Neural Networks and Deep Learning

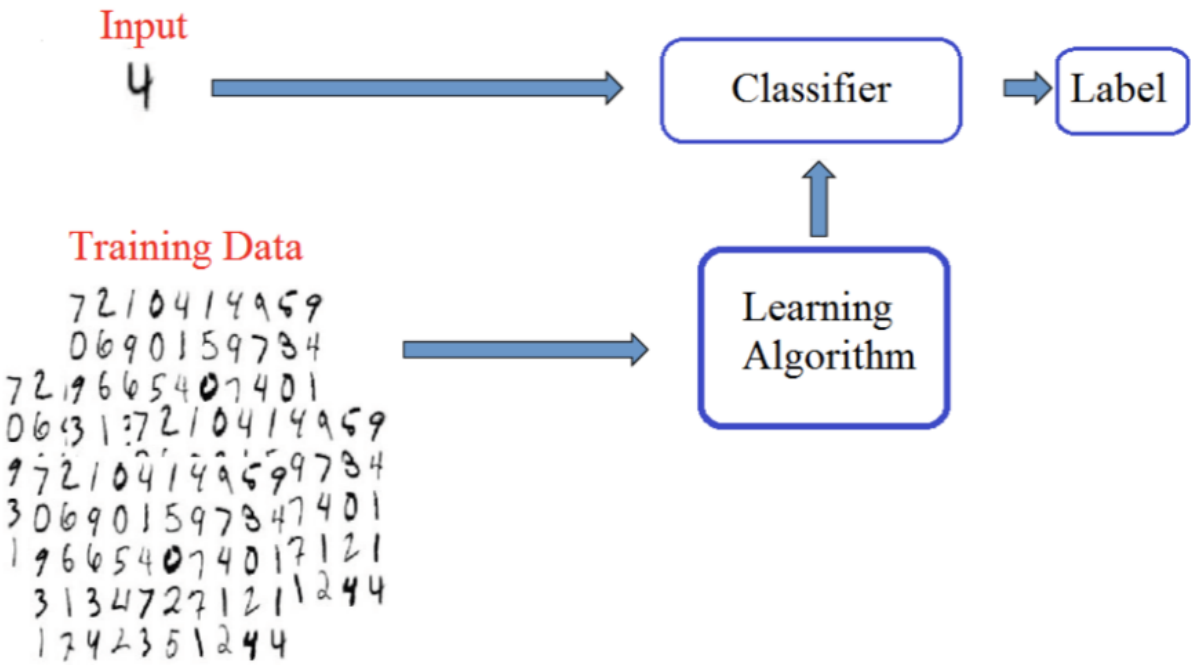
M. Vazirgiannis

Data Science and Mining Team (DASCIM), LIX
École Polytechnique
http://www.lix.polytechnique.fr/dascim
Google Scholar: https://bit.ly/2rwmvQU
Twitter: @mvazirg

October, 2022

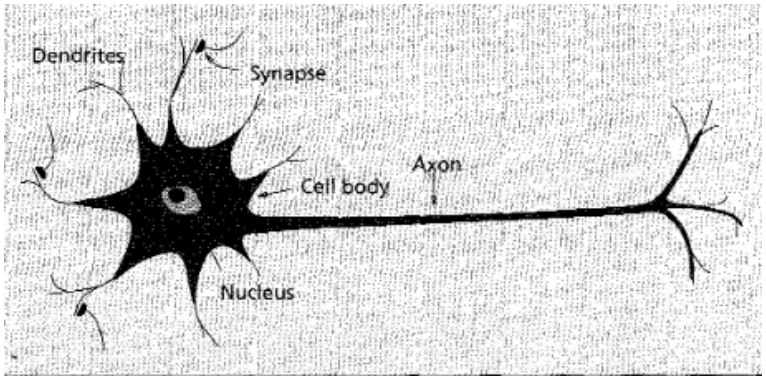
Outline

Machine Learning....



- Data
- Performance
- Model
- Optimization

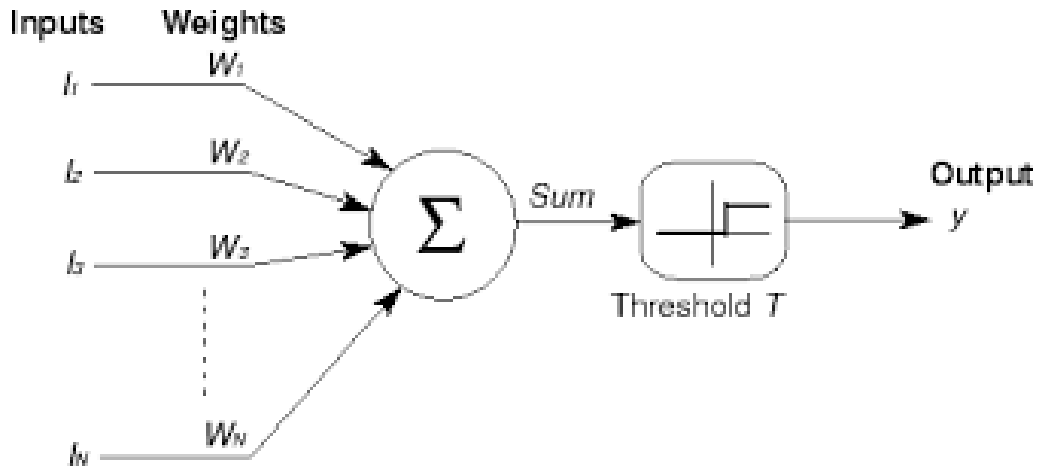
Neural Networks



- 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)

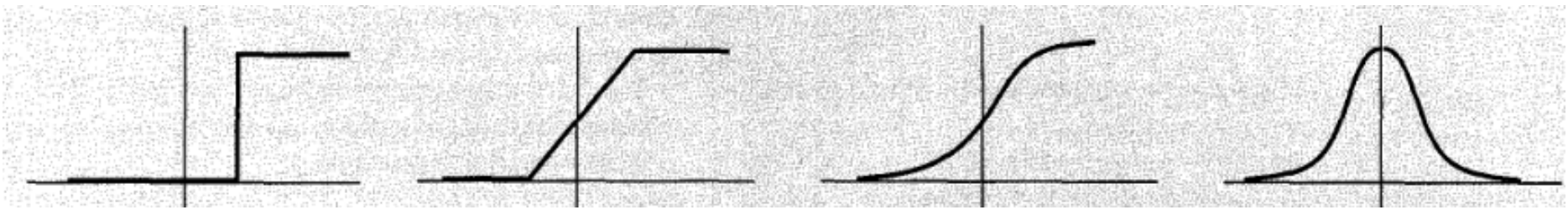
Artificial neural networks: A tutorial, AK Jain, J Mao, KM Mohiuddin - Computer, 1996

Neural Networks



McCulloch-Pitts Neuron Model $y = \theta \sum_{j=1}^{n} w_j X_j - \mu$

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



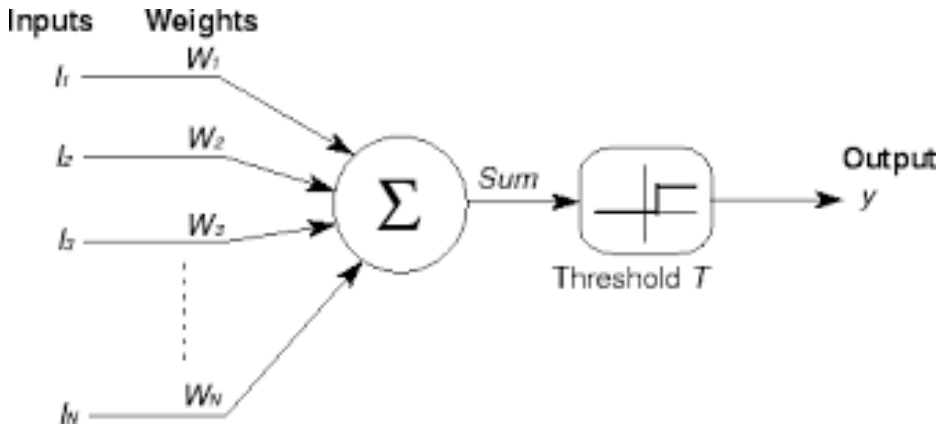
Activation functions

- Idendity: $f(x) = x$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic Tangent: $\tanh(x) = \frac{1-e^{-x}}{1+e^{-x}}$
- Step function: $f(x) = 1_{x>0}$
- 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

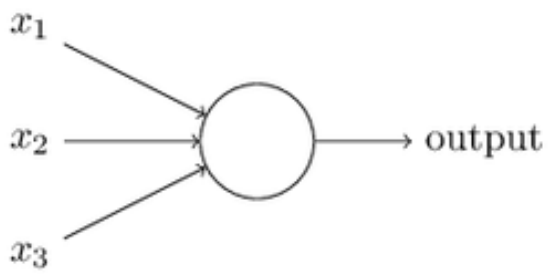


Perceptron learning algorithm

- developed in the 1950-60 by Frank Rosenblatt
- takes several binary inputs, x1, x2... produces a single binary output

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

- Device making decisions by weighing up evidence.
- Varying weights and threshold, we get different models of decision-making



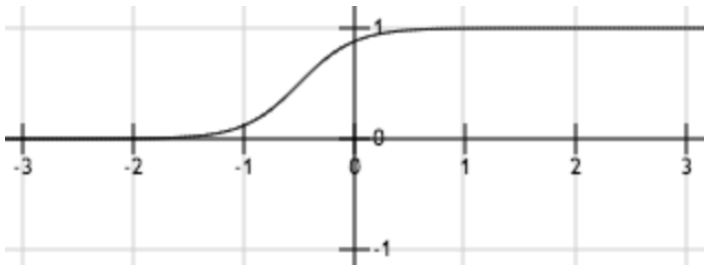
Perceptron learning algorithm

- Initialize the weights and threshold (small random numbers)
- Present an input vector $\mathbf{x} = \{x_1, ..x_n\}$ and evaluate the output of the neuron $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b$
- Update the weights $w_{t+1} = w_t + \eta(y - f(\mathbf{x}))\mathbf{X}$ where
  η: learning rate, y: the correct output.
  $\sum_{j=1}^{n} w_j x_j - b = 0$ defines the class separation hyper plane
  - learning occurs only when the perceptron makes an error.
  - Perceptron convergence theorem (Rosenblatt, 1962)

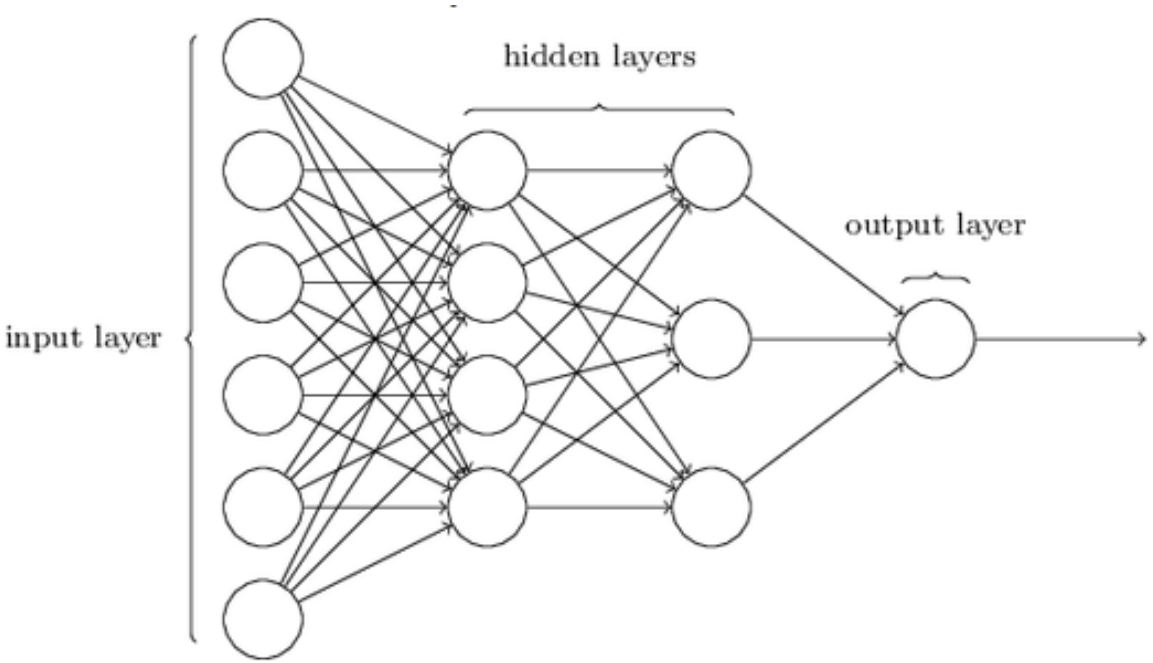
If training data are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations.

Neural Networks – sigmoid neuron

- In perceptron small changes to the weights may cause huge difference to output ($0 \rightarrow 1$)
- 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: $\sigma(z) = \frac{1}{1+e^{(-\Sigma_j w_j x_j - b)}}$



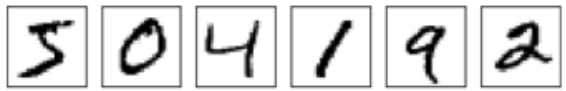
NNs architecture - Multilayer perceptron (MLP)

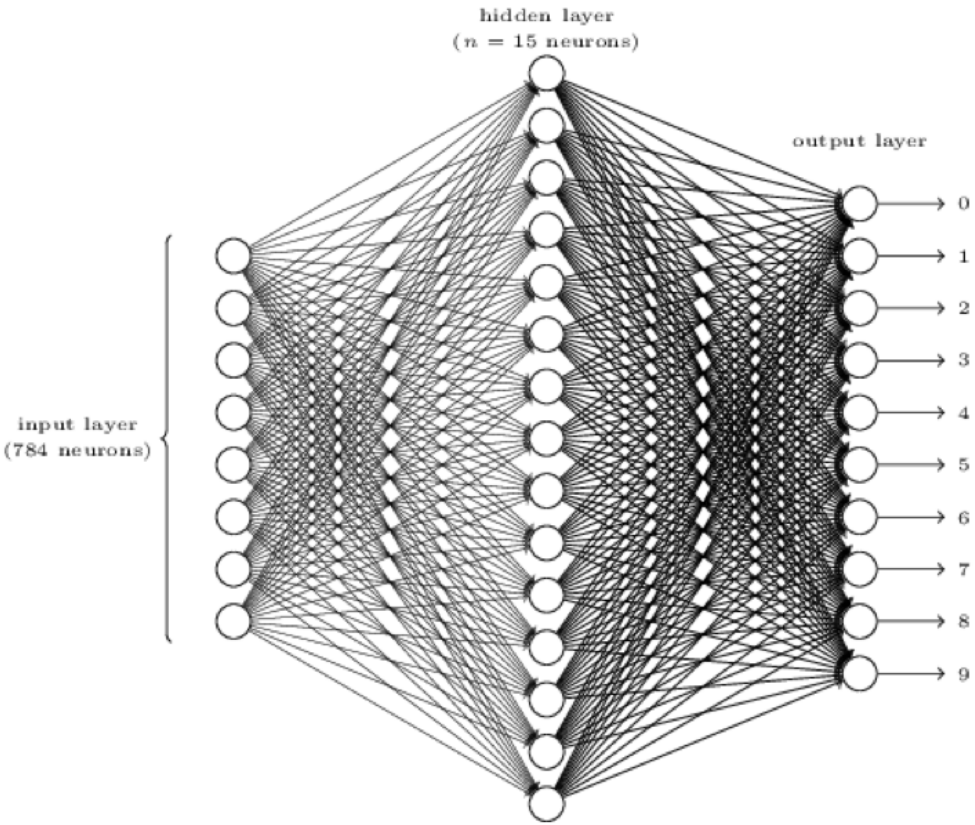


- i.e. case of 64x64 greyscale image representing a single number (i.e. ”8”)
- MLP with (64x64) 4096 input neurons
- A single neuron as output ($>$ 0.5: “is 8” , $<$ 0.5 not ”8”)

Learning with gradient descent

- Assume hand written digits (MNIST database – 60.000 images):
  - Each of them is a 28x28=784 pixels





- 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 $C(w,b) = \frac{1}{2n}\Sigma_x |y - f(x)|^2$
  - w: weights, b: biases, y: 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 GD: $C = \frac{1}{n}\Sigma_x C_x$ where $C_x = \frac{\|y-f(x)\|^2}{2}$

To compute the gradient ∇C we must compute gradients ∇C_x for each training point and then average them: $\nabla C = \frac{1}{n}\Sigma_x \nabla C_x$

$\Rightarrow$ very slow learning

Learning with stochastic gradient descent (SGD)

- estimate ∇C by computing ∇C_x for of randomly chosen samples of training points
- averaging over sample: estimate of the true gradient ∇C - faster
- Chose m random subsets (with replacement) of the training set: $X_1, X_2, \ldots, X_m$ - mini-batches.
- Assuming sample size m large enough we expect that the average value of the ∇C_{X_j} will be roughly equal to the average over all ∇C_x:

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

Learning with stochastic gradient descent (SGD)

- w_k and b_l: weights and biases of the perceptron connections in the neural network. SGD trains on randomly choses a mini-batch of training points.

$$w_k \rightarrow w_k^{'} = w_k - \frac{\eta}{m} \Sigma_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b_l^{'} = b_l - \frac{\eta}{m} \Sigma_j \frac{\partial C_{X_j}}{\partial b_l}$$

- X_j: raining examples in the current mini-batch.
- pick another randomly chosen mini-batch and train with those.
- until all training points used (complete an *epoch* of training).

Backpropagation

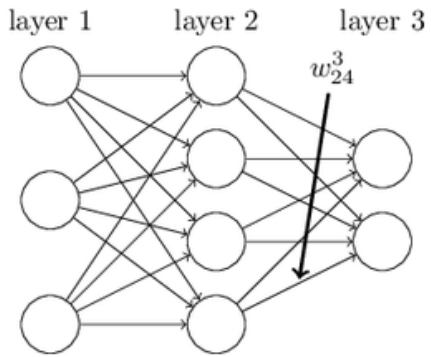
- Learning parameters with gradient descent

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

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

- How to compute the gradients: backpropagation

w_{jk}^{l} : weight among the k^{th} neuron 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 α^l for the input layer.
- **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l \alpha^{l-1} + b^l$ and $\alpha^l = \sigma(z^l)$.
- **Output error** $delta^L$: Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$
- **Backpropagate error**: For each $l = L-1, L-2, \ldots, 2$ compute

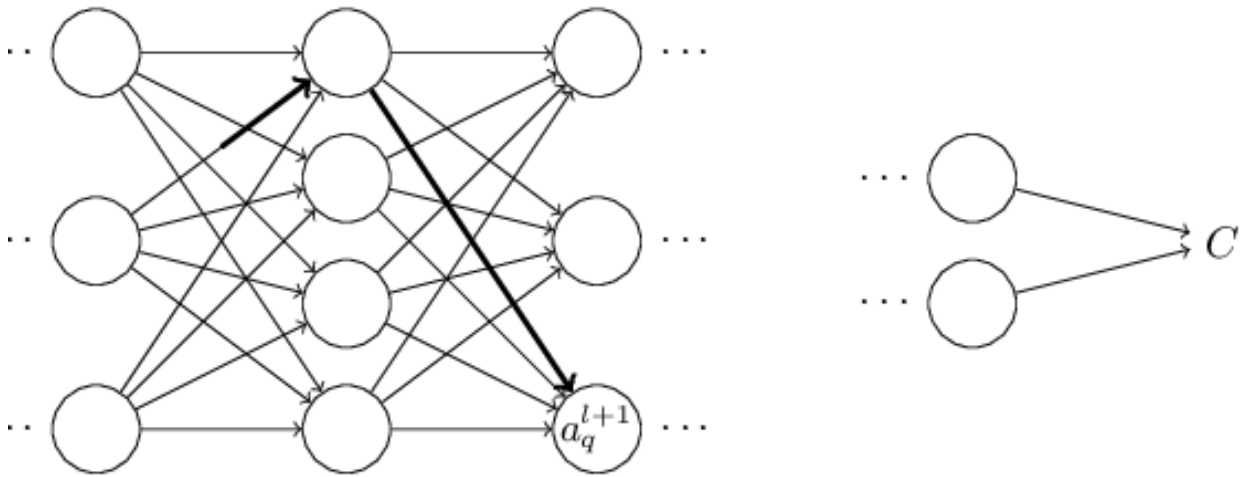
$$\delta^l = ((w^{l+1})^T \sigma^{l+1}) \odot \sigma'(z^L).$$

- **Output**: gradient of the cost function:

$$\partial C / \partial w^l_{jk} = a^{l-1}_k \delta^l_j$$
$$\partial C / \partial b^l_j = \delta^l_j$$

Back propagation - the big picture

- Δw_{lj}^k : small change in weight w_{lj}^k
- causes change of the output activation from the corresponding neuron: $\Delta \alpha_j^l \approx \frac{\partial \alpha_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$
- causing changes in all activations in the next layer
- Thus: $\Delta \alpha_q^{l+1} \approx \frac{\partial \alpha_q^{l+1}}{\partial \alpha_j^l} \Delta \alpha_j^l$



Back propagation - the big picture

Cascading changes to next layers cause change in the output:

- 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:

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

- Thus effect of a change in a single weight on the total cost/error:

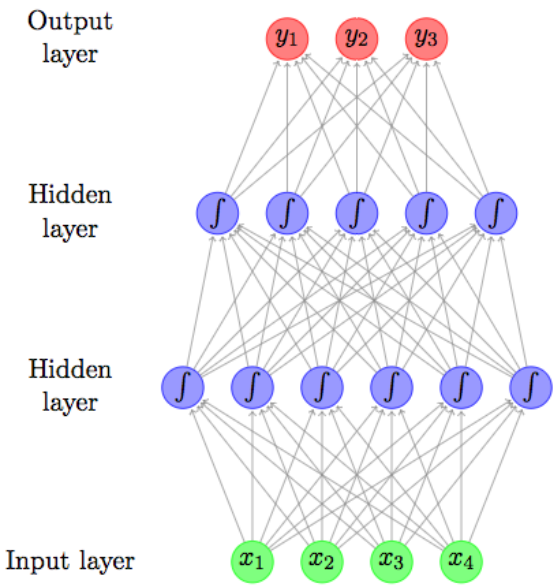
$$\frac{\partial C}{\partial w_{jk}^l} \approx \Sigma_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}$$

- Weights update:

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

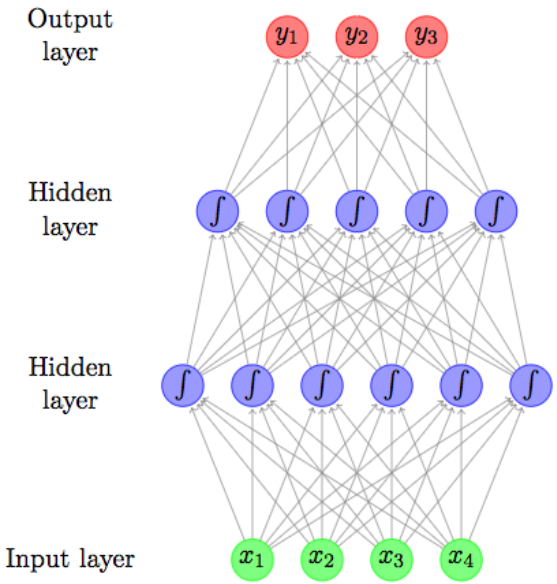
$$b_l^{'} = b_l - \eta\frac{\partial C}{\partial b_l}$$

Multilayer Feed-forward NNs



- top layer: output of the network (3 class problem).
- other layers are considered “hidden”.
- hidden layer neurons represent non-linear functions.
- applied to neuron’s value before passing it to the output.
- fully-connected: each neuron connected to all next layer neurons.

Multilayer Feed-forward NNs



- **x**: input
- **h**: output of first hidden layer
- **y**: output

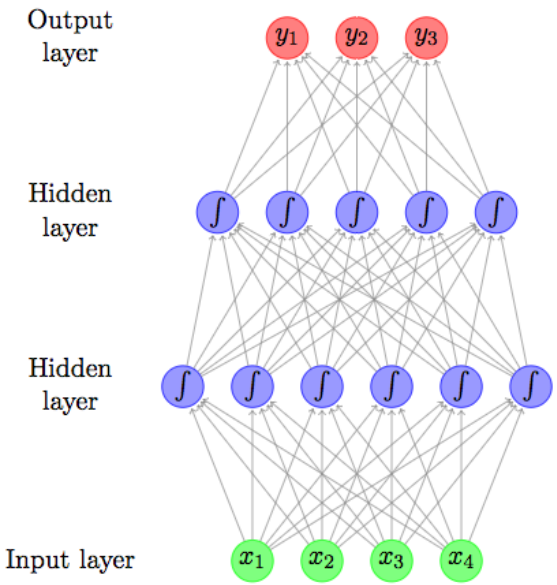
$$NN_{MLP2}(\mathbf{x}) = (g^2(g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)\mathbf{W}^2 + \mathbf{b}^2))\mathbf{W}^3$$

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

- Alternatively:

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

Multilayer Feed-forward NNs



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

Common non linear functions

- Sigmoid ($x \rightarrow [0, 1]$) $\sigma(x) = \frac{1}{1+e^{-x}}$
- 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.

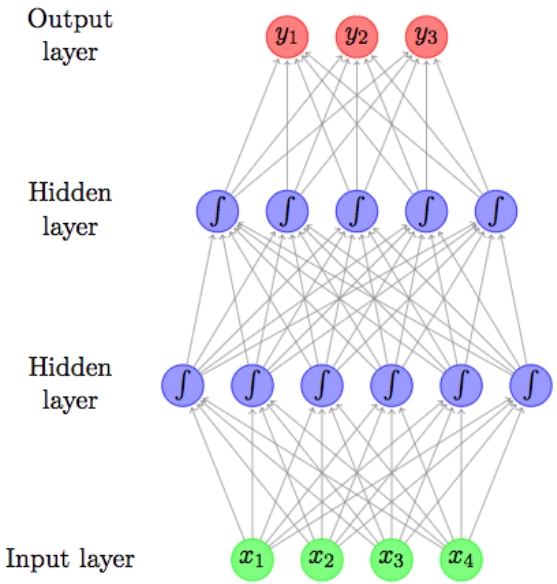
$$hardtanh(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & otherwise \end{cases}$$

- Relu:

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

ReLU $>$ **tanh** $>$ **sigmoid**

Multilayer Feed-forward NNs



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

Classification: only one output activated - softmax:

$$\mathbf{y} = [y_1, \ldots, y_3]$$
$$softmax(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}}$$

- 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

Deep Learning - Hyperparameters tuning - Preliminaries

- consider a deep neural network on a training dataset $\{(x_i, y_i)\}_{i=1,\ldots,N}$

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

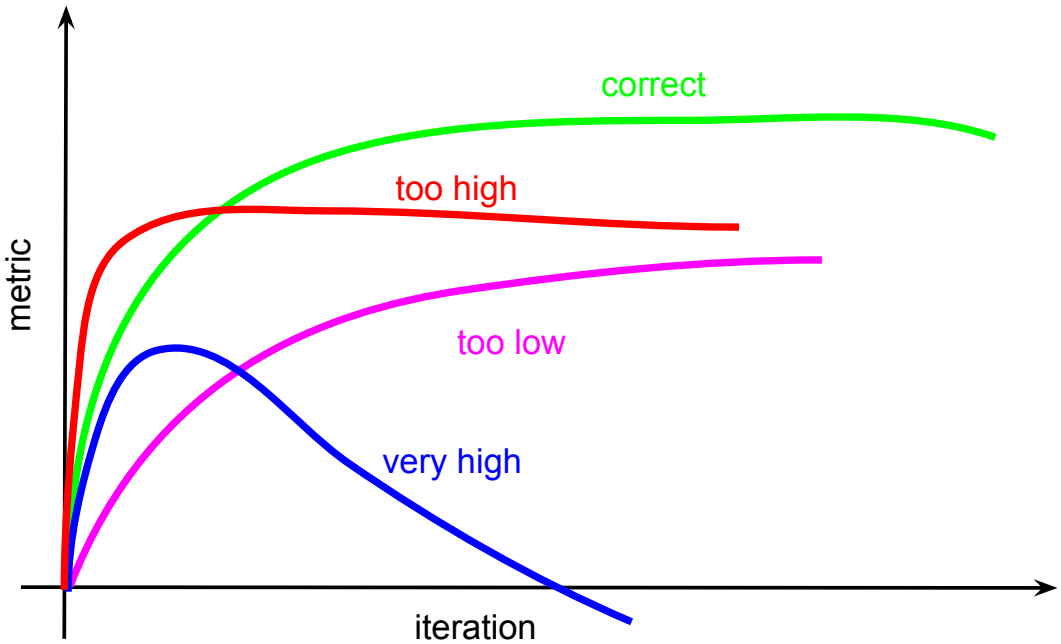
- Parameters $\theta = w$: parameters of the network, F: function representing the network, L : loss funtion of the network and l: loss for each training point.
- **Batch Stochastic Gradient Descent (SGD)** - general algorithm to train deep neural networks
- At each iteration t:

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

- Where η : learning rate, $\hat{L}_t = \frac{1}{B}\sum_{i=1}^{B} l(y_{b_i}, y_{b_i})$, $\{b_1, \ldots, b_B\}$ indices of the elements in the batch.

Deep Learning - Hyperparameters tuning - Learning Rate Schedules

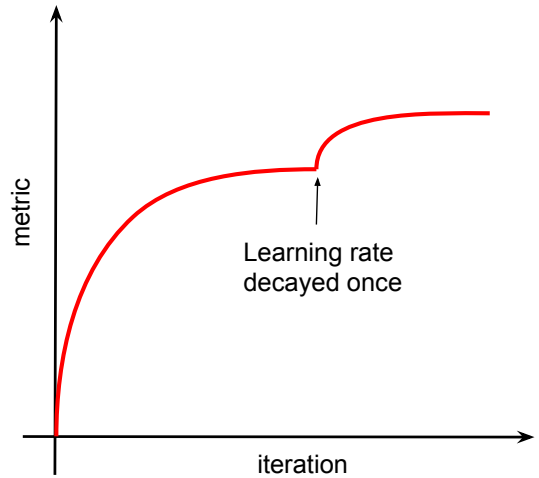
- In vanilla SGD learning rate is fixed, this version is still used today
- but we need to tune the learning rate.Typically try $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$.



How to evaluate the learning rate.

Deep Learning - Hyperparameters tuning - Decaying the Learning Rate

- Covergence of training curve due to the fact that the 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.



Step decay of the learning rate once

- Learning rate can be decayed on schedule (after a predefined number of iterations), or when the network stops learning

Deep Learning - Hyperparameters tuning - Learning Rate Annealing / Step-Wise Decay

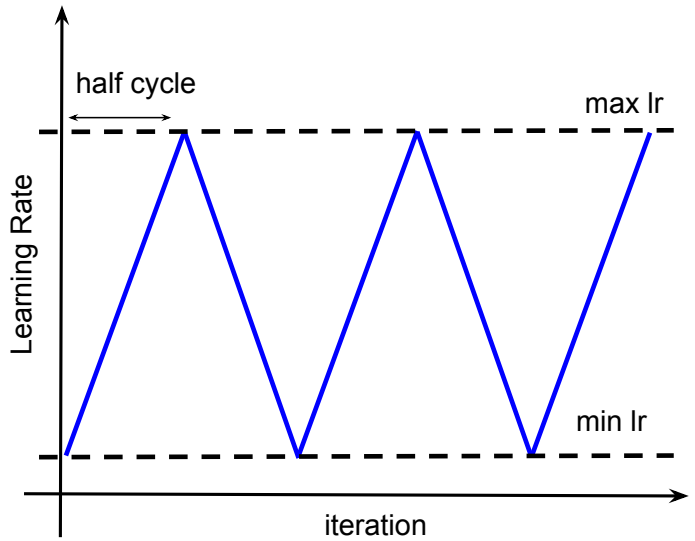
- Learning rate annealing: continuous changes of the learning rate during training.

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

- : For example $decay = 10^{-6}$. The point is to reach a minimum for the loss as fast as possible.
- Variants: exponential decay $\eta_t = \eta_0 e^{-decay * t}$.

Deep Learning - Hyperparameters tuning - Triangular Schedule

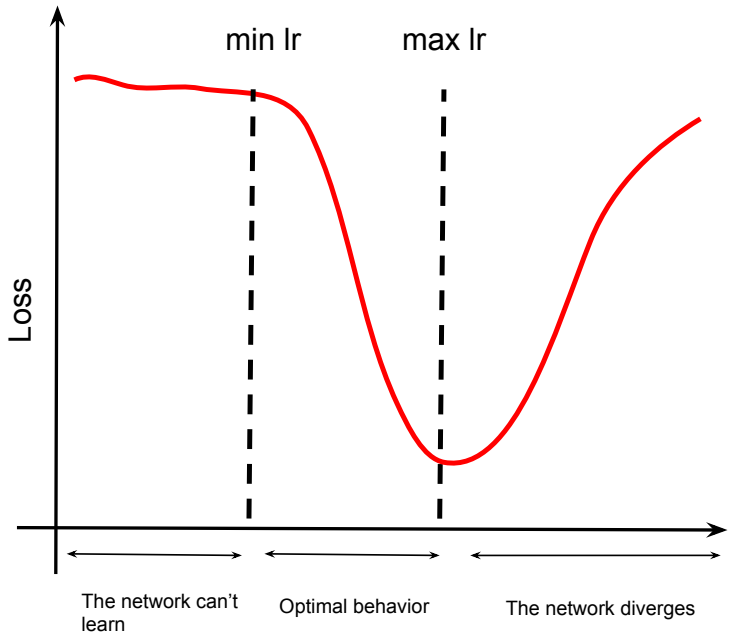
- Sequentially increase and deacrease the learning rate.
- Intuition: increasing the learning rate allows to escape local minima, deacreasing it allows to correctly train the network.



Example of triangular schedule

Choosing the learning rate boundaries

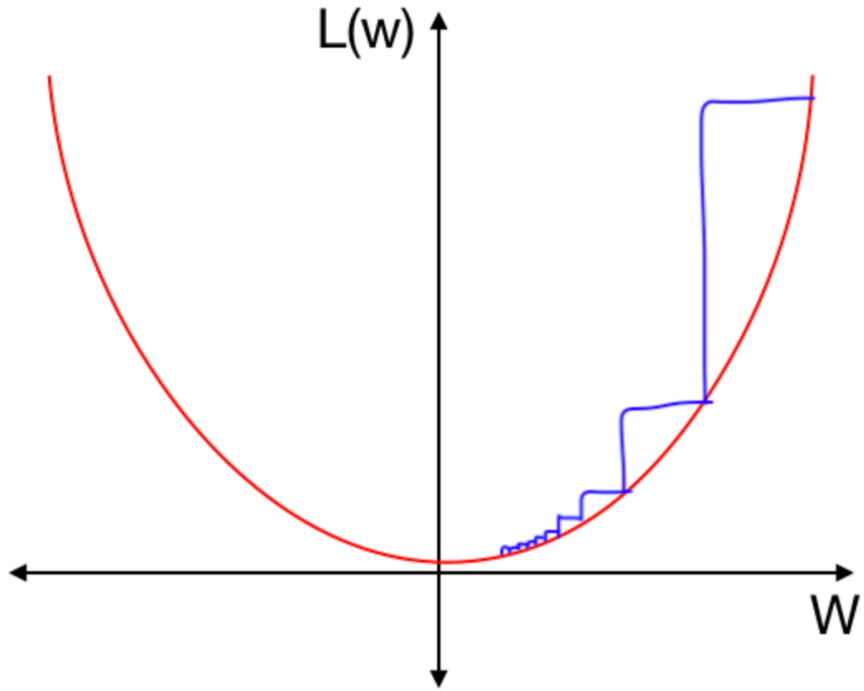
- We want to choose extreme values of learning rates for which the network is able to learn.
- Start very low (i.e. 10^{-5}) and augment the learning rate for one half cycle to a high value (i.e. 10^{-2}). Then choose with respect to the training curve



Search for learning rate boundaries

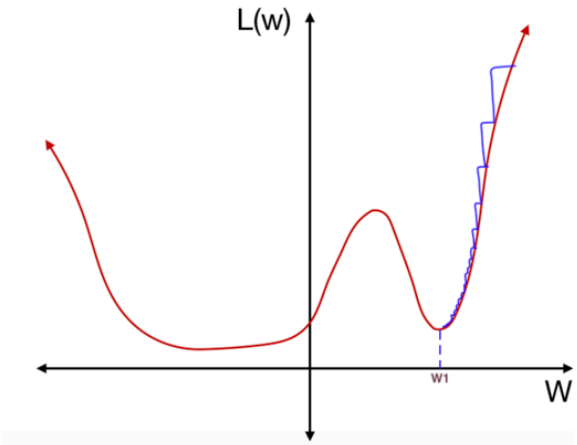
Deep Learning - Stochastic Gradient Descent With Restarts

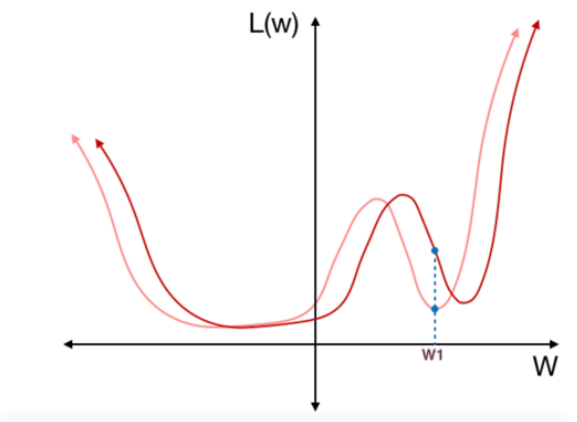
- variant of learning rate annealing
- gradually decreases the learning rate through training
- start with relatively high learning rates for several iterations
- quickly approach a local minimum,
- then gradually decrease the learning rate as we get closer to the minimum



Deep Learning - Stochastic Gradient Descent With Restarts

- assume multiple minima
- small change in weights may result in a large change in loss.
- If test this network on a different dataset, loss function might be slightly different (small shift).
- small shift of the loss function in this local minimum (w_1) result in a large change of loss
- search for ”flatter” error valleys

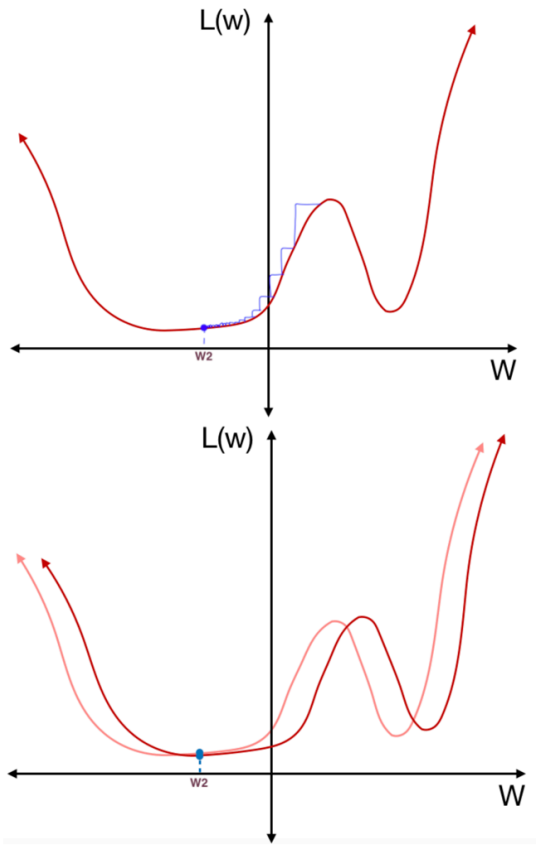




decreasing learning rate approached minimum

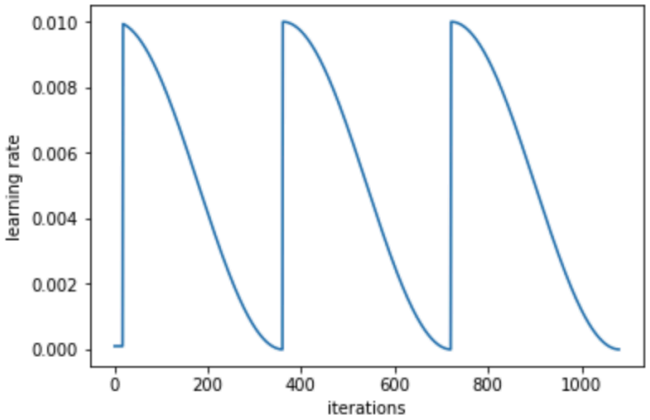
Deep Learning - Stochastic Gradient Descent With Restarts

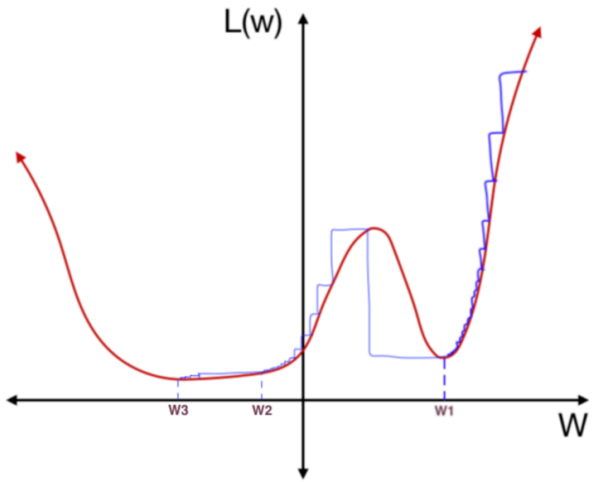
- search for ”flatter” error valley - w_2
- offers an accurate and stable solution- generalization
- better ability to react to new data.
- to find a more stable local minimum, we increase the learning rate from time to time, encouraging the model to “jump” from one local minimum to another if it is in a steep trough.
- SGDR - with restarts



Deep Learning - Stochastic Gradient Descent With Restarts

- In the first “half cosine” period in the graph above, we descend into a local minimum - w_1
- restart: increase learning rate
- gradually decrease learning rate again until minimize the loss function - w_1
- finding a stable solution



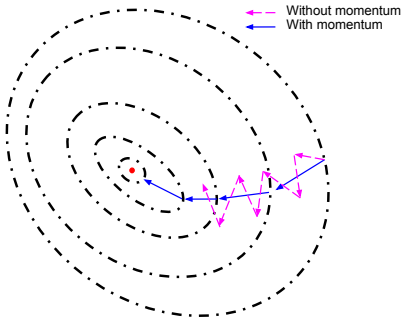


Deep Learning - Stochastic Gradient Descent With Momentum

- SGD is noisy, because submited to the data in the batch.
- To converge more smoothly and faster momentum keeps a memory of the previous update.

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

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



Convergence with momentum

Deep Learning - ADAM and RMSProp

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

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

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

ADAM combines momentum and adaptative gradient.

Deep Learning - Regularization - Batch size

- Noise is a shield against overfitting.
- based on this: the smaller the batch, the more noisy the convergence.
- deacreasing *batch size* regularizes the network.

Setting a good batch size is a trade off between regularization and computation time

Deep Learning - Regularization - Dropout

- Dropout, randomly sets to zero a proportion of the input of a layer.
- At each step of training, a new subnetwork is selected.
- As a result an adaptation appears in the final network only if it exists in a sufficient part of the training data.
- Another interpretation of dropout: large numbers of subnetworks are learned and aggregated at test time.
- In practice, dropout is applied to the input of every layer, with a rate of about 50%

Deep Learning - Regularization - Weight decay

- Regularization of a neural network can also be standard
- L_2-constraint called weight decay.
- corresponds to adding the following penalty to the loss $\lambda \sum_w w^2$.
- Then weight updates:

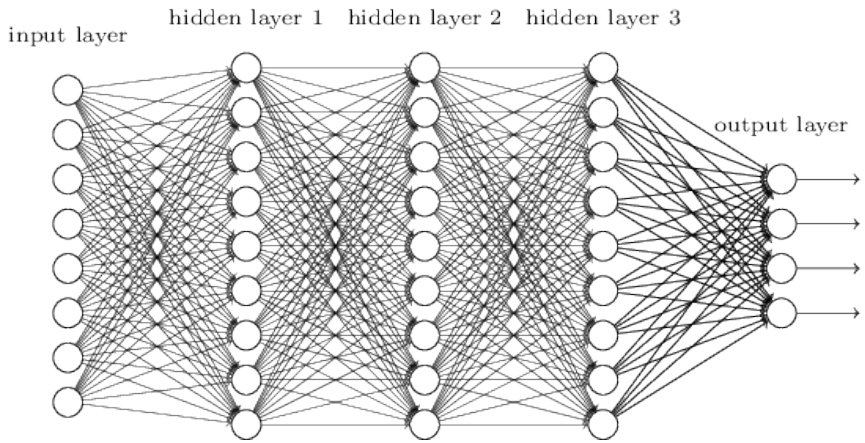
$$w_{t+1} = w_t - \eta \nabla_w \hat{L}_t - 2\lambda w$$

Outline

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*

Convolutional Neural Networks

- introduced by Yann LeCun and Yoshua Bengio [1](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.

[1] 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

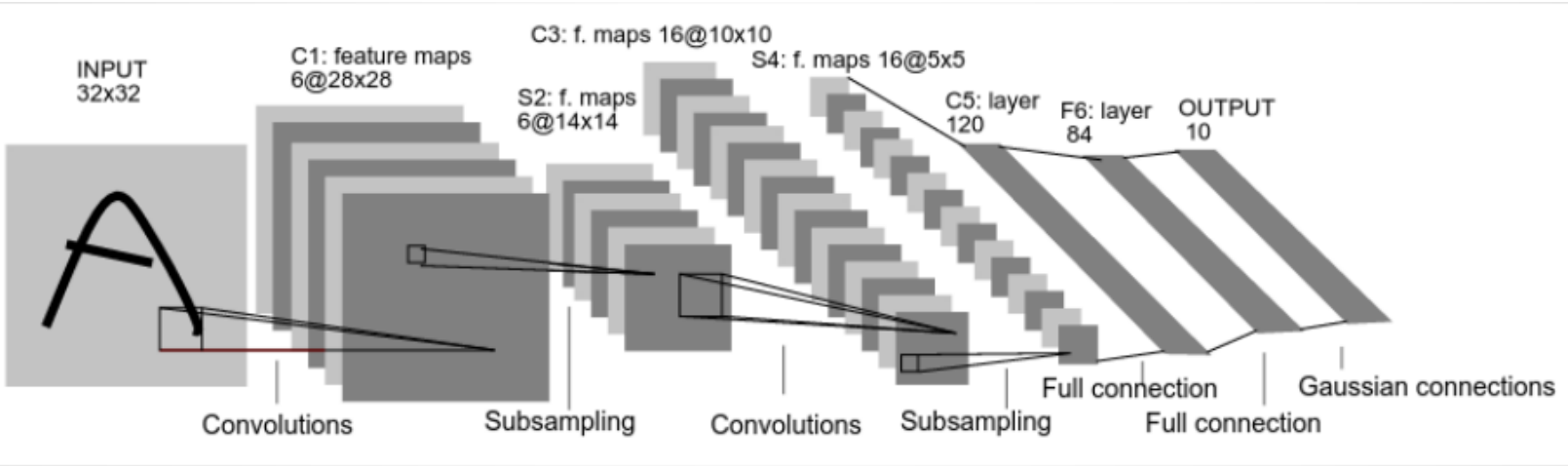
- inspired by studies of the cat's visual cortex [1], developed in computer vision to work on regular grids such as images [2].
- Feed-forward NNs , each neuron receives input from a neighborhood of the neurons (receptive fields) in the previous layer.
- Receptive fields, allow CNNs to recognize complex patterns in a hierarchical way, by combining lower-level, elementary features into higher-level features compositionality.
  - raw pixels $\Rightarrow$ edges$\Rightarrow$ shapes $\Rightarrow$ objects.
- absolute positions of features in the image are not important – only useful respective positions are useful to compose 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

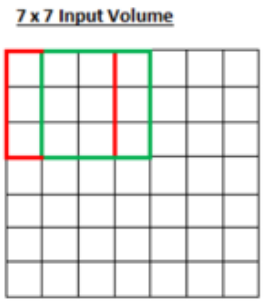
- The convolution of f and g, written as $f * g$, : integral of the product of the two functions after one is reversed and shifted

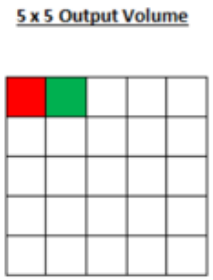
$$s(t) = \int x(a)w(t-a)da \quad s(t) = (x * w)(t)$$

- 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

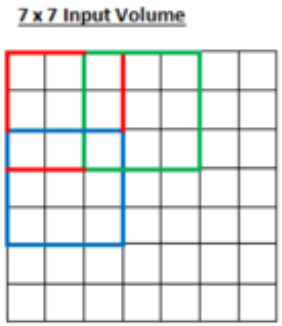
$$s[t] = (x * w)(t) = \Sigma_{a=-\infty}^{\infty} x[a]w[t-a]$$

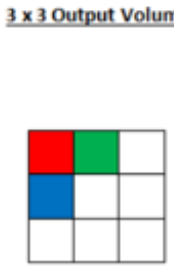
Example





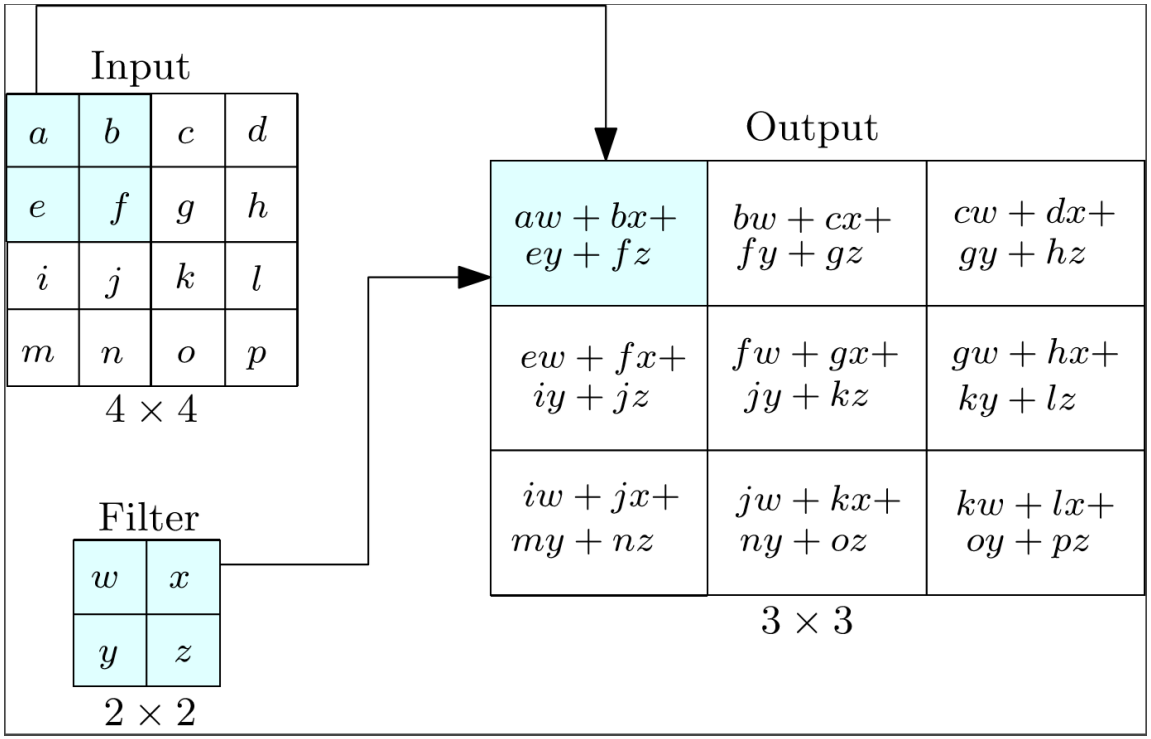
Stride = 1





Stride = 2

Example

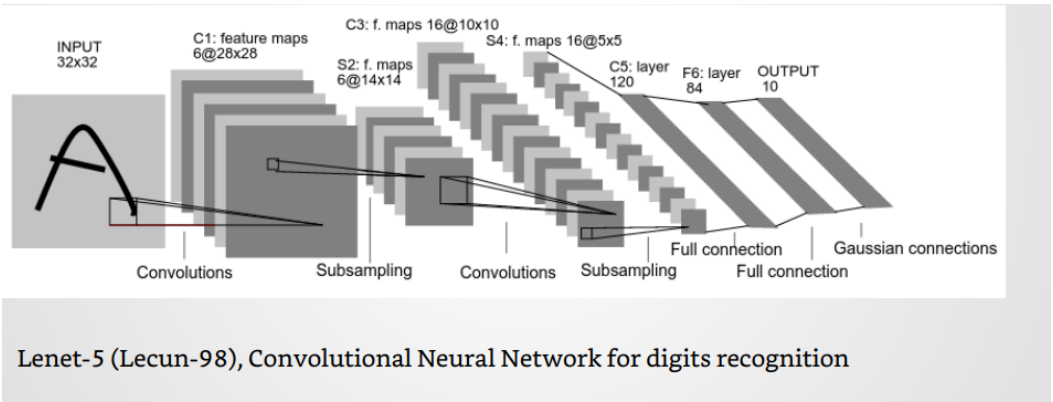


- Convolution exploits spatial local correlations in the feature space by enforcing local connectivity pattern between neurons of adjacent layers
- Drastic reduction of free parameters compared to fully connected network reducing over fitting and computational complexity of the network

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) $\rightarrow$ Most popular choice avoids saturation issues, makes learning faster

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

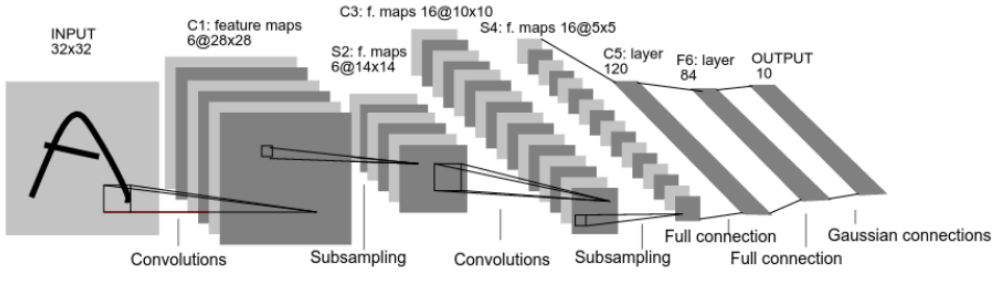


Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

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

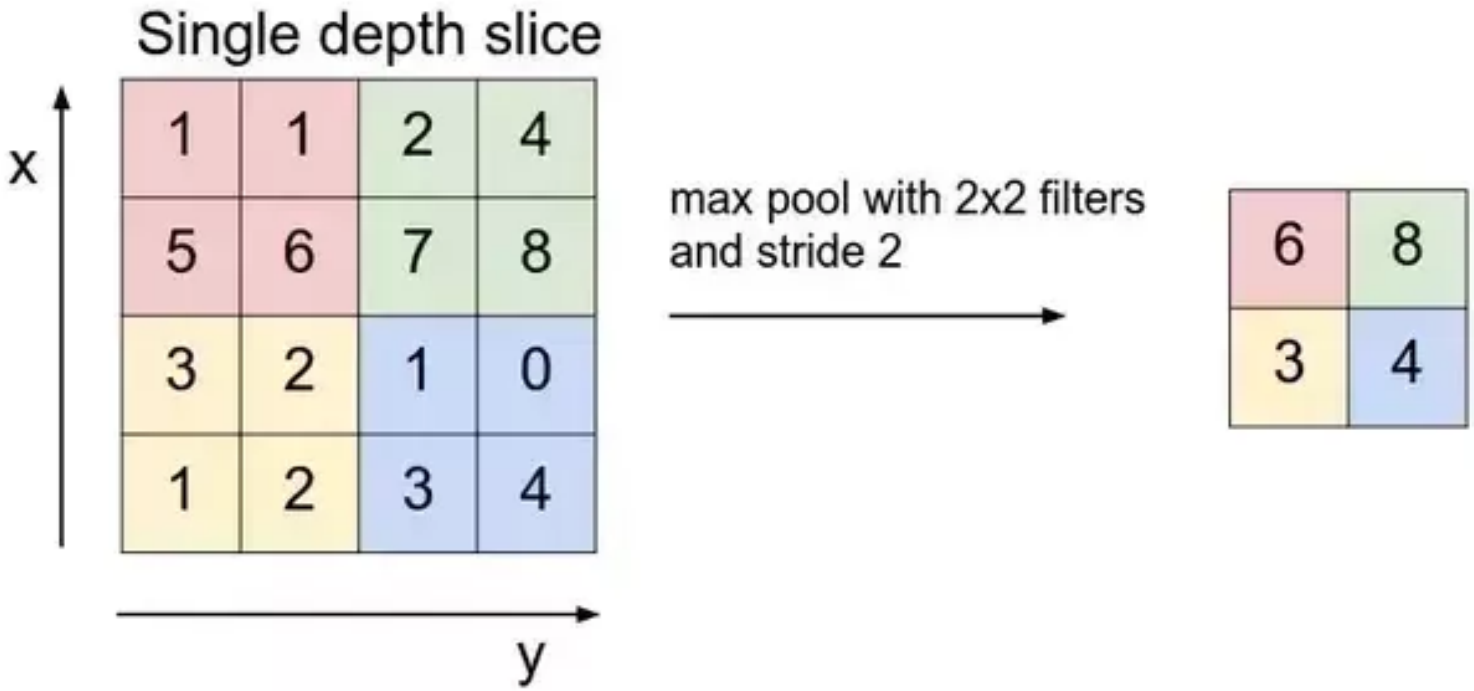
Pooling

- 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

Outline

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

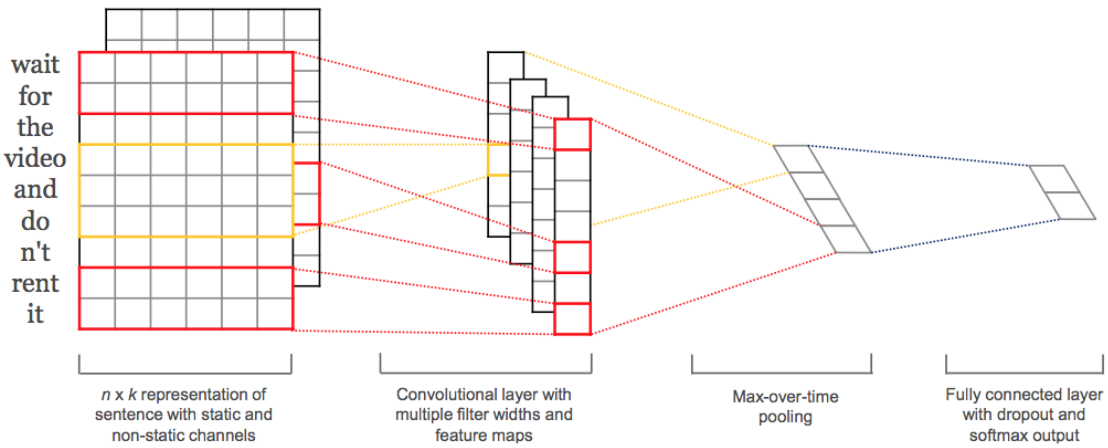
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \ldots \mathbf{x}_n$$

- 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

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



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

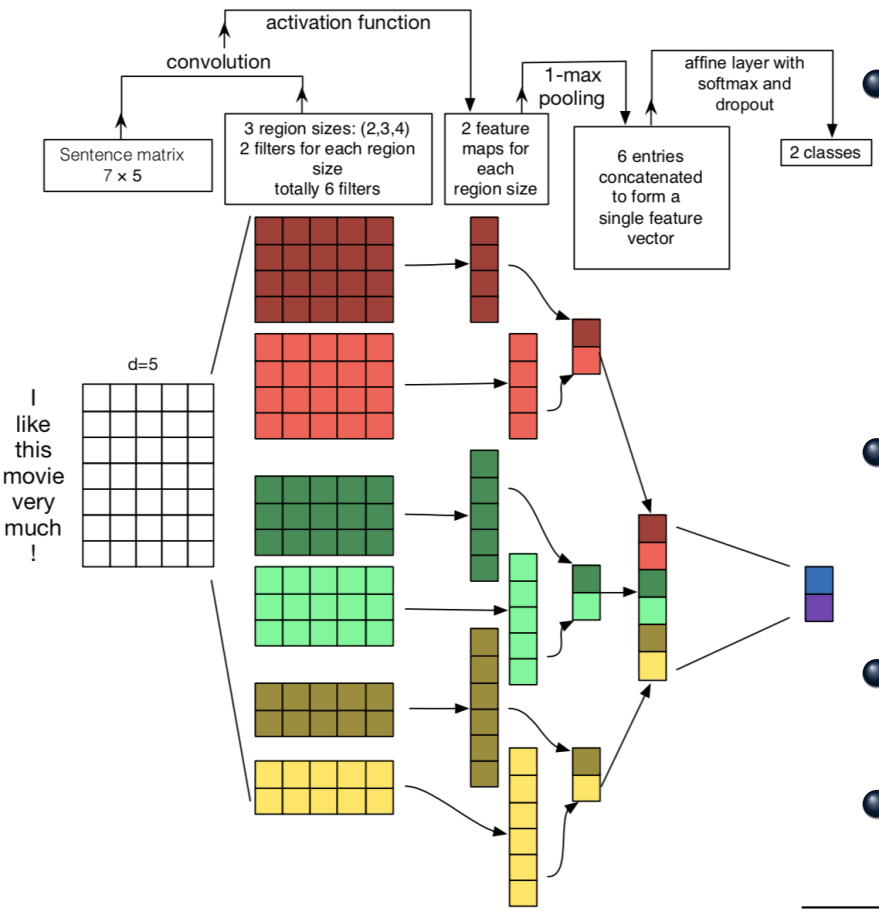
- Combine multiple word embeddings
- Each set of vectors is treated as a ‘channel’
- Filters applied to all channels
- Gradients are back-propagated only through one of the channels
- Fine-tunes one set of vectors while keeping the other static

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 for text classification



- Data (text) only 1^{st} 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 $\Rightarrow$ classification via softmax

---

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

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

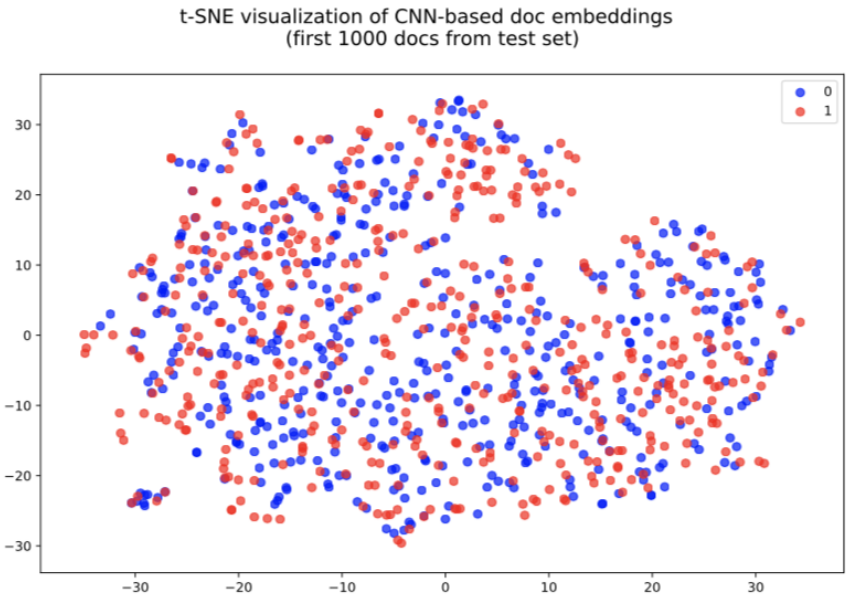


Figure 2: Doc embeddings before training.

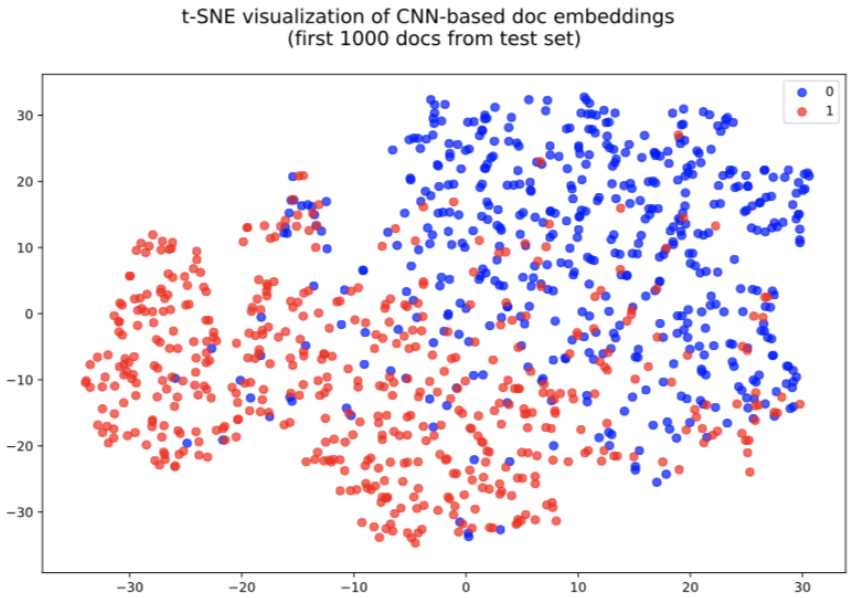


Figure 3: Doc embeddings after 2 epochs.

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

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

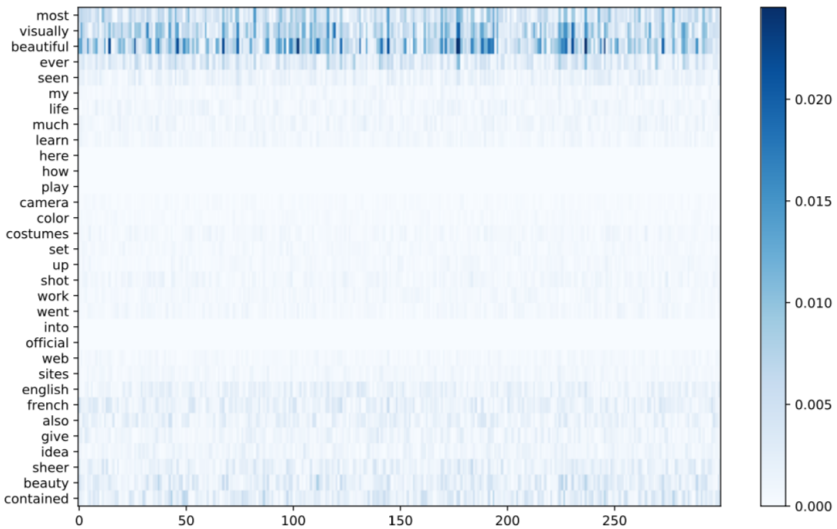


Figure 4: Saliency map for document 1 of the IMDB test set (true label: positive)

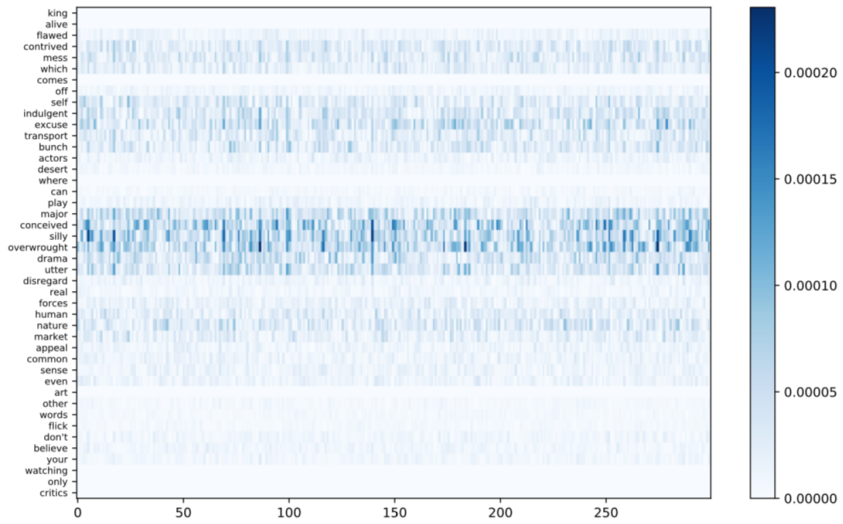
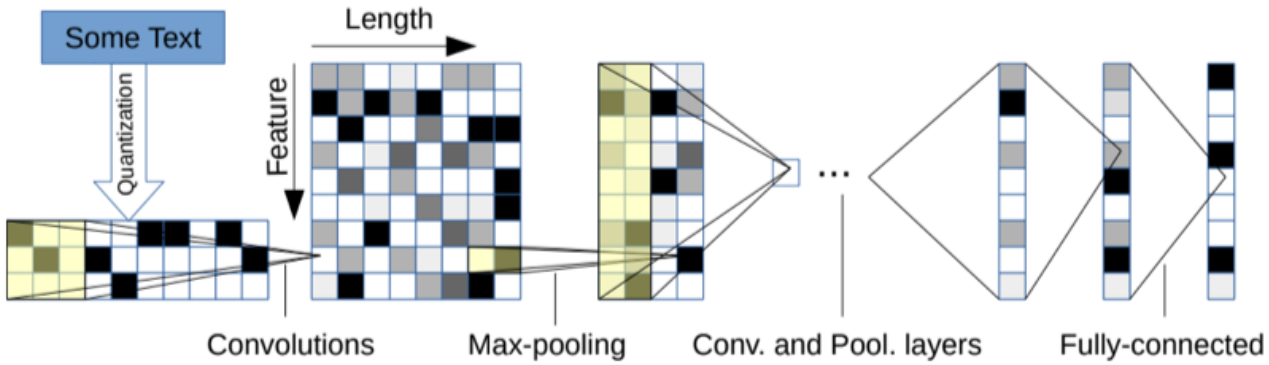


Figure 5: Saliency map for document 15 of the IMDB test set (true label: negative)

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

THANK YOU

Acknowledgements:
Dr. P. Meladianos - for his contributions in the CNN part
J.B. Remy - for his contributions in the DL Hyperparameter tuning part