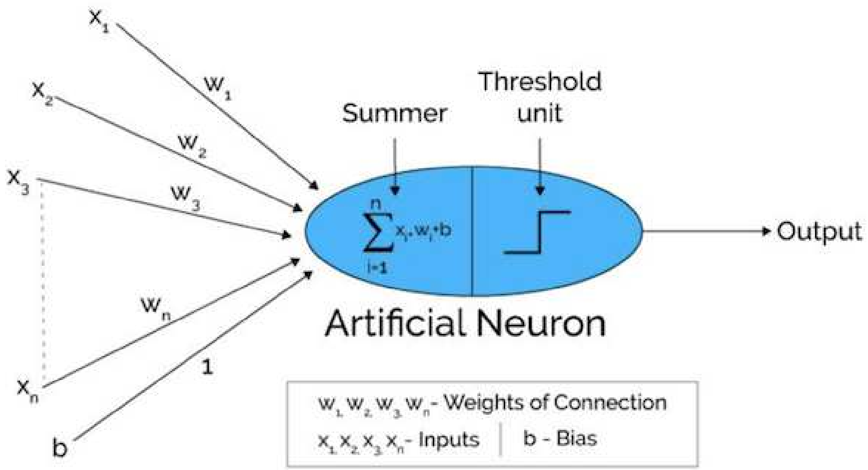
Deep Neural Networks

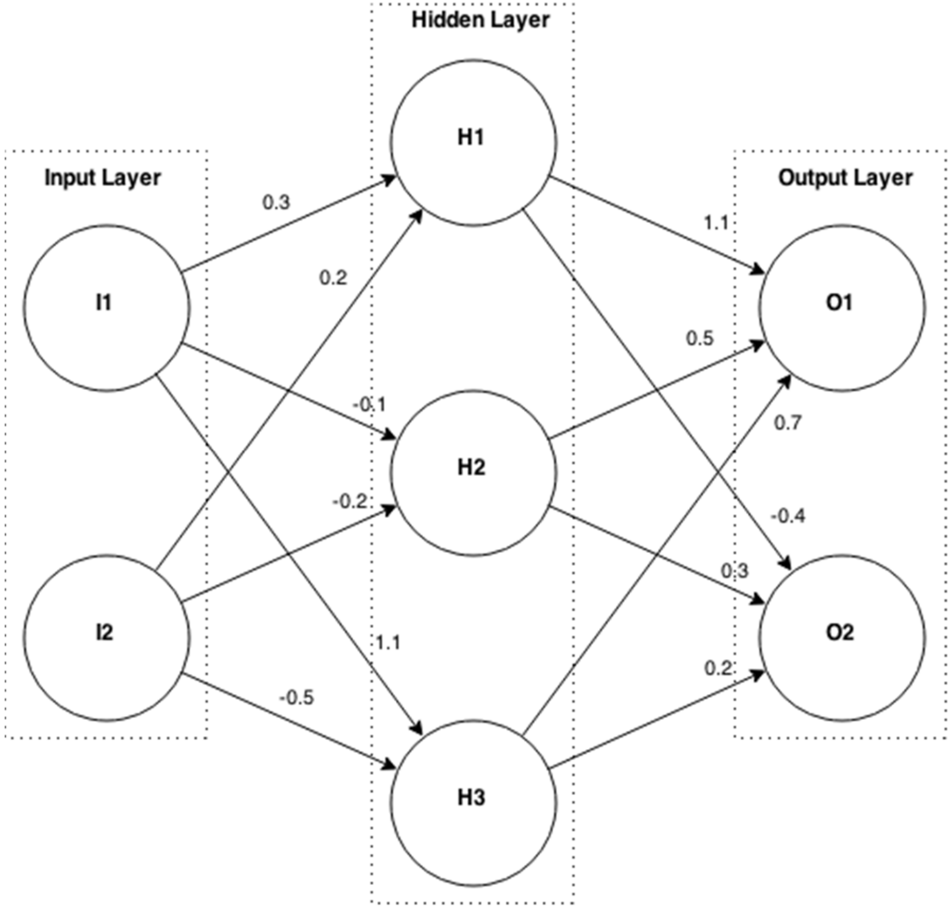
NSES – Lecture

Miroslav Hlaváč



Perceptron

- Mathematical approximation of biological neuron
  - Weighted sum of inputs and bias
  - Followed by activation function

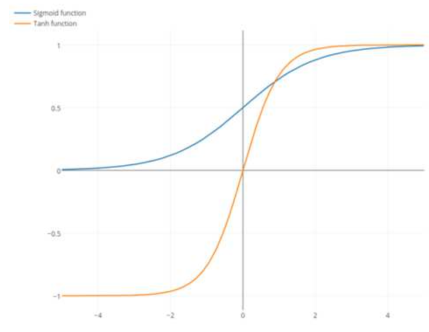


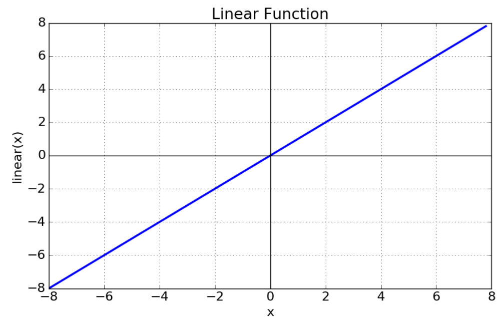
MLP – Multi Layer Perceptron

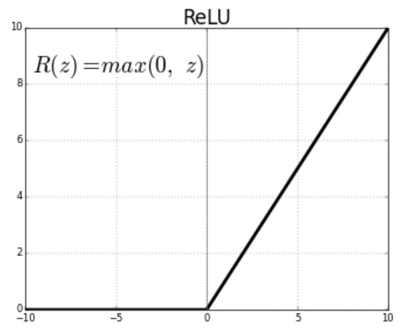
- Basic example of Multi Layer Perceptron
  - Input layer
  - Hidden layer
  - Output layer

Activation Functions

- Linear
  - $f(x) = x$
- Sigmoid
  - $f(x) = \frac{1}{(1+e^{-x})}$
- Hyperbolic Tangent
  - $f(x) = \tanh(x)$
- Rectified Linear Unit - ReLU
  - $f(x) = max(0, x)$







Error/Loss Function

- Performance metric based on desired and actual output of the network
  - The first thing that comes to mind – $error = output_{desired} - output_{actual}$
    - This error has a problem in being negative when the network overshoots and positive when it undershoots – absence of real minimum
  - It leads us to absolute error - $error = |output_{desired} - output_{actual}|$
    - This error function has minimal value when both outputs are the same – 0, but the training algorithm cannot differentiate between lot of small errors and few big errors
  - Lets introduce sum of squares of the absolute errors to differentiate between small and big errors
    - $error = \sum|output_{desired} - output_{actual}|^2$

Backpropagation

- The full meaning is “the backward propagation of errors”
- Backpropagation algorithm
  - Propagation
    - Forward pass to generate network output
    - Error calculation
    - Backpropagation of errors through the network to generate the difference between targeted and actual real values for all outputs and hidden neurons
  - Weight update

The network is considered deep when it has more than one hidden layer

- Basic example is Input layer + 2 Hidden layers + Output layer

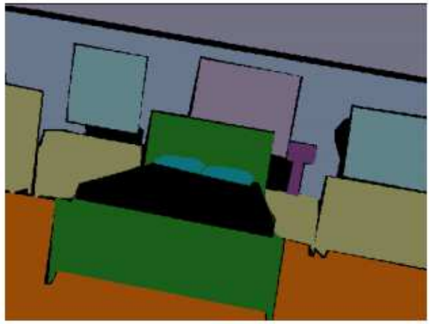
As the computer hardware evolves the number of hidden layers – complexity of DNNs - is increasing

- A research was done to find a relation between number of hidden layers (parameters) of a network and it's ability to solve complex problems – paper Deep Residual Learning for Image Recognition (2015) [ResNet]
- This paper introduced methods to achieve trainable networks with hundreds of layers and compared their results on performing the task of image classification

Deep Neural Network

Test samples

Ground Truth



SegNet

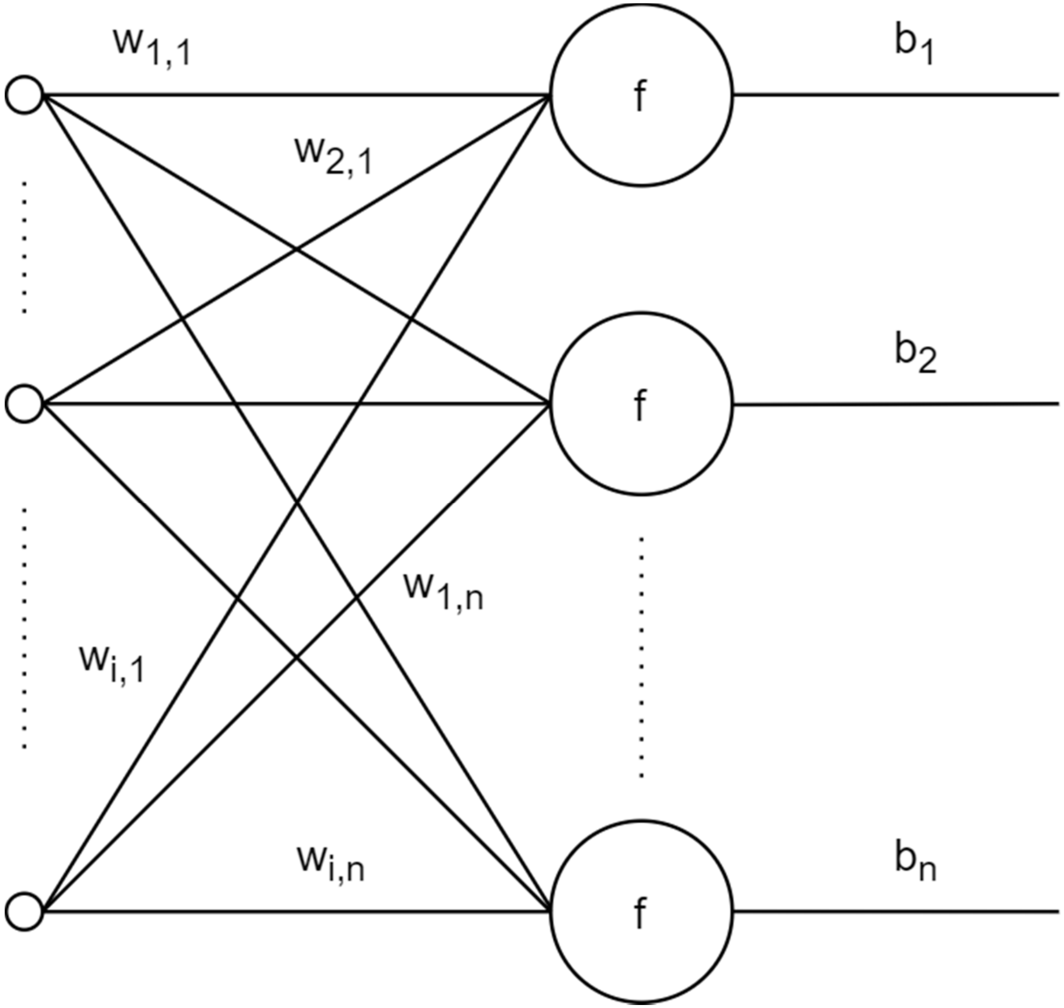
What can we use DNN for?

- Computer Vision
    - Face identification
    - Object recognition
- Speech
    - Audio and visual speech recognition
    - Text-to-speech
- Dimension reduction
    - Alternative to PCA
- Prediction
    - Stock exchange predictions

Deep Neural Network Topology

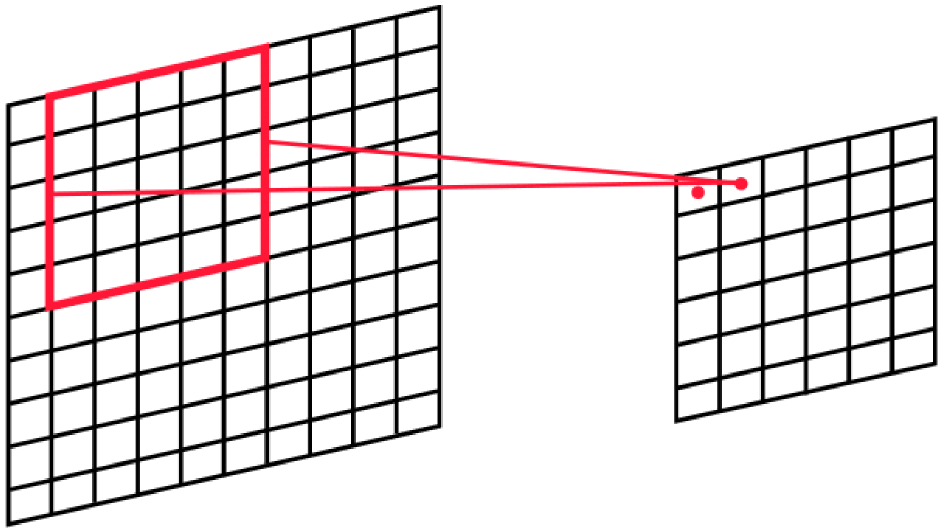
- The hidden layers in DNN can be arranged in different structures
  - Most common is the Fully-connected layer
- Different layers are suitable for different tasks
- There is no exact recipe how to form a network topology for a given task
- DNN can combine forward and recurrent layers, apply regularization and pooling after each layer output and many more trick to improve the ability of the whole network to perform a given task better
- Each layer is usually followed by an activation function

Different types of layers



Dense(Fully-connected) Layer

- The layer is represented by the number of neurons - n
- Each neuron is connected to every neuron in the previous layers - i
- The number of outputs is equal to the number of neurons
- Number of weights is $n \times i$
- This matrix is usually represented as a matrix of weights $\boldsymbol{W}$ and a vector of biases $\boldsymbol{b}$
- $f(\boldsymbol{x}) = \varphi(\boldsymbol{xW} + \boldsymbol{b})$

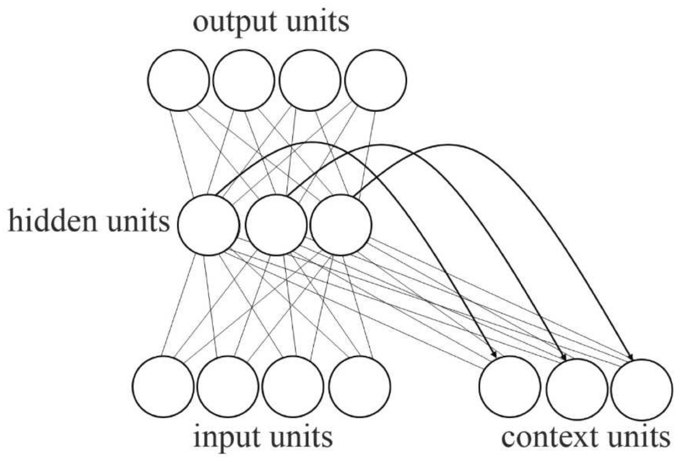


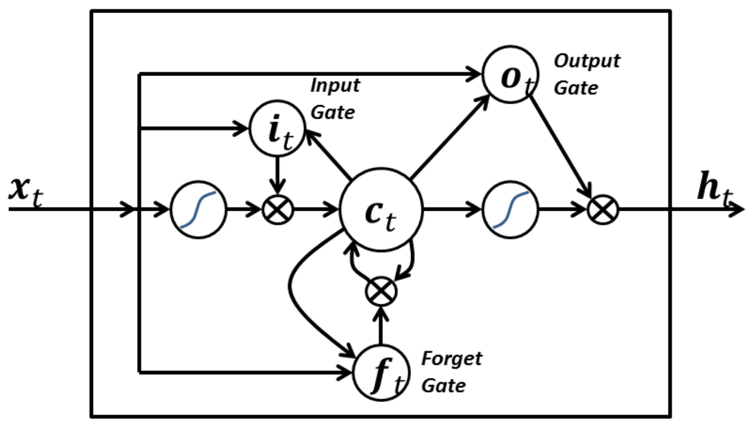
Convolutional Layer

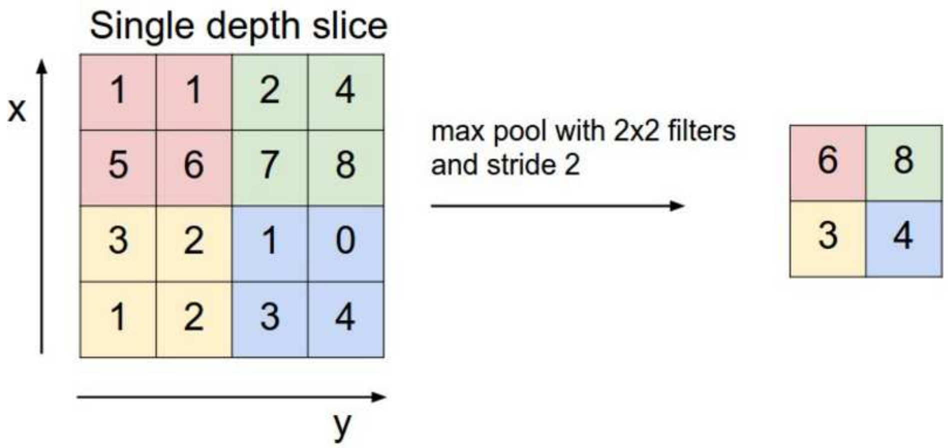
- This layer is composed of a set of trainable filters
- It is defined by the number of filters, size of the filters and the stride in each dimension the kernel is moved
- During forward pass the filter is slid(convolved) across the input data – (the picture represents a 2D example) and produces a dot product between the data and the weights in the filter
- The output is a stack of activation maps produced by the filters

Recurrent Layer

- Implemented as a set of recurrent units(cells)
- Types of cells:
  - Simple RNN – classical recurrent network
  - Gated Recurrent Unit (GRU) – implements reset gate
  - LSTM – implements input, output, and forget gate
  - Convolutional LSTM – input and recurrent transformations are implemented as convolutions

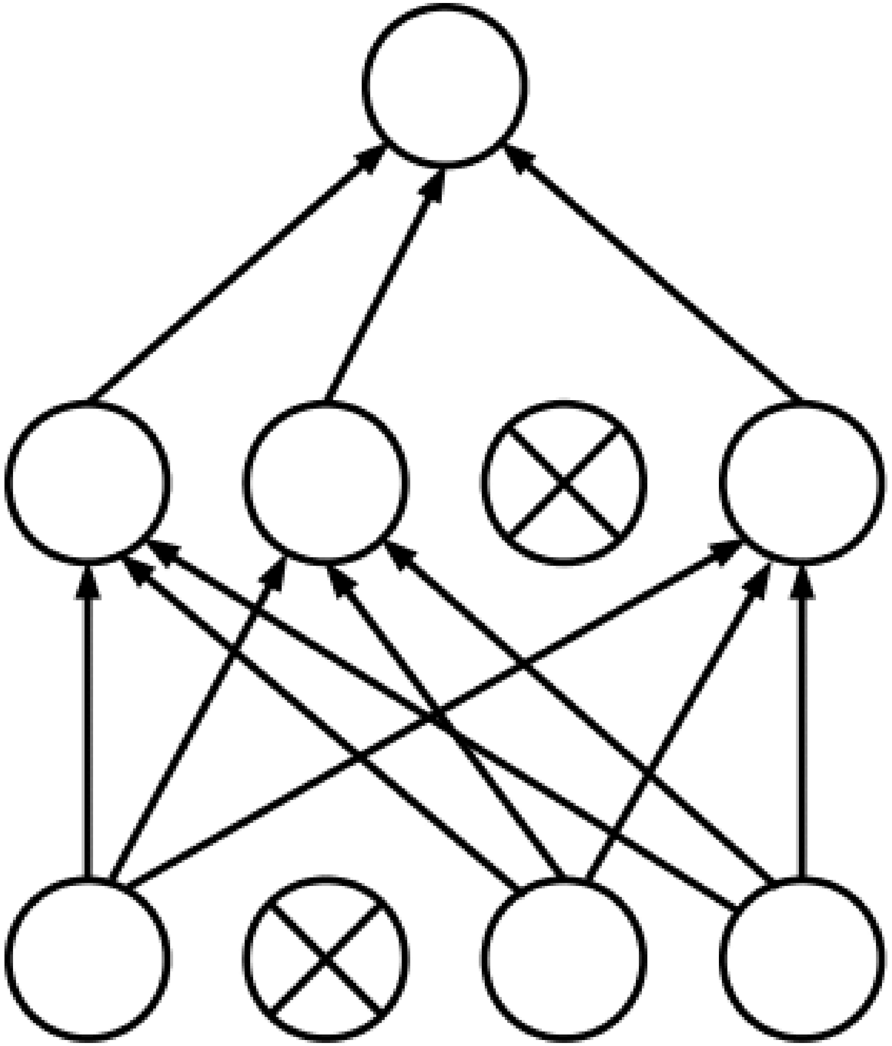






Pooling Layers

- Pooling is a process with no trainable parameters
- Rectangular window is slid over the data to compute:
  - Average
  - Maximum
  - Etc....



Response Normalization

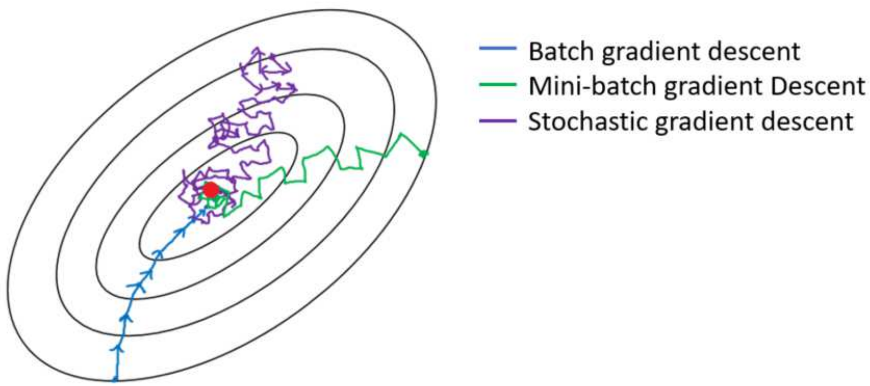
- Simulates biological concept of lateral inhibition
  - Capacity of excited neuron to outweigh the activity of neighbors
- Batch Normalization
  - Normalizes the outputs of the connected layer to have zero mean/unit variance
- Dropout
  - Sets the output of randomly selected outputs to zero

Classification Layer

- Softmax
  - Defined by a function
    - $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$
  - This function transforms the input vector to values between $\langle 0; 1 \rangle$ and their sum to be one
  - Used for classification tasks
  - Output values $\sigma(\mathbf{z})_j$ correspond to probabilities of the input to belong to the class j
  - Targets for the training are so called one-hot-vectors

Learning Objective

- Two types of objectives are currently used in DNN
  - Classification
    - Finding the probability that a given input belongs to a certain class
    - For classification with softmax we use categorical cross-entropy
      - $L(p, q) = -\sum_x p(x) \log q(x)$
      - Where x is the index of a class, p is the distribution(one-hot-vector), and q is the approximated distribution (softmax)
  - Regression
    - Approximation of the desired output given some input values
      - Mean Squared Error(MSE)/ RMSE
      - Mean Absolute Error
      - Hinge Loss $L\left(Y, \hat{Y}\right) = \frac{1}{N} \sum_i \max(1 - y_i \cdot \widehat{y_i}, 0)$



Optimization – Training the Network

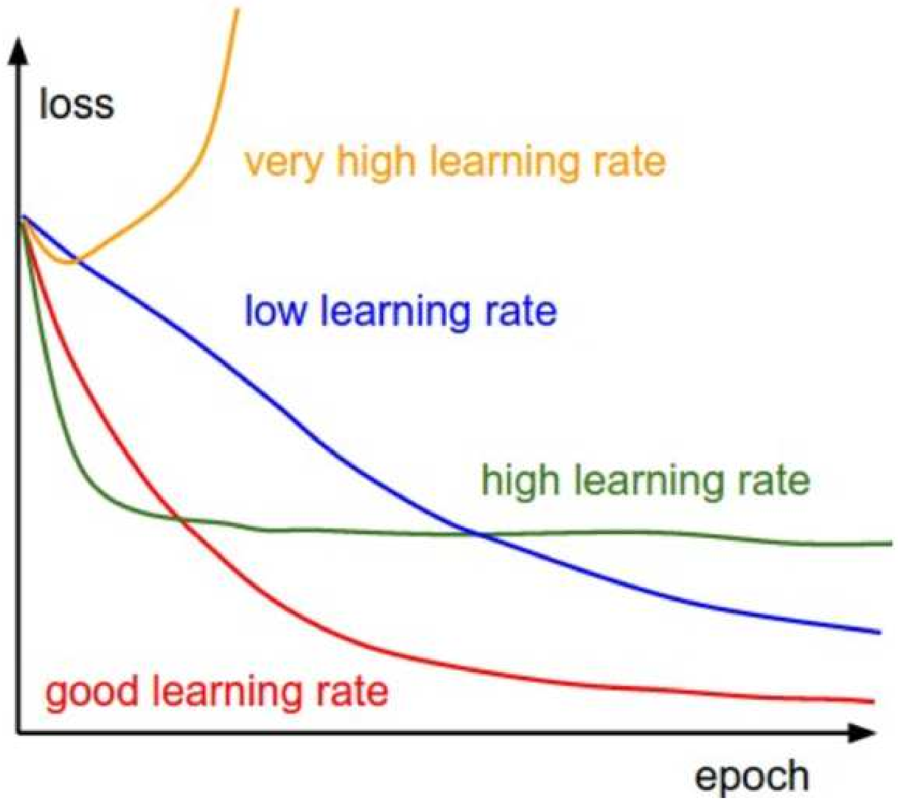
- Variations to the basic algorithm of Gradient Descent(GD)
  - Because the computer memory is limited and we can't always fit all the training data into memory
- Batch GD
  - Computes the error for each sample but the update is done after the whole training set
- Stochastic GD
  - Updates are done after evaluating each sample in the training set
- Mini-Batch GD

Optimization – Mini-Batch Gradient Descent

- Most commonly used implementation
- Splits the dataset into small batches
- The errors and gradients are calculated for each sample in the batch
- No need to keep all the data in memory, just the batch
- Hyperparameters
  - Learning rate
  - Batch size
  - Learning rate decay

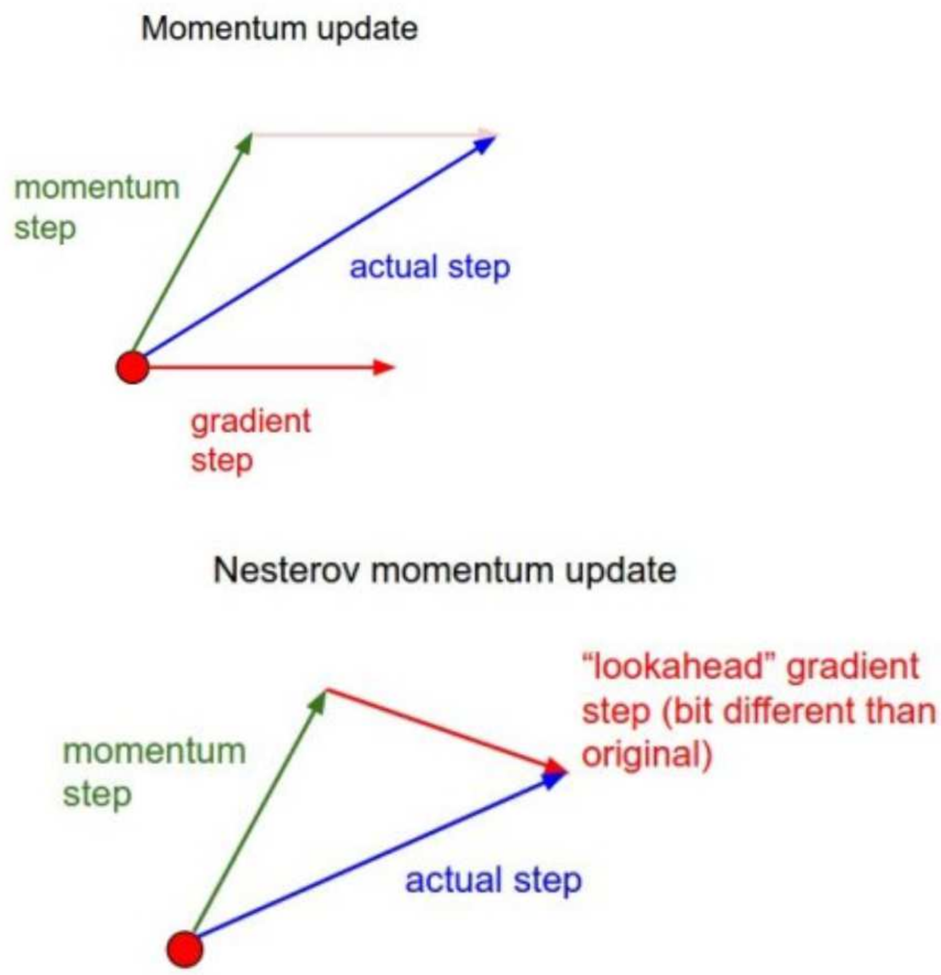
Optimization – Tricks to improve the convergence of GD

- Selection of initial value of learning rate is very important for convergence of GD
- Learning rate decay
  - Progressive
  - Step



Optimization – Tricks to improve the convergence of GD

- $\omega_{t+1} = \omega_t - \gamma \nabla L(\omega_t)$
- Momentum
- Nesterov Momentum
- Weight decay
  - $\omega_{t+1} = \omega_t - \gamma \nabla L(\omega_t) - \gamma \lambda \omega_t$



Optimization – Adagrad

- Adapts the learning rate to the parameters
  - Smaller updates to frequent features
  - Larger updates to infrequent features
- The learning rate is updated based on a sum of past gradients computed for each parameter separately
- Eliminates the need for selecting the starting learning rate
- The problem of the cumulative sum is it will grow indefinitely during the training process effectively shrinking the learning rate to zero

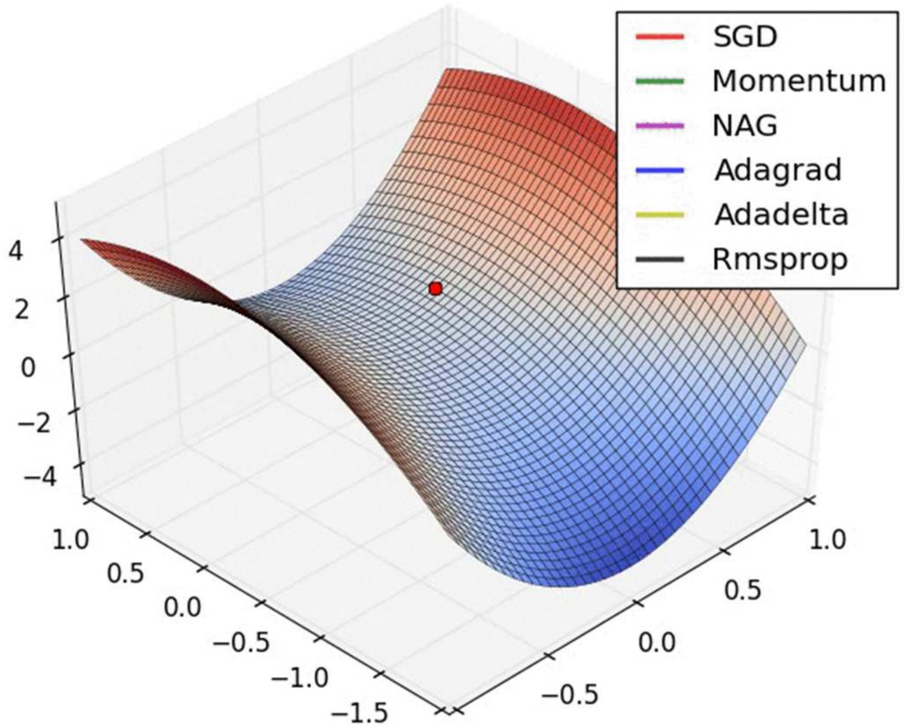
Optimization – RMSprop and Adadelta

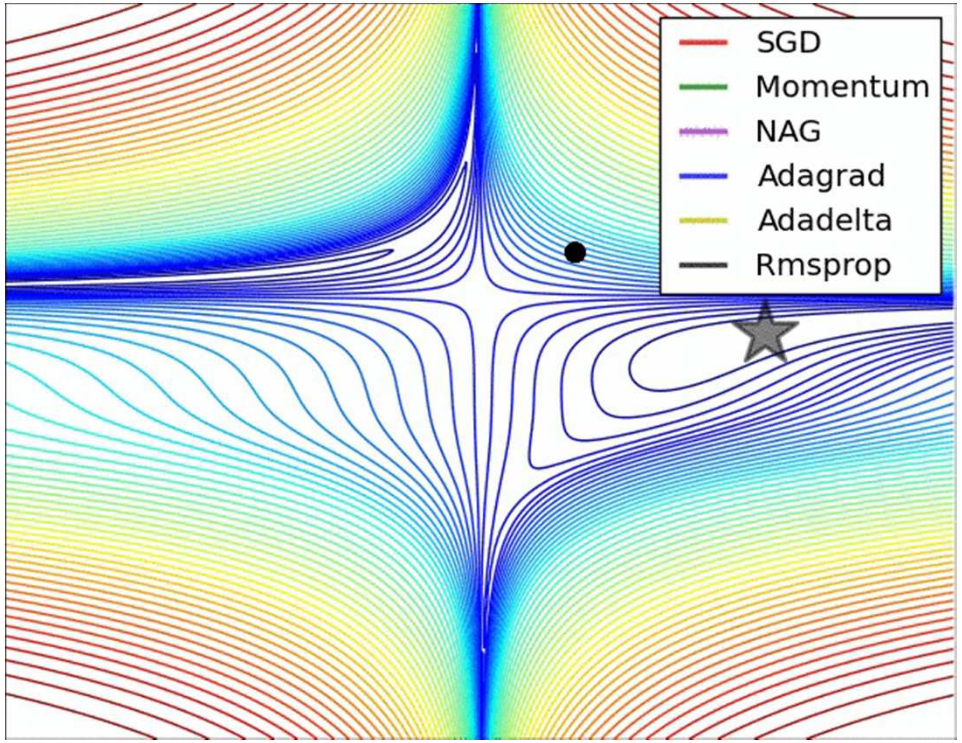
- Developed simultaneously to solve the problem of diminishing learning rate of Adagrad
- Adadelta takes only selected window of past gradients into account
  - Implemented as decaying average of past squared gradients
  - Solves the difference between hypothetical units of updates and parameters by approximating as the running average of previous updates
- RMSprop
  - Same update as Adadelta but neglects the difference in units
  - Not published in any paper, proposed in a Lecture on Cursera

Optimization – Adam

- Adaptive Moment Estimation
  - Computes adaptive learning rate for each parameter
  - In addition to Adadelta stores also the exponentially decaying average of past gradients, similar to momentum

Comparison of GD extensions





Deep Neural Network

- Main problems during training of DNN
  - Gradient vanishing and gradient explosion
    - The backpropagation algorithm updates the consecutive weights proportionally to the partial derivative of the error function
  - Overfitting
    - Very good result on training data, bad results on testing data
  - Degradation
    - Despite increasing the number of layers the training accuracy is lowering

Data augmentation

- The more parameters the network has the more data it will need to train itself for a given task
- Data resources are still limited or protected by law
  - For example medical images
- If we have only limited number of data we can increase the number by performing augmentations
  - Generally we can add noise with zero mean and variation of one
  - For images we can add rotation, translation, etc..
- This will also make the network more robust to variations in testing data

Data augmentation

- General augmentations:
  - Additional Gaussian Noise

- Typical types of augmentations for images:
  - Flip
  - Rotation
  - Scale
  - Crop
  - Translation

Programing your own DNN

- Frameworks
  - Tensorflow – Developed by Google
  - Caffe
  - Torch - PyTorch
  - CNTK – Developed by Microsoft
  - Chainer
- Hi-level API
  - Keras

Keras

- High level API for neural networks
- Written in Python
- Easy and fast
- Supports all currently used types of layers
    - Possibility to create own layers
- Utilizes both CPU and GPU for computations
- www.keras.io

Simple examples from Keras

- MLP – definition

```
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax’))
```

- MLP – optimizer

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,
          nesterov=True)
```

Simple examples from Keras

- MLP – compilation

```
model.compile(loss='categorical_crossentropy',
optimizer=sgd, metrics=['accuracy'])
```

- MLP – training

```
model.fit(x_train, y_train, epochs=20, batch_size=128)
```

- MLP – evaluation

```
score = model.evaluate(x_test, y_test, batch_size=128)
```