

Neural Networks and Deep Learning - I

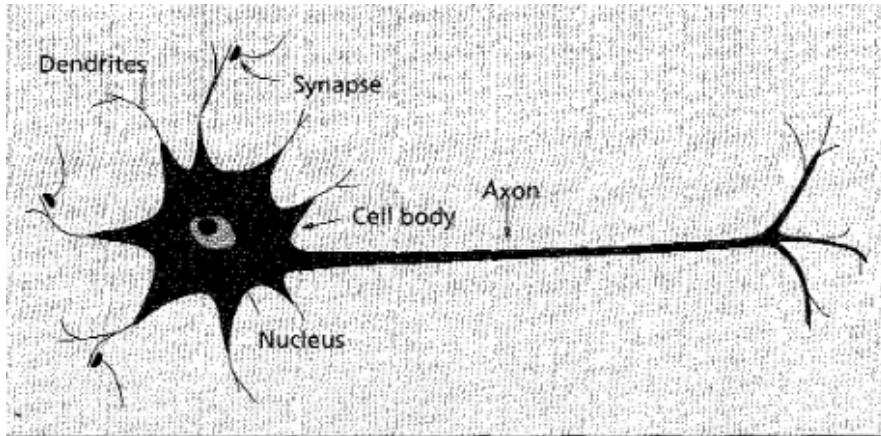
M. Vazirgiannis

October 2018

Outline

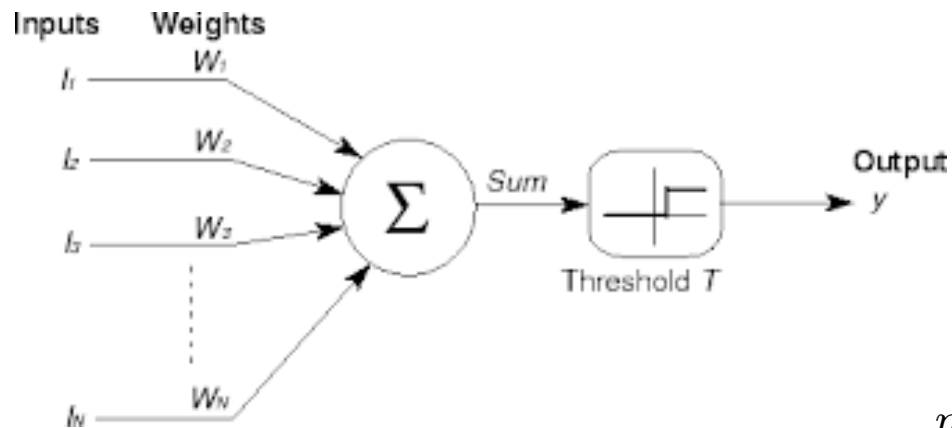
- **Neural Networks - Introduction**
- Back Propagation
- CNNs
- RNNs

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)

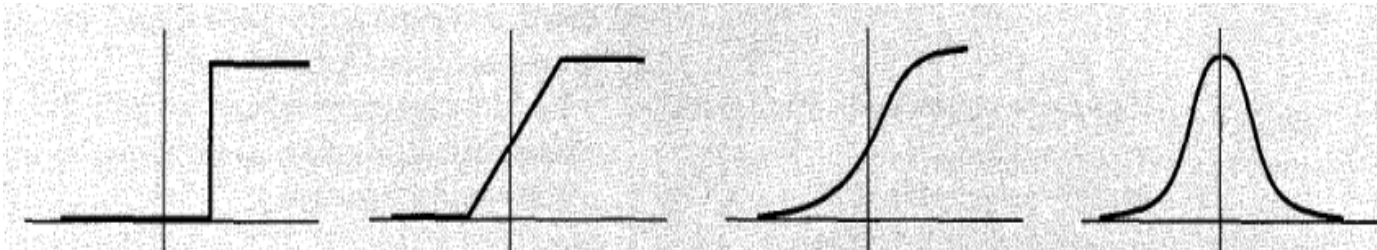
Neural networks



McCulloch-Pitts Neuron Model

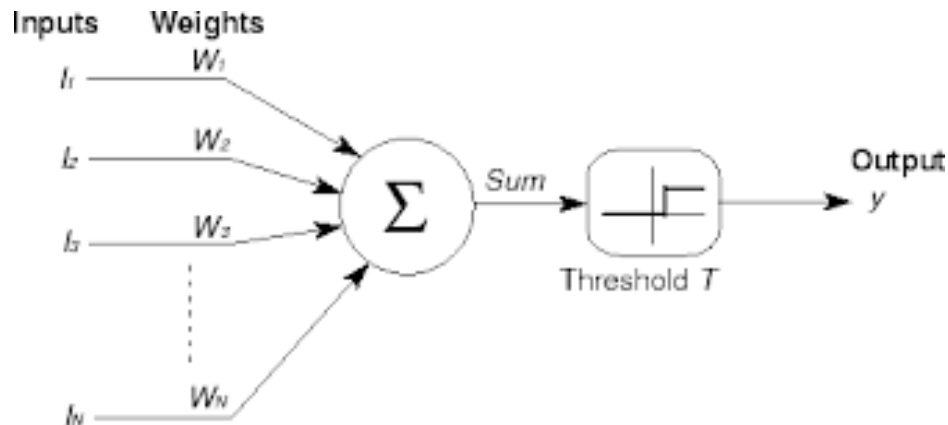
- θ : step function at 0
- w_j : weight of the j -th input
- u : bias
- activation functions: piecewise linear, sigmoid, or Gaussian

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



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.

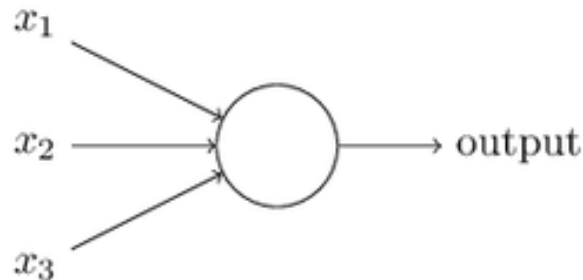


Perceptron learning algorithm

- developed in the 1950 -60 by Frank Rosenblatt
- takes several binary inputs, x_1, x_2, \dots 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 $X = \{x_1, \dots, x_n\}$ 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.

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

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

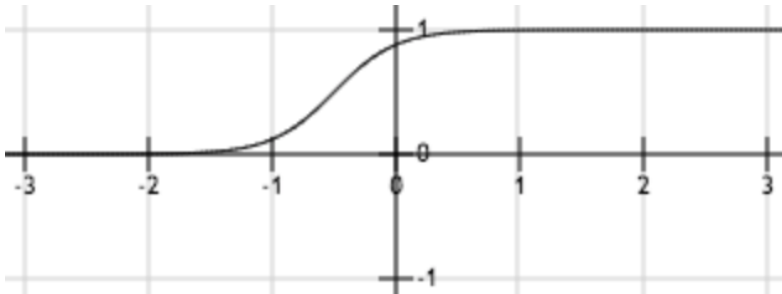
If training data are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations.

Neural Networks – sigmoid neuron

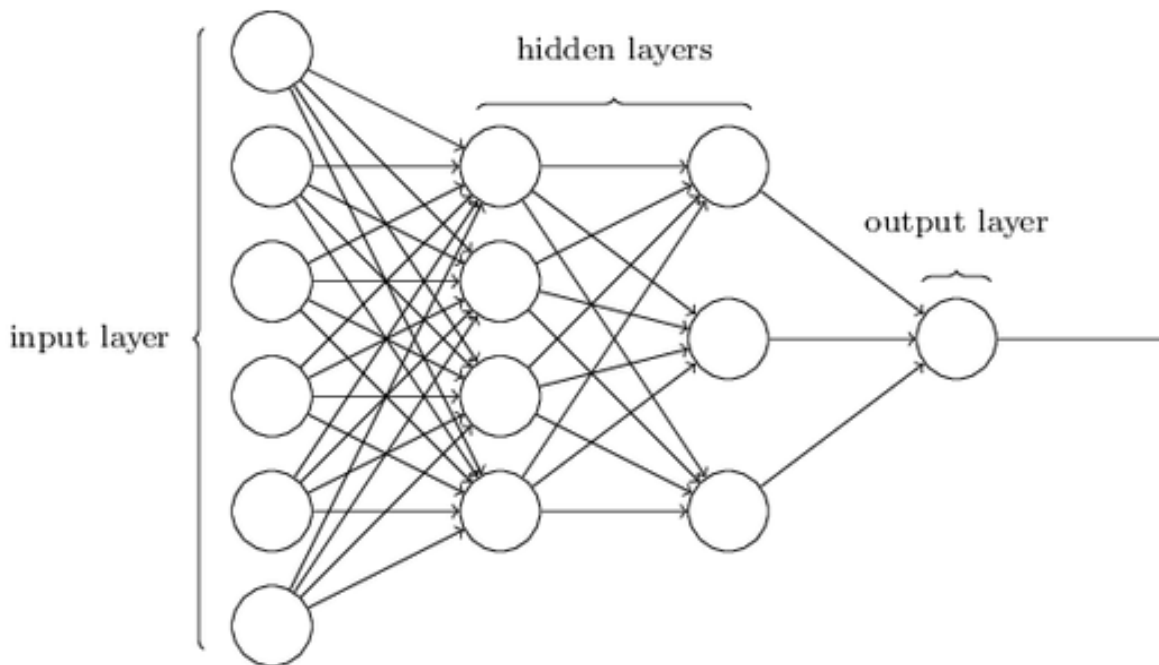
- In perceptron small changes to the weights may cause huge difference to output (0->1)
- We want to impose: small changes to weights (or bias) small change to the output.
- the sigmoid neuron has weights for each input, w_1, w_2, \dots , and an overall bias b . The output is $\sigma(w \cdot x + b)$,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

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

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




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

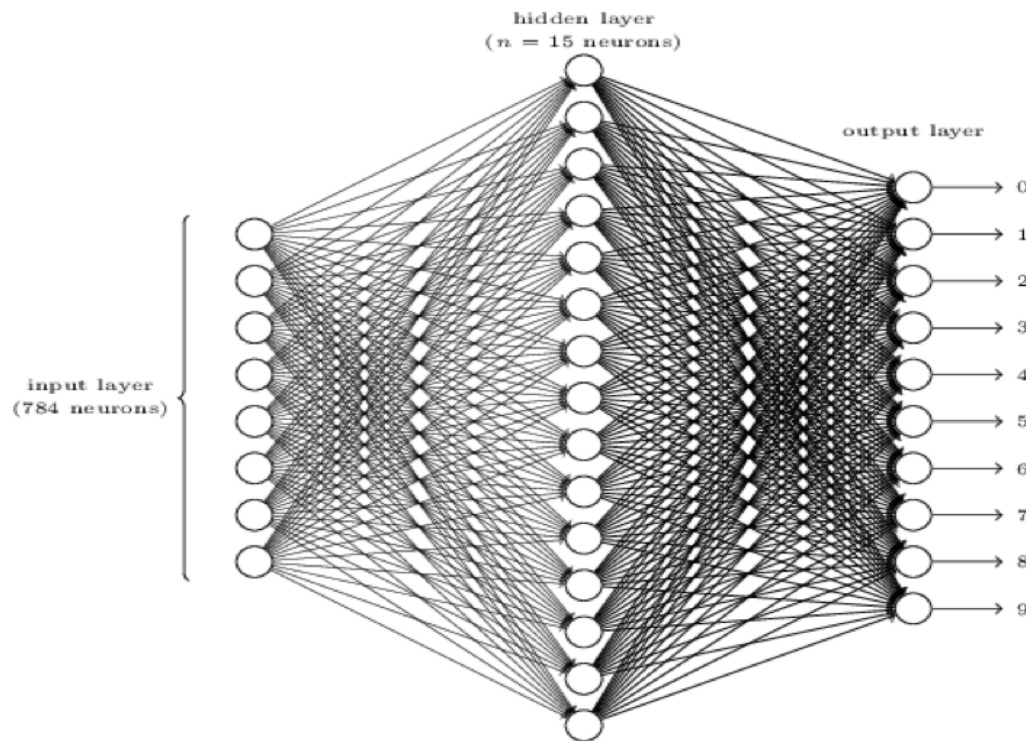
- Assume hand written digits (MNIST database – 60.000 images) :
- Each of the 
- Assume a NN with 784 inputs, a hidden layer of 15 neurons

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.



Learning with gradient descent

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

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

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

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, \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

- 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 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

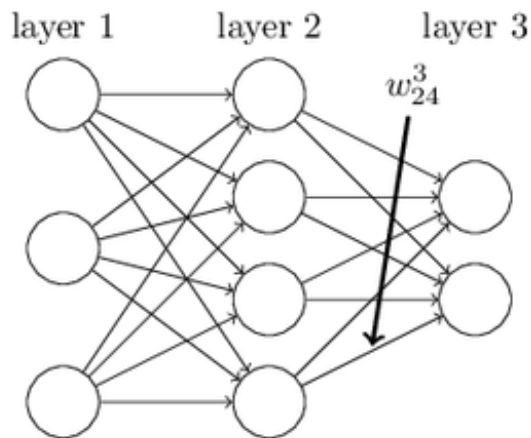
- Learning parameters with gradient descent

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

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

- 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.

Backpropagation

- Thus the activation a_j^l of the j^{th} neuron in layer l related to the activations of the $(l-1)^{th}$ layer:

$$\alpha_j^l = \sigma\left(\sum_k w_{jk}^l \alpha_k^{l-1} + b_j^l\right)$$

- k : the number of neurons in the $(l-1)^{th}$ layer.
- σ : activation function, w_{jk}^l weight among the k^{th} neuron of the $l-1$ layer to the j^{th} neuron of the l layer, α_k^{l-1} : the output of the activation function of the k th neuron at $(l-1)^{th}$ layer, b_j^l the intercept.
- Rewriting it becomes in matrix form:

$$\alpha^l = \sigma(w^l \alpha^{l-1} + b^l)$$

Let $z^l = w^l \alpha^{l-1} + b^l$: the weighted input values to the neurons of layer l :

$$\alpha^l = \sigma(z^l)$$

Backpropagation

- Assumption 1:

- Error is the average of individual errors for each training data point x

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

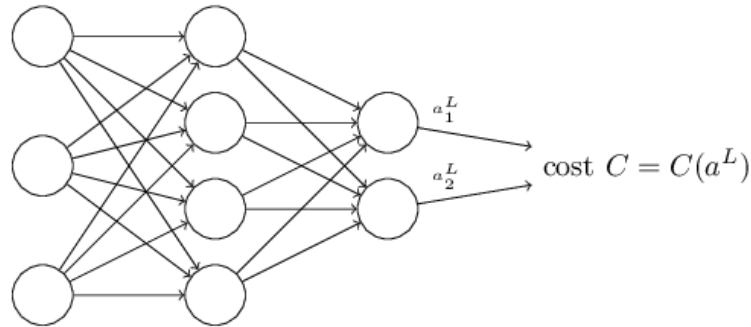
- The individual point error is $C_x = \frac{1}{2} ||y - a^L||^2$
- Where a^L is the vector of activation out put at level L

- Thus we can

- compute the partial derivatives $\partial C_x / \partial w$ and $\partial C_x / \partial b$ for each training data point.
- **Compute $\partial C / \partial w$ and $\partial C / \partial b$ by averaging over training points**

Backpropagation

- Assumption 2:
 - cost can be written as a function of outputs only of the neural network:



$$C = \frac{1}{2} ||y - \alpha^L||^2 = \frac{1}{2} \sum_j (y_j - \alpha_j^L)^2$$

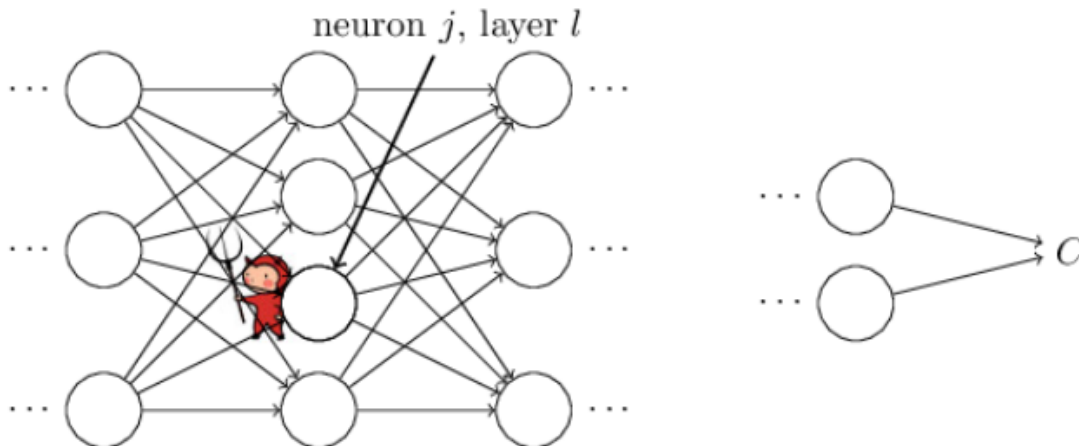
* *Hadamard product* –

$$s \odot t = s_i t_j$$

- Used in backpropagation

Backpropagation

- understanding how changing the weights and biases in a network changes the cost function
=> computing the partial derivatives $\partial C / \partial w_{jk}^l$ $\partial C / \partial b_j^l$.
- introduce an intermediate quantity, δ_j^l , : *error* in *j*th neuron in the *l*th layer.
- Backpropagation will give us a procedure to compute the error δ_j^l , then relate it to $\partial C / \partial w_{jk}^l$ and $\partial C / \partial b_j^l$.



Back propagation: error in the output layer

- error in the output layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

- $\partial C / \partial a_j^L$: how fast the cost is changing as a function of the j th output activation.
 - If, C doesn't depend on output neuron, j , then δ_j^L will be small,
- $\sigma'(z_j^L)$: measures how fast the activation function σ is changing at z_j^L
- In matrix based form:
$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$
- For quadratic error function:
$$\delta^L = (a^L - y) \odot \sigma'(z^L).$$

Backpropagation: error as function of errors in next layer

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad (1)$$

- moves error backward through the activation function in layer l , giving error δ^l in the weighted input to layer l .

- combining $\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$ with (1)
- we compute the error δ^l for any layer in the network.
 - We start by using (BP1) to compute δ^L ,
 - apply (1) to compute δ^{L-1} , then (1) again to compute δ^{L-2} ,
 - Until the input layer

Backpropagation

- the rate of change of the cost with respect to any bias in the network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \quad (3)$$

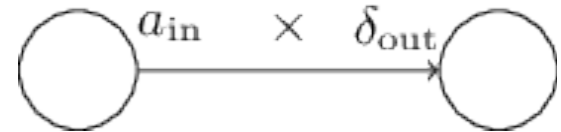
- rate of change of the cost with respect to any weight in the network:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l.$$

- already know how to compute δ^l and a^{l-1} , $\frac{\partial C}{\partial w} =$

- Equivalently

$$\frac{\partial C}{\partial w} = a_{in} \delta_{out}$$



- a_{in} : activation of the neuron input to the weight w , δ_{out} : error of the neuron output from the weight w .

Back propagation - Summary

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

The backpropagation algorithm

- **Input x :** Set the corresponding activation a^l for the input layer.
- **Feedforward:** For each $l=2,3,\dots,L$ 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 $l=L-1, L-2, \dots, 2$ compute

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l).$$

- **Output:** The gradient of the cost function is given by

$$\begin{aligned} \partial C / \partial w_{jk}^l &= a_k^{l-1} \delta_j^l \\ \partial C / \partial b_j^l &= \delta_j^l. \end{aligned}$$

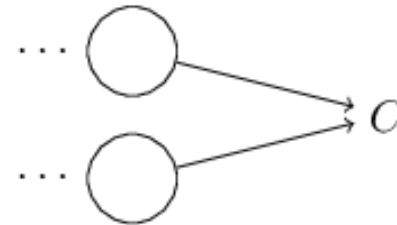
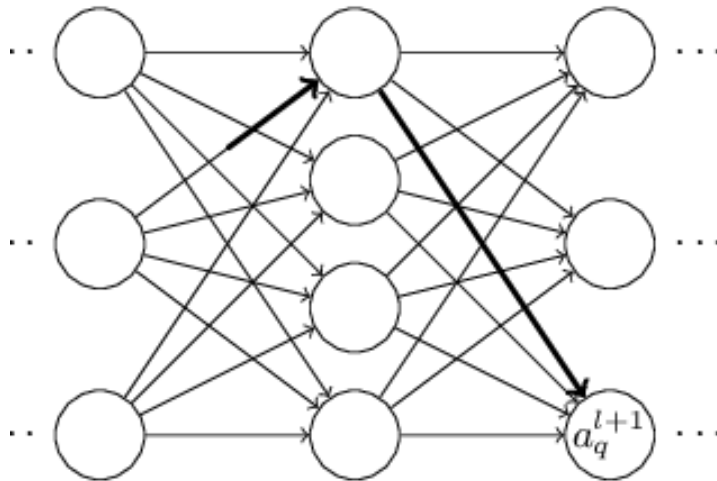
Back propagation - the big picture

- Assume a small change in Δw_{jk}^l to some weight w_{jk}^l in the network
- It will cause a change in the output activation from the corresponding neuron

$$\Delta a_j^l \approx \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l.$$

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

- Thus:
- $$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \Delta a_j^l.$$



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:

$$\Delta C \approx \sum_{mnp \dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

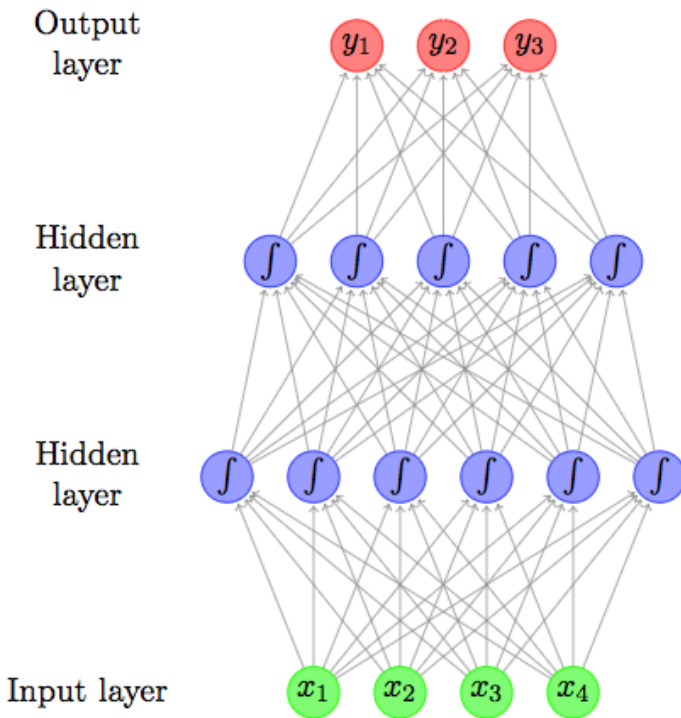
- Thus, the effect of the cost of a change in a single weight is:

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp \dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$

- Thus learning takes place: $w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$

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

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.

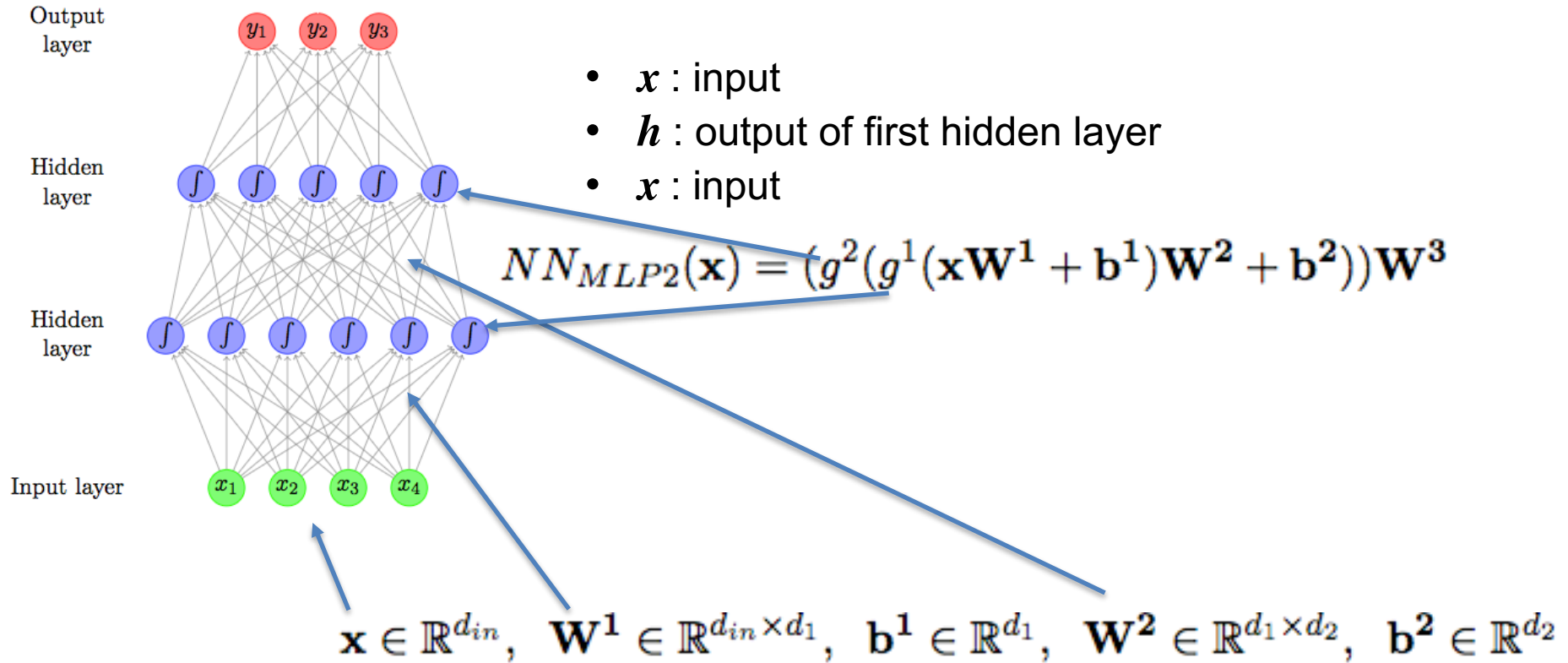
- top-most layer: output of the network.
- The other layers are considered “hidden”.

- 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



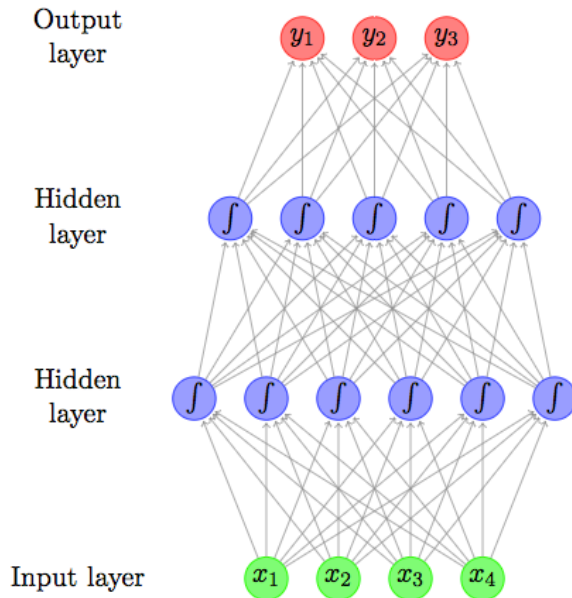
- Alternatively: $NN_{MLP2}(\mathbf{x}) = \mathbf{y}$

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

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

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

Multilayer Feed-forward NNs



Common non linear functions

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

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

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

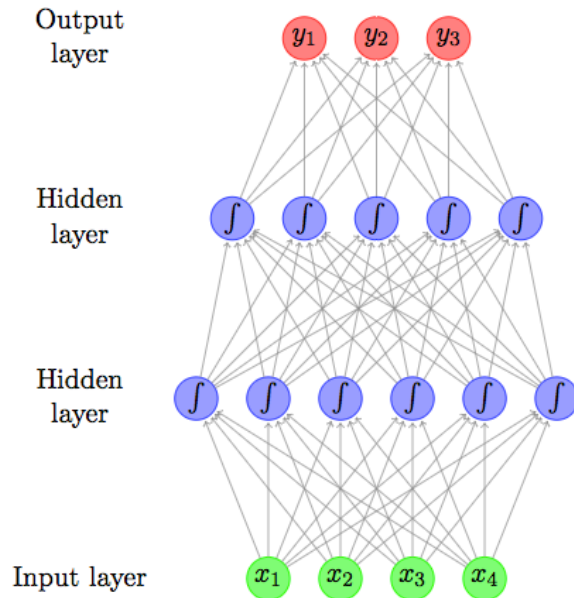
$$\text{hardtanh}(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$$

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

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

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 \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.

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

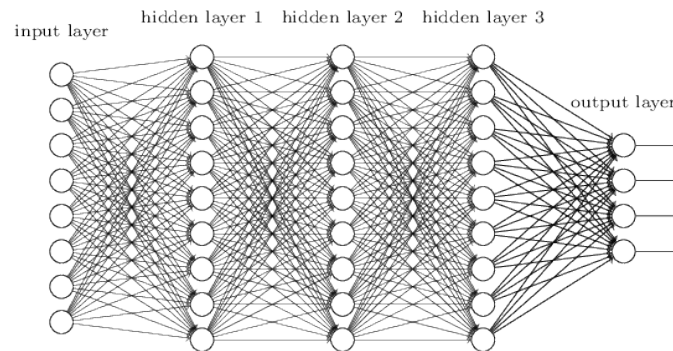
$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1.$$

- sigmoid would not produce it.

Convolutional Neural Networks (CNNs)

NNs fully-connected layers to classify images.

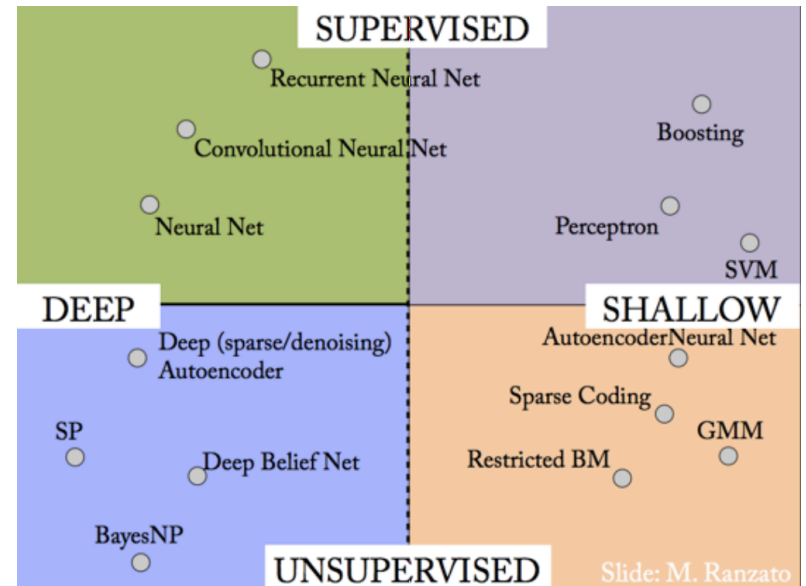
- do not take into account the spatial structure of the images.
 - treat input pixels far apart and close together in the same way.
 - spatial structure must be inferred from the training data.



- 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*.

Deep Models

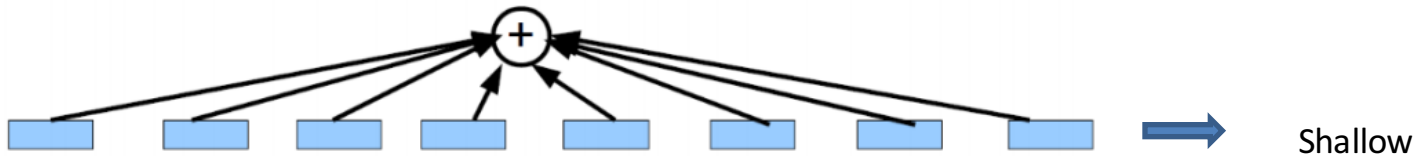
- Motivation: Automatic Feature Representation Learning
- Instead of designing features, design feature learners



Learning non-linear functions

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

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



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



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.

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.

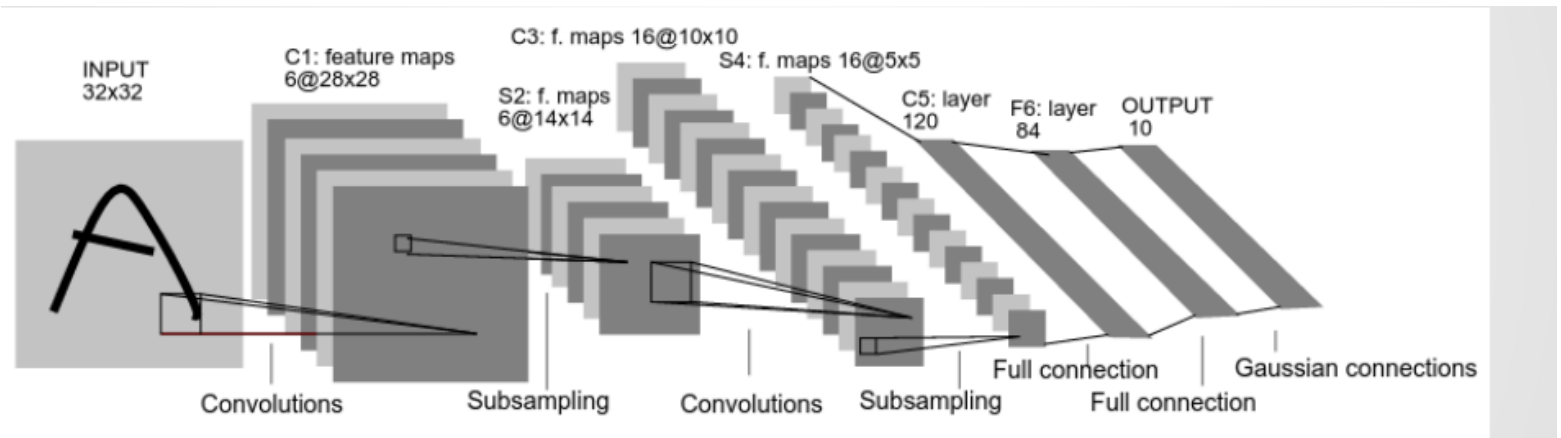
[1] Hubel, David H., and Torsten N. Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology 160.1:106-154. 4

[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 vs NN

- From a memory and capacity standpoint the CNN is not much bigger than a regular two layer network.
- At runtime the convolution operations are computationally expensive and take up about 67% of the time.
- CNN's are about 3X slower than their fully connected equivalents (size-wise).

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

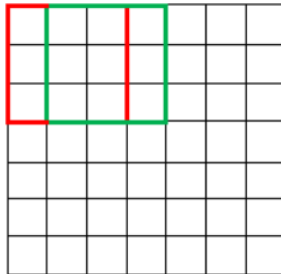
$$s(t) = \int x(a)w(t-a)da \quad s(t) = (x * w)(t)$$

- 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:

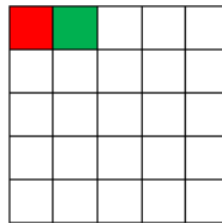
$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

Example

7 x 7 Input Volume

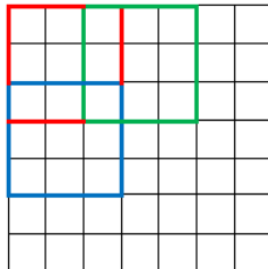


5 x 5 Output Volume



Stride = 1

7 x 7 Input Volume



3 x 3 Output Volume

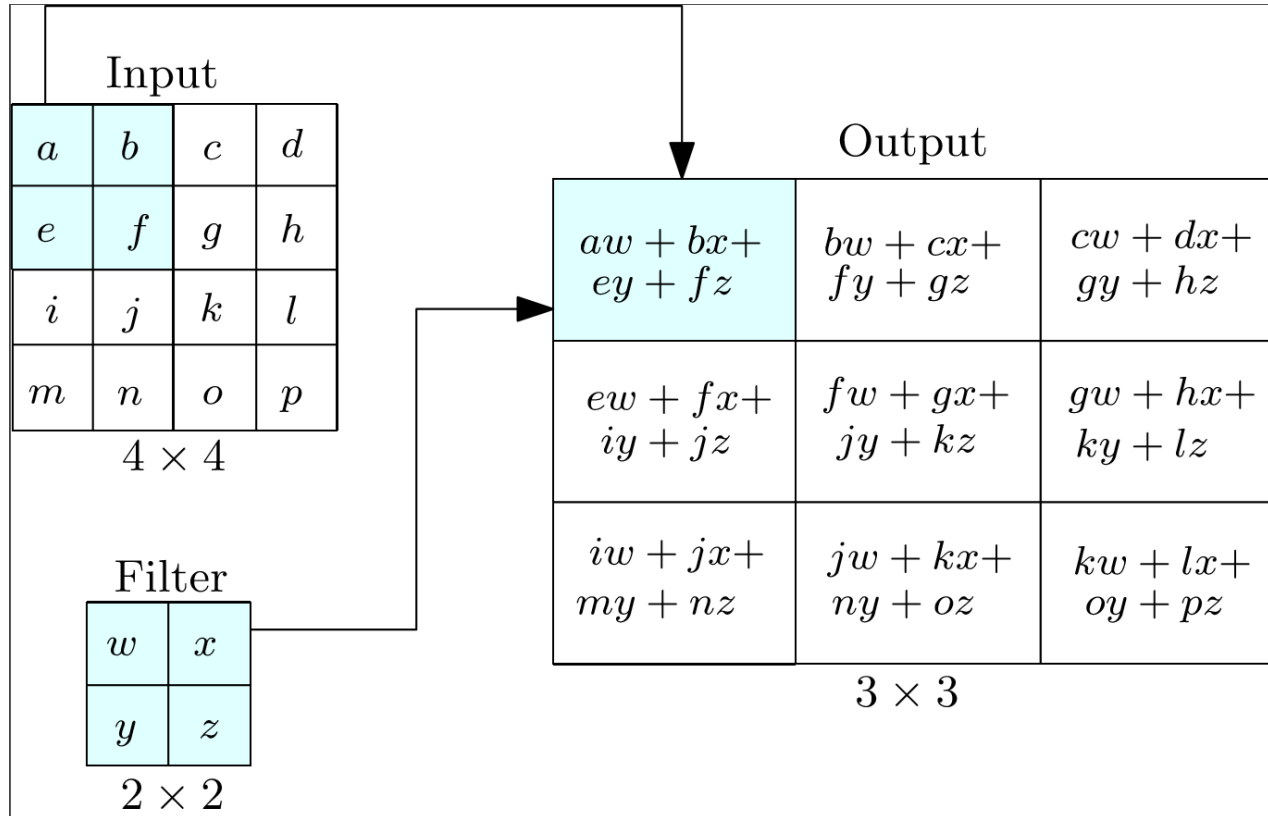


Stride = 2

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:

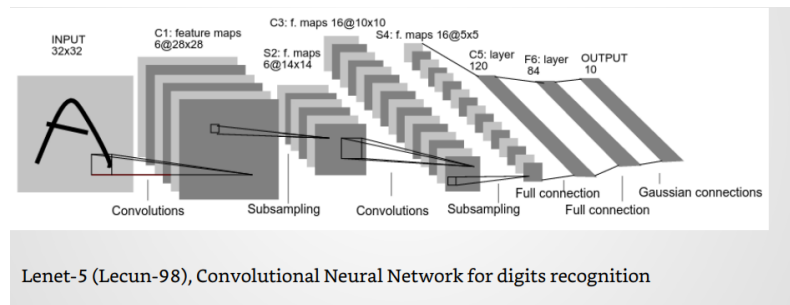
- Sigmoid
$$\frac{1}{1 + e^{-x}}$$

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

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

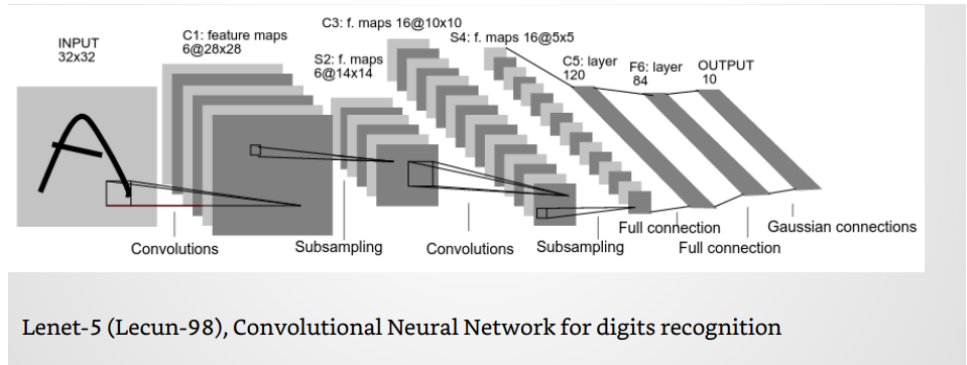
$$f(x) = \max(0, x)$$

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

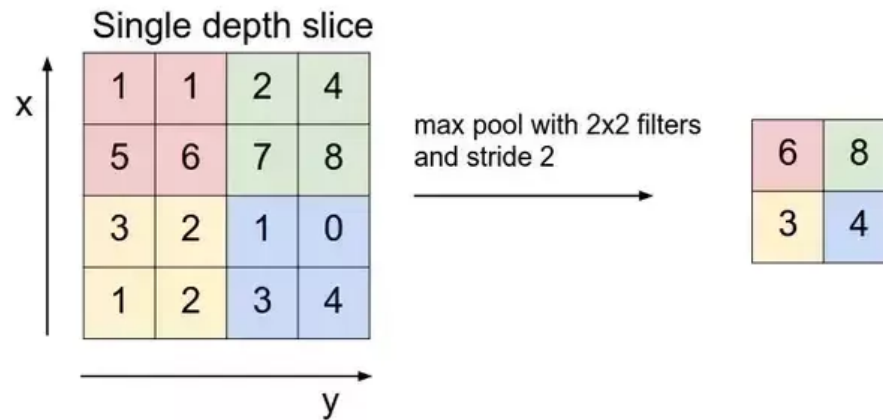


Pooling

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



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

CNN for text classification

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

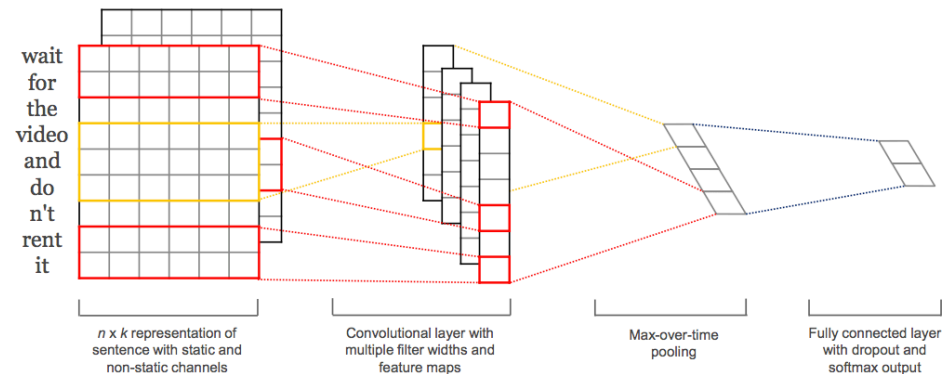
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$$

- 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

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



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

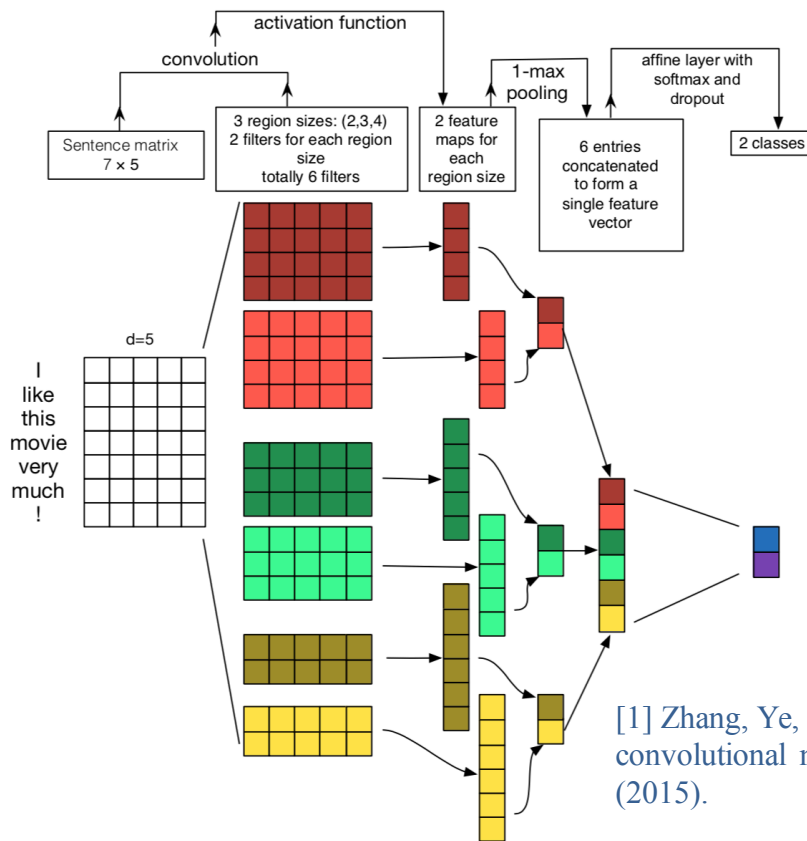
- Combine multiple word embeddings
- Each set of vectors is treated as a 'channel'
- Filters applied to all channels
- Gradients are back-propagated only through one of the channels
- Fine-tunes one set of vectors while keeping the other static

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^[1]



- 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 $\sim 10^2$ 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

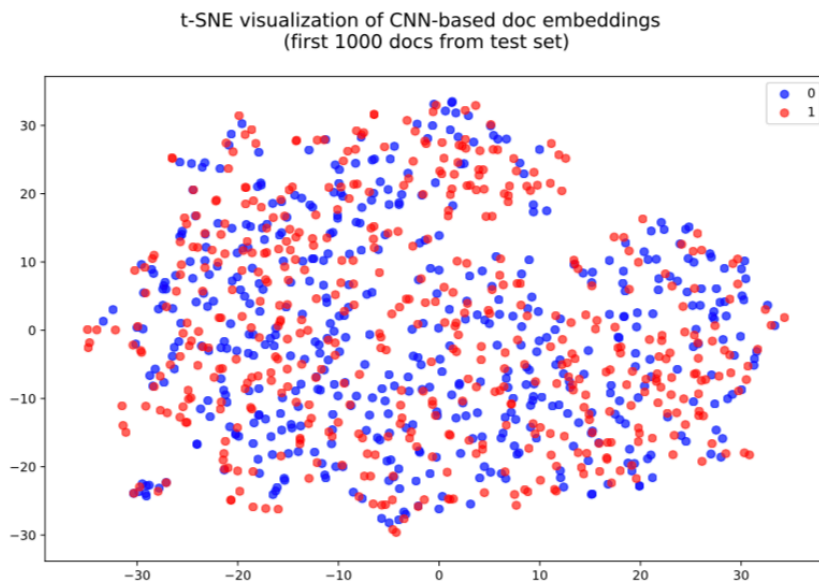


Figure 2: Doc embeddings before training.

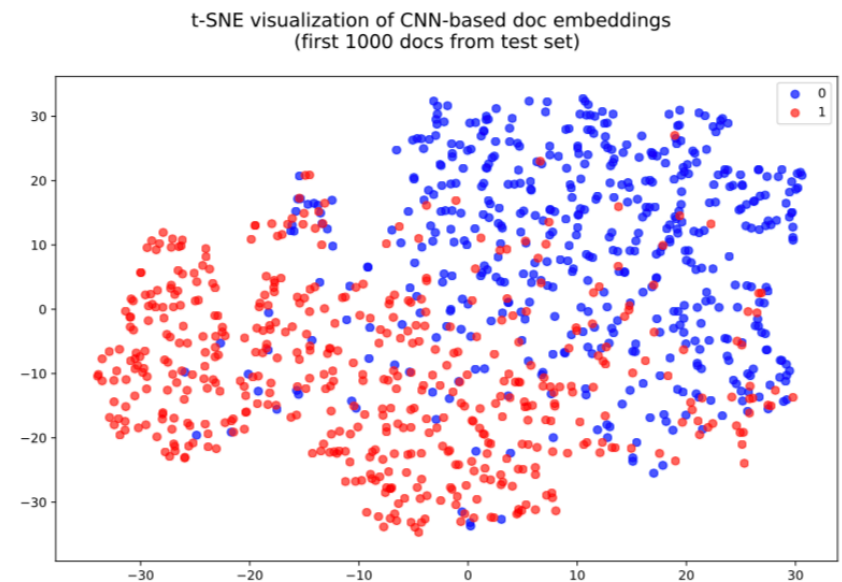


Figure 3: Doc embeddings after 2 epochs.

CNN architecture for (short) document classification - Saliency maps

- words are most related to changing the class

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

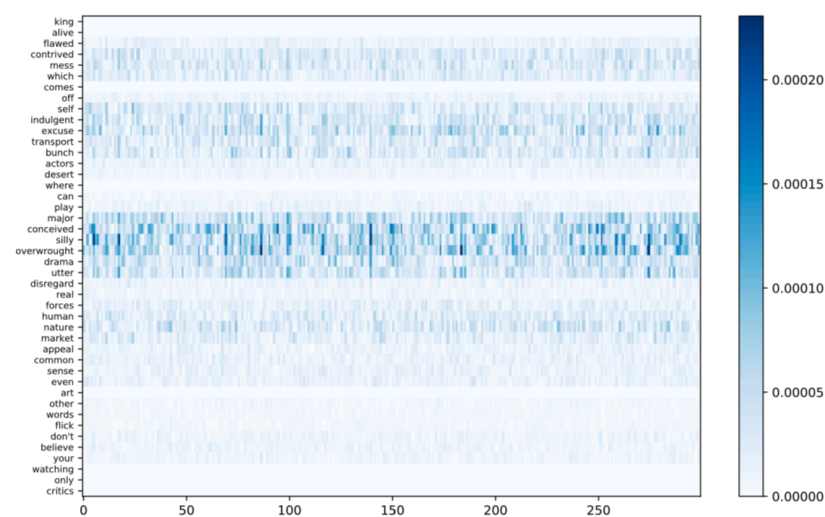
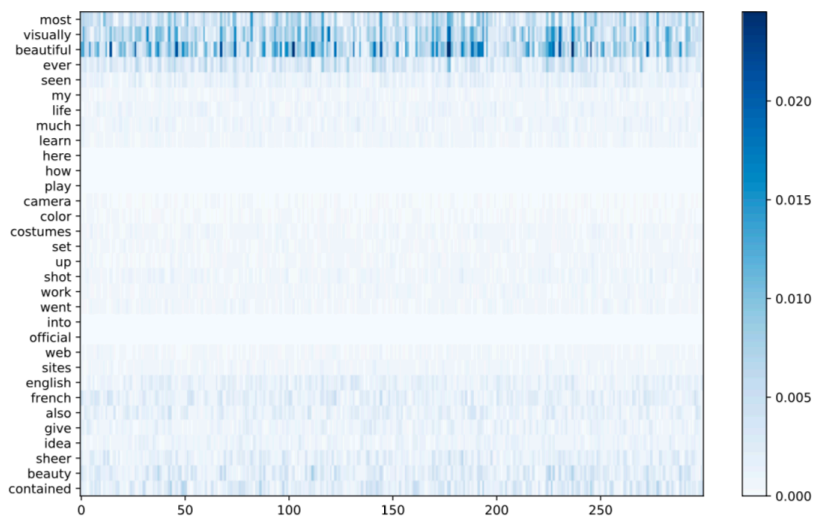
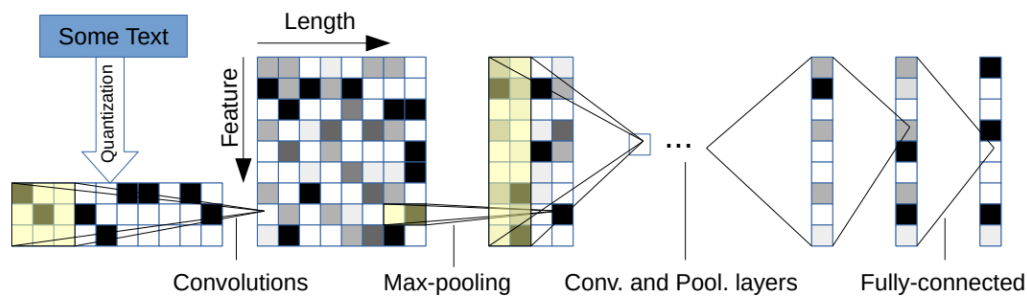


Figure 4: Saliency map for document 1 of the IMDB test set (true label: positive) Figure 5: Saliency map for document 15 of the IMDB test set (true label: negative)

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/>

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