

Neural Networks for NLP/ Text Mining

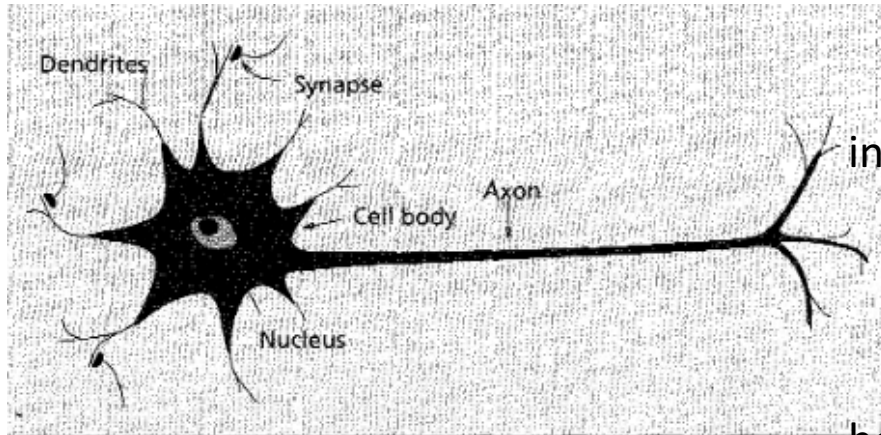
M. Vazirgiannis

March 2016

Outline

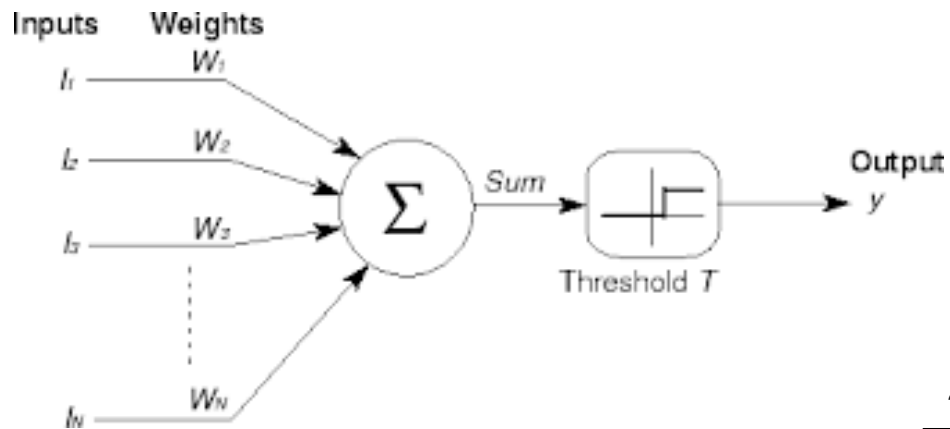
- **Neural Networks - Introduction**
- CNNs
- RNNs
- Gaussian Document Representation
from Word Embeddings (EACL 2017 paper)

Neural Networks



- Neuron: biological cell that processes information
- composed of a cell body, the axon and the dendrites.
- Cell body contains information about hereditary traits and a plasma that holds the molecular equipment for producing material needed by the neuron.
- receives signals (impulses) from other neurons through its *dendrites* (receivers)
- transmits signals generated by its cell body along the *axon* (transmitter). At the terminals of these strands are the synapses.
- *synapse* functional unit between two neurons (an axon strand and a dendrite)

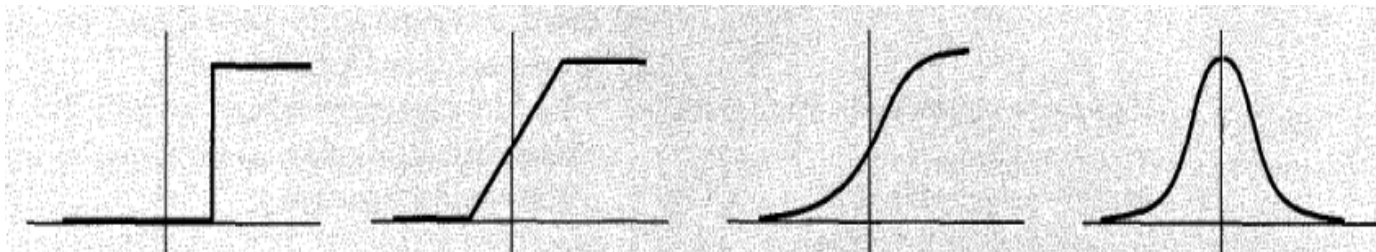
Neural networks



McCulloch-Pitts Neuron Model

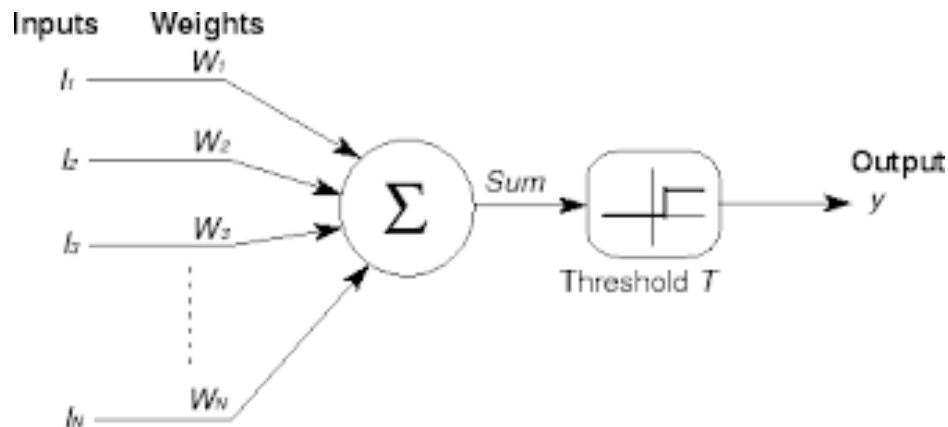
- θ is a step function at 0
- w_j : the weight of the j -th input
- activation functions: piecewise linear, sigmoid, or Gaussian,

$$y = \theta \sum_{j=1}^n w_j X_j - u$$



Neural Network architectures

- Weighted directed graphs in which artificial neurons are nodes and directed edges (with weights) are connections between neuron outputs and inputs.
- Based on the connection pattern NNs can be grouped into
 - feed-forward networks, in which graphs have no loops
 - recurrent (or feedback) networks, in which loops occur because of feedback connections.

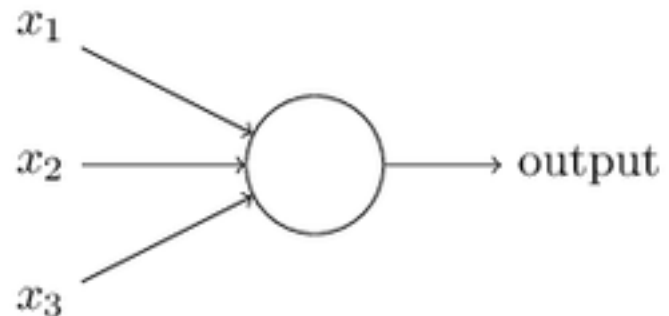


Perceptron learning algorithm

- developed in the 1950 -60 by Frank Rosenblatt
- takes several binary inputs, x_1, x_2, \dots and produces a single binary output

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

- device that makes decisions by weighing up evidence.
- varying the weights and the threshold, we can get different models of decision-making



Neural Networks – Perceptron Learning

- Initialize the weights and threshold (small random numbers)
- Present an input vector $X = \{x_1, \dots, x_n\}$ and evaluate the output of the neuron
$$y = \sum_{j=1}^n w_j X_j - u$$
- Update the weights $w_{t+1} = w_t + \alpha(d - y)X$ where α : learning rate, d : the correct output.

$$\sum_{j=1}^n w_j X_j - u = 0 \quad \text{defines the class separation hyperplane}$$

- learning occurs only when the perceptron makes an error.
- Perceptron convergence theorem (Rosenblatt, 1962)

If training data are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations.

Neural Networks – sigmoid neuron


- In perceptron small changes to the weights may cause huge difference to the output (0->1)
- We want to impose: small changes to weights (or bias) small change to the output.
- the sigmoid neuron has weights for each input, w_1, w_2, \dots , and an overall bias, b . But the output is not 0 or 1. Instead, it's $\sigma(w \cdot x + b)$, where σ is the *sigmoid function*

- *The output of the neuron is:*

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

Learning with gradient descent

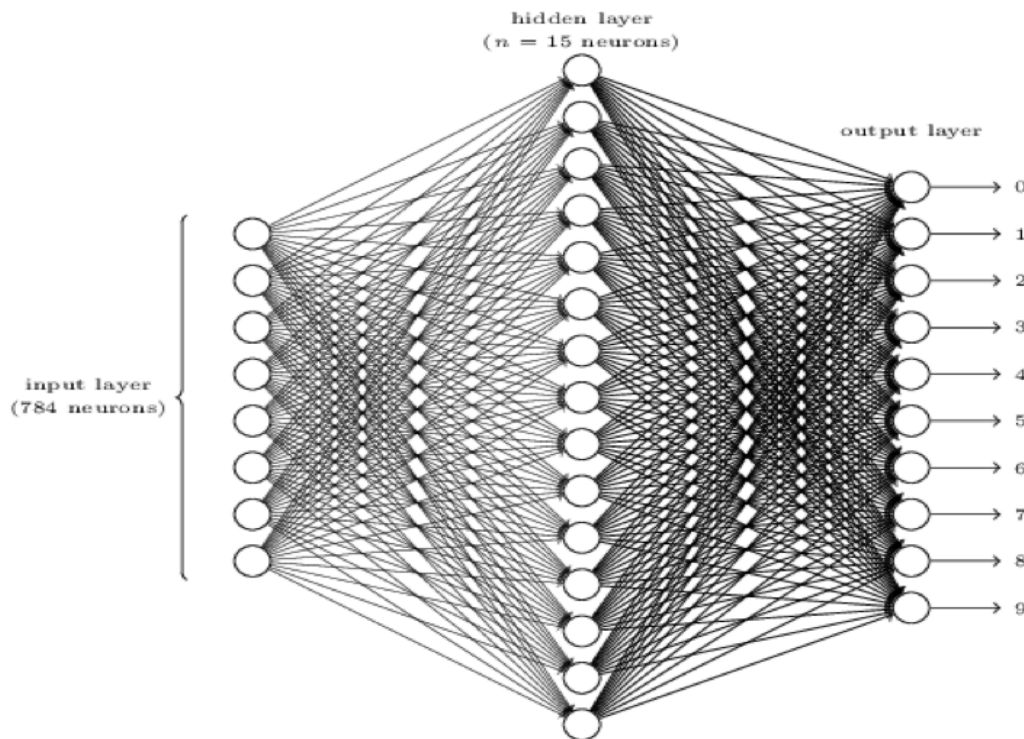
- Assume hand written digits (MNIST database – 60.000 images) :
- Each of the 
- Assume a NN with 784 inputs, a hidden layer of 15 neurons

and 10 outputs:

i.e.: if x is an image (=7)

$y(x)$: (0,0,0,0,0,0,0,1,0,0,)

is the desired output.



Learning with gradient descent

- Cost function $C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$.
 - w : weights, b : biases, a : the vector of correct outputs, n total number of training samples.
- Searching for w, b to minimize $C(w, b)$

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Problems with gradient descent

$$C = \frac{1}{n} \sum_x C_x \quad \text{where} \quad C_x \equiv \frac{\|y(x) - a\|^2}{2}$$

Thus to compute the gradient ∇C we must compute gradients: ∇C_x for each training input and then average them: $\nabla C = \frac{1}{n} \sum_x \nabla C_x \Rightarrow$ huge cost

Learning with stochastic gradient descent

- estimate the gradient ∇C by computing ∇C_x for a small sample of randomly chosen training inputs
- averaging over this small sample we can get a good estimate of the true gradient ∇C , faster
- randomly pick out a small sample size m of randomly chosen training inputs. label those X_1, X_2, \dots, X_m and refer to them as a *mini-batch*. Assuming sample size m is large enough we expect that the average value of the ∇C_{X_j} will be roughly equal to the average over all ∇C_x :

$$\frac{\sum_{j=1}^m \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

Learning with stochastic gradient descent

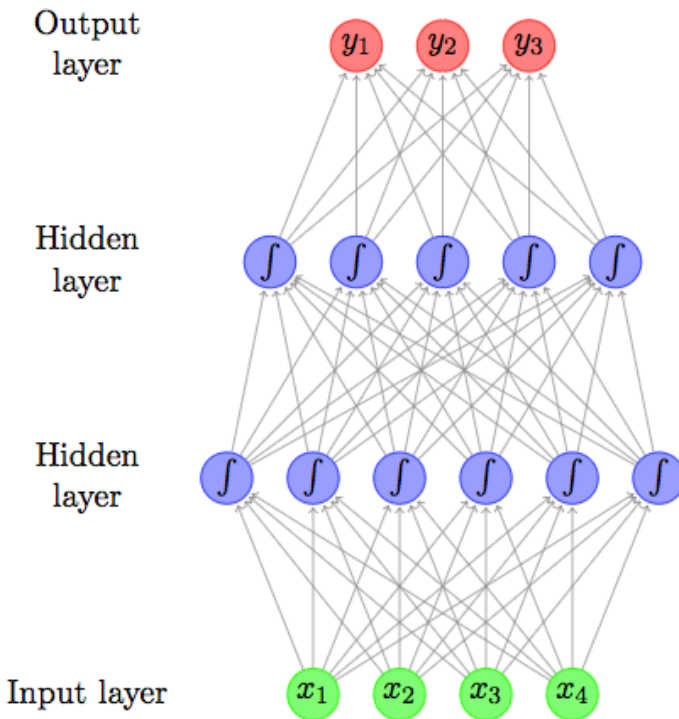
- suppose w_k and b_l the weights and biases in the neural network. Then stochastic gradient descent works by picking out a randomly chosen mini-batch of training inputs, and training with those

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

- sums are over all the training examples X_j in the current mini-batch.
- Then pick out another randomly chosen mini-batch and train with those.
- until we've exhausted the training inputs, (complete an *epoch* of training).

Multilayer Feed-forward NNs



- bottom layer: input to the network.
- Neurons layers, reflecting the flow of information.
- Circle: neuron,
 - incoming arrows neuron's inputs,
 - outgoing arrows neuron's outputs.
 - Arrow: weighted, reflecting its importance.

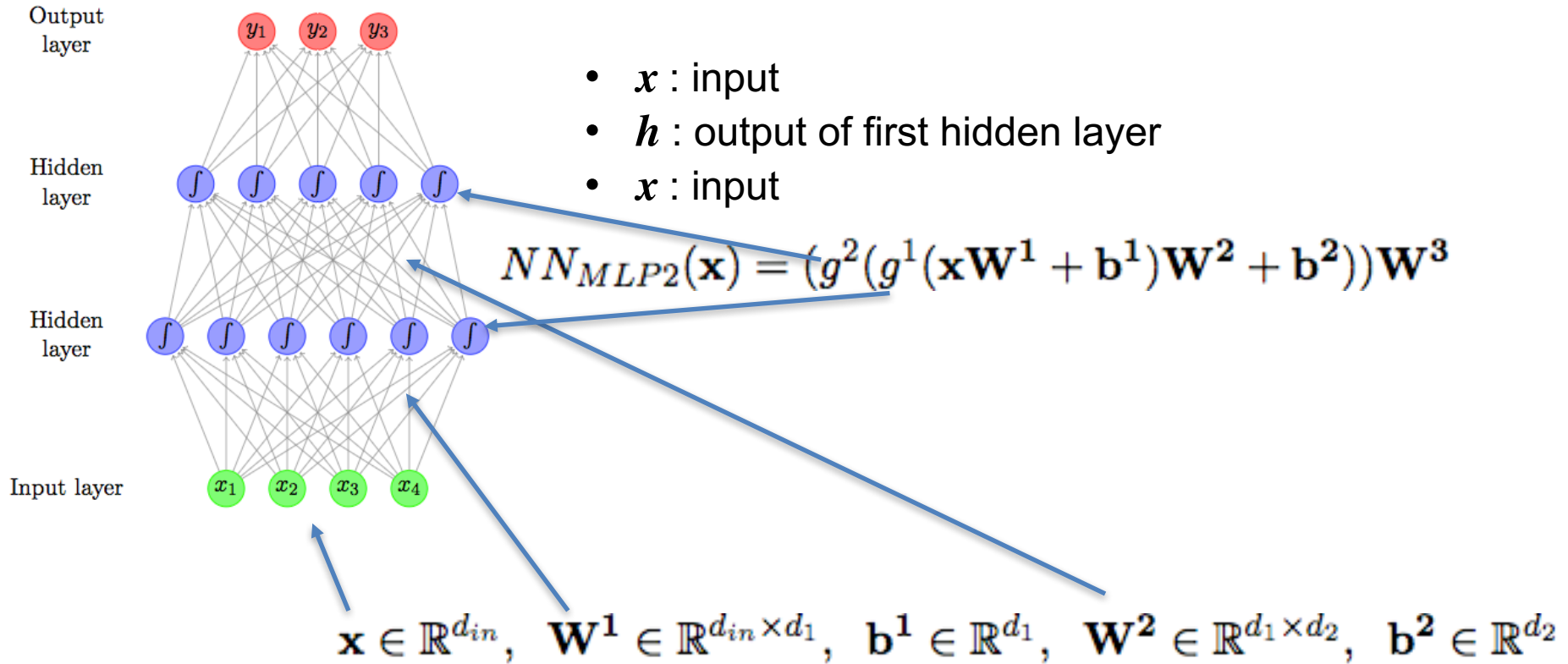
- top-most layer: output of the network.
- The other layers are considered “hidden”.

- Sigmoid shape inside the neurons in the hidden layers represent a non-linear function - typically

$$\frac{1}{1 + e^{-x}}$$

- applied to neuron's value before passing it to the output.
- fully-connected layer or affine layer
 - each neuron is connected to all of the neurons in the next layer

Multilayer Feed-forward NNs



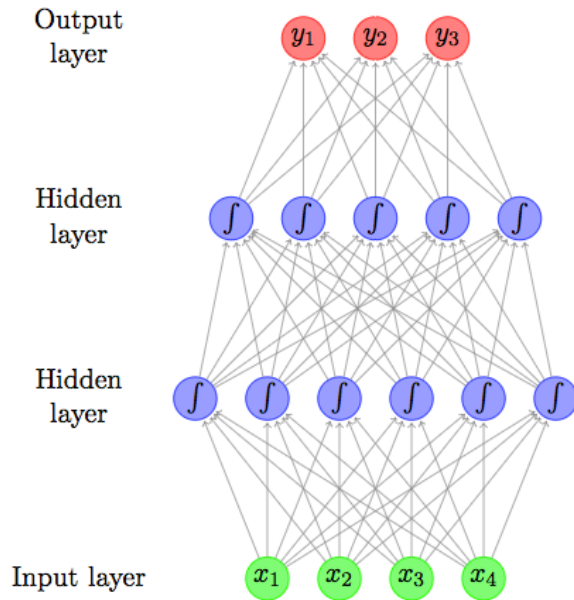
- Alternatively: $NN_{MLP2}(\mathbf{x}) = \mathbf{y}$

$$\mathbf{h}^1 = g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$$

$$\mathbf{h}^2 = g^2(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3$$

Multilayer Feed-forward NNs



$$\begin{aligned} NN_{MLP2}(\mathbf{x}) &= \mathbf{y} \\ \mathbf{h}^1 &= g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \\ \mathbf{h}^2 &= g^2(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \\ \mathbf{y} &= \mathbf{h}^2\mathbf{W}^3 \end{aligned}$$

Common non linear functions

- Sigmoid ($x \rightarrow [0,1]$) $\sigma(x) = \frac{1}{1 + e^{-x}}$
- Hyperbolic tangent (tanh)
S-shaped function, x into the range $[-1,1]$.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

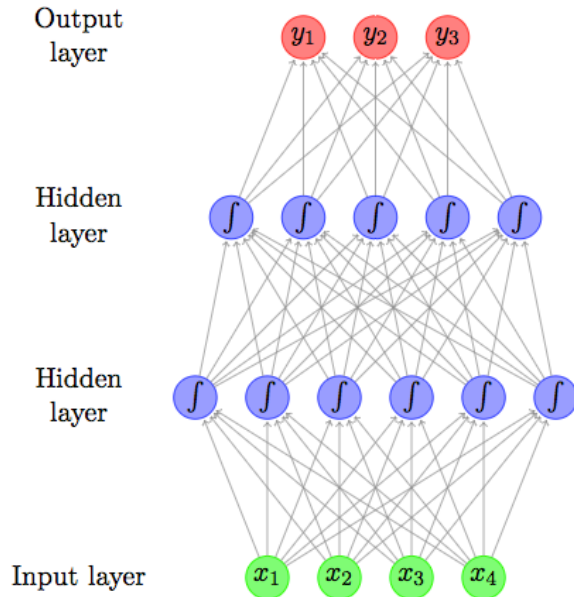
- Hard tanh: approximation of the tanh function faster to compute.

$$\text{hardtanh}(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$$

- $ReLU(x) = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & \text{otherwise} \end{cases}$

As a rule of thumb - $ReLU > \tanh > \text{sigmoid}$.

Multilayer Feed-forward NNs



$$NN_{MLP2}(\mathbf{x}) = \mathbf{y}$$

$$\mathbf{h}^1 = g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$$

$$\mathbf{h}^2 = g^2(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3$$

Output transformations

in several cases the output \mathbf{X} can also be transformed. The most common is softmax:

$$\mathbf{X} = x_1, \dots, x_k$$
$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}}$$

- vector of non-negative real numbers that sum to one,
- Softmax transformation used when we need a probability distribution over the possible output classes.

Convolutional Neural Networks (CNNs)

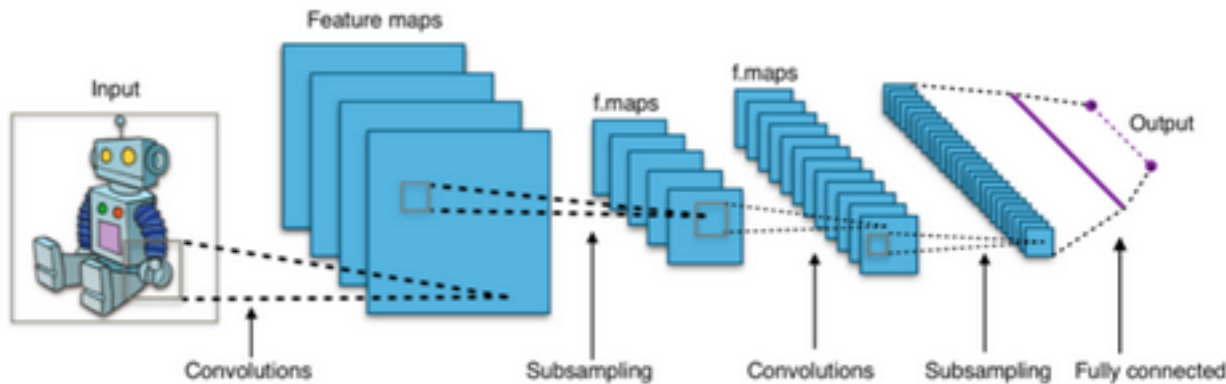
feedforward neural networks, very good at

- capturing local patterns from grids,
- learning more complex features by combining those patterns in a hierarchical way.
 - i.e. edges inferred from raw pixels, edges used to detect simple shapes, shapes used to recognize objects.

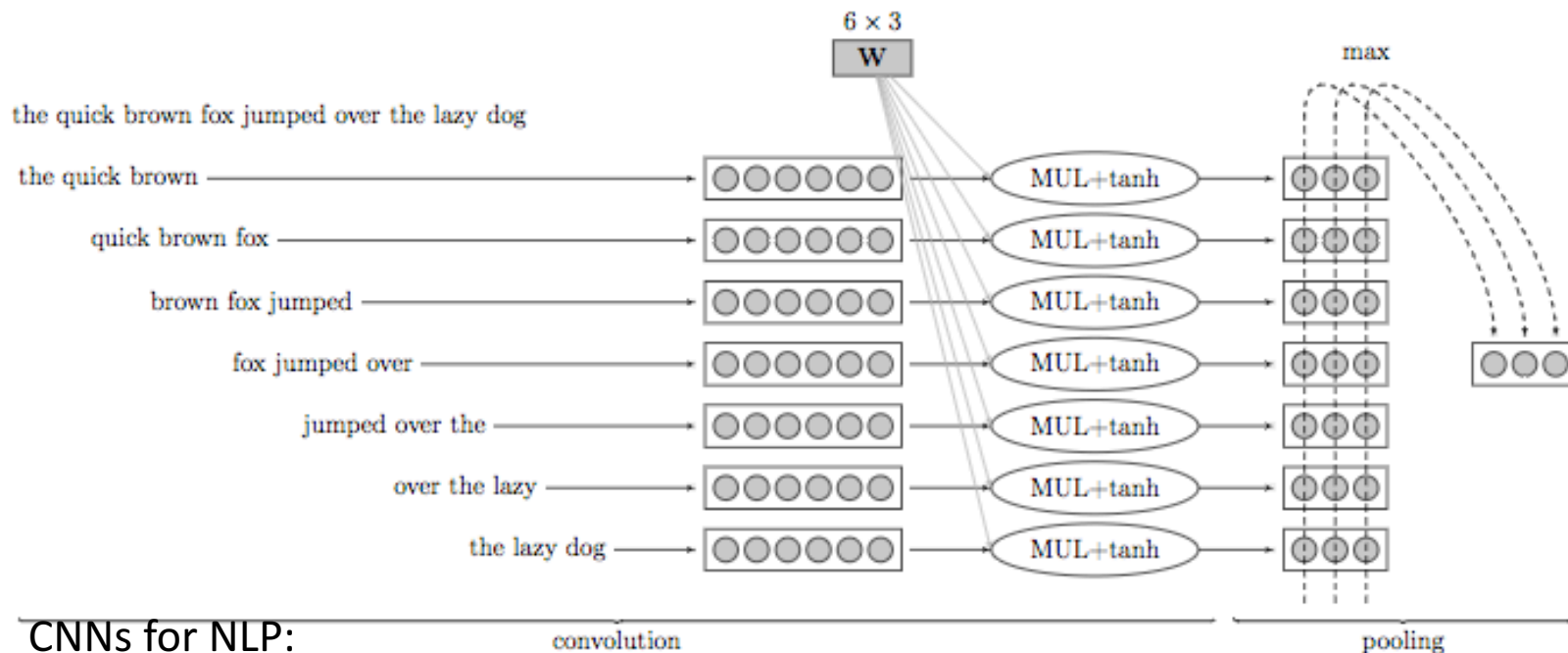
Convolutional Neural Networks (CNNs)

Fundamental properties CNNs

- local invariance: Where exactly the object is placed within the image is not important, the goal is to detect it regardless of its position in the grid.
- Compositionality: The ability to learn more and more complex features starting from small regions is called



Convolutional Neural Networks (CNNs)



CNNs for NLP:

- sentence or document classification - ordering is crucial locally (“not bad, quite good”, “not good, quite bad”, “do not recommend”),
- global location of the features within the sentence or the document does not really matter.
- i.e. polarity of a opinion: “positive, I liked it” can be anywhere in the document
- CNNs are not able to encode long range dependencies,
 - LSTMs are used instead.

CNN for document classification

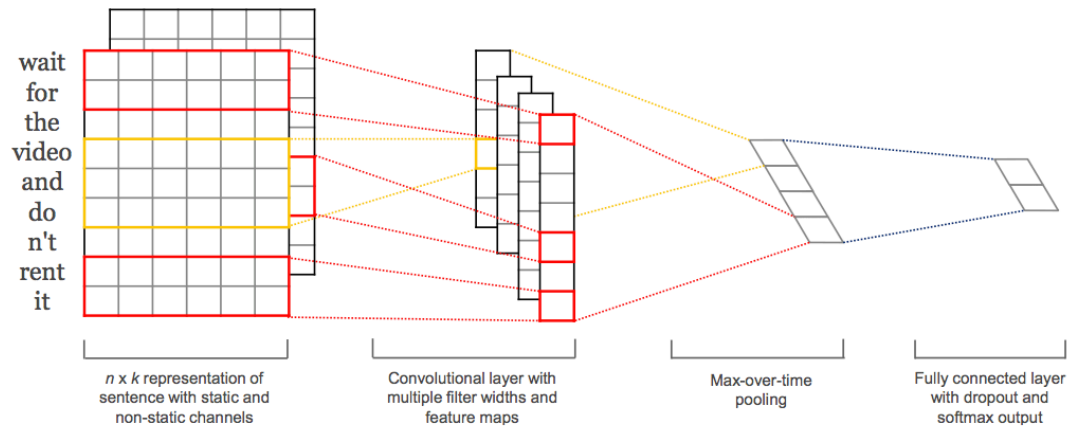
- Use the high quality embeddings as input for Convolutional Neural Network
- Input must be fixed size
- max-pooling deals with variable document lengths
- Applies multiple filters to concatenated word vectors

- Produces new features for every filter
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$$
- And picks the max as a feature for the CNN

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}]$$

$$\hat{c} = \max\{\mathbf{c}\}$$



Yoon Kim - Convolutional Neural Networks for Sentence Classification

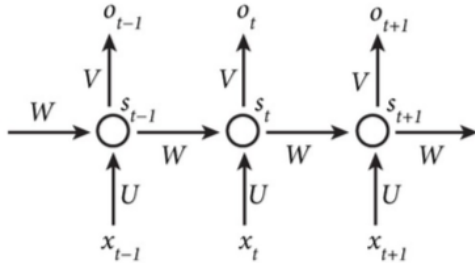
CNN for document classification

- Many variations of the model
- use existing vectors as input (CNN-static)
- learn vectors for the specific classification task through backpropagation (CNN-rand)
- Modify existing vectors for the specific task through backpropagation(CNN-non-static)
- Combine multiple word embeddings
 - Each set of vectors is treated as a 'channel'
 - Filter is applied to both channels
 - Gradients are backpropagated only through one of the channels
 - Fine-tunes one set of vectors while keeping the other static

CNN for document classification

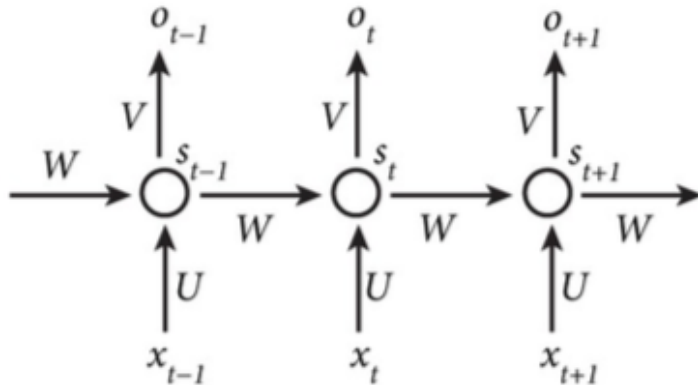
Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	89.6
CNN-non-static	81.5	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	88.1	93.2	92.2	85.0	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	—	—	—	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	—	—	—	—
RNTN (Socher et al., 2013)	—	45.7	85.4	—	—	—	—
DCNN (Kalchbrenner et al., 2014)	—	48.5	86.8	—	93.0	—	—
Paragraph-Vec (Le and Mikolov, 2014)	—	48.7	87.8	—	—	—	—
CCAE (Hermann and Blunsom, 2013)	77.8	—	—	—	—	—	87.2
Sent-Parser (Dong et al., 2014)	79.5	—	—	—	—	—	86.3
NBSVM (Wang and Manning, 2012)	79.4	—	—	93.2	—	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	—	—	93.6	—	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	—	—	93.4	—	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	—	—	93.6	—	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	—	—	—	—	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	—	—	—	—	—	82.7	—
SVM _S (Silva et al., 2011)	—	—	—	—	95.0	—	—

RNNs



- CNNs are naturally efficient with grids⁸,
- RNNs were specifically developed to be used with sequences
 - time series, or, in NLP, words (sequences of letters) or sentences (sequences of words).
 - language modeling $P[w_n | w_1, \dots, w_{n-1}]$.
 - RNNs trained with such objectives can be used to generate new and quite convincing sentences from scratch, as well demonstrated in this very interesting blogpost⁹.
- RNN can be considered as a chain of simple neural layers that share the same parameters.

RNN architecture



The output vector o_t depends on the current hidden state.

In the case of a classification task, it is computed as:

$$o_t = \text{softmax}(V s_t)$$

- V is parameter matrix shared across all time steps,
- $o_t \in \mathbb{R}_{\text{d}_{\text{out}}}$, where d_{out} the number of classes (i.e. $\text{d}_{\text{out}} = 3$ for a three-class classification task)
- RNNs can run very deep: for instance sequences of words, T reaches 15.

RNN input

- an ordered list of input vectors x_0, \dots, x_T
- an initial state vector s_{-1} (initialized with 0s)
- Output: an ordered list of state vectors s_0, \dots, s_T , or “hidden states” (the memory of the network), as well as an ordered list of output vectors o_0, \dots, o_T .
- At each time step t (or position in the document sequence) the hidden state s_t is defined in terms of the previous hidden state s_{t-1} and the current input vector x_t (new training example) in the following recursive way:

$$s_t = f(Ux_t + Ws_{t-1})$$

- Where f is a nonlinear function (i.e. \tanh) and U and W are parameter matrices shared in all steps.
- $x_t \in \mathbb{R}^{d_{\text{in}}}$, where d_{in} can be the size of the vocabulary,
- $s_t \in \mathbb{R}^H$, where H : number of neurons composing that layer.
- The larger this layer, the greater the capacity of the memory – usually ranges in 10^2 .

Gaussian Document Representation from Word Embeddings

- How to represent a document in a collection?
- How to use the high quality word embeddings?
- State of the art approaches
- How to measure similarity/distance between documents?
- Evaluate performance on text classification tasks

*G.Nikolentzos, P. Meladianos, F. Rousseau, M. Vazirgiannis, Y.Stavrakas,
EACL 2017, Valencia*

Document representation

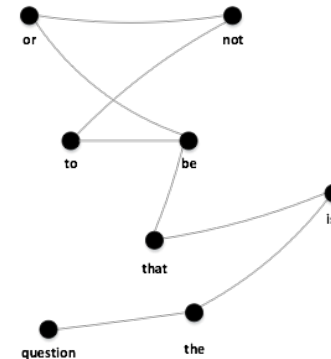
▸ Bag-of-Words

- Document-term matrix
- high dimensionality
- sparse vectors
- dimensions = $|V|$ $|V| > 10^6$
- unable to capture semantic similarity

	T1	T2	...Tn-1	Tn
D1				
D2				
D3				

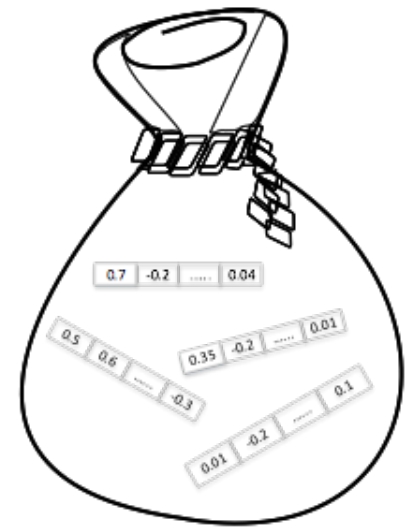
• Graph-of-Words

- captures co-occurrence
- graph algorithms and techniques
- graph kernels



Bag-of-Vectors

- Distributed representations of words
 - store contextual information in a low-dimensional vector
 - dimensionality reduction
 - dimensions=m $100 < m < 500$
 - able to capture semantic similarity between words
 - many learning methods (word2vec, GloVe, SVD)
- Document is represented by a bag-of-vectors
- Goal: meaningful document representations and distance metrics based on the representations of their words



Related work

- Centroid of vectors *[Lebret and Collobert, 2015]*
- Paragraph Vector *[Mikolov, 2014]*
 - vector representations for paragraphs by inserting an additional memory vector in the input layer.
- Word Mover's Distance *[Kusner, 2015]*
 - Calculates the cumulative edit distance between two documents
- CNN for document classification *[Kim, 2014]*
 - Use the high quality embeddings as input for Convolutional Neural Network

Word Mover's distance

- Edit distance of 2 documents
- Based on word embedding representations
- Incorporate semantic similarity between individual word pairs into the document distance metric
- Based on “travel cost” between two words
- Calculates the cost of moving d to d'
- hyper-parameter free
- highly interpretable
- high retrieval accuracy



CNN for document classification

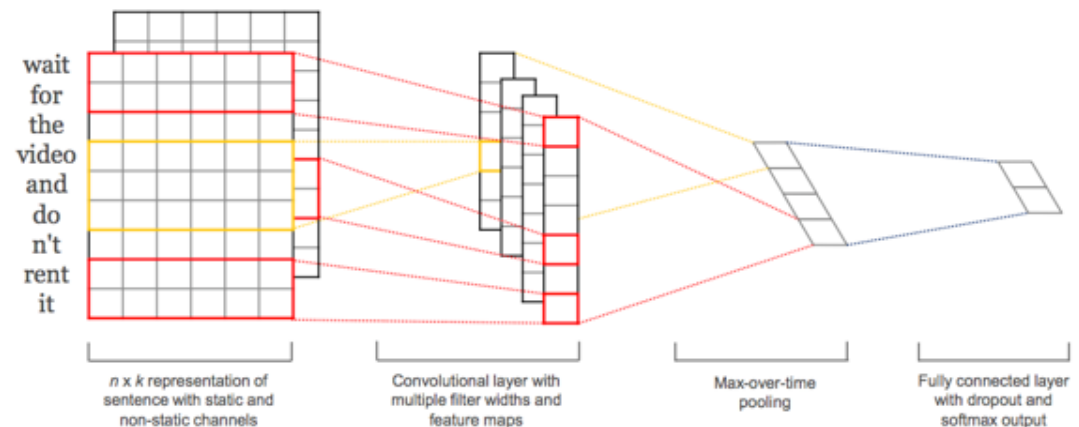
- Use the high quality embeddings as input for Convolutional Neural Network
- Input must be fixed size
- max-pooling deals with variable document lengths
- Applies multiple filters to concatenated word vectors

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$$

- Produces new features for every filter

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

- And picks the max as a feature for the CNN



Gaussian Document Representation from Word Embeddings

- assume that words w present in a document as i.i.d. samples drawn from a multivariate Gaussian distribution $\mathcal{D} = \{d_1, d_2, \dots, d_m\}$
- document is represented as a multivariate Gaussian distribution $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- mean vector $\boldsymbol{\mu} = \frac{1}{|d|} \sum_{w \in d} \mathbf{w}$
- covariance matrix $\boldsymbol{\Sigma} = \frac{1}{|d|} \sum_{w \in d} (\mathbf{w} - \boldsymbol{\mu})(\mathbf{w} - \boldsymbol{\mu})^T$
- Words contained in the vocabulary, but not contained in the embeddings model are initialized to random vectors

Document Similarity

- centroid similarity

$$\text{sim}(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) = \frac{\boldsymbol{\mu}_1 \cdot \boldsymbol{\mu}_2}{\|\boldsymbol{\mu}_1\| \|\boldsymbol{\mu}_2\|}$$

- covariance similarity

$$\text{sim}(\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2) = \frac{\sum \boldsymbol{\Sigma}_1 \circ \boldsymbol{\Sigma}_2}{\|\boldsymbol{\Sigma}_1\|_F \times \|\boldsymbol{\Sigma}_2\|_F}$$

- similarity between two documents

$$\text{sim}(d_1, d_2) = \alpha(\text{sim}(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)) + (1 - \alpha)(\text{sim}(\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2)) \quad \alpha \in [0, 1]$$

- valid kernel function

Why Gaussian?

- Each document follows a distribution described by its topic
- Word embeddings capture lexico-semantic regularities
- Words with similar syntactic and semantic properties are found to be close to each other in the embedding space
- Semantically related words are localized in space
- Gaussian distributions capture a notion of centrality in space
- The choice of a Gaussian parameterization is justified by both analytic convenience and observations that Euclidean distances between embeddings correlate with semantic similarity

Experiments

Datasets

Dataset	# training examples	# test examples	# classes	vocabulary size	<i>word2vec</i> size
Reuters	5,485	2,189	8	23,585	15,587
Amazon	8,000	CV	4	39,133	30,526
TREC	5,452	500	6	9,513	9,048
Snippets	10,060	2,280	8	29,276	17,067
BBCSport	348	389	5	14,340	13,390
Polarity	10,662	CV	2	18,777	16,416
Subjectivity	10,000	CV	2	21,335	17,896
Twitter	3,115	CV	3	6,266	4,460

Baselines

- BOW-SVM
- NBSVM
- Centroid-SVM
- WMD-KNN
- CNN

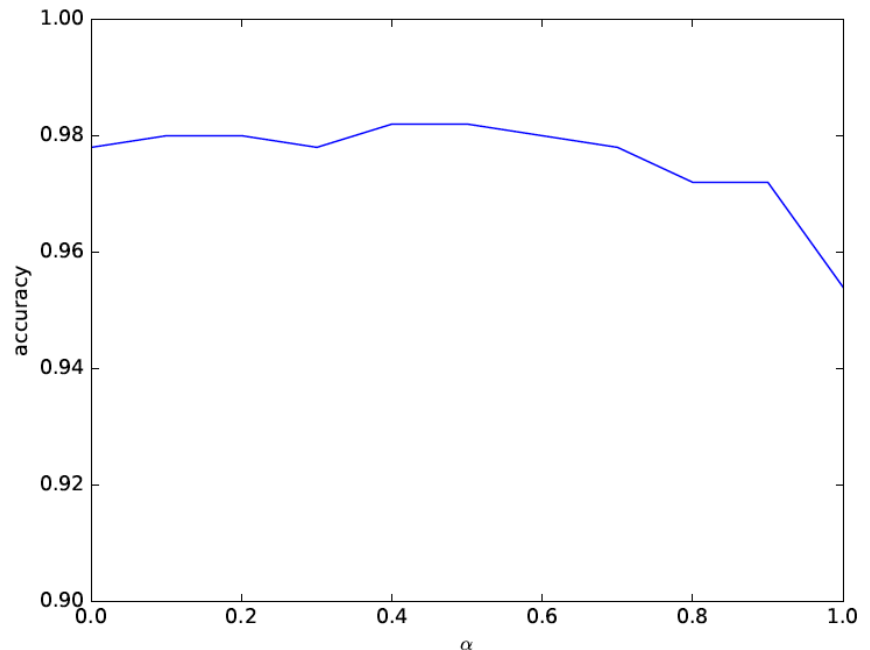
Results

Dataset Method	Reuters		Amazon		TREC		Snippets	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
BOW (binary)	0.9571	0.8860	0.9126	0.9127	0.9660	0.9692	0.6171	0.5953
Centroid	0.9676	0.9171	0.9311	0.9312	0.9540	0.9586	0.8123	0.8170
WMD	0.9502	0.8204	0.9200	0.9201	0.9240	0.9336	0.7417	0.7388
NBSVM	0.9712	0.9155	0.9486	0.9486	0.9780	0.9805	0.6474	0.6357
CNN	0.9707	0.9297	0.9448	0.9449	0.9800	0.9800	0.8478	0.8466
Gaussian	0.9712	0.9388	0.9498	0.9497	0.9820	0.9841	0.8224	0.8244

Dataset Method	BBCSport		Polarity		Subjectivity		Twitter	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
BOW (binary)	0.9640	0.9690	0.7615	0.7614	0.9004	0.9004	0.7467	0.6205
Centroid	0.9923	0.9915	0.7783	0.7782	0.9100	0.9100	0.7361	0.5727
WMD	0.9871	0.9866	0.6642	0.6639	0.8604	0.8603	0.7031	0.4436
NBSVM	0.9871	0.9892	0.8698	0.8698	0.9369	0.9368	0.7852	0.6191
CNN	0.9486	0.9461	0.8037	0.8031	0.9315	0.9314	0.7549	0.6137
Gaussian	0.9974	0.9974	0.8021	0.8020	0.9310	0.9310	0.7534	0.6443

Results

- sensitivity of the classification to parameter α
- TREC dataset
- Centroid performance drops significantly
- highest accuracy $\alpha=0.5$



Conclusion

- Model each document as a Gaussian distribution based on the embeddings of its words
- Similarity between two documents based on the similarity of their distributions
- Empirical evaluation demonstrates the effectiveness of the approach across a range of data
- Performance gain is attributed to the high quality of the embeddings and the ability to effectively utilize them

References and online resources

- **Artificial neural networks: A tutorial**, AK Jain, J Mao, KM Mohiuddin - Computer, 1996
- introduction from a coder's perspective: <http://karpathy.github.io/neuralnets/>
- <http://cs231n.github.io/>
- online book: <http://neuralnetworksanddeeplearning.com/index.html>
- history of neural nets: <http://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network>
- nice blog post on neural nets applied to NLP: <http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/>
- **A Primer on Neural Network Models for Natural Language Processing**, Y. Goldberg, u.cs.biu.ac.il/~yogo/nnlp.pdf