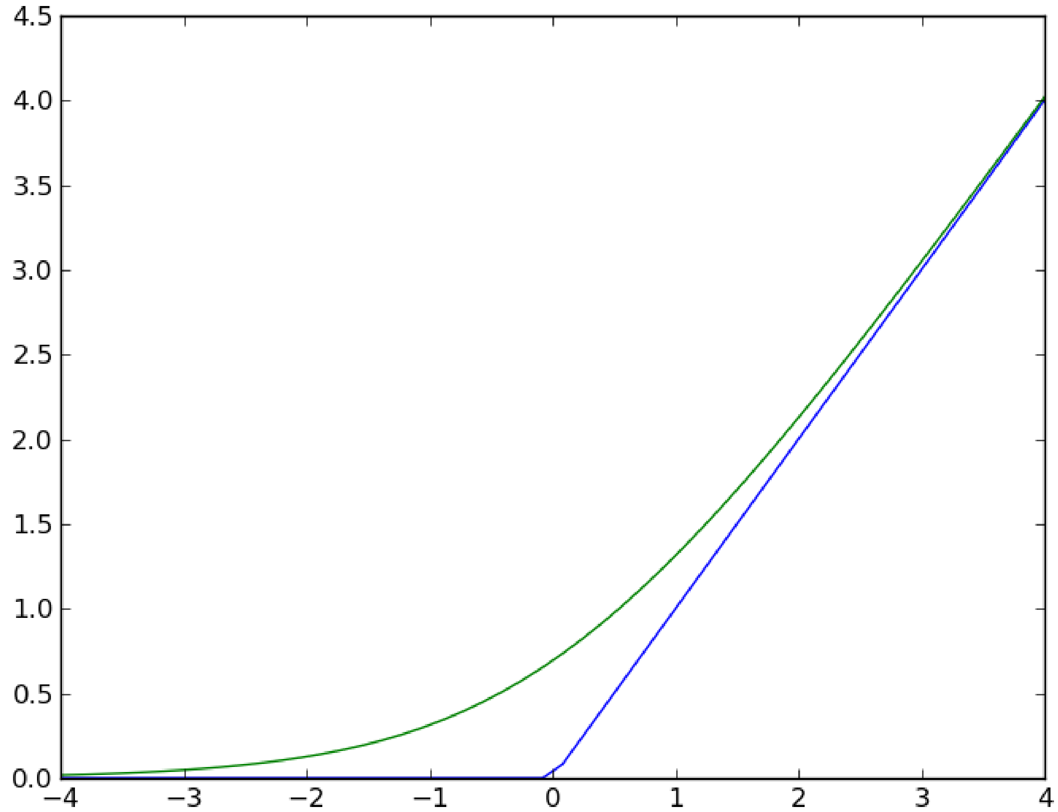
- Rectifier (blue):

$$f(x) = \max(0, x)$$

- Smooth approximation (green):

$$f(x) = \ln(1 + e^x)$$

- A unit using the rectifier function is called rectified linear unit (ReLU)

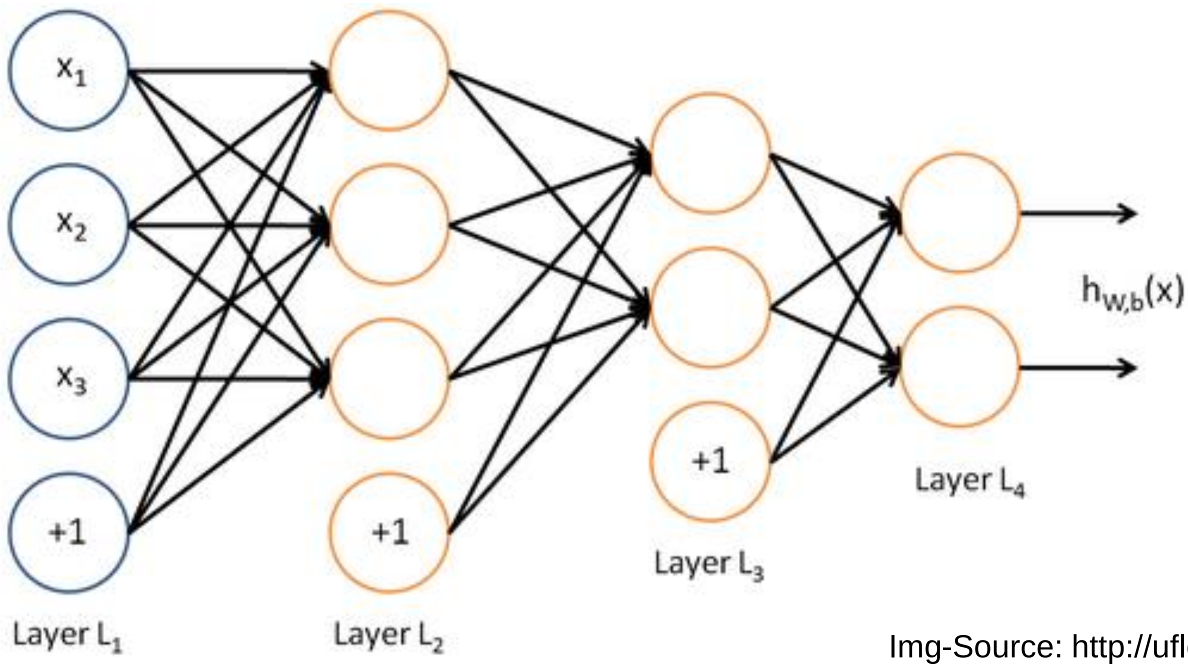


Img-Source: Wikipedia

Which Activation Function to use?

- Most papers use the *tanh* or ReLU
- For certain problems ReLU can be useful, especially for convolutional neural networks
  - Be careful with your input data, the activation can become arbitrarily large -> “*one input dimension could rule them all”*

Feed Forward Neural Networks



Img-Source: http://ufldl.stanford.edu/wiki/

Feed Forward Networks

- The most simple network architecture
- Information flows only in one direction
- Input Layer: Represents my data
- Output layer: Represents possible labels

Hidden-Layer

- The hidden layer (L_2, L_3) represent learned non-linear combination of input data
- For solving the XOR problem, we need a hidden layer
  - some neurons in the hidden layer will activate only for some combination of input features
  - the output layer can represent combination of the activations of the hidden neurons
- Neural network with one hidden layer **is a universal approximator**
  - Every function can be modeled as a shallow feed forward network
  - Not all functions can be represented *efficiently* with a single hidden layer
    $\Rightarrow$ we still need deep neural networks

Matrix Notation

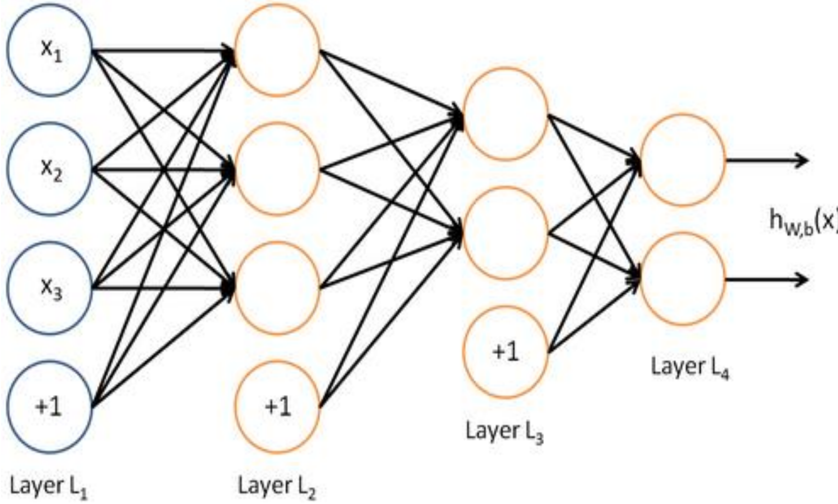
TECHNISCHE UNIVERSITÄT DARMSTADT

- Given 3-dimensional input $x \in \mathbb{R}^3$
- Given first weight matrix: $W_1 \in \mathbb{R}^{3\times 3}$
- Given first bias vector: $b_1 \in \mathbb{R}^3$

- Given second weight matrix: $W_2 \in \mathbb{R}^{3\times 2}$
- Given second bias vector: $b_2 \in \mathbb{R}^2$

- Given third weight matrix: $W_3 \in \mathbb{R}^{2\times 2}$
- Given third bias vector: $b_3 \in \mathbb{R}^2$

- Computation L_2: $l_2 = \tanh(W_1 x + b_1)$
- Computation L_3: $l_3 = \tanh(W_2 l_2 + b_2)$
- Computation L_4: $l_4 = \mathrm{softmax}(W_3 l_3 + b_3)$



Img-Source: http://ufldl.stanford.edu/wiki/

Softmax-Classifier

- For classification, most networks apply a softmax-classifier as final layer
- Softmax regression is a generalization of logistic regression to the case of multiple classes

- Given we have *K* classes (*K* = number of output units). Compute the activation *z* for the last layer:

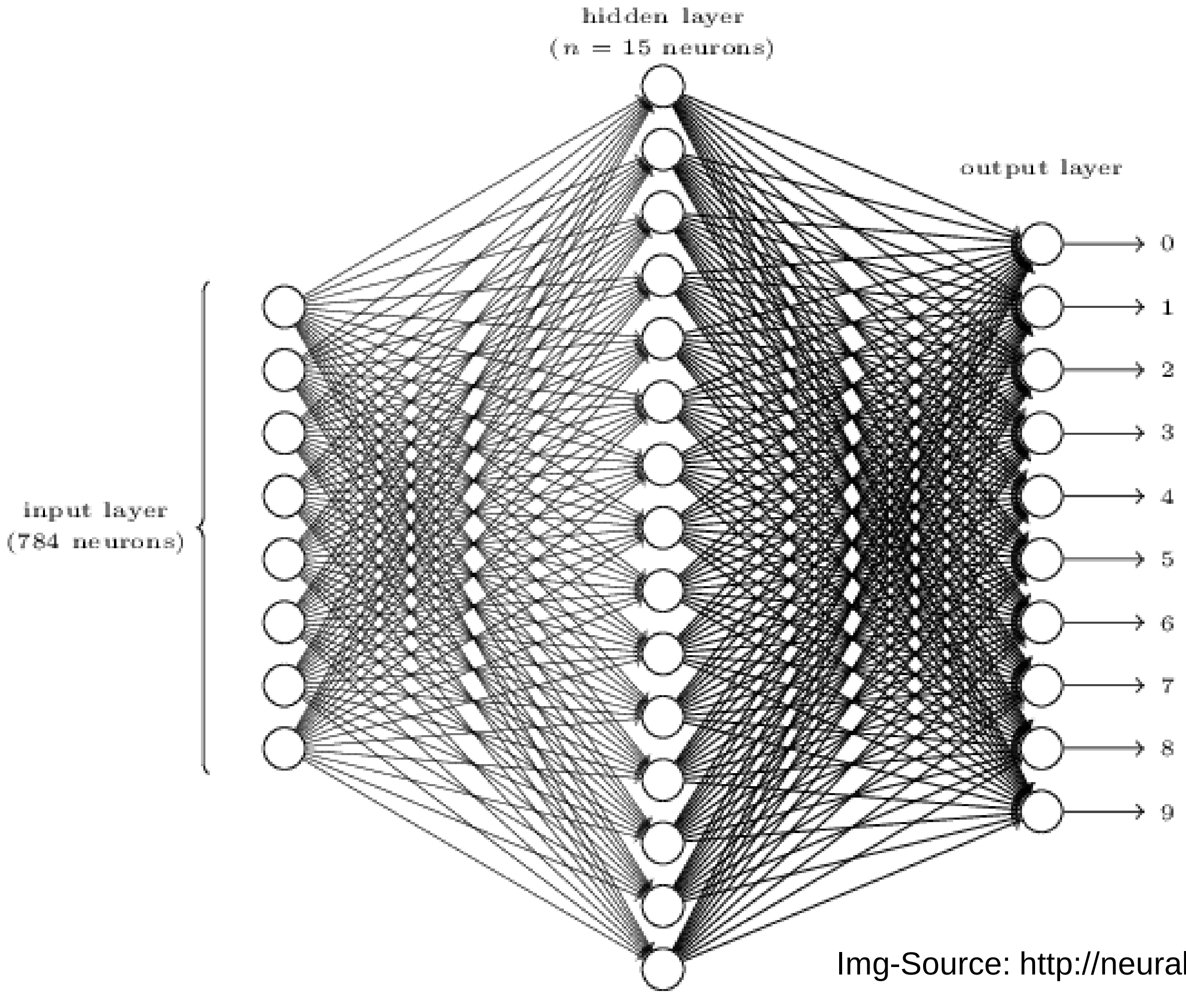
$$z = W_3 l_3 + b_3 \in \mathbb{R}^K$$

Compute the final output *y:*

$$y_j = \mathrm{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- y_j can have values between 0 and 1
- y_j sums up to 1 -> It can be interpreted as probability distribution

Feed-Forward Network for Handwritten Digit Recognition



Img-Source: http://neuralnetworksanddeeplearning.com

Hidden Layers

Rule of Thumb

- The number and the size of the hidden layers can have a large impact on the performance
- More hidden layers $\Rightarrow$ more parameters to learn $\Rightarrow$ more data needed
- Start with a small number of hidden layers, i.e. with 1
- Increase number of hidden layers stepwise until you find an optimum
- Typically decreasing sizes for the hidden layers, for example
  2000 dim $\Rightarrow$ 1500 dim $\Rightarrow$ 750 dim $\Rightarrow$ 100 dim $\Rightarrow$ 10 dim

Initialization of Weights and Bias

- The weights and bias vectors are the parameters that we learn during training
- The bias is typically initialized to 0
- The weights are randomly initialized – Use Glorot-style uniform initialization
  - Xavier Glorot & Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks* (http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf)
  - Explanation of the algorithm: http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization
  - Implementation / Usage in Theano & Keras is straight forward
  - Be careful: The maximal size of the weights is different for the *tanh* and the *sigmoid* activation function.

Training Neural Networks

Img-Source: https://www.flickr.com/photos/jakerust/

Back Propagation

Defining an Error Function

- First we need an error function which computes the difference between the output of the network and the expected output (true label)

- In case of single label classification, we can use the negative log-likelihood:

$$E(x, W, b) = -\log(o_y)$$

  with *y* denoting the expected label and *o* the output vector.

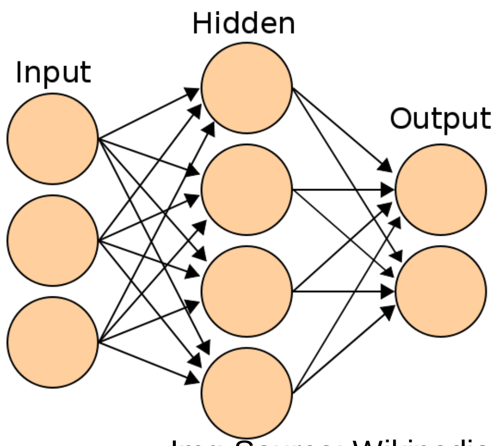
- In case of a distributional classification, we can use the mean squared error:

$$E(x, W, b) = \frac{1}{2}\sum(y_i - o_i)^2$$

  with y_i denoting the expected value for node *i*

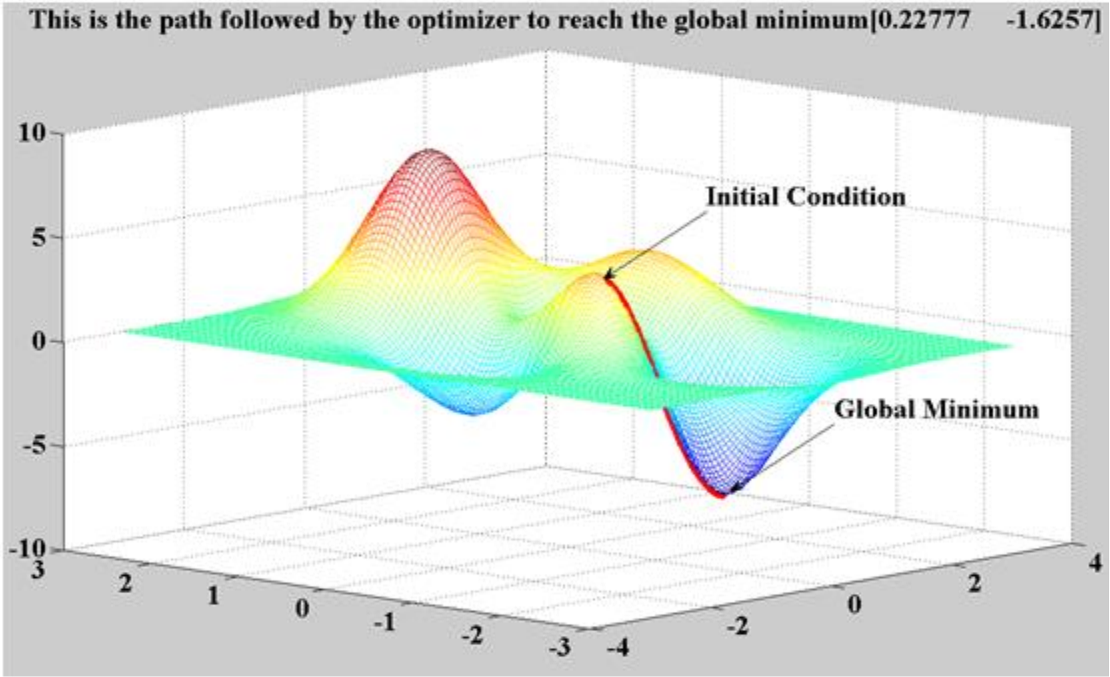
- We then want to minimize the error function

- Defining the error function is an **important aspect** in designing a deep neural network
  - Changing the error function changes what our network learns



Img-Source: Wikipedia

Back Propagation – Gradient Descent



Source: mathworks.de

Gradient Descent

- We want to minimize the error function by tuning the trainable parameters (weights and biases)
- The gradient (the derivative of a multi dim. function) points us towards a local minima
- We follow the gradient by a certain step length. This length is called *learning rate*

Computation of Gradients in Theano/TF

- Computation of the gradient (the derivative of a multi dimensional function) can be cumbersome
  - Especially as the computation must be computationally efficient, as we will perform billions of such computations

- Theano/TensorFlow is great, as it provides us automatic gradient computation
  - We don't need to compute the derivative
  - And we don't need to implement it into our program code
  - -> Awesome!

- Next slides look a bit complicated, but doing back propagation via stochastic gradient descent (SGD) is quite simple in Theano/TensorFlow

Back Propagation – Gradient Descent

Training with back propagation:

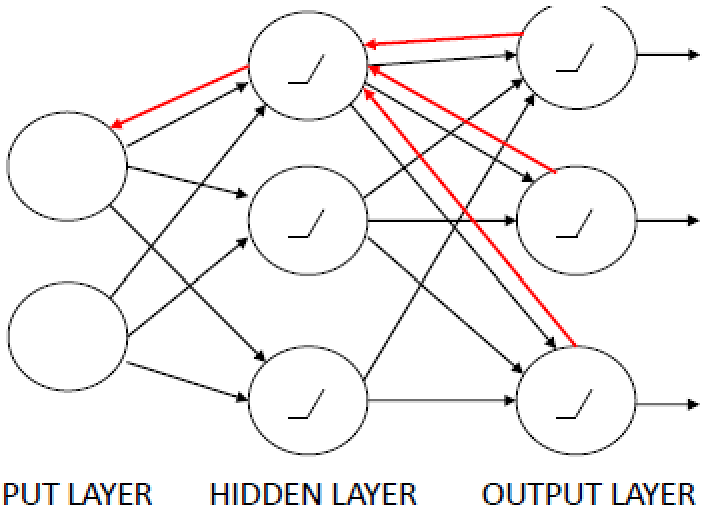
1. Given the input data, we compute the values for the output neurons
2. Compare the output to the gold labels – compute error function
3. Compute the derivative for all tunable parameters (weights and parameters)
4. Update the parameters:

$$W^{(i)} := W^{(i)} - \lambda \frac{\partial}{\partial W^{(i)}} E(x, W, b)$$

$$b^{(i)} := b^{(i)} - \lambda \frac{\partial}{\partial b^{(i)}} E(x, W, b)$$

λ is denoting the learning rate

5. Each iteration is called an *epoch*. With each epoch, the error function decreases, converging to a local minima

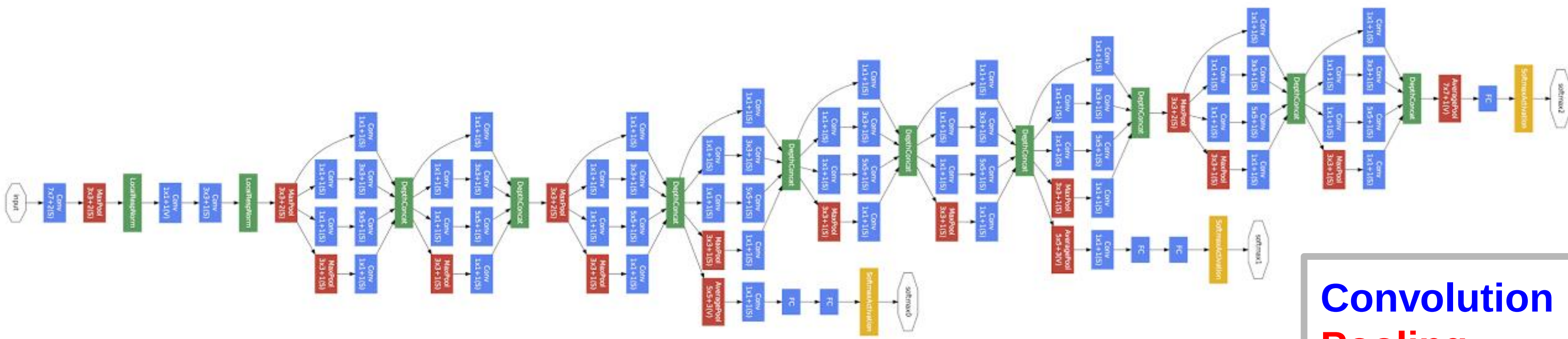


Source: Mikolov, 2014

Motivation for Deep Learning

- Feed forward networks with a single hidden layer can compute any function $\Rightarrow$ in theory, no need for deep architectures
- However, learning shallow architectures is not always efficient
- To learn the parity function (N bits at input, output is 1 if the number of active input bits is odd) requires an exponential number of hidden units $\Rightarrow$ requires exponentially more training data
- See the 2009 Yoshua Bengio Book – *Chapter 2 Theoretical Advantages of Deep Architectures for more information*
  - http://www.iro.umontreal.ca/%7Ebengioy/papers/ftml_book.pdf

What does a Deep Network can look like?

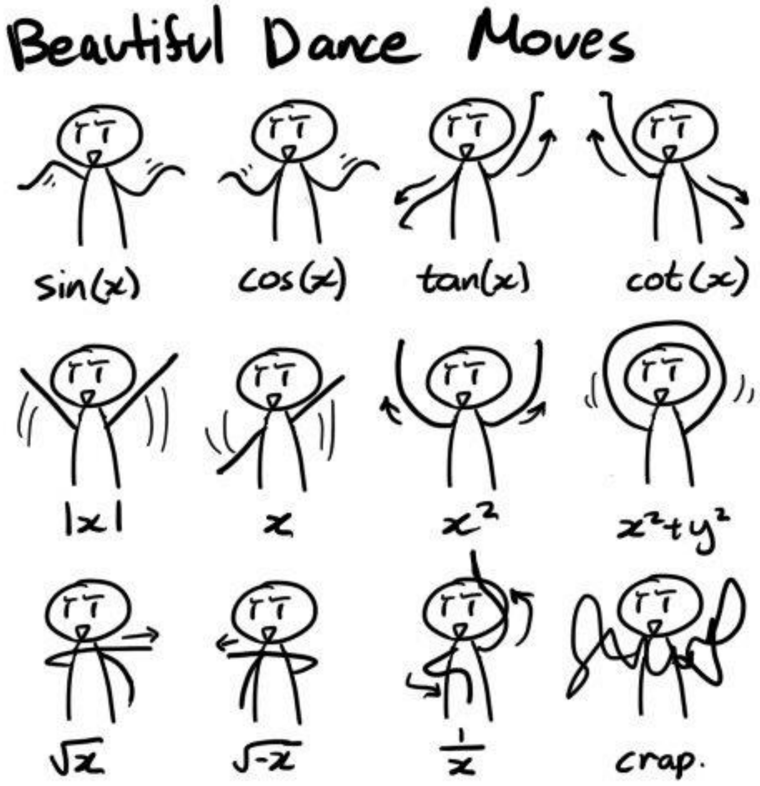


Convolution
Pooling
Softmax
Other

- Google's Entry for the 2014 ImageNet Challenge
- "Just" 5 million parameters – 20MB model size
- Uses ReLU (sigmoid/tanh does not work in really deep networks)
  - See Glorot et al., 2011, *Deep Sparse Rectifier Neural Networks*

Source: Szegedy et al., 2014, Going Deeper with Convolutions

Everything is a Vector



Source: https://www.flickr.com/photos/dylanng/

Vectors in Deep Learning

- Deep Learning loves dense vectors
- Everything is represented and understood as a vector
- Vectors allows an end-to-end-training
- No more in-between mapping to tag sets etc.
  - No problem with modeling hard cases and ambiguities
  - No problem with error propagation between pipeline components

Dense Vector

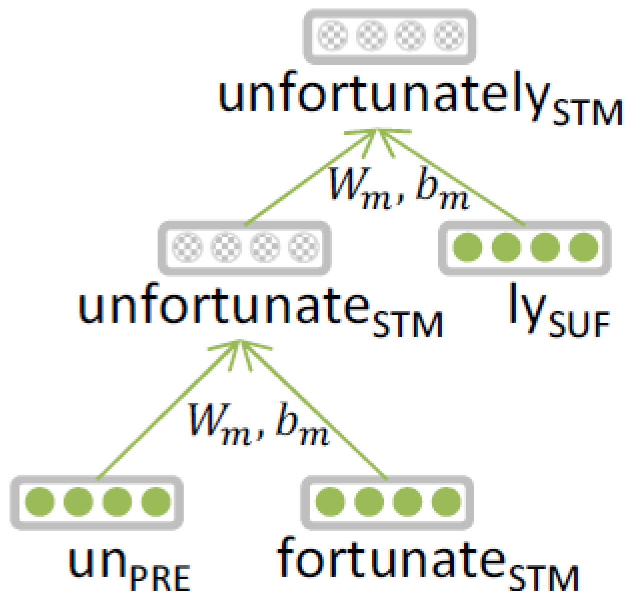
- In a dense vector most values are non-zero

Representations at NLP Levels: Morphology

- Traditional: Morphemes

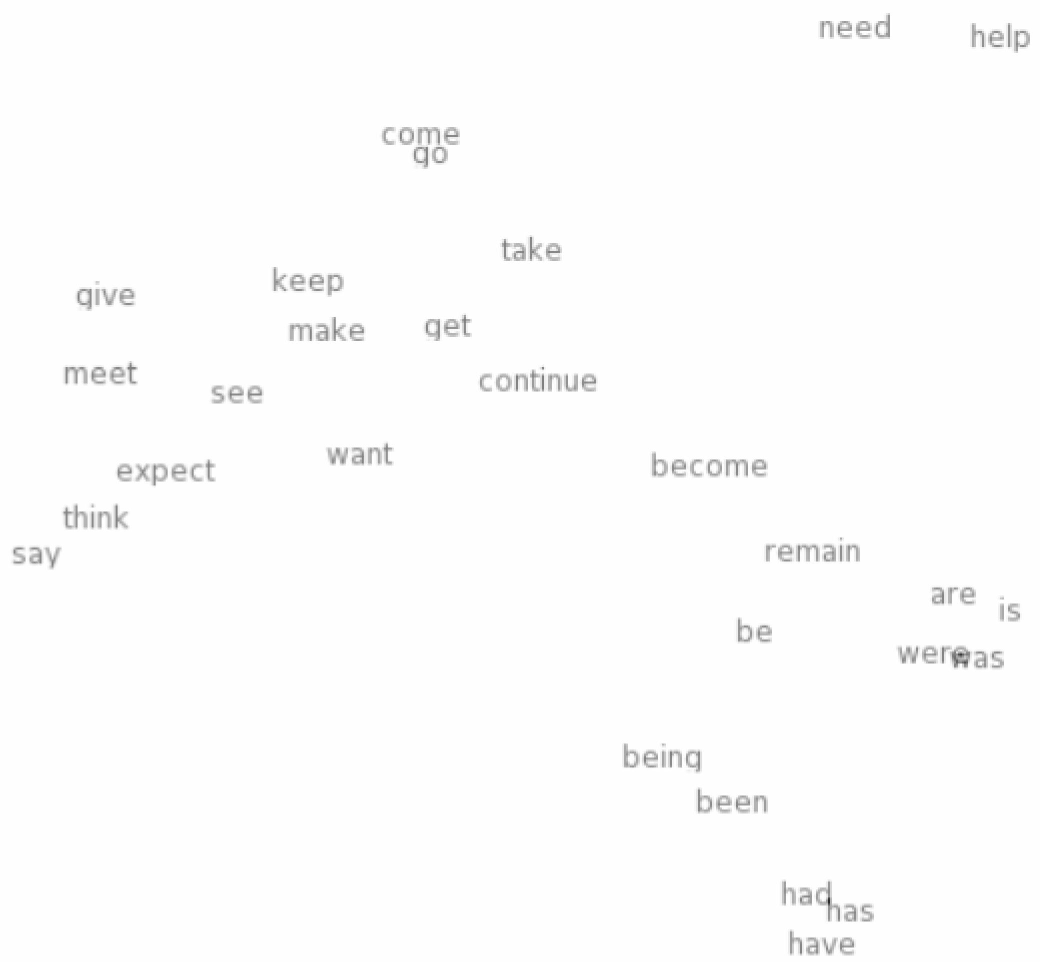
prefix	stem	suffix
un	interest	ed

- DL:
  - every morpheme is a vector
  - a neural network combines two vectors into one vector
  - Thang et al. 2013



Source: Richard Socher, CS224d, http://cs224d.stanford.edu/syllabus.html

Neural word vectors - visualization

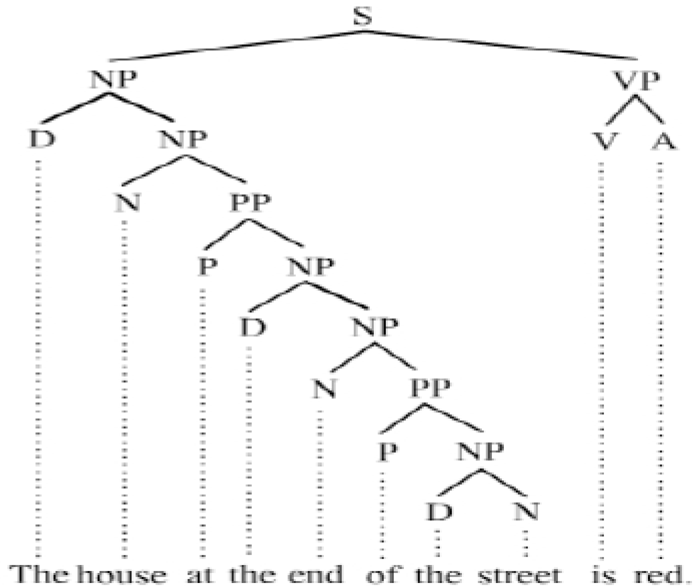


Source: Richard Socher, CS224d, http://cs224d.stanford.edu/syllabus.html

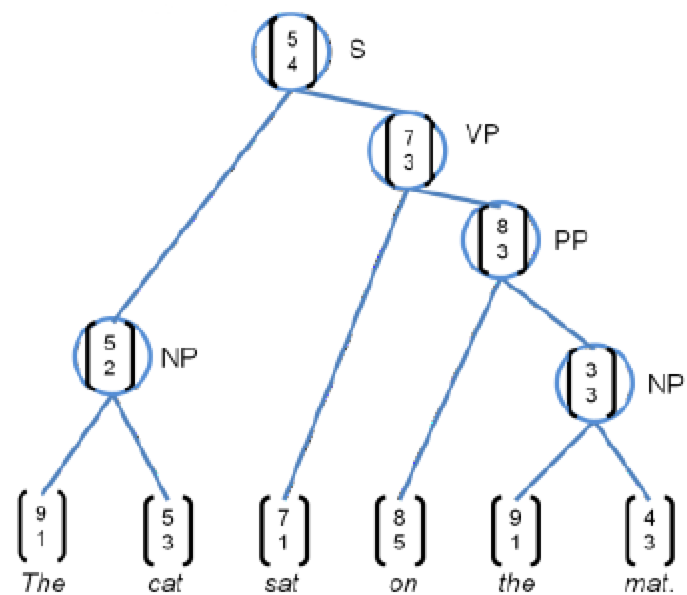
Source: Richard Socher, CS224d, http://cs224d.stanford.edu/syllabus.html

Representations at NLP Levels: Syntax

- Traditional: Phrases
  Discrete categories like NP, VP



- DL:
  - Every word and every phrase is a vector
  - a neural network combines two vectors into one vector
  - Socher et al. 2011



Question Answering

- Common: A lot of feature engineering to capture world and other knowledge, e.g. regular expressions, Berant et al. (2014)

Is main verb trigger?

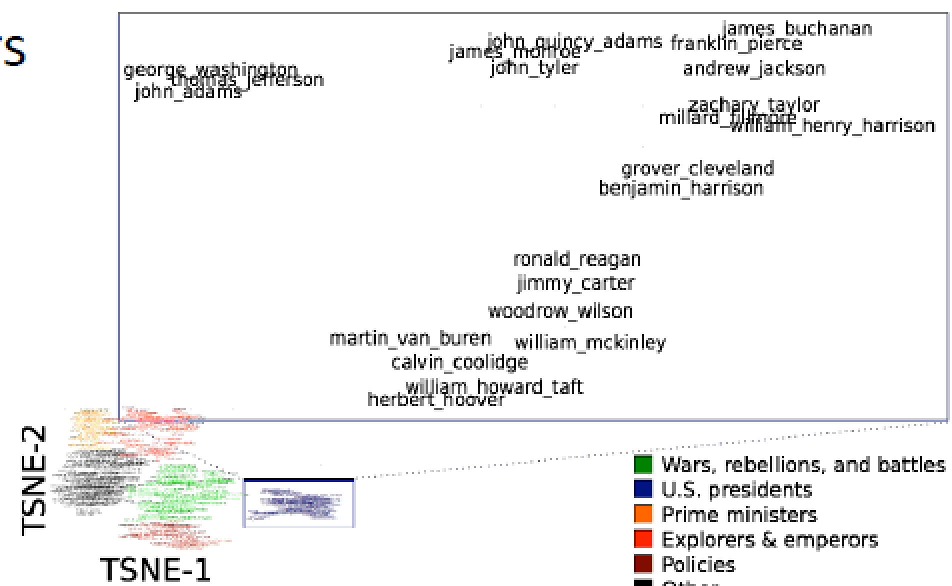
Yes

Condition	Regular Exp.
Wh- word subjective?	AGENT
Wh- word object?	THEME

No

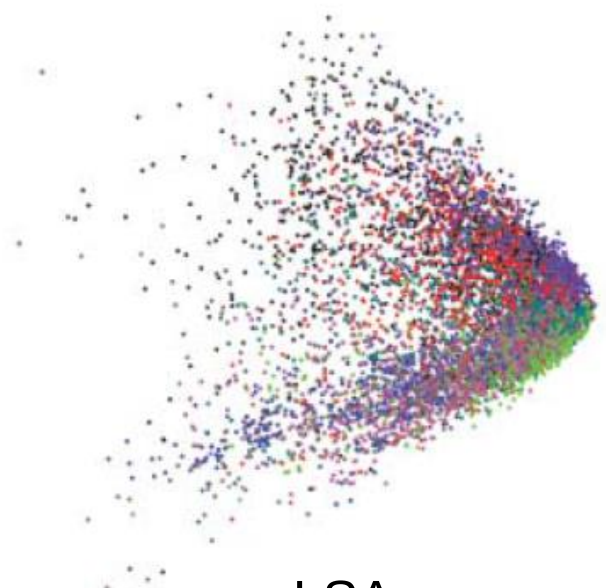
Condition	Regular Exp.
default	(ENABLE|SUPER)$^{+}$
DIRECT	(ENABLE|SUPER)
PREVENT	(ENABLE|SUPER)* PREVENT(ENABLE|SUPER)*

- DL: Same deep learning model that was used for morphology, syntax, logical semantics and sentiment can be used!
- Facts are stored in vectors

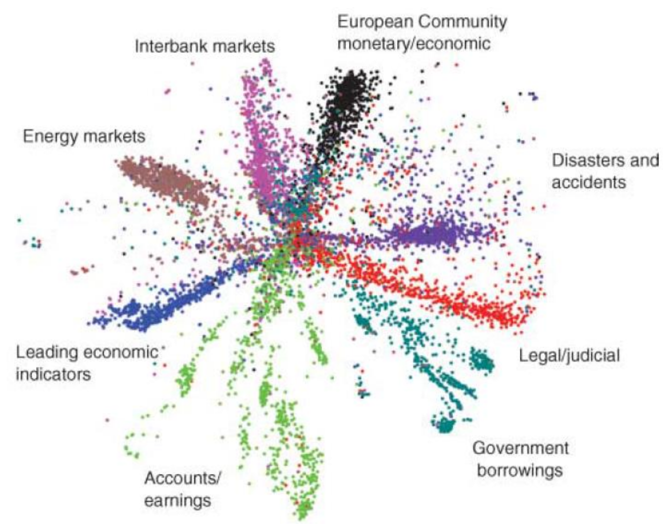


Learning Multiple Levels of Representation

- Biologically inspired: The brain has a deep architecture
- Instead of solving a task directly, we learn intermediate representations
- Highly non-linear properties can be captured this way
- Insufficient model depth can be exponentially inefficient



LSA



Deep Autoencoder

Source: Hinton et al., 2006