Neural Networks and Deep Learning

M. Vazirgiannis

Data Science and Mining Team (DASCIM), LIX,

Ecole Polytechnique

http://www.lix.polytechnique.fr/dascim

Google Scholar: https://bit.ly/2rwmvQU

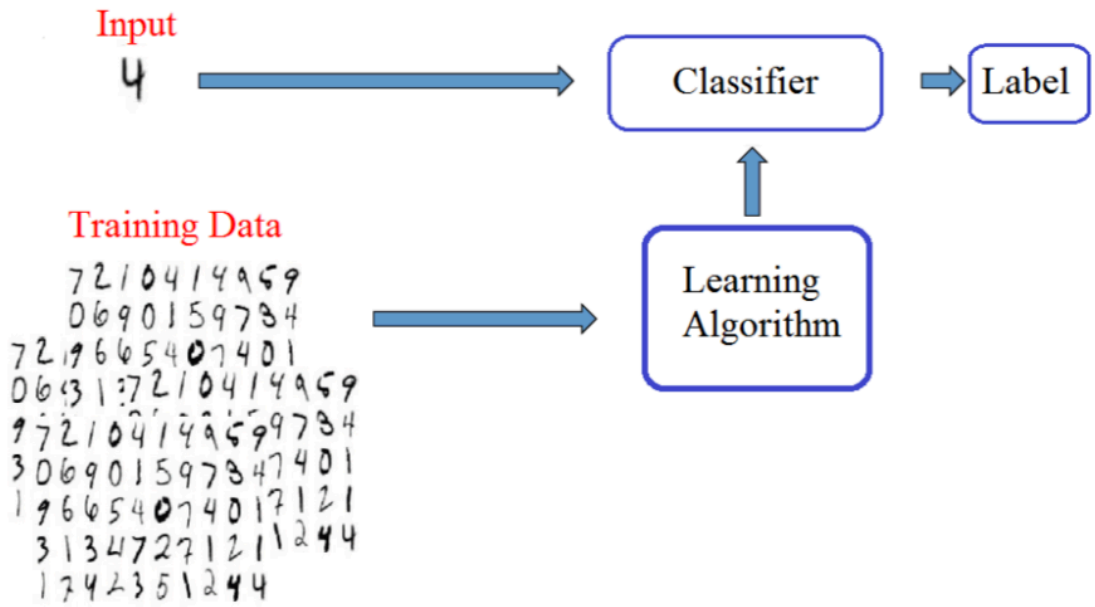
Twitter: @mvazirg

December 2019

Outline

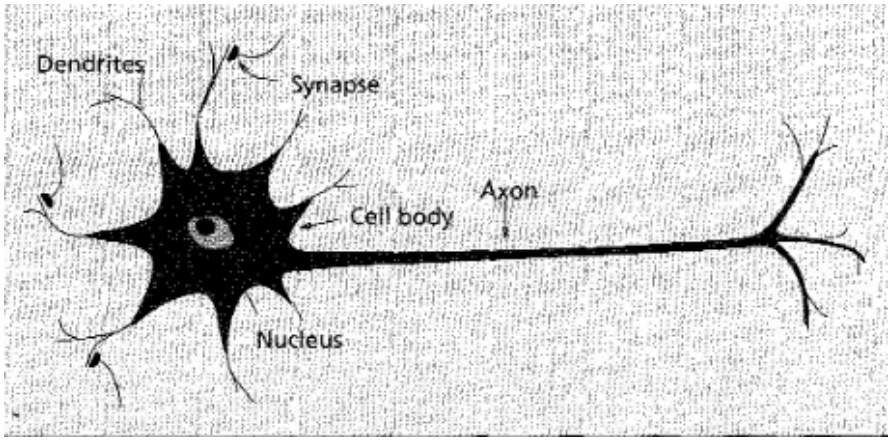
- **Neural Networks – Introduction**
- CNNs
- Applications in NLP
- MLP Hyper-parameter tuning

Machine Learning…



- Data
- Performance
- Model
- Optimization

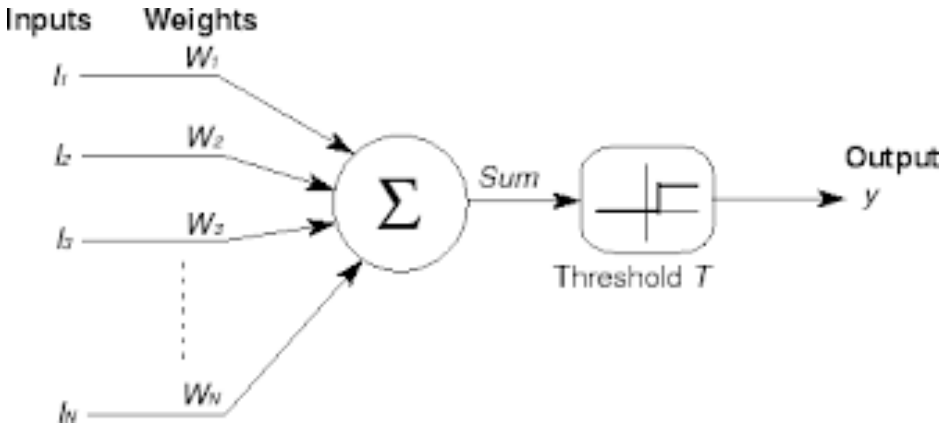
Neural Networks



- Neuron: biological cell processing information
- composed of a cell body, the axon and the dendrites.
- Cell body
  - *receives* signals (impulses) from other neurons through its *dendrites*
  - *transmits* signals generated by its cell body along the *axon*
- *synapse* functional unit between two neurons (an axon strand and a dendrite)

Artificial neural networks: A tutorial, AK Jain, J Mao, KM Mohiuddin - Computer, 1996

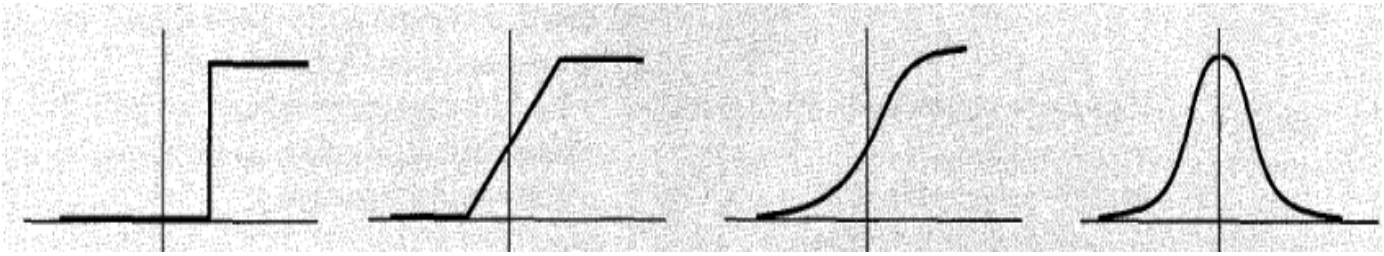
Neural networks



$$y = f(\sum_{j=1}^{n} w_j X_j + b)$$

McCulloch-Pitts Neuron Model

- f: activation function
- w_j: weight of the j-th input Xj
- b : bias
- activation functions: piecewise linear, sigmoid, or Gaussian



Activation functions

$$Identity : f(x) = x$$

$$Sigmoid : \sigma(x) = \frac{1}{1 + e^{-x}}$$

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

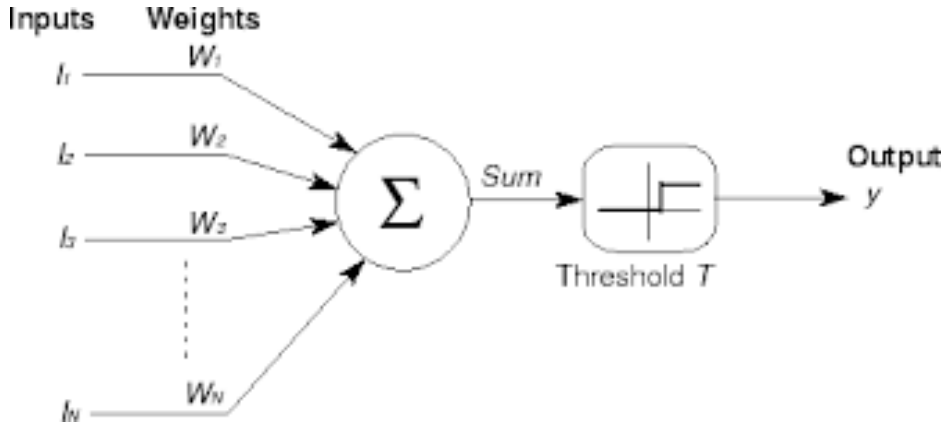
$$Stepfunction : f(x) = 1_{x>0}$$

$$ReLU : f(x) = max(0, x)$$

- Non linearity for decision…

Neural Network architectures

- Weighted directed graphs with artificial neurons as nodes, directed edges (with weights) 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.

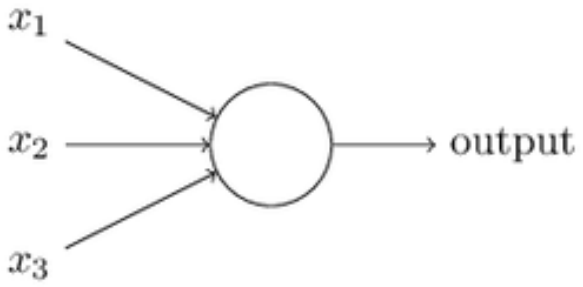


Perceptron learning algorithm

- developed in the 1950 -60 by Frank Rosenblatt
- takes several binary inputs, x_1, x_2… 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 making decisions by weighing up evidence.
- Varying weights and threshold, we get different models of decision-making



Perceptron learning algorithm

- Initialize the weights and threshold (small random numbers)
- Present an input vector $X = \{x_1, .. x_n\}$ 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.

$\sum_{j=1}^{n} w_j X_j - u = 0$ defines the class separation hyper plane

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

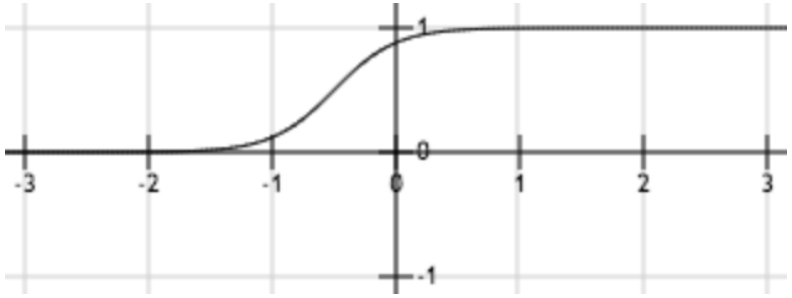
If training data are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations.

Neural Networks – sigmoid neuron

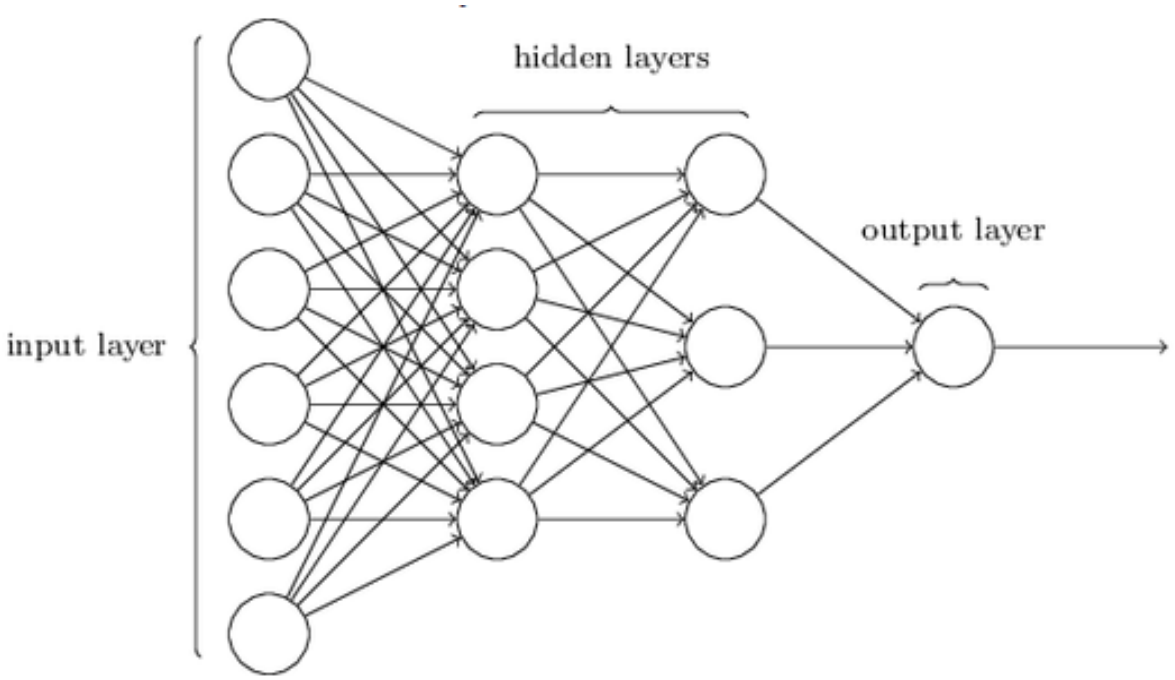
- In perceptron small changes to the weights may cause huge difference to output (0->1)
- We want to impose: small changes to weights (or bias) small change to the output.
- the sigmoid neuron has weights for each input, $w_1, w_2, \ldots$, and an overall bias b. The output is $\sigma(w \cdot x + b)$, where σ is the *sigmoid function*

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

- *The output of the neuron the is:* $\sigma(z) = \dfrac{1}{1 + e^{(-\sum_j w_j x_j - b)}}$

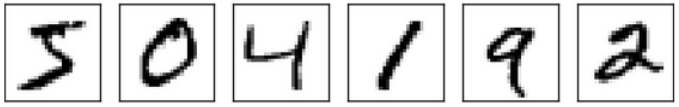


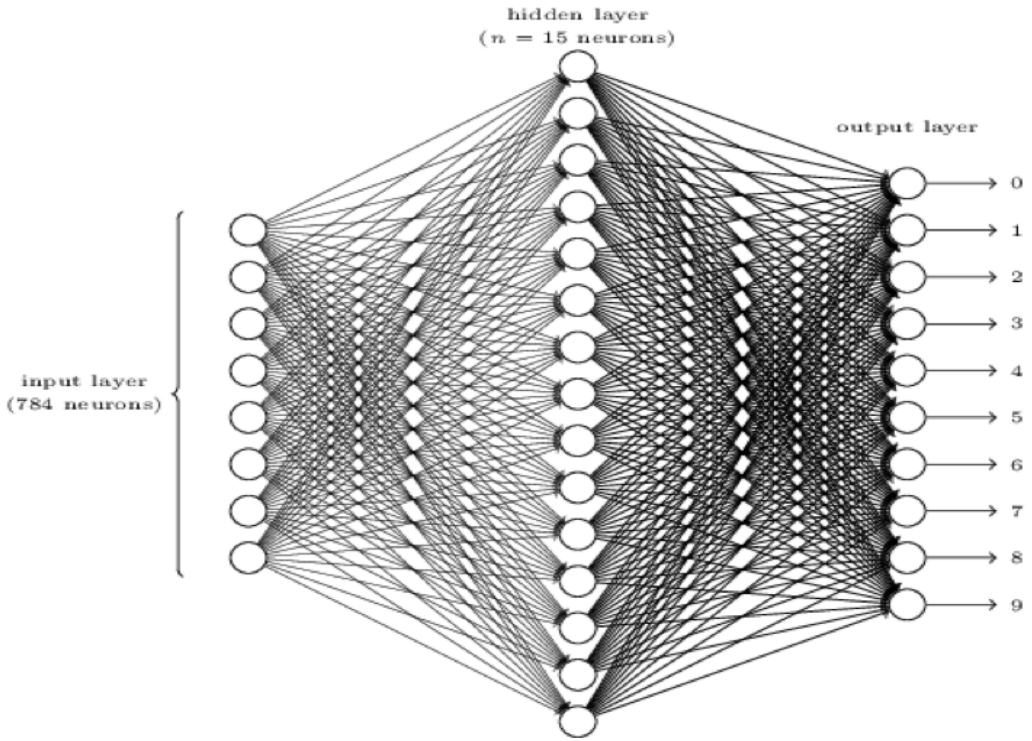
NNs architecture - Multilayer perceptron (MLP)



- i.e. case of 64x64 greyscale image representing a single number (i.e. 8)
- MLP with (64x64) 4096 input neurons
- A single neuron as output (>0.5: “is 8” , <0.5 “not 8”

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.

http://neuralnetworksanddeeplearning.com/chap1.html

Learning with gradient descent

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

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

$$b_l' = b_l - \eta \frac{\partial C}{\partial b_l}$$

Problems with gradient descent

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

Thus to compute the gradient ∇C we must compute gradients: ∇C_x for each training input and then average them:

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x$$

=> very slow learning

Learning with stochastic gradient descent

- estimate ∇C computing ∇C_x for a small sample of randomly chosen training points
- averaging over sample: get a good estimate of the true gradient ∇C, faster
- randomly pick m randomly chosen training inputs. label those $X_1, X_2, \ldots, X_m$ 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 ∇Cx:

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

Learning with stochastic gradient descent

- 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

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

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

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

Backpropagation

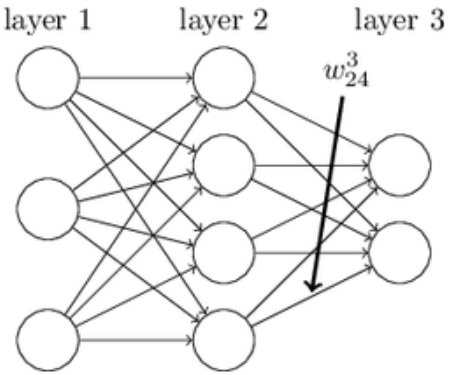
- Learning parameters with gradient descent

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

- How to compute the gradients: backpropagation

w_{jk}^{l} : is the weight among the neuron k^{th} of the l-1 layer to the j^{th} neuron of the l layer.



Letters to Nature, *Nature* **323**, 533-536 (9 October 1986) | doi:10.1038/323533a0; **Learning representations by back-propagating errors,** David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams‡.

The backpropagation algorithm

- **Input *x*:** Set the corresponding activation a^1 for the input layer.
- **Feedforward:** For each $l=2,3,\ldots,L$ compute
$$z^l=w^l a^{l-1}+b^l \text{ and } a^l=\sigma(z^l).$$
- **Output error** δ^L**:** Compute the vector $\delta^L=\nabla_a C\odot\sigma'(z^L)$.
- **Backpropagate the error:** For each $l=L-1,L-2,\ldots,2$ compute
$$\delta^l=((w^{l+1})^T\delta^{l+1})\odot\sigma'(z^l).$$
- **Output:** The gradient of the cost function is given by
$$\partial C/\partial w^l{}_{jk}=a_k{}^{l-1}\delta_j{}^l$$
$$\partial C/\partial b_j{}^l=\delta_j{}^l.$$

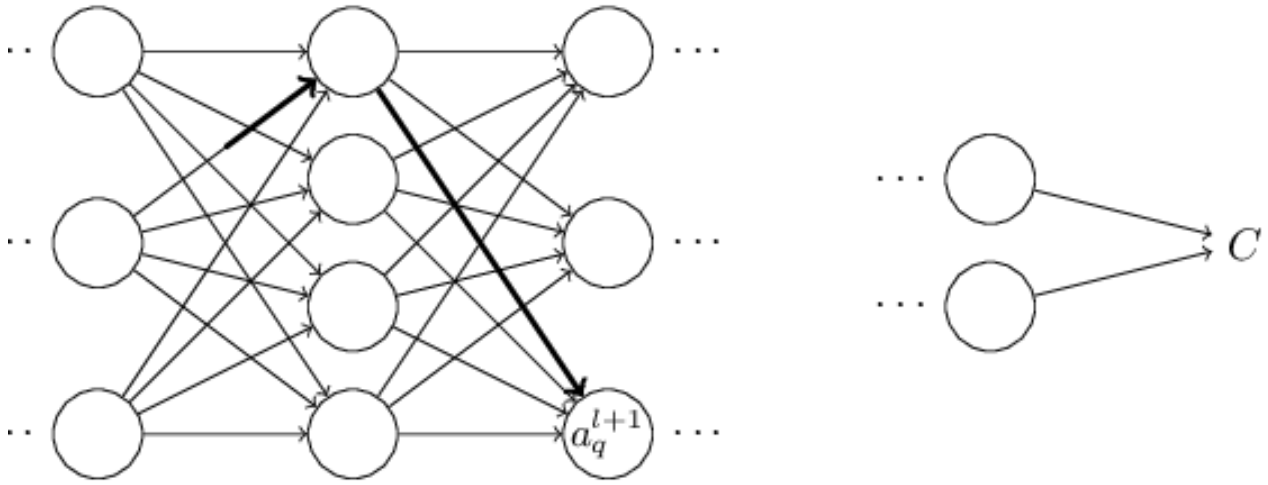
Back propagation - the big picture

- Assume a small change in Δw^l_{jk} to some weight w^l_{jk} in the network
- It will cause a change in the output activation from the corresponding neuron

$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}.$$

- That, in turn, will cause a change in *all* the activations in the next layer
- Thus:

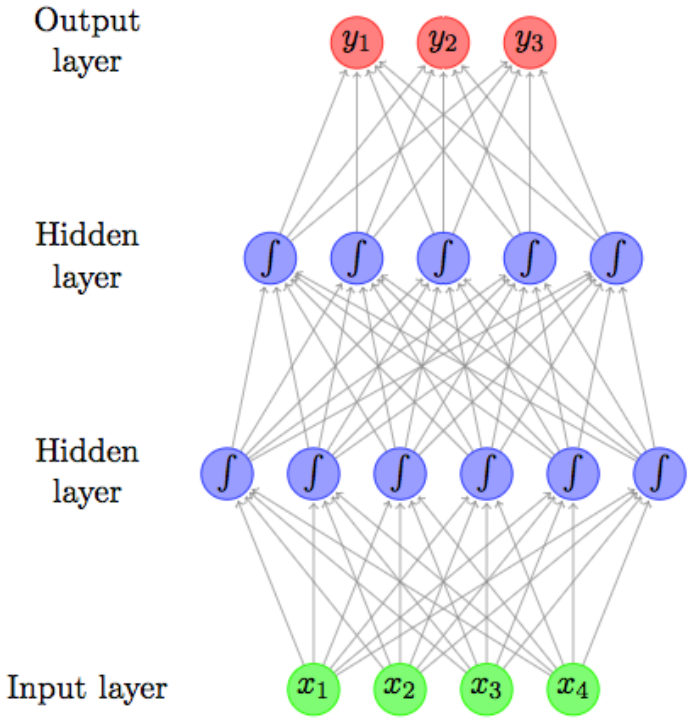
$$\Delta a^{l+1}_q \approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \Delta a^l_j.$$



Back propagation - the big picture

- Cascading changes to the next layers until a change in the final layer:
  - Single path: $$\Delta C \approx \frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l,$$
  - Sum over all paths: $$\Delta C \approx \sum_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l$$
  - Thus, the effect of the cost of a change in a single weight is: $$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}$$
  - Thus learning takes place: $$w_k \to w_k' = w_k - \eta\frac{\partial C}{\partial w_k}$$ $$b_l \to b_l' = b_l - \eta\frac{\partial C}{\partial b_l}.$$

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.

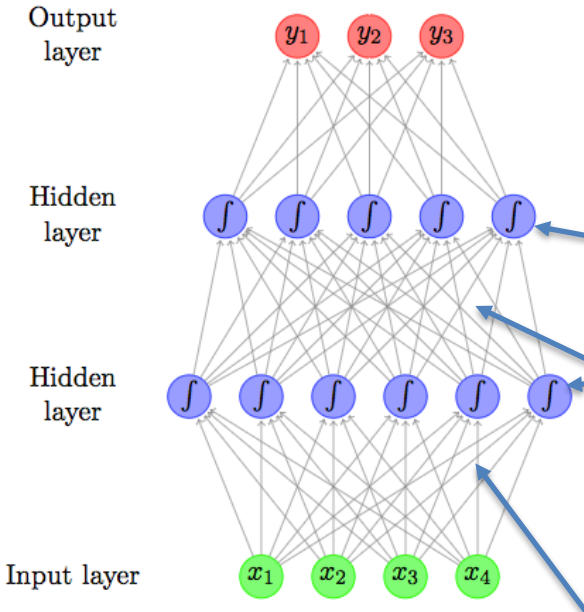
- top-most layer: output of the network.
- The other layers are considered "hidden".

- 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



- $\boldsymbol{x}$: input
- $\boldsymbol{h}$: output of first hidden layer
- $\boldsymbol{x}$: input

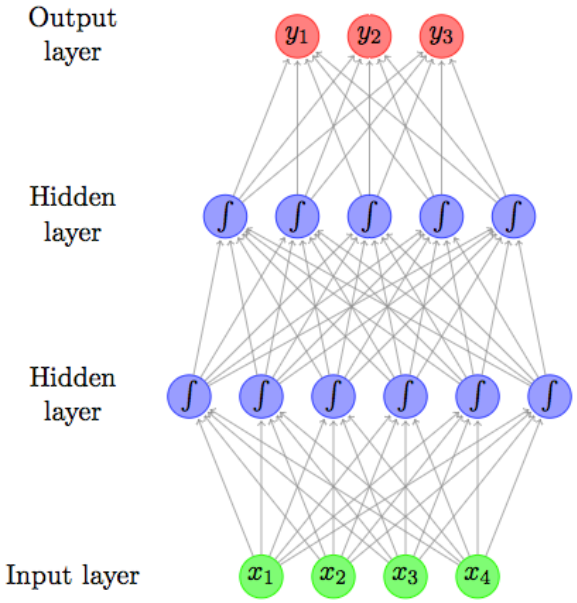
$$NN_{MLP2}(\mathbf{x}) = (g^2(g^1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2}))\mathbf{W^3}$$

$$\mathbf{x} \in \mathbb{R}^{d_{in}},\ \mathbf{W^1} \in \mathbb{R}^{d_{in} \times d_1},\ \mathbf{b^1} \in \mathbb{R}^{d_1},\ \mathbf{W^2} \in \mathbb{R}^{d_1 \times d_2},\ \mathbf{b^2} \in \mathbb{R}^{d_2}$$

- Alternatively:

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

Multilayer Feed-forward NNs



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

Common non linear functions

- Sigmoid (x ->$[0,1]$) $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent (*tanh*)
  S-shaped function, x ->$[-1,1]$

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

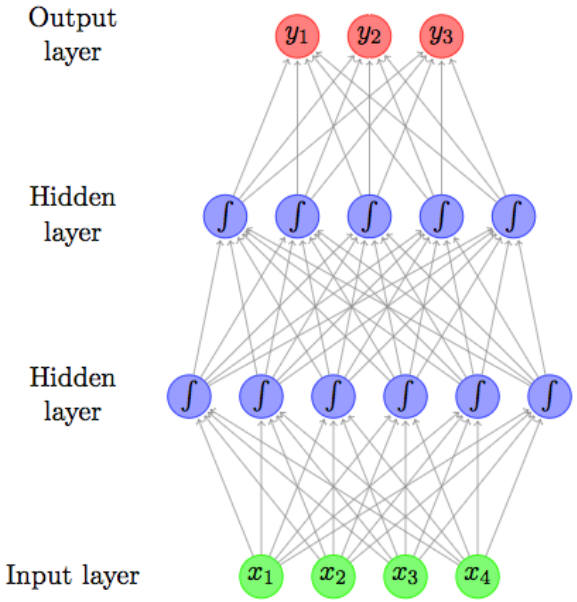
- Hard tanh: approximation of the tanh function faster to compu

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

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

As a rule of thumb - ReLU > tanh > 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 X can also be transformed. The most common is softmax:

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

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

Softmax

- Change the activation function in the output layer:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

  - Denominator: sum over all (k) output neurons
- Output is a probability distribution

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1.$$

  - sigmoid would not produce it.

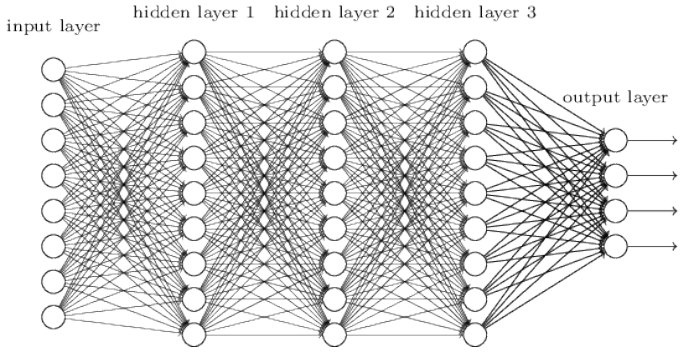
Outline

- Neural Networks – Introduction
- **CNNs**
- Applications in NLP

Convolutional Neural Networks (CNNs)

NNs fully-connected layers to classify images.

- do not take into account the spatial structure of the images.
  - treat input pixels far apart and close together in the same way.
  - spatial structure must be inferred from the training data.

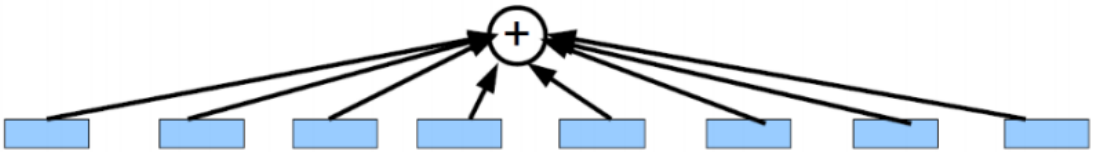


- CNNs takes advantage of the spatial structure
- fast to train thus feasibility for training deep, many-layer networks,
- deep CNNs used for image recognition, text classification etc.
- Basic components:
  - *local receptive fields*,
  - *shared weights*
  - *pooling*.

Learning non-linear functions

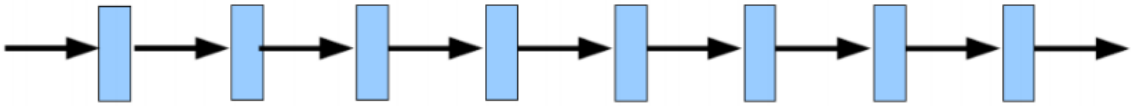
Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

- Proposal 1: linear combination $f(x) \approx \sum_j g_j$



Shallow

- Proposal 2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



Deep

Convolutional Neural Networks

- introduced by Yann LeCun and Yoshua Bengio (1995)
- Neuro-biologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.
- They designed a network structure that implicitly extracts relevant features.
- Convolutional Neural Networks are a special kind of multi-layer neural networks.

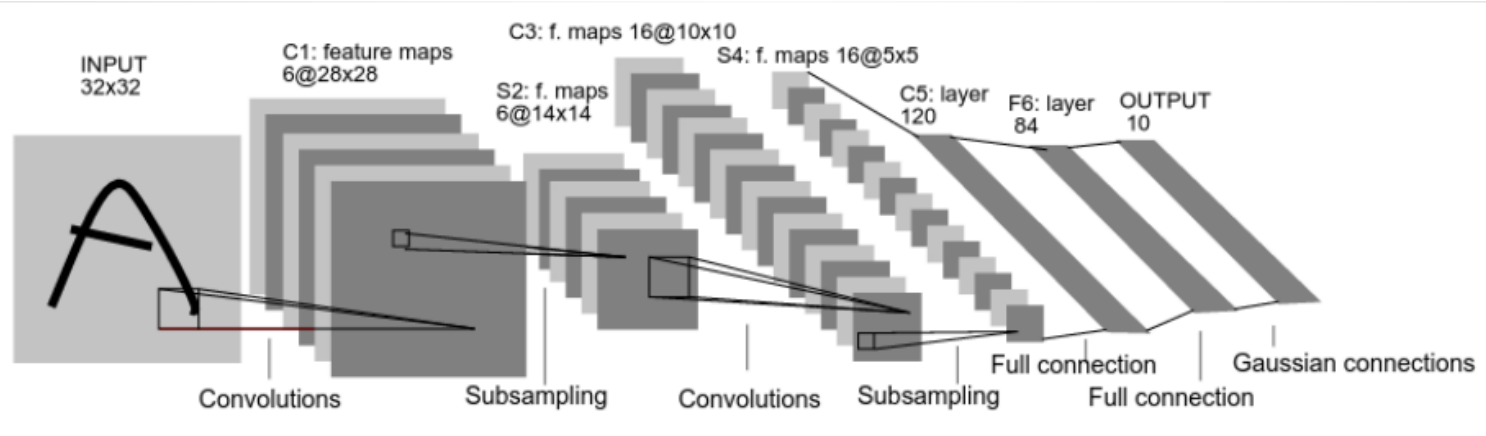
[2] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. 3, 4

Convolutional Neural Networks

- inspired by studies of the cat's visual cortex [1], developed in computer vision to work on regular grids such as images [2].
- Feed-forward NNs , each neuron receives input from a neighborhood of the neurons (receptive fields) in the previous layer.
- Receptive fields, allow CNNs to recognize complex patterns in a hierarchical way, by combining lower-level, elementary features into higher-level features *compositionality*.
  - raw pixels=> edges =>shapes =>objects.
- absolute positions of features in the image are not important – only useful respective positions is useful composing higher-level patterns.
- Model detect a feature regardless of its position in the image - local invariance.
- Compositionality, local invariance two key concepts of CNNs.

[1] Hubel, David H., and Torsten N. Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology 160.1:106-154. 4
[2] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. 3, 4

CNN architecture



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Convolution

- The convolution of f and g, written as $f*g$, is defined as the integral of the product of the two functions after one is reversed and shifted

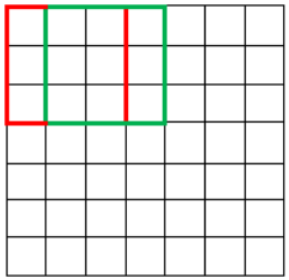
$$s(t) = \int x(a)w(t-a)da \qquad s(t) = (x * w)(t)$$

- Convolution is commutative.
- Can be viewed as a weighted average operation at every moment (for this w need to be a valid probability density function)
- Discrete Convolution:

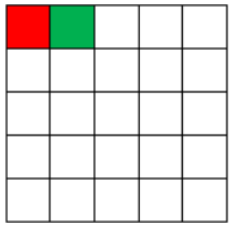
$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

Example

7 x 7 Input Volume

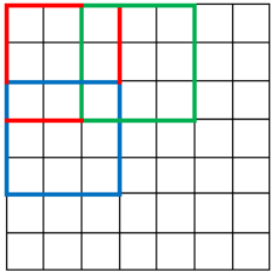


5 x 5 Output Volume

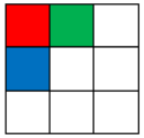


Stride = 1

7 x 7 Input Volume



3 x 3 Output Volume

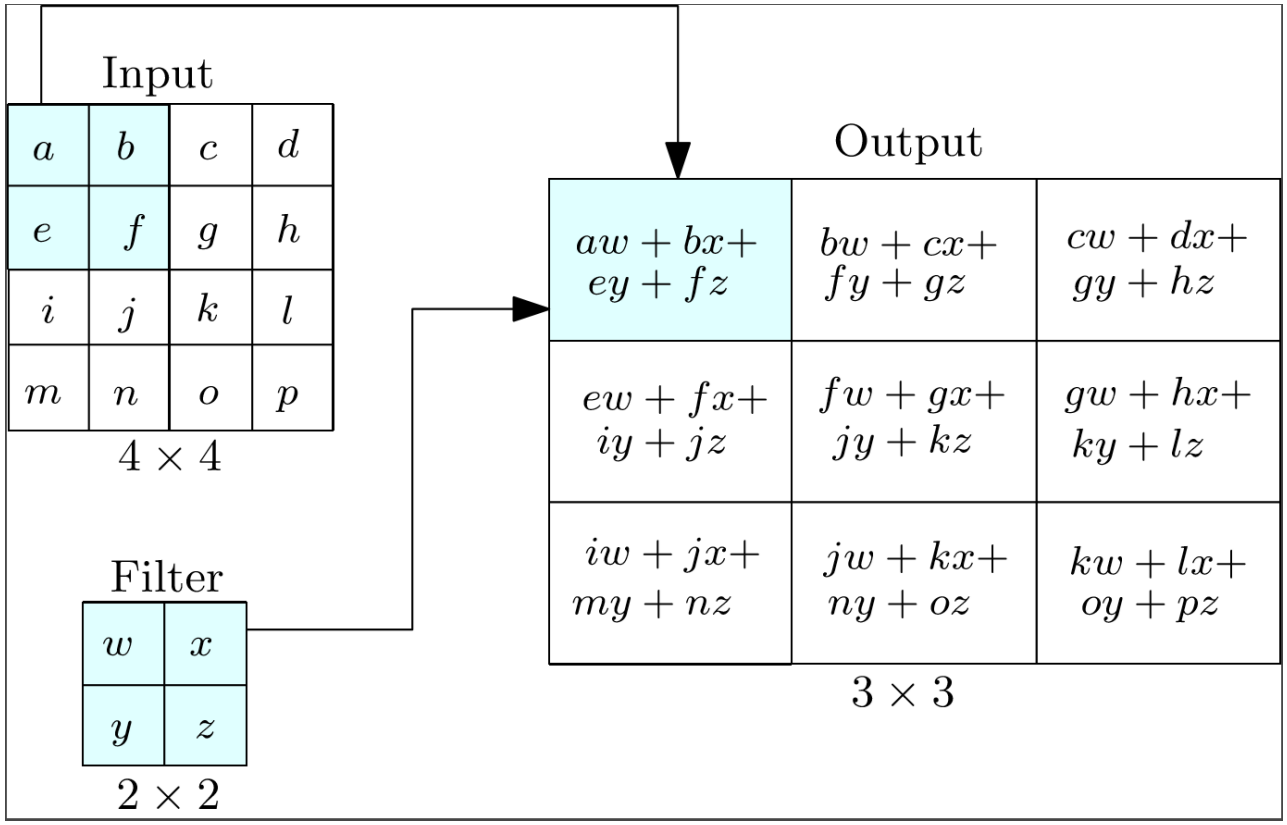


Stride = 2

Convolution Variants

- **Full**: Add zero-padding to the feature matrix enough for every feature to be visited *k* times in each direction
  - (the maximum padding which does not result in a convolution over just padded elements)
- **Valid**: With no zero-padding, kernel is restricted to traverse only within the feature matrix
- **Same**: Add zero-padding to the feature matrix to have the output of the same size as the feature matrix
- **Stride**: Down-sampling the output of convolution by sampling only every *s* features in each direction.

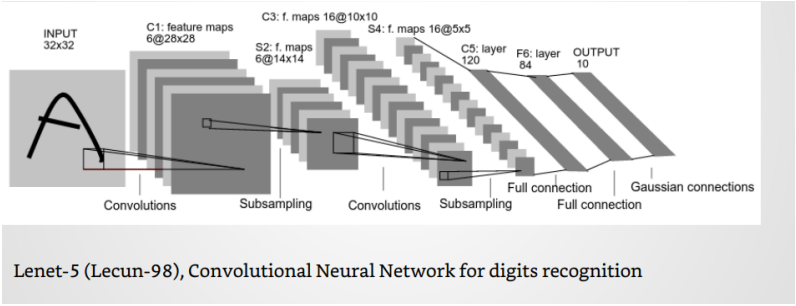
Example



- Convolution exploits spatial local correlations in the feature space by enforcing local connectivity pattern between neurons of adjacent layers
- Drastic reduction of free parameters compared to fully connected network reducing over fitting and computational complexity of the network

Feature maps

- Feature Map - Obtained by convolution of the feature matrix with a linear filter, adding a bias term and applying a non-linear function
- Non-linear functions:
  - Sigmoid $\frac{1}{1+e^{-x}}$
  - Tanh $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

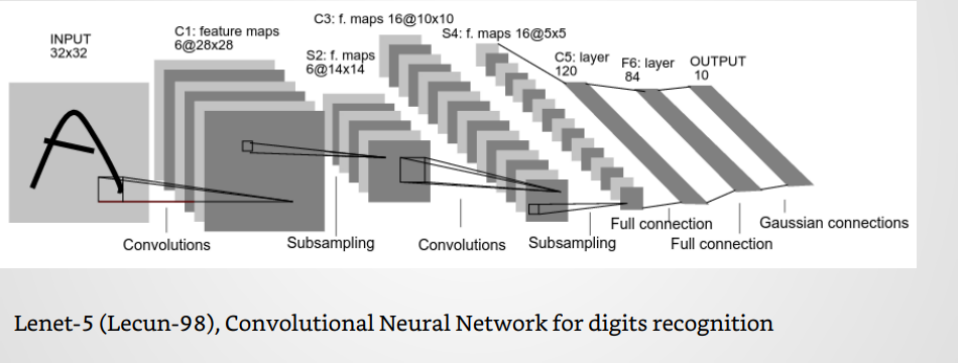
  - Rectified Linear Unit (ReLU) -> Most popular choice avoids saturation issues, makes learning faster

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

- Require a number of such feature maps at each layer to capture sufficient features

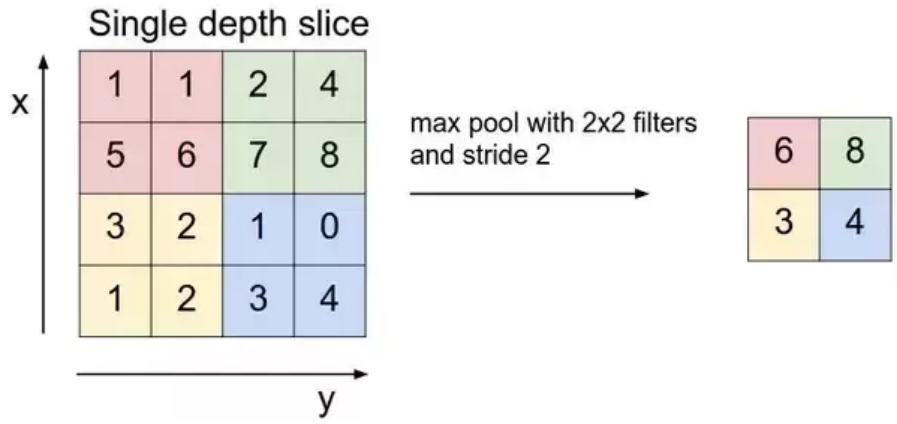
Pooling

- Sub-sampling layer
- Variants:
  - Max pooling
  - Weighted average
  - L2 norm of neighborhood
- Provides translation invariance
- Reduces computation



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Max pooling - Example



See demo: http://cs231n.github.io/understanding-cnn/

How to reduce overfitting in CNNs

- Dropout: Randomly set the output value of network neurons to 0
  - Works as a regularization alternative
- Weight decay: keeps the magnitude of weights close to zero
- Data Augmentation: Slightly modified instances of the data

CNN for Text Classification

- Use the word embeddings of the document terms as input for Convolutional Neural Network
- Input must be fixed size
- Applies multiple filters to concatenated word vectors
- Produces new features for every filter
- picks the max as a feature for the CNN

Outline

- Neural Networks – Introduction
- **CNNs**
- Applications in NLP

CNN for text classification

- Use the high quality embeddings as input for Convolutional Neural Network
- Applies multiple filters to concatenated word vector

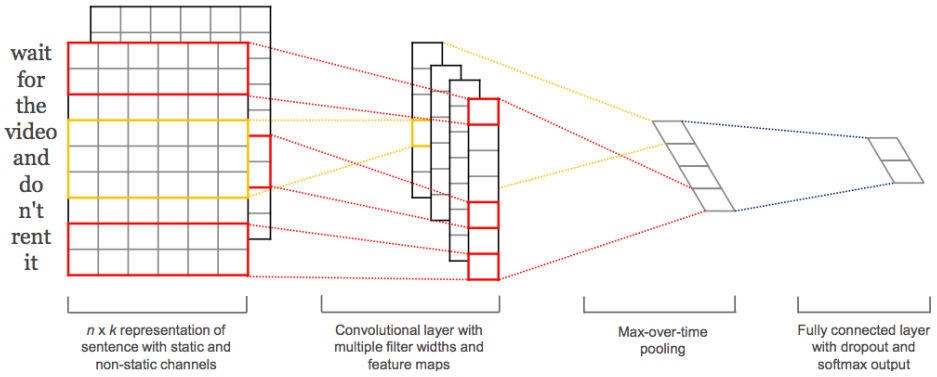
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$$

- 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

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \qquad \hat{c} = \max\{\mathbf{c}\}$$



Yoon Kim - Convolutional Neural Networks for Sentence Classification

CNN for text classification

Many variations of the model [1]

- 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)

[1] Y. Kim, Convolutional Neural Networks for Sentence Classification, EMNLP 2014

CNN for text classification

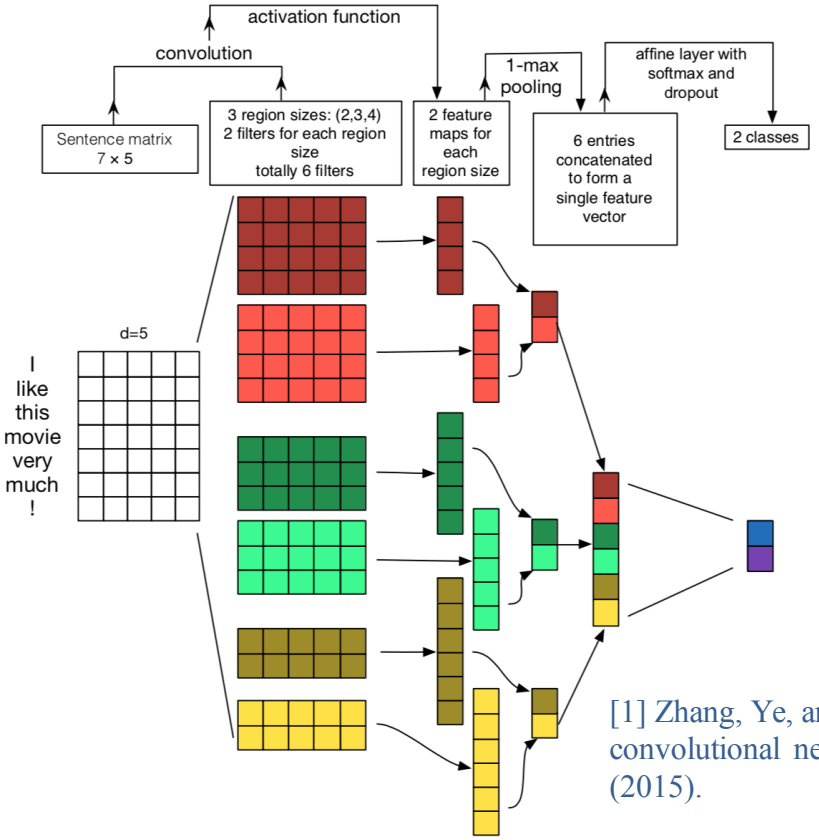
- Combine multiple word embeddings
- Each set of vectors is treated as a ‘channel’
- Filters applied to all channels
- Gradients are back-propagated only through one of the channels
- Fine-tunes one set of vectors while keeping the other static

CNN for text 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**	–	–

Accuracy scores (Kim et al vs others)

CNN architecture for (short) document classification



- Data (text) only 1st column of input
- Rest of each row: embedding (in images 2D+RGB dimension)
- Filters of different sizes (4x5, 3x5 etc.)
  - Each size captures different features (need ~10^2 filters/size)
- Feature maps:
  - As many as the times filter fits on data matrix
- Max pooling maintains the "best features"
- Global feature map => classification via softmax

[1] Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification." arXiv preprint arXiv:1510.03820 (2015).

CNN architecture for (short) document classification – T-SNE visualization

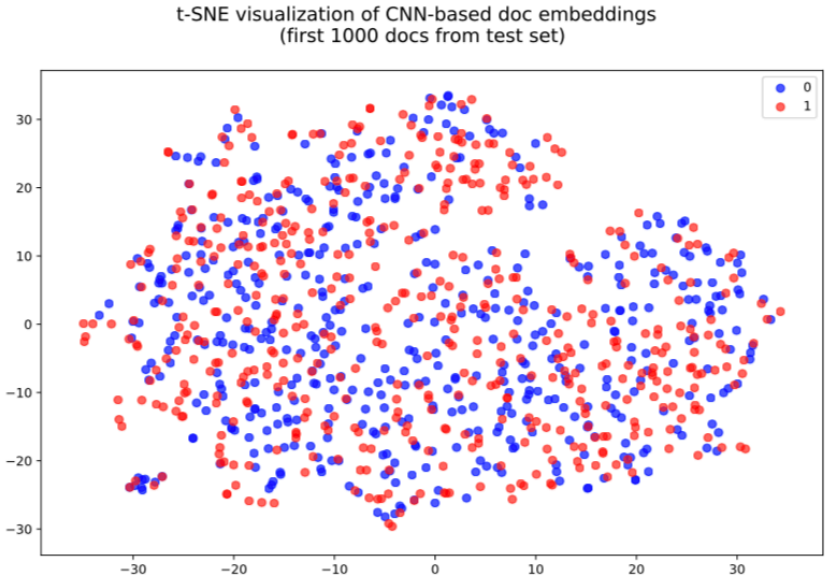


Figure 2: Doc embeddings before training.

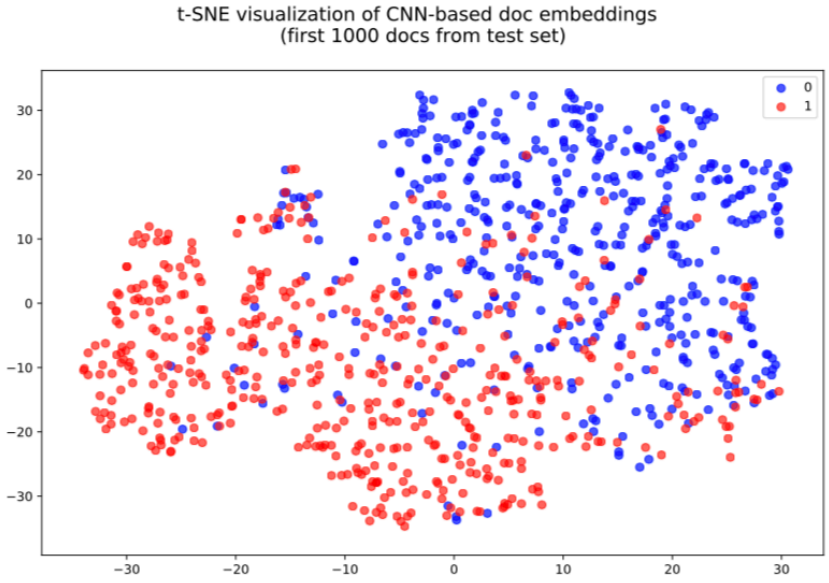


Figure 3: Doc embeddings after 2 epochs.

CNN architecture for (short) document classification - Saliency maps

- Words most related to changing the doc classification
- A in R^{sxd}, s:# sentence words, d:size of embeddings

$$\text{saliency}(a) = \left| \frac{\partial(\text{CNN})}{\partial a} \Big|_a \right|$$

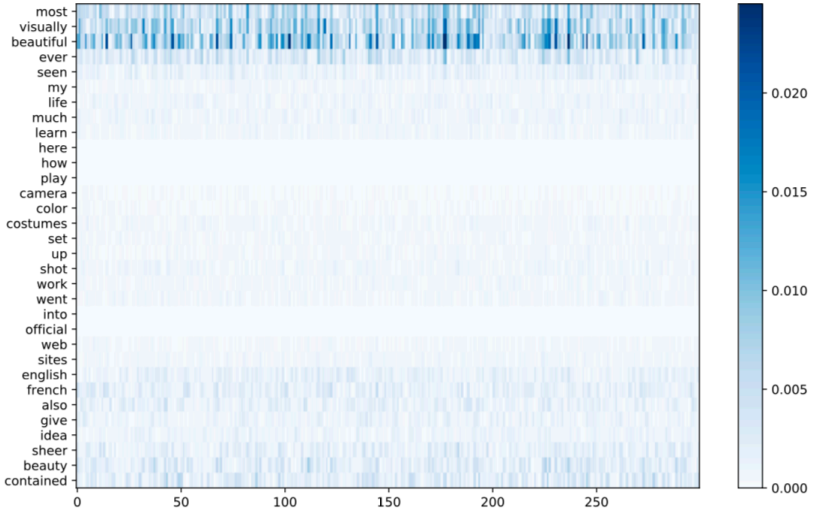


Figure 4: Saliency map for document 1 of the IMDB test set (true label: positive)

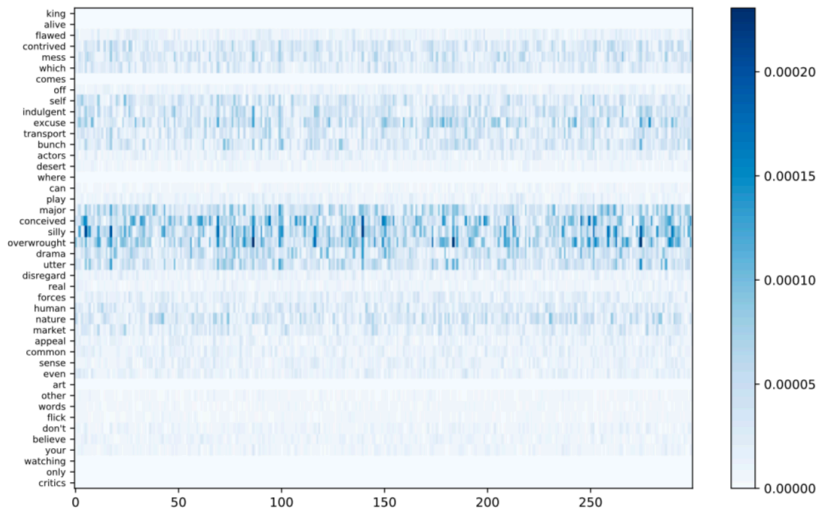
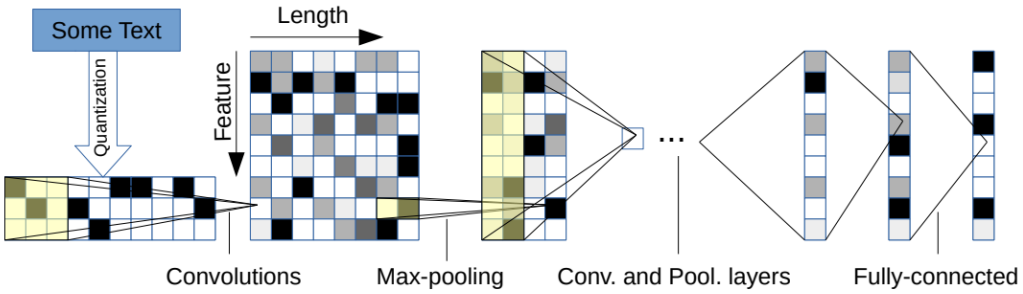


Figure 5: Saliency map for document 15 of the IMDB test set (true label: negative)

Character-level CNN for Text Classification

- Input: sequence of encoded characters
- quantize each character using “one-hot” encoding
- input feature length is 1014 characters
- 1014 characters able capture most of the texts of interest
- Also perform Data Augmentation using Thesaurus as preprocessing step

Model Architecture



- 9 layers deep
- 6 convolutional layers
- 3 fully-connected layers
- 2 dropout modules in between the fully-connected layers for regularization

Model Comparison

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg. w2v Conv.	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm. w2v Conv.	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg. w2v Conv. Th.	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm. w2v Conv. Th.	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg. Lk. Conv.	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm. Lk. Conv.	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg. Lk. Conv. Th.	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

Testing errors for all models

Blue->best, Red->worst

Links

- http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- https://arxiv.org/pdf/1509.01626.pdf
- http://www.aclweb.org/anthology/D14-1181
- http://cs231n.github.io/convolutional-networks/
- http://ufldl.stanford.edu/tutorial/supervised/Pooling/

- **MLP + CNN Applications**
  - Word embeddigs

Language model

- Goal: determine $P(s = w_1 \dots w_k)$ in some domain of interest

$$P(s) = \prod_{i=1}^{k} P\,(w_i \mid w_1 \dots w_{i-1})$$

e.g., $P\,(w_1 w_2 w_3) = P\,(w_1)\,P\,(w_2 \mid w_1)\,P\,(w_3 \mid w_1 w_2)$

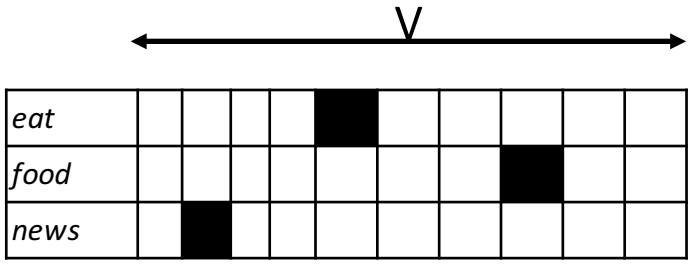
- Traditional n-gram language model assumption:
  "the probability of a word depends only on **context** of $n-1$ previous words"

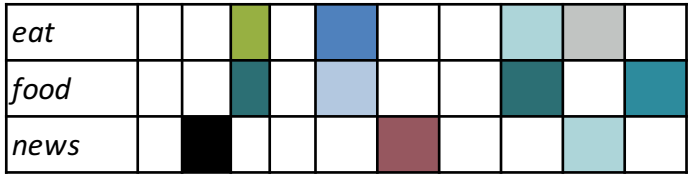
$$\Rightarrow \widehat{P}(s) = \prod_{i=1}^{k} P\,(w_i \mid w_{i-n+1} \dots w_{i-1})$$

- Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute $\widehat{P}\,(w_i \mid w_{i-n+1} \dots w_{i-1}) = \frac{\#w_{i-n+1} \dots w_{i-1} w_i}{\#w_{i-n+1} \dots w_{i-1}}$ on training corpus
  2. smooth to avoid zero probabilities

Representing Words

- **One-hot vector**
  - high dimensionality
  - sparse vectors
  - dimensions=|V| (10^6<|V|)
  - unable to capture semantic similarity between words
- **Distributional vector**
  - words that occur in similar contexts, tend to have similar meanings
  - each word vector contains the frequencies of all its neighbors
  - dimensions=|V|
  - computational complexity for ML algorithms

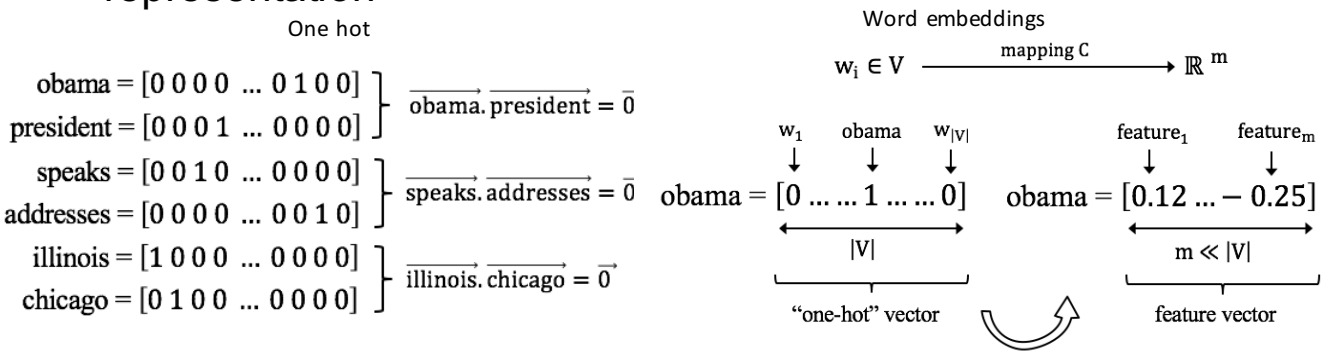




Example

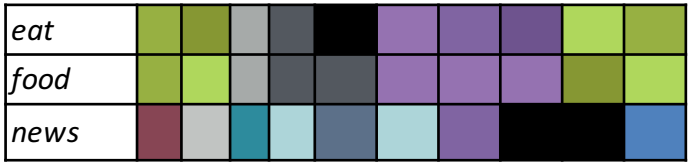
- We should assign similar probabilities (discover similarity) to *Obama speaks to the media in Illinois* and the *President addresses the press in Chicago*

- This does not happen because of the "one-hot" vector space representation



Representing Words

- **Word embeddings**
  - store the same contextual information in a low-dimensional vector
  - **densification** (sparse to dense)
  - **compression**
    - dimensionality reduction
    - dimensions=m 100<m<500
  - able to capture semantic similarity between words
  - learned vectors (unsupervised)
  - Learning methods
    - **SVD**
    - **word2vec**
    - **GloVe**



SVD word embeddings

- Dimensionality reduction on co-occurrence matrix
- Create a |V|x|V| word co-occurrence matrix X
- Apply SVD $X = USV^T$
- Take first k columns of U
- Use the k-dimensional vectors as representations for each word
- Able to capture semantic and syntactic similarity

Latent Semantic Indexing (LSI)

- The initial matrix is SVD decomposed as: $A=ULV^T$
- Choosing the top-k singular values from L we have:

$$A_k=U_kL_kV_k^T,$$

- L_k square kxk - top-k singular values of the diagonal in matrix L,
- $U_{k,}$ mxk matrix - first k columns in U (left singular vectors)
- $V_k^{T,}$ kxn matrix - first k lines of V^T (right singular vectors)

Typical values for κ~200-300 (empirically chosen based on experiments appearing in the bibliography)

LSI capabilities

- Term to term similarity: $A_k A_k^T = U_k L_k^2 U_k^T$

  Where $A_k = U_k L_k V_k^T$

- Document-document similarity: $A_k^T A_k = V_k L_k^2 V_k^T$

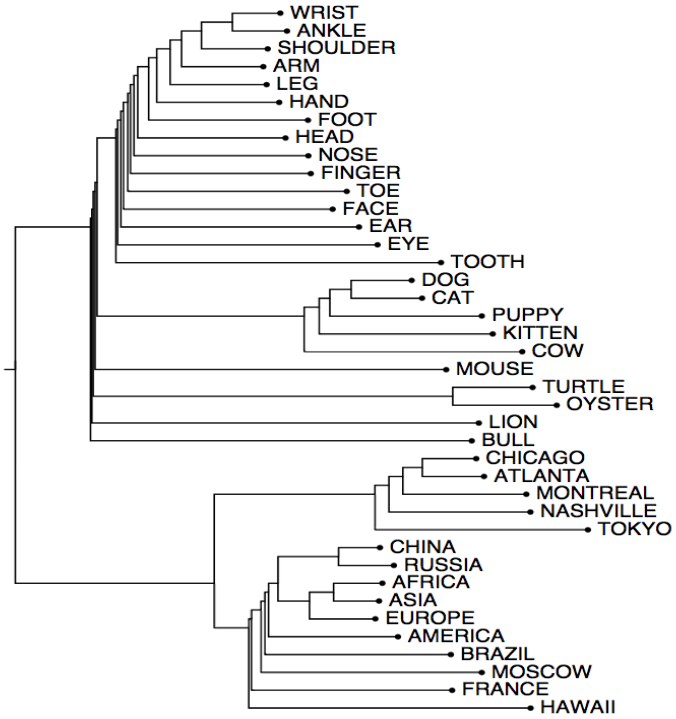
- Term document similarity (as an element of the transformed – document matrix)

- Extended query capabilities transforming initial query q to q_n :

$q_n = q^T U_k L_k^{-1}$

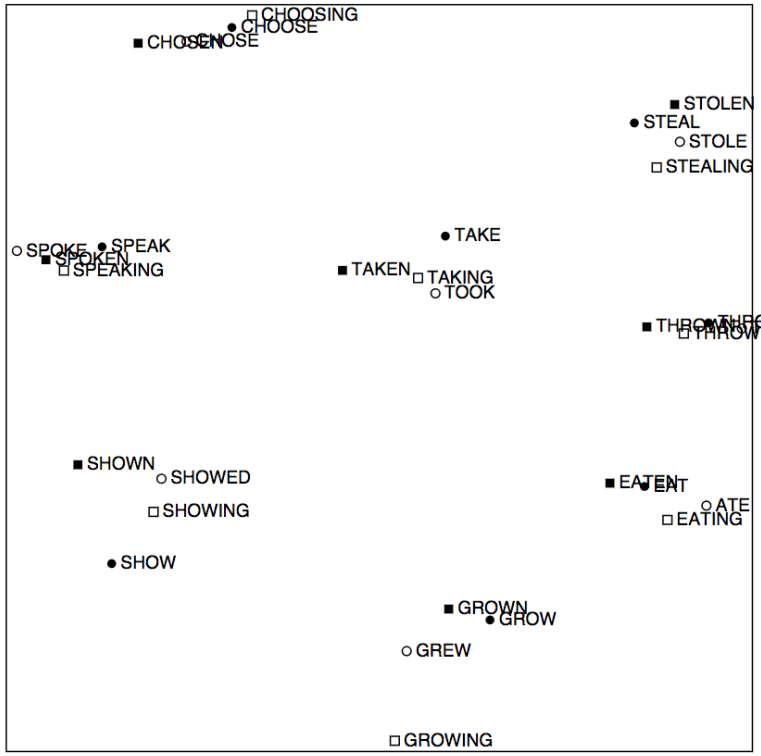
- Thus q_n can be regarded a line in matrix V_k

Patterns in SVD Space



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Patterns in SVD Space



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

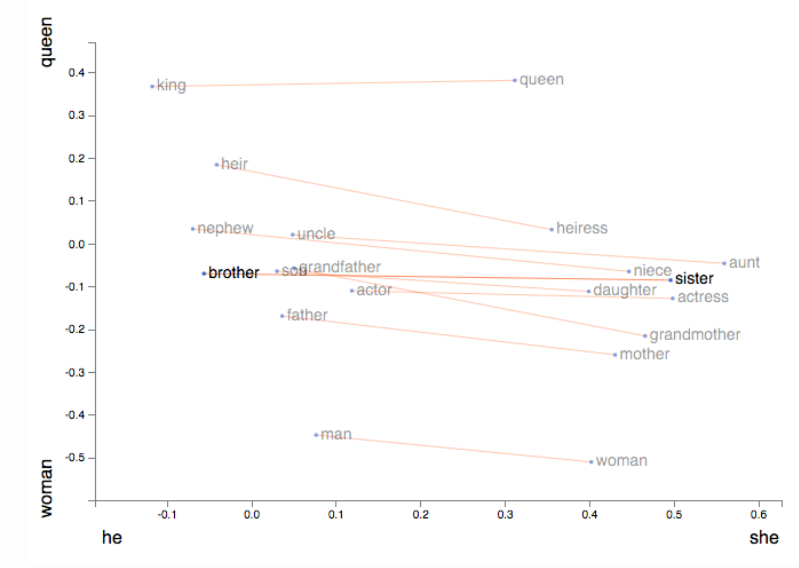
SVD problems

- The dimensions of the matrix change when dictionary changes
- The whole decomposition must be re-calculated when we add a word
- Sensitive to the imbalance in word frequency
- Very high dimensional matrix
- Not suitable for millions of words and documents
- Quadratic cost to perform SVD
- Solution: Directly calculate a low-dimensional representation

Word analogy

- Words with similar meaning laying close to each other
- Words sharing similar contexts may be analogous
  - Synonyms
  - Antonyms
  - Names
  - Colors
  - Places
  - Interchangeable words

- Vector computations with analogies
- i.e. **king - man + woman = queen**



https://lamyiowce.github.io/word2viz/

But why?

• what's an analogy?

$$\frac{p(w'|man)}{p(w'|woman)} \approx \frac{p(w'|king)}{p(w'|queen)}$$

Assume PMI approximated by low rank approximation of the co-occurrence matrix.

1. $PMI(w',w) \approx v_w v_{w'}$ *inner product of the vector representations*
2. Isotropic: $E_{w'}[(v_{w'} v_u)]^2 = ||v_u||^2$

Then

3. $argmin_w \; E_{w'}[ln\frac{p(w'|w)}{p(w'|queen)} - ln\frac{p(w'|man)}{p(w'|woman)}]^2$

4. $argmin_w \; E_{w'}[\big(PMI(w'|w) - PMI(w'|queen)\big) - (PMI(w'|man) - PMI(w'|woman))]^2$

5. $argmin_w ||\big(v_w - v_{queen}\big) - (v_{man} - v_{woman})||^2$
6. $v_w \approx v_{queen} - v_{woman} + v_{man}$ which is an analogy!

- *Arora et al (ACL 2016)* shows that if (2) holds then (1) holds as well
- So we need to construct vectors from co-occurrence that satisfy (2)
- d<<|V| in order to have isotropic vectors

A Latent Variable Model Approach to PMI-based Word Embeddings, *Arora et al (ACL 2016)*

Learning Word Vectors

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

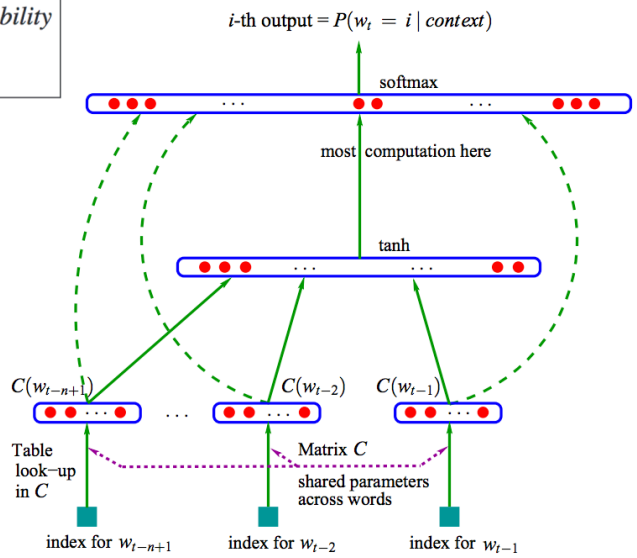
- Corpus containing T training words
- Objective: $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$
- Decomposed in two parts:

$$w_i \in V \xrightarrow{\text{mapping C}} \mathbb{R}^m$$

  - Mapping **C** (1-hotv => lower dimensions)
  - Mapping any **g** s.t. (estimate prob t+1 | t previous)

$$f(w_{t-1}, \cdots, w_{t-n+1}) = g(C(w_{t-1}), \cdots, C(w_{t-n+1}))$$

- C(i): i-th word feature vector (Word embedding)
- Objective function: $J = \frac{1}{T} \sum f(w_t, \ldots, w_{t-n+1})$

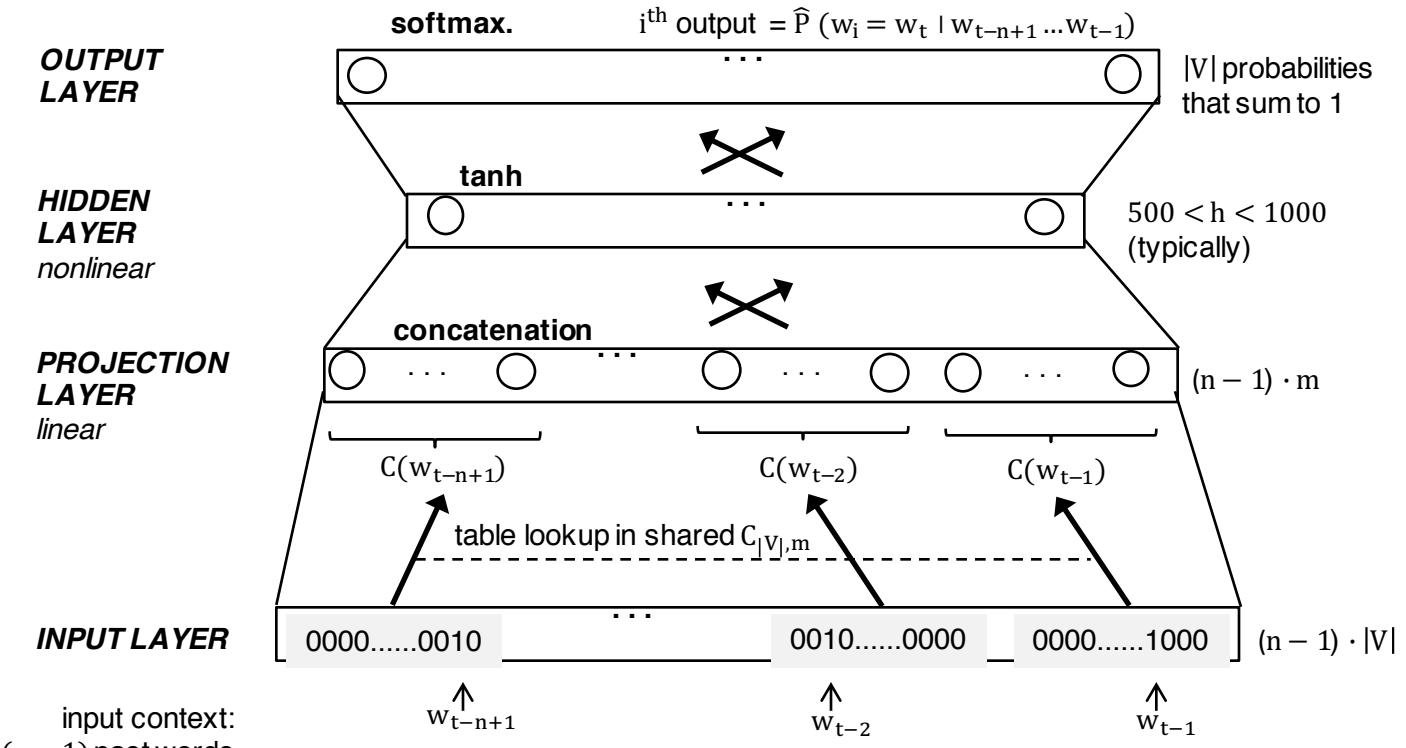


Bengio, Yoshua, et al. "A neural probabilistic language model." *The Journal of Machine Learning Research* 3 (2003): 1137-1155.

Neural Net Language Model

For each training sequence: input = (context, target) pair: $(w_{t-n+1} \ldots w_{t-1}, w_t)$

objective: minimize $E = -\log \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$



Objective function

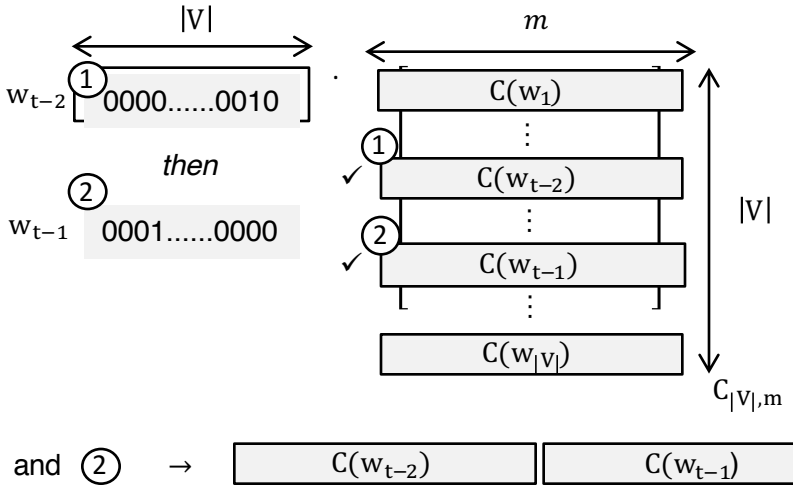
- $E = -log\ \hat{P}(w_t\ |w_{t-n+1} \dots w_{t-1})$
- a probability between 0 and 1.
- On this support, the log is negative => –log term positive.
- makes sense to try to minimize it.

  - Probability of word given the context be as high as possible (1 for a perfect prediction).
  - case the error is equal to 0 (global minimum).

p	log(p)	-log(p)
0,7	-0,15490196	0,15490196
0,2	-0,698970004	0,698970004

NNLM Projection layer

- Performs a simple table lookup in $C_{|V|,m}$: concatenate the rows of the shared mapping matrix $C_{|V|,m}$ corresponding to the context words
  Example for a two-word context $w_{t-2}w_{t-1}$:



- $C_{|V|,m}$ is **critical**: it contains the weights that are tuned at each step. After training, it contains what we're interested in: the **word vectors**

NNLM hidden/output layers and training

- Softmax (log-linear classification model) outputs positive numbers that sum to one (a multinomial probability distribution):

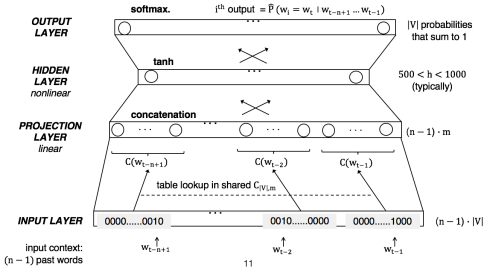
  i^{th} unit in the output layer: $\hat{P}(w_i = w_t \mid w_{t-n+1} \dots w_{t-1}) = \frac{e^{y_{w_i}}}{\sum_{i'=1}^{|V|} e^{y_{w_{i'}}}}$

  where:
  - $y = b + U.tanh(d + H.x)$
  - $\tanh$: nonlinear squashing (link) function
  - x : concatenation $C(w)$ of the context weight vectors seen previously
  - b : output layer biases ($|V|$ elements)
  - d : hidden layer biases (h elements). Typically $500 < h < 1000$
  - U : $|V| * h$ matrix storing the *hidden-to-output* weights
  - H : $(h * (n-1)m)$ matrix storing the *projection-to-hidden* weights

  $\rightarrow \boldsymbol{\theta} = (\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{U}, \boldsymbol{H}, \boldsymbol{C})$

  - Complexity per training sequence: $n * m + n * m * h + \mathbf{h} * |\mathbf{V}|$

    computational bottleneck: **nonlinear hidden layer** ($h * |V|$ term)



- **Training** performed via stochastic gradient descent (learning rate ε):

$$\theta \leftarrow \theta + \varepsilon \cdot \frac{\partial E}{\partial \theta} = \theta + \varepsilon \cdot \frac{\partial \log \hat{P}\,(w_t \mid w_{t-n+1} \dots w_{t-1})}{\partial \theta}$$

(weights are initialized randomly, then updated via backpropagation)

NNLM facts

- tested on Brown (1.2M words, $|V| \cong 16K$) and AP News (14M words, $|V| \cong 150K$ reduced to 18K) corpuses
- h: # hidden units, n=# words in context, m: dimensionality of word vectors.
- Brown: $h = 100, n = 5, m = 30$
- AP News: $h = 60, n = 6, m = 100$, **3 week** training using **40 cores**
- 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs
- in terms of test *set perplexity*: geometric average of $1/\widehat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$
- Due to **complexity**, NNLM can't be applied to large data sets → poor performance on rare words
- Bengio et al. (2003) claims main contribution was a more accurate LM. They let the interpretation and use of the word vectors as **future work**
- On the opposite, Mikolov et al. (2013) focus on the **word vectors**

Word2Vec

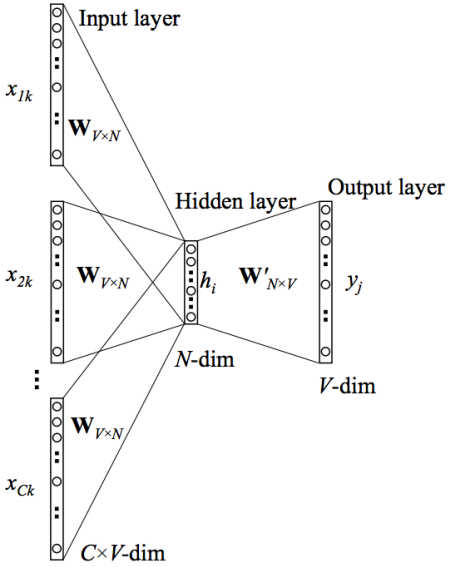
- Mikolov et al. in 2013
- Key idea of word2vec: achieve better performance not by using a more complex model (i.e., with more layers), but by allowing a **simpler (shallower) model** to be trained on **much larger amounts of data**
- no hidden layer (leads to 1000X speedup)
- projection layer is shared (not just the weight matrix) - C
- context: words from both history & future:

• Two algorithms for learning words vectors:

- **CBOW**: from context predict target
- **Skip-gram**: from target predict context

Continuous Bag Of Words (CBOW)

- continuous bag-of-words
- continuous representations whose order is of no importance
- uses the surrounding words to predict the center word
- n-words before and after the target word

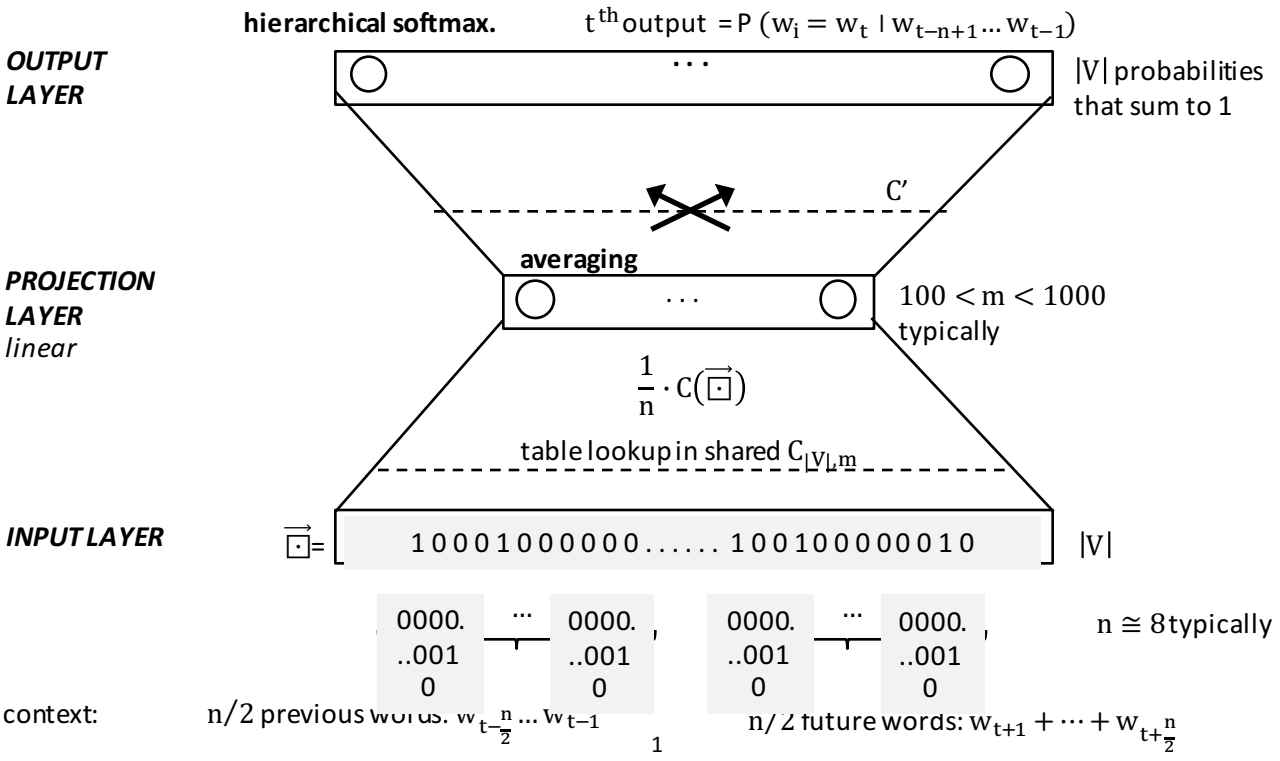


Efficient Estimation of Word Representations in Vector Space- Mikolov et al.

Continuous Bag-of-Words (CBOW)

For each training sequence: input = (context, target) pair: $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

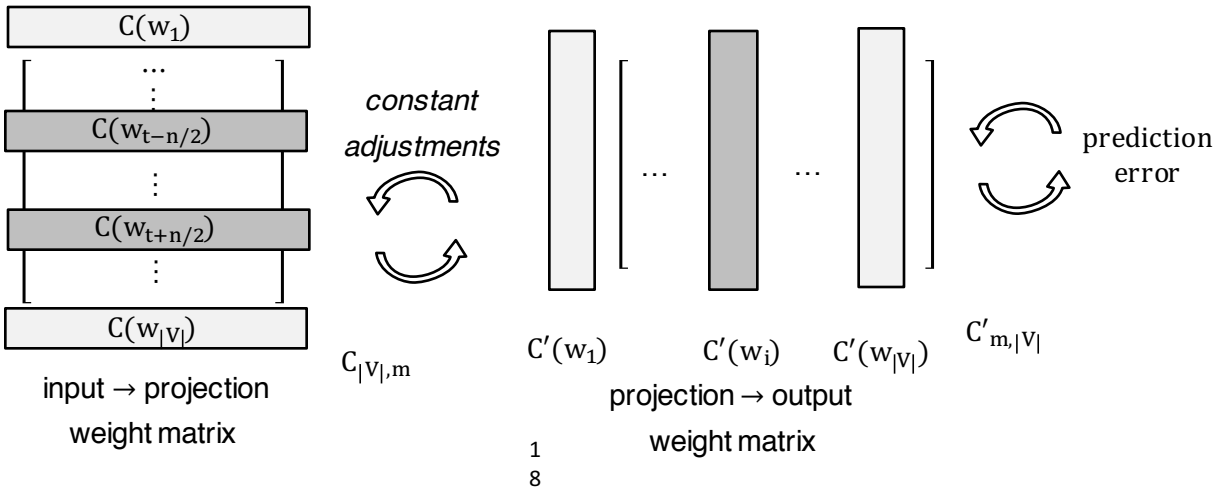
objective: minimize $-\log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$



Weight updating

- For each (context, target=w_t) pair, only the word vectors from matrix C corresponding to the context words are updated
- Recall that we compute $P(w_i = w_t \mid \text{context}) \forall w_i \in V$. We compare this distribution to the true probability distribution (1 for w_t, 0 elsewhere)
- **Back propagation**
- If $P(w_i = w_t \mid \text{context})$ is **overestimated** (i.e., > 0, happens in potentially $|V| - 1$ cases), some portion of $C'(w_i)$ is **subtracted** from the context word vectors in C, proportionally to the magnitude of the error
- Reversely, if $P(w_i = w_t \mid \text{context})$ is **underestimated** (< 1, happens in potentially 1 case), some portion of $C'(w_i)$ is **added** to the context word vectors in C

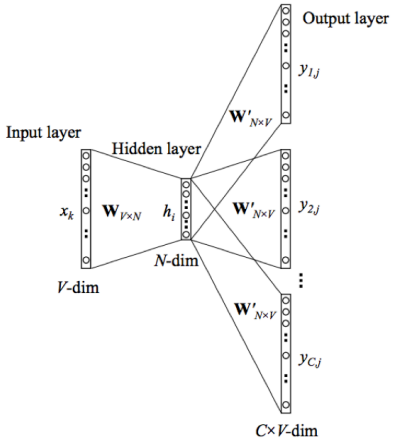
→ at each step the words move away or get closer to each other in the feature space → clustering



Skip-gram

- skip-gram uses, in a context, the center word to predict the surrounding words
- instead of computing the probability of the target word w_t given its previous words, we calculate the probability of the surrounding word w_{t+j} given w_t
- $\mathrm{p}(w_{t+j}|w_t) = \frac{\exp(\boldsymbol{v}_{w_t}^{T}\boldsymbol{v}'_{w_{t+j}})}{\sum_{w\in V}\exp(\boldsymbol{v}_{w_t}^{T}\boldsymbol{v}'_{w_{t+j}})}$
- $\boldsymbol{v}^{T}{}_{wt}$ is a column of $\boldsymbol{W}_{VxN}$ and $\boldsymbol{v}'_{\boldsymbol{w_{t+j}}}$ is a column of $\boldsymbol{W'}_{NxV}$
- Objective function

$$J = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\le j\le n}\log \mathrm{p}(w_{t+j}|w_t)$$



Efficient Estimation of Word Representations in Vector Space- Mikolov et al. 2013

word2vec facts

- Complexity is $\mathbf{n} * \mathbf{m} + \mathbf{m} * \mathbf{log}|\mathbf{V}|$ (Mikolov et al. 2013a)
- **n**:size of the context window (~10) **nxm**: dimensions of the projection layer, **|V|** size of the vocabulary
- On Google news 6B words training corpus, with $|\mathbf{V}| \sim 10^6$:
  - CBOW with $\mathrm{m} = 1000$ took **2 days** to train on **140 cores**
  - Skip-gram with $\mathrm{m} = 1000$ took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for $\mathrm{m} = 100$ only!
    (note that $\mathrm{m} = 1000$ was not reasonably feasible on such a large training set)
- word2vec training speed ≅ 100K-5M words/s
- Quality of the word vectors:
  - ↗ significantly with **amount of training data** and **dimension of the word vectors** (m), with diminishing relative improvements
  - measured in terms of accuracy on 20K semantic and syntactic association tasks.
    e.g., words in **bold** have to be returned:

Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%
- References: http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd
  https://code.google.com/p/word2vec/

GloVe

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

- Ratio of co-occurrence probabilities best distinguishes relevant words

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \quad \Rightarrow \quad w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

- *Cast this into a least square problem:*

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}.$$

- X co-occurrence matrix
- f weighting function,
- b bias terms
- $w_i = word\ vector$
- $\widetilde{w_j} = context\ vector$

model that utilizes
- count data
- bilinear prediction-based methods like word2vec

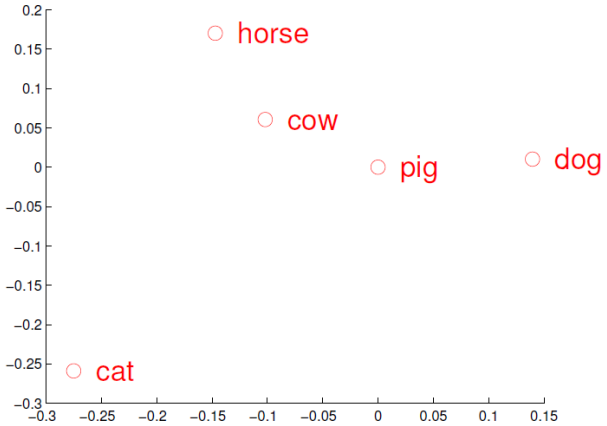
https://nlp.stanford.edu/pubs/glove.pdf

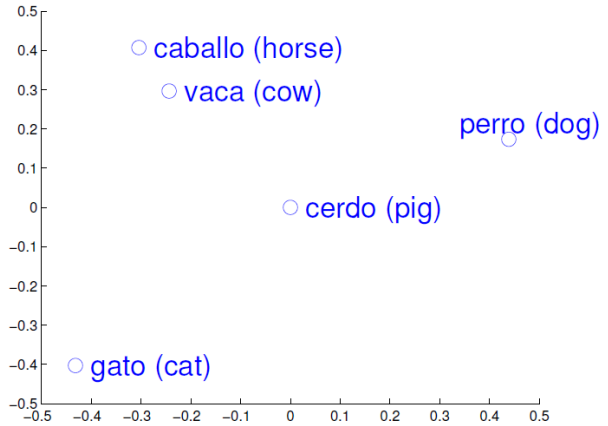
Which is better?

- Open question
- SVD vs word2vec vs GloVe
- All based on co-occurrence
- Levy, O., Goldberg, Y., & Dagan, I. (2015)
  - SVD performs best on similarity tasks
  - Word2vec performs best on analogy tasks
  - *No single algorithm consistently outperforms the other methods*
  - *Hyperparameter tuning is important*
  - 3 out of 6 cases, tuning hyperparameters is more beneficial than increasing corpus size
  - word2vec outperforms GloVe on all tasks
  - *CBOW is worse than skip-gram on all tasks*

Applications

- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval…
- Example for English to Spanish machine translation:

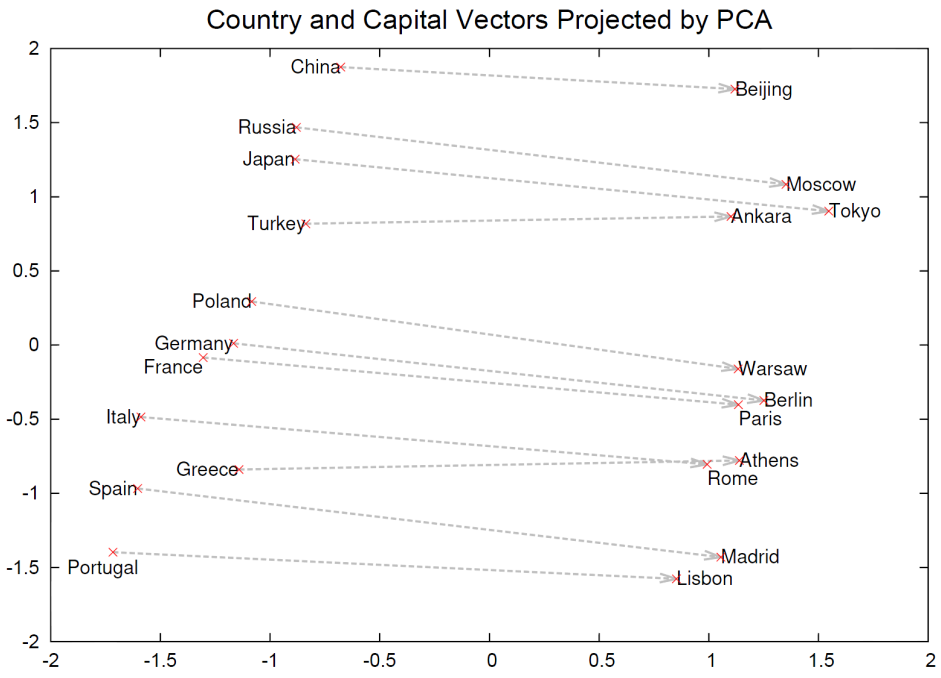




About 90% reported accuracy (Mikolov et al. 2013c)

Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Remarkable properties of word vectors



regularities between words are encoded in the difference vectors
e.g., there is a constant **country-capital** difference vector

Mikolov et al. (2013b)
Distributed representations of words and phrases and their compositionality

Outline

- Neural Networks – Introduction
- CNNs
- Applications in NLP
- **MLP Hyper-parameter tuning**

Hyper Parameterizing Deep Learning

Contributions by J.B. Remy@LIX

Hyper Parameterizing Deep Learning

In this lecture we are going to consider that we want train a deep neural network on a supervised dataset $\{(x_i, y_i)\}_{i=1,\dots,N}$:

$$F(\theta, x_i) = \hat{y}_i \text{ and } L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(\hat{y}_i, y_i)$$

Where $\theta = w$ is the set of parameters of the network, F is the function representing the network, L is the general loss of the network and l the loss for one element.

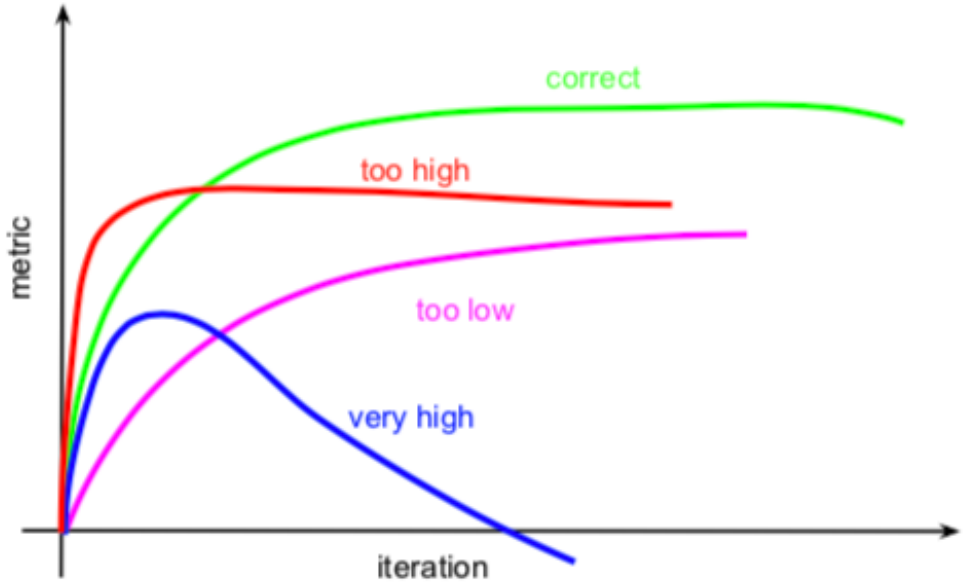
The general algorithm to train deep neural networks is **Batch Stochastic Gradient Descent (SGD)**. At each iteration t:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \hat{L}_t$$

Where η is the learning rate . And $\hat{L}_t = \frac{1}{B}\sum_{i=1}^{B} l(\hat{y}_{b_i}, y_{b_i})$, with $\{b_1, \dots, b_B\}$ the indicies of the elements in the batch.

Learning rate Schedules

- For SGD the learning rate is fixed,
- but we need to tune the learning rate. Typically try $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$.



Decaying learning rate

- Flattening of the training curve： learning rate is high to reach an optimum.
- i.e. θ oscilates arround an optimum. Decaying the learning rate once (by an order of magnitude) allows to reach closer to this optimum.

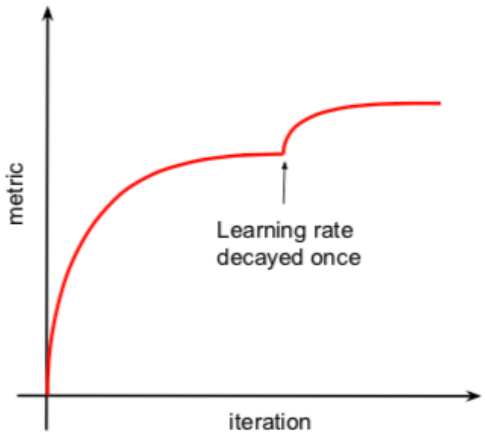


Figure: Decaying the learning rate once

Learning rate can be decayed

- on schedule (after a predifined number of iterations), or
- when network stops learning

Learning Rate Annealing / Step-Wise Decay

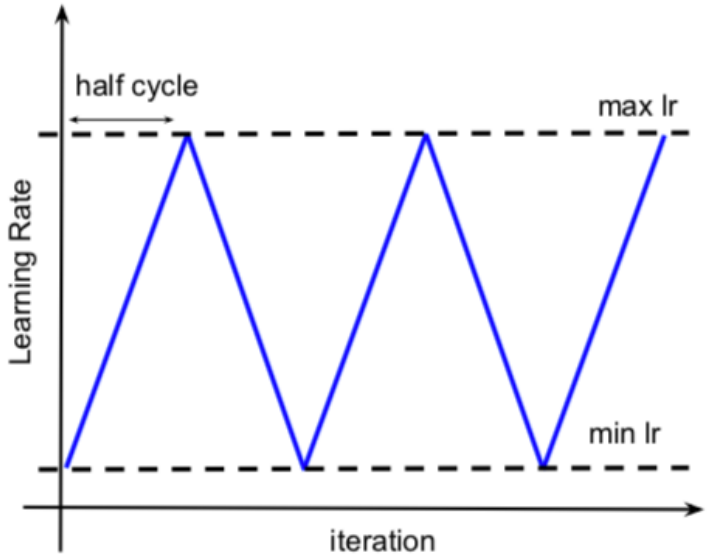
Learning rate annealing, consists in continuously the learning rate during training.

$$\eta_t = \frac{\eta_0}{1 + decay * t}$$

For example $decay = 10^{-6}$. The point is to reach a minimum as fast as possible. Many variations of this decay exist, like exponential decay $\eta_t = \eta_0 e^{-decay*t}$.

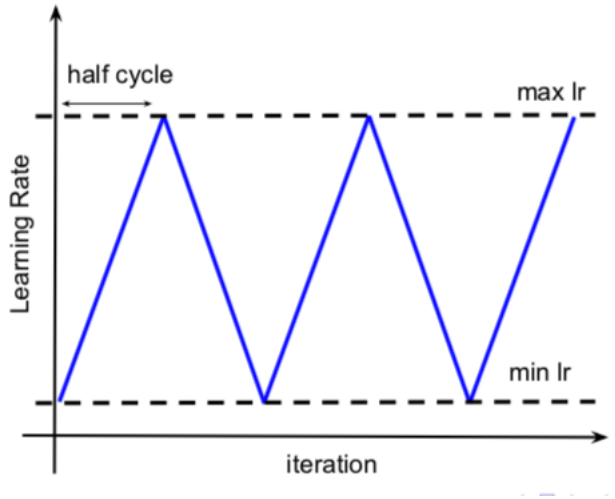
Learning rate - Triangular Schedule

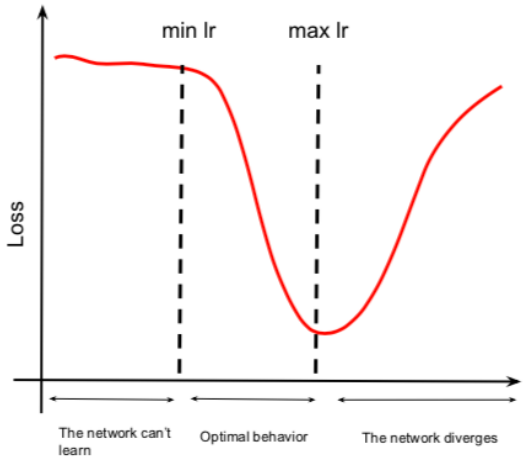
- triangular schedule: sequentially increase and decrease the learning rate.
- Intuition:
  - increasing the learning rate allows to escape local minima,
  - deacreasing it allows to correctly train the network.



Triangular schedule- boundaries

- choose extreme values of learning rates for which the network is able to learn.
- start very low (10^{-3}) and augment the learning rate for one half cycle to a very high value (1).
- choose with respect to the training curve.





Momentum and adaptive agforithms

- SGD is extremely noisy.
- To converge smoothly and faster: momentum keeps a memory of the previous updates.

$$U_t = \beta U_{t-1} + (1-\beta)\nabla_\theta \hat{L}_t$$

$$\theta_{t+1} = \theta_t - \eta U_t$$

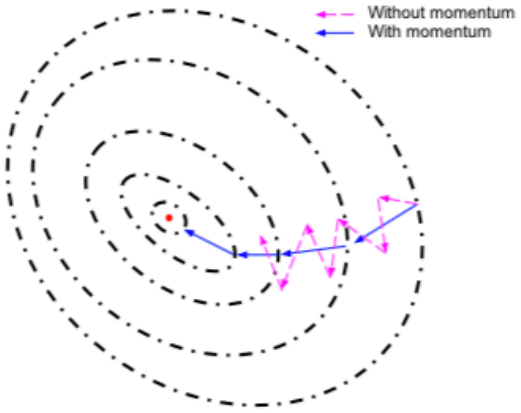


Figure: Convergence with momentum

ADAM and RMSProp

- Adaptive gradients, adapts the learning rate independently for every parameter, thus convergence is faster.
- RMSProp writes, for every parameter w:

$$v_t = \beta v_{t-1} + (1-\beta)\left(\nabla_w \hat{L}_t\right)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \nabla_w \hat{L}_t$$

- ADAM combines momentum and adaptive gradient.
- Adaptive algorithm are very efficient to speed up convergence. BUT They have a major fall off, they tend to stay trapped in local minima or gradients deserts.
- The recent trend tends to standard SGD with learning rate decay, or cyclical learning rate schedules.
- However, if computation time is an issue, adaptive gradients are a suitable solution.

Regularization

Regularization of a neural network can also be standard, with a constrained. With a L_2-constraint it is called weight decay . This correspond to adding the following penalty to the loss $\lambda \sum_w w^2$. Then the update writes:

$$w_{t+1} = \theta_t - \eta \nabla_w \hat{L}_t - 2\lambda w$$

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*