LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Deep Learning for Physicists

Lecture #2: Training neural networks

Kosmas Kepesidis

Some of things covered in the previous lecture…

- DNNs are trained using data by minimizing an **objective function**

- At each node of an DNN, two operations are performed:
  - ➢ Linear transformation with displacement (affine mapping)
  - ➢ Nonlinear transformation (activation function)

- Most common tasks where DNNs are used:
  - ➢ Regression: high-dimensional function approximation
  - ➢ Classification: classify objects into *m* categories

Outline

- Basics of numerical optimization
- Optimization of network parameters

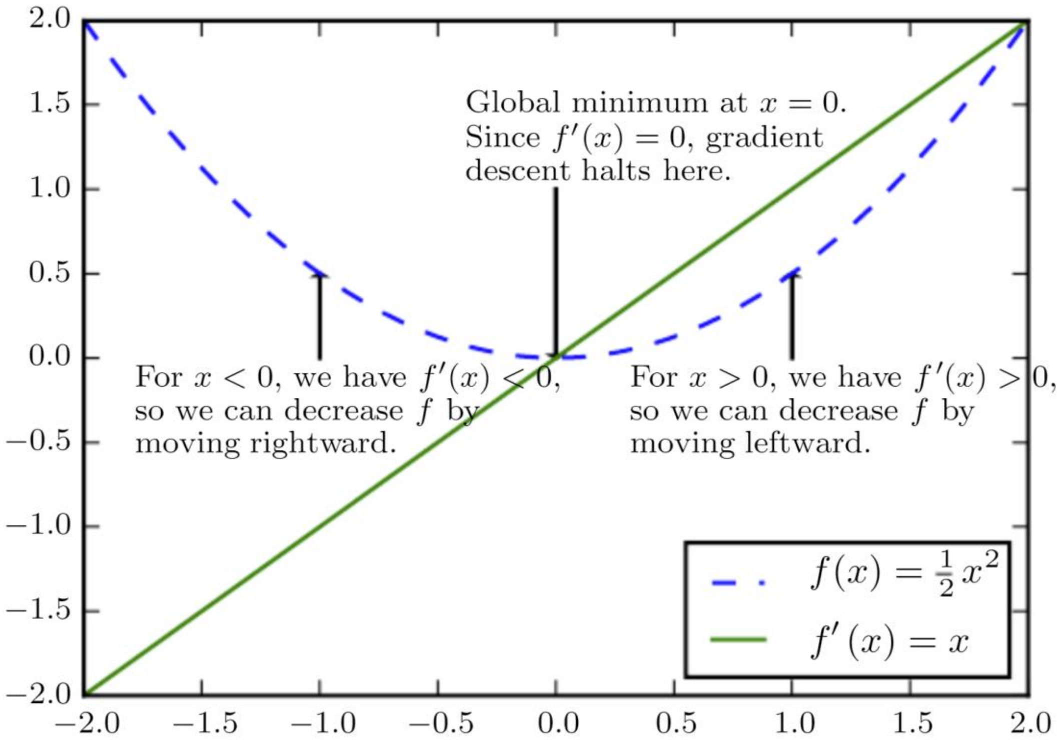
Basics of numerical optimization

Basics of numerical optimization

- Gradient-based optimization
  - Training deep-learning algorithms corresponds to an optimization problem
  - This is the task of minimizing (or maximizing) a certain function $f(\mathbf{x})$ (or $-f(\mathbf{x})$)
    - Typically functions with multiple input and a scalar output $f\colon \mathbb{R}^n \to \mathbb{R}$
  - Such a function is called **objective function**
    - Other names used: **criterion**, **cost function**, **loss function**, **error function**
  - Minimization using a method known as **gradient descent**

Basics of numerical optimization

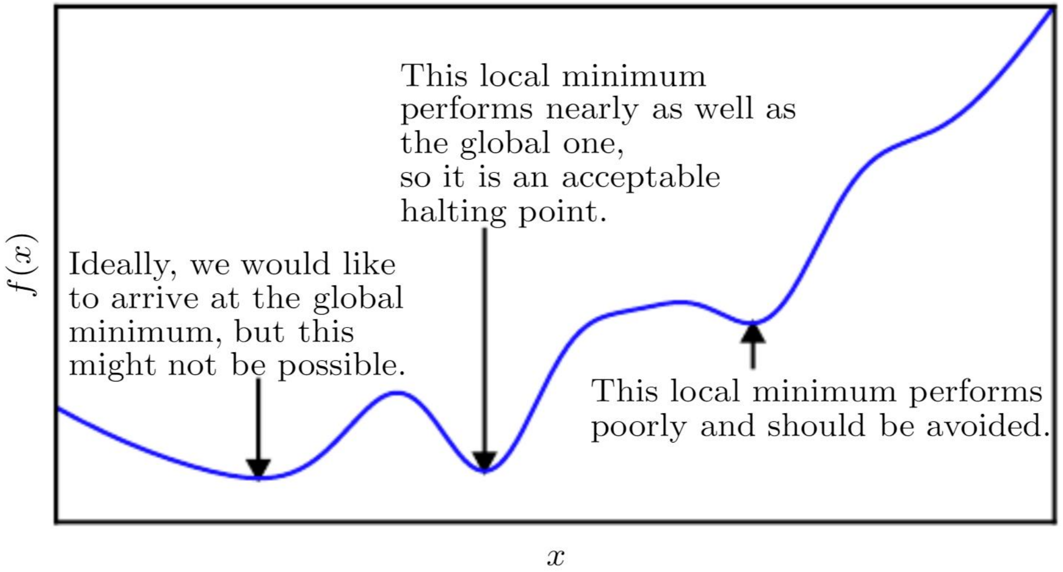
- Gradient-based optimization
  - Illustration of **gradient descent** in 1D convex case
    - Use first derivative $f'(x)$ for minimization
    - Look for the **global minimum**



Source: deeplearningbook.org

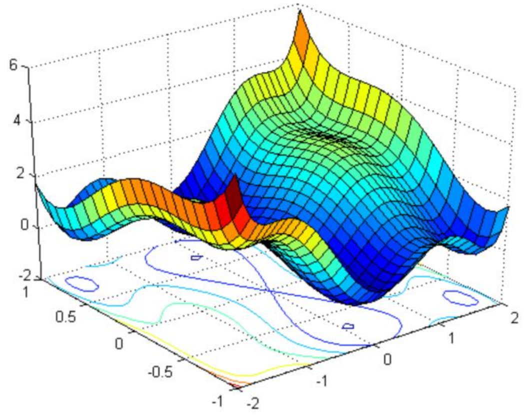
Basics of numerical optimization

- Gradient-based optimization
    - Issues in more realistic cases
        - There are local **minima/maxima** and **saddle points** (critical points $f'(x) = 0$)
        - Finding **global minimum** turns out to be a difficult job
        - Satisfying solutions of a "good" local minimum
        - Avoid poor-performing "bad" local minima



Source: deeplearningbook.org

Basics of numerical optimization



Source: https://algorithmia.com

- Gradient-based optimization
  - ➢ Typically, functions with multiple input and a scalar output are used $f: \mathbb{R}^n \rightarrow \mathbb{R}$
  - ➢ In multiple dimensions, the location of minima (critical points in general) is defined the partial derivative in each direction needs to be zero $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$
  - ➢ To minimize f one needs to find the direction in which f decreases the fastest – this is done using the **directional derivative** (slope of f in direction of unit vector $\boldsymbol{u}$)
    - ➢ See https://www.deeplearningbook.org/contents/numerical.html for more details
  - ➢ f can be decreased by moving in the opposite direction of the negative gradient – a method known as the **steepest descend** or **gradient descent**: $\mathbf{x}' = \mathbf{x} - \alpha \, \nabla_{\mathbf{x}} f(\mathbf{x})$
    - ➢ α is known as the **learning rate**

Basics of numerical optimization

- Gradient-based optimization

  - Performance of **gradient descent** depends on the differences in scale between input features of the data

  - Large differences in scales can lead to very long convergence times

Optimization of network parameters

Optimization of network parameters

- Input data preprocessing and numerical stability

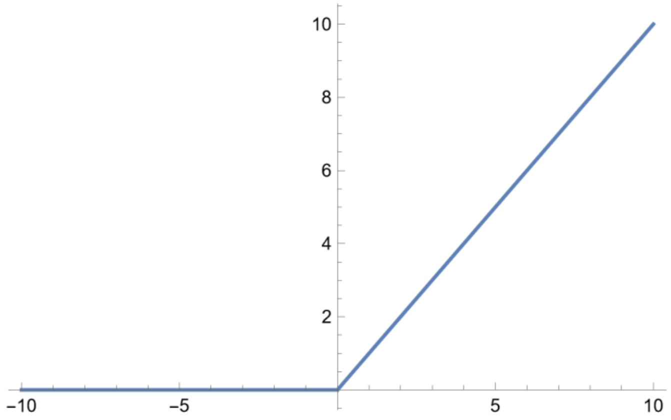
  - Numerical stability is crucial for successful training of neural networks

  - If the ranges of the input data vary by many orders of magnitude and are different for the individual components, adjusting network parameters becomes difficult

  - Careful preparation and pre-processing of the input data are essential for successful model building

Optimization of network parameters

- Input data preprocessing and numerical stability
  - Zero-centering:
    - Gradient of widely-used ReLU activation changes dramatically close to zero
    - This can affect the importance of results of the affine mappings $\vec{y} = \mathbf{W}\,\vec{x} + \vec{b}$
    - Solution: subtract mean values $x_i \rightarrow x_i - \langle x_i \rangle$



Optimization of network parameters

- Input data preprocessing and numerical stability

  - Scaling of data

    - If the values of different observables x_i are of different orders of magnitude, larger values might be considered more “significant”, when training a neural network

    - Solution: standard scaling $\quad x_i \to \frac{x_i - \langle x_i \rangle}{\sigma_i}$

    - If values in fixed interval $[x_{i,min}, \ \ x_{i,max}]$: $\quad x_i \to 2\frac{x_i - x_{i,min}}{x_{i,max} - x_{i,min}} - 1$

Optimization of network parameters

- Input data preprocessing and numerical stability
  - Logarithmic scale
    - For variables whose values fluctuate, it might be useful to redistribute values by transforming them using the logarithm (or exponential)

$$x_i \rightarrow \log(x_i)$$

$$x_i \rightarrow e^{-x_i}$$

Optimization of network parameters

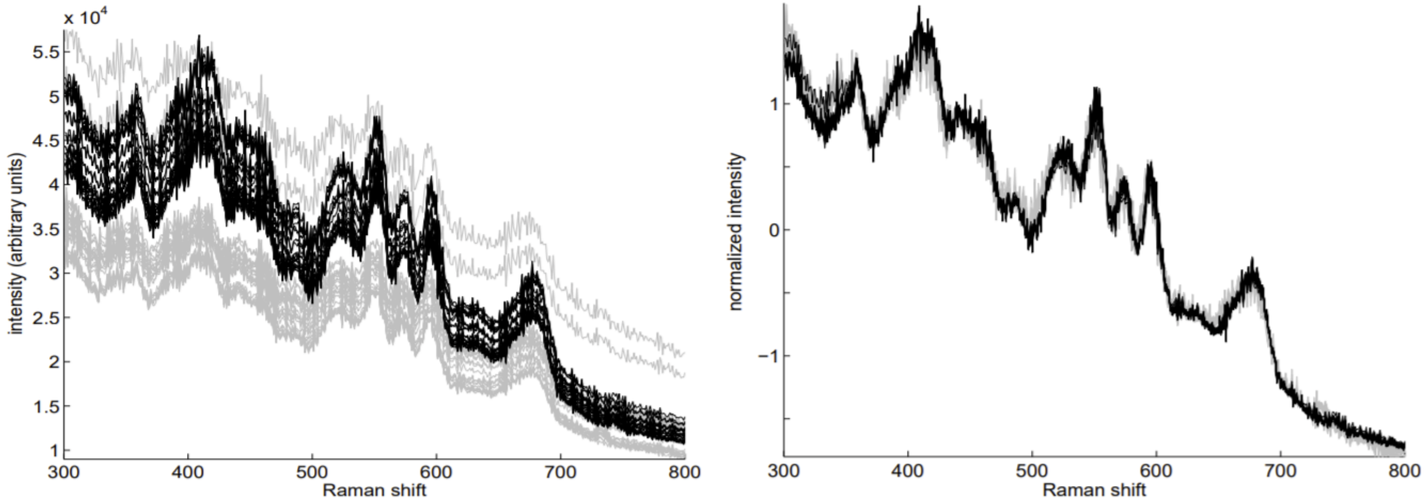
- Input data preprocessing and numerical stability
  - Decorrelation – reduce redundancy
    - Identify colinear variables and keep only one: $x_i = c * x_j$
    - Identify variables with a known functional form between them: $x_i = f(x_j)$

Optimization of network parameters

- Input data preprocessing and numerical stability
  - Normalization (global)
    - Series of measurements containing same quantity can be normalized with respect to a global value such as lobal average, maximum etc.
    - Examples:
      - Pixel intensity of an image
      - Absorbance in infrared (IR) spectroscopy

Optimization of network parameters

- Input data preprocessing and numerical stability
    - ➢Real example: Raman spectroscopy



Optimization of network parameters

- Input data preprocessing and numerical stability

  - Thought experiment with seismic stations
    - 50 measuring stations record seismic movements as function of time
    - 100k earthquake events collected, in time span of 5 years
    - Data includes the times $t_{p,i}$ and $t_{s,i}$ of maximum amplitudes $A_{s,i}$ and $A_{p,i}$ of compression and shear waves respectively

  - Scaling the time information:
    - $t_{p,i}$ and $t_{s,i}$ are measured in seconds after Unix timestamp (01.01.1970 at 00:00)
    - Such big numbers affect network performance
    - Solution: scaling using first and last recordings

$$t_{p,i} \rightarrow 2\,\frac{t_{p,i} - t_{p,i,min}}{t_{p,i,max} - t_{p,i,min}} - 1$$

Optimization of network parameters

- Input data preprocessing and numerical stability

  - Thought experiment with seismic stations
    - 50 measuring stations record seismic movements as function of time
    - 100k earthquake events collected, in time span of 5 years
    - Data includes the times $t_{p,i}$ and $t_{s,i}$ of maximum amplitudes $A_{s,i}$ and $A_{p,i}$ of compression and shear waves respectively

  - Log-transform amplitude:
    - Amplitude values are assumed to be in an interval $[0,1000]$ – with the majority close to zero
    - Log-transform is appropriate: $A_{s,i} \rightarrow \log(A_{s,i})$

Optimization of network parameters

- After data preprocessing:
  - ➢Training of DNNs is an iterative process
  - ➢Available data is used repeatedly and as efficiently as possible
  - ➢ Two terms are associated with this procedure: **epoch** and **minibatch**

Optimization of network parameters

- **Epoch**

  - One epoch of a network training denotes the one-time, complete use of all training data

Optimization of network parameters

- **Minibatch**

  ➢ The parameters of a network are iteratively optimized in many small steps

  - Using all training data in each step would be time-consuming and highly inefficient
  - Use instead a randomly selected sample of the training data in each iterative step
  - Size of the batch, in powers of 2 ($k = 2^{\mathrm{m}}$) to be chosen depends on the problem to be solved
  - Tradeoff: computing **costs** $\sim k$ to be balanced by **precision** $\sim \frac{1}{\sqrt{k}}$

Optimization of network parameters

- Network initialization

  - ➢While b_i are set to zero, initial weights **W** in mappings $\vec{y} = \boldsymbol{W}\vec{x} + \vec{b}$ are usually selected to be random numbers taken from
    1. uniform $[-s, s]$
    2. or standard normal $N(\mu = 0, \sigma)$ distribution

  - ➢For normally-distributed random weights, standard deviation should be optimally scaled by the numbers of input/output neurons ($n_{in},\ n_{out}$)
    - For tanh activation, recommend: $\sigma^2 = \frac{2}{n_{in}+n_{out}}$ (Glorot normal or Xavier normal initialization)
    - For ReLU activation, recommend: $\sigma^2 = \frac{2}{n_{in}}$

Optimization of network parameters

- Network initialization

  - While b_i are set to zero, initial weights **W** in mappings $\vec{y} = \boldsymbol{W}\vec{x} + \vec{b}$ are usually selected to be random numbers taken from
    1. uniform $[-s, s]$
    2. or standard normal $N(\mu = 0, \sigma)$ distribution

  - In the case of uniformly-distributed random weights, to recover the same spread of weights **W** as for the Gaussian initialization, use $s = 3 \times \sigma$

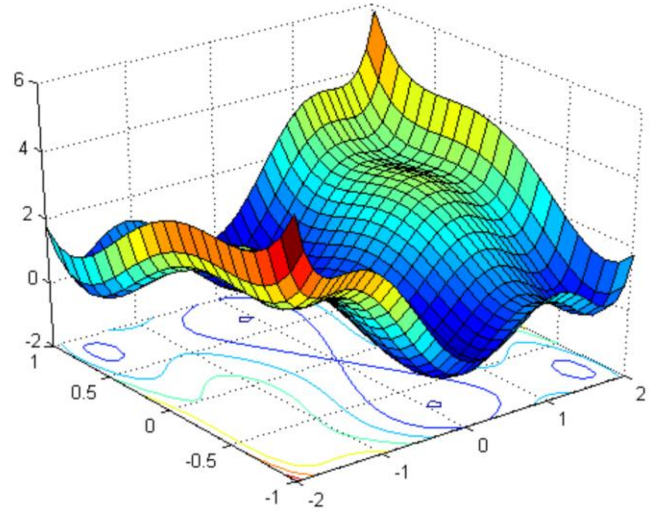
Optimization of network parameters

- **Objective function** (also known as **cost, loss** function) is a measure for evaluating networks predictions

  - ➢ It depends on all (to-be-optimized) parameters of the neural network

  - ➢ Approach a “good” local minimum on the hyper-plane of all parameters

  - ➢ The “goodness” of the local minimum is assessed by evaluation of the network predictions



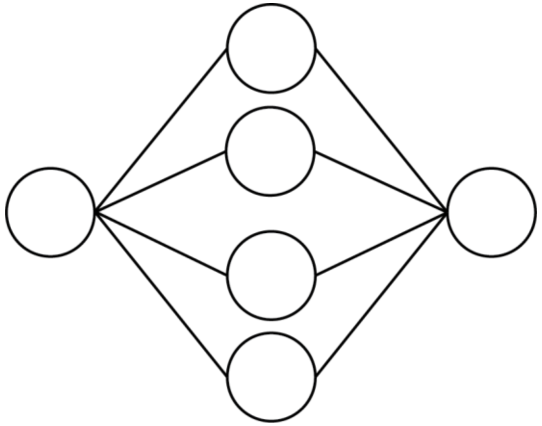
Source: https://algorithmia.com

Optimization of network parameters

- Objective function for **regression**:
  - Distance measure between predictions $f(x_i)$ and target values $y(x_i)$, where i runs over data points
    - **Mean absolute error** (MAE) – *Manhattan norm* $$\mathcal{L} = \frac{1}{k}\sum_{i=1}^{k}|f(x_i) - y(x_i)|$$
    - **Mean squared error** (MSE) $$\mathcal{L} = \frac{1}{k}\sum_{i=1}^{k}[f(x_i) - y(x_i)]^2$$
    - **Root mean squared error** (RMSE) – *Euclidean norm* $$\mathcal{L} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}[f(x_i) - y(x_i)]^2}$$

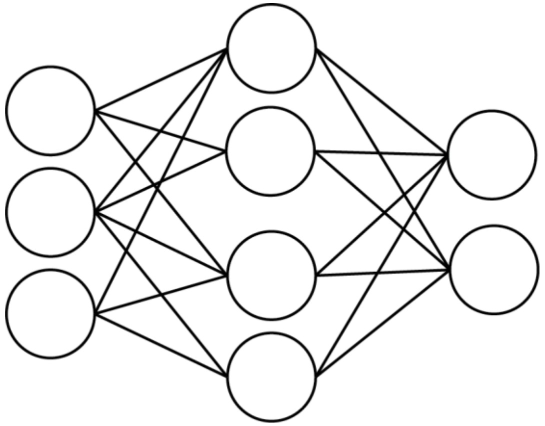
Optimization of network parameters

- Objective function for **regression**:
  - Distance measure between predictions $f(x)$ and target values $y(x)$
    - **Mean squared error** (MSE) $$\mathcal{L} = \frac{1}{k}\sum_{i=1}^{k}[f(x_i) - y(x_i)]^2$$
      - For 1D input
      - Does not account for multiple observables



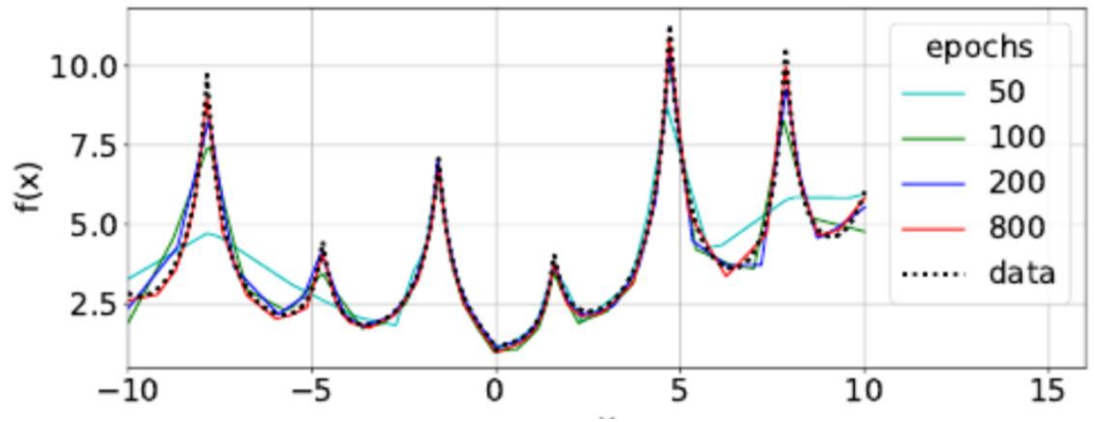
Optimization of network parameters

- Objective function for **regression**:
  - ➢distance measure between predictions $f(x)$ and target values $y(x)$
    - ▪ **Mean squared error** (MSE) $$\mathcal{L} = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{m}[f(\vec{x_i}) - y(\vec{x_i})]^2$$
      - Extended MSE to ***n*-dimensional input** and ***m*-dimensional output**
      - $\vec{x_i} = \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,n-1} \\ x_{i,n} \end{pmatrix}$



Optimization of network parameters

- Objective function for **regression**:
  - ➢distance measure between predictions $f(x)$ and target values $y(x)$
    - **Mean squared error** (MSE) $\mathcal{L} = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{m}[f(\vec{x_i}) - y(\vec{x_i})]^2$
      - Example: Function interpolation



Source: M. Erdmann et al., Deep Learning for Physics Research

Optimization of network parameters

- Objective function for **classification**
  - ➢Objective function needs to process probabilities
  - ➢Solution comes from statistical mechanics:
    - **Boltzmann entropy**: $S = k_B \log W$
      - k_B: Boltzmann constant
      - W: number of all equally-probable configurations in a system
    - Probability: $\mathrm{p} = 1/W$, therefore $S = -k_B \log(\mathrm{p})$
    - Generalize to non-uniform probability distributions:
      - $S = -k_B \langle \log(p_j) \rangle = -k_B \sum_j p_j \log(p_j)$

Optimization of network parameters

- Objective function for **classification**

  - **Shannon's entropy**: measure of ignorance in information theory

    - $S = -k_S \sum_j p_j \log(p_j)$, where $k_S = \frac{1}{\log(2)}$

    - $S = -\sum_j p_j \log_2(p_j)$

    - It is a measure of entropy in bits

Optimization of network parameters

- Objective function for **classification**

  ➢ **Cross-entropy**: distinguishes between true probabilities (p_j) and estimated probabilities (q_j)

  $$H = -\sum_j p_j \log(q_j)$$

    - Interesting and useful property: H becomes minimal for $p_j = q_j$

  ➢ Objective function based on cross entropy:

  $$\boldsymbol{\mathcal{L}} = -\frac{1}{k}\sum_{i=1}^{k}\left[\sum_{j=1}^{m} p_j \log\left(q_j\right)\right]_i$$

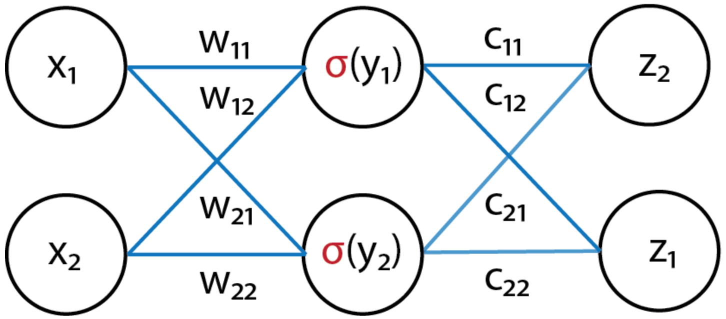
Optimization of network parameters

- Objective function for **classification**

  - For binary classification problem i.e. $m = 2$ and $p_1 + p_2 = 1$, cross-entropy is given by:

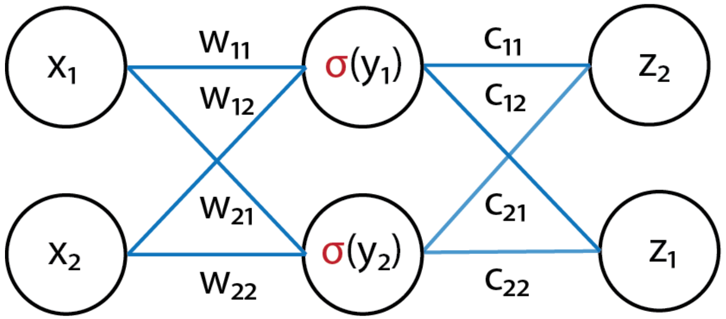
$$\mathcal{L} = -\frac{1}{k}\sum_{i=1}^{k}[\mathrm{p}_1\log(q_1) + (1 - p_1)\log(1 - q_1)]_i$$

Optimization of network parameters



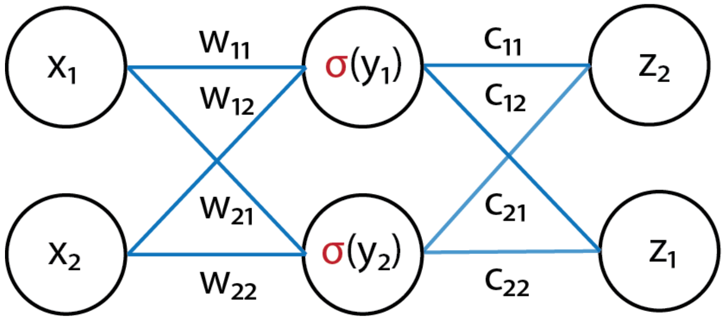
- Objective function for **classification**
  - ➢Example: separating signal from noise
    - NN with two output nodes: $\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \begin{pmatrix} \sigma(y_1) \\ \sigma(y_2) \end{pmatrix} + \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$
    - Activation function: hyperbolic tangent $\sigma(y) = \tanh(y)$
    - Want to transform results of output layer into probabilities, i.e. $\hat{z}_1 + \hat{z}_2 = 1$
    - This is done using the **softmax** function: $\vec{q} \equiv \begin{pmatrix} \hat{z}_1 \\ \hat{z}_2 \end{pmatrix} = \frac{1}{e^{z_1} + e^{z_2}} \begin{pmatrix} e^{z_1} \\ e^{z_2} \end{pmatrix}$

Optimization of network parameters



- Objective function for **classification**
  - ➢ Example: separating signal from noise
    - NN with two output nodes: $\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \begin{pmatrix} \sigma(y_1) \\ \sigma(y_2) \end{pmatrix} + \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$
    - Activation function: hyperbolic tangent $\sigma(y) = \tanh(y)$
    - Transform results of output layer into probabilities, i.e. $\widehat{z_1} + \widehat{z_2} = 1$
    - This is done using the **softmax** function: $\vec{q} \equiv \begin{pmatrix} \widehat{z_1} \\ \widehat{z_2} \end{pmatrix} = \frac{1}{e^{z_1} + e^{z_2}} \begin{pmatrix} e^{z_1} \\ e^{z_2} \end{pmatrix}$
    - Signal: $\widehat{z_1} > \widehat{z_2}$, noise: $\widehat{z_1} < \widehat{z_2}$

Optimization of network parameters



- Objective function for **classification**
  - Example: separating signal from noise
    - True values encoded in **one-hot-encoding**: $\vec{p}^T = (p_1 \quad p_2) = \begin{cases}(1 \quad 0):\ Signal \\ (0 \quad 1):\ Noise\end{cases}$
    - For training the network we use the cross-entropy for a mini-batch of size k with data and $m = 2$ classes

$$\mathcal{L} = -\frac{1}{k}\sum_{i=1}^{k}\left[\sum_{j=1}^{m=2} p_j(\vec{x_i}) \log\left(q_j(\vec{x_i})\right)\right] = -\frac{1}{k}\sum_{i=1}^{k} \vec{p}^T(\vec{x_i}) \log\left(\vec{q}^T(\vec{x_i})\right)$$

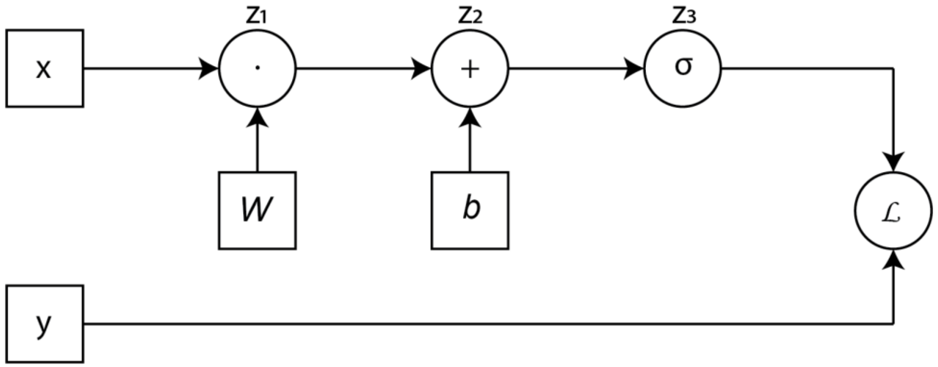
Optimization of network parameters

- Numerical optimization is achieved using **gradient descent**
  - ➢it is based on gradients of objective function with respect to the network parameters $\overrightarrow{W}$ and $\vec{b}$
  - ➢The gradients are obtained using a method called **backpropagation**

  - ➢Example with a single node:

$z_1 = Wx \qquad z_3 = \sigma(z_2)$

$z_2 = z_1 + b \qquad \mathcal{L} = (y - z_3)^2$



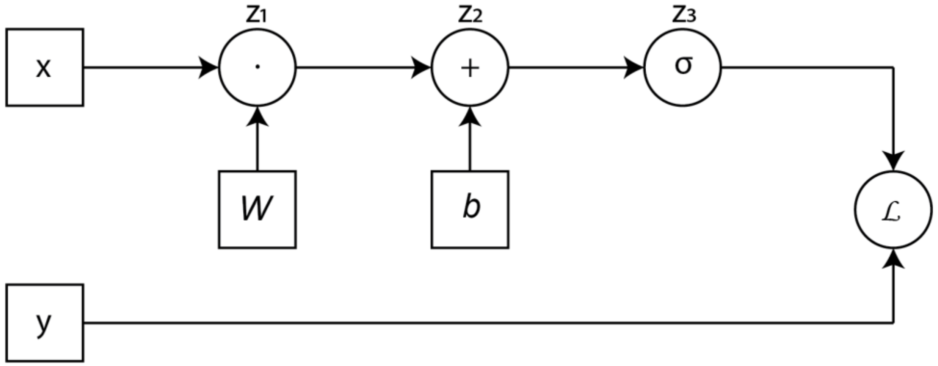
Question: How can parameters be changed to achieve minimal values of $\mathcal{L}$?

Optimization of network parameters

- Using the chain rule, calculate partial derivatives of $\mathcal{L}$ with respect to the W and b:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial W}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial b}$$



➢ Calculation of derivative done opposite to the direction of forward pass: **backpropagation**

Optimization of network parameters

- Optimizing with **stochastic gradient descent** (SGD)

  - Gradients of objective function calculated as average over k data points of the minibatch

$$\mathrm{E}\left[\frac{\partial \mathcal{L}}{\partial W}\right] = \frac{1}{k}\sum_{i=1}^{k}\frac{\partial \mathcal{L}}{\partial W} \qquad \mathrm{E}\left[\frac{\partial \mathcal{L}}{\partial b}\right] = \frac{1}{k}\sum_{i=1}^{k}\frac{\partial \mathcal{L}}{\partial b}$$

  - SGD leads to greater variance
    - Performing optimization in terms of minibatches enables multiple parameter updates in a single epoch
    - This approach is more robust against “unwanted” local minima

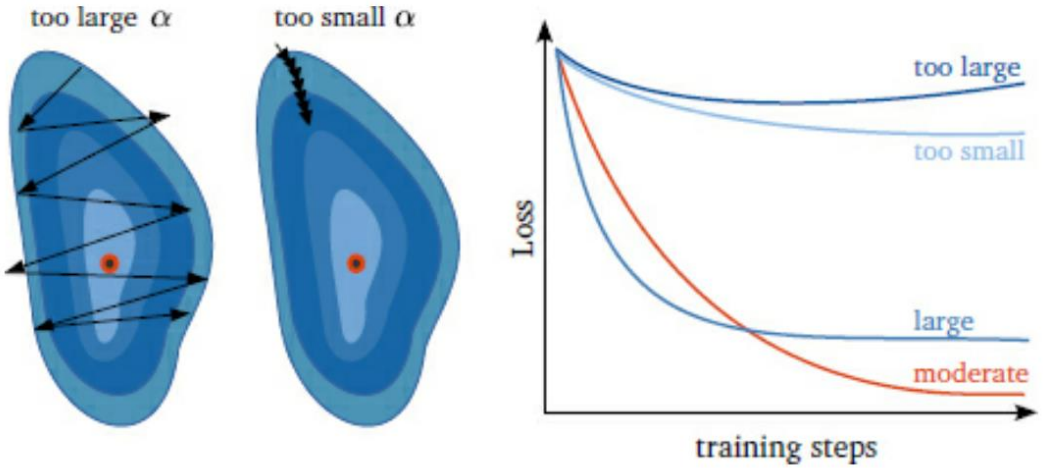
Source: M. Erdmann et al., Deep Learning for Physics Research

Optimization of network parameters

- SGD evaluates whether parameter W (and/or b) needs to be increased or decreased in the next iteration step
- How large the change in the parameters will be (step size) is determined by the **learning rate:** α

$$W_{t+1} = W_t - \alpha \, \mathrm{E}\left[\frac{\partial \mathcal{L}}{\partial W}\right]_t$$

$$b_{t+1} = b_t - \alpha \, \mathrm{E}\left[\frac{\partial \mathcal{L}}{\partial b}\right]_t$$



Source: M. Erdmann et al., Deep Learning for Physics Research

Optimization of network parameters

- SGD evaluates whether parameter W (and or b) needs to be increased or decreased in the next iteration step
- How large the change in the parameters will be (step size) is determined by the **learning rate:** α

- Learning rates are varied in the training process
  - Initially, parameters are examined with larger step sizes
  - Usually in range: $\alpha = 10^{-5} - 10^{-2}$
  - Subsequently, α is gradually reduced: examining parameters for smaller and smaller step size

Optimization of network parameters

- Learning strategies: methods for accelerating network training!

1. **Adagrad** (adaptive gradient): adaptive learning rates

   - During optimization, α reduces continuously
   - Changes in α, are adapted individually for each parameter
   - Takes into account sum of squares of all previous gradients

$$v_t = \sum_{\tau=1}^{t} \left(\frac{\partial \mathcal{L}}{\partial W} \right)^2 \qquad \alpha_t = \frac{\alpha}{\sqrt{v_t} + \epsilon}$$

   - $\epsilon \approx 10^{-8}$ guaranties not division by zero

J. Duchi et al. *Adaptive subgradient methods for online learning and stochastic optimization.* Journal of machine learning research 12.7 (2011)

Optimization of network parameters

- Learning strategies: methods for accelerating network training!

2. **RMSprob**: adaptive learning rates as well

    - Includes decay parameter to suppress influence from gradients of older steps
    - Decay parameter with typical value: $\beta = 0.9$

$$v_t = \beta\ v_{t-1} + (1-\beta)\left(\frac{\partial \mathcal{L}}{\partial W}\right)^2 \qquad \alpha_t = \frac{\alpha}{\sqrt{v_t} + \epsilon}$$

G. Hinton, Lecture series, lecture 6e in 2014, (2012)

Optimization of network parameters

- Learning strategies: methods for accelerating network training!

3. **Momentum**: considers parameters and gradients geometrically

$$\vec{\theta} = \begin{pmatrix} W \\ b \end{pmatrix} \qquad \nabla\vec{\theta} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial W} \\ \frac{\partial \mathcal{L}}{\partial b} \end{pmatrix}$$

➢ Goal the most efficient step size in the right direction in parameter space

$$\vec{u}_t = -\alpha \nabla \vec{\theta}_t \qquad \vec{\theta}_{t+1} = \vec{\theta}_t + \vec{u}_t$$

B. Polyak, Some methods of speeding up the convergence of iteration meth-ods, USSR Computational Mathematics and Mathematical Physics 4 (1964)

Optimization of network parameters

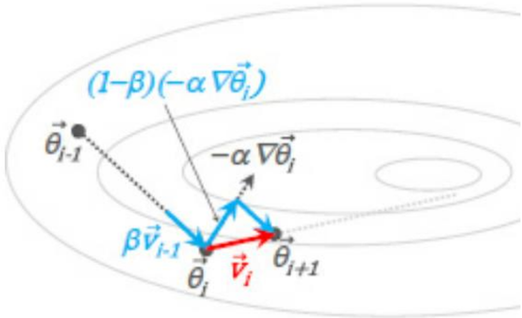
- Learning strategies: methods for accelerating network training!

  3. **Momentum**: considers parameters and gradients geometrically

  ➢ Essential aspect of momentum method: Stabilize direction of optimization using the history of velocity

$$\vec{u}_t = \beta \vec{u}_{t-1} + (1-\beta)\left(-\alpha \nabla \vec{\theta}_t\right)$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \vec{u}_t$$



Source: M. Erdmann et al., Deep Learning for Physics Research

  ➢ Coefficient β balances influence of previous velocity and its modification
    ➢ Typical values: $\beta = 0.5, \dots, 0.9$

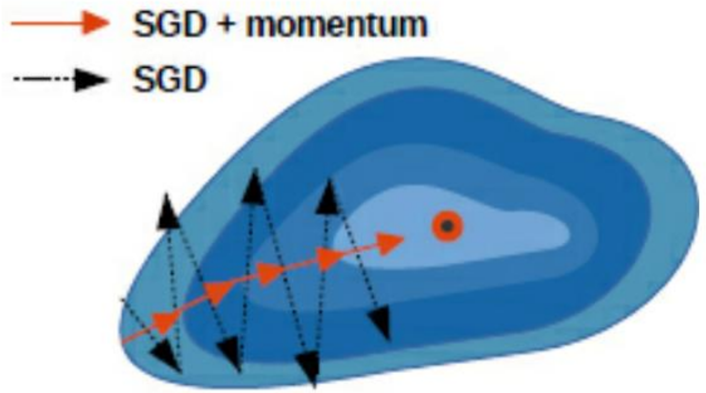
B. Polyak, Some methods of speeding up the convergence of iteration meth-ods, USSR Computational Mathematics and Mathematical Physics 4

Optimization of network parameters

- Learning strategies: methods for accelerating network training!

3. **Momentum**: considers parameters and gradients geometrically

➢ Leads to oscillating behavior in the parameter space – with damping!



Source: M. Erdmann et al., Deep Learning for Physics Research

B. Polyak, Some methods of speeding up the convergence of iteration meth-ods, USSR Computational Mathematics and Mathematical Physics 4

Optimization of network parameters

- Learning strategies: methods for accelerating network training!

    4. **Adam** (adaptive moments): combines ideas of RMSprob and the momentum method

        - Both gradients and their squares are subject to decay:

$$m_t = \frac{1}{1-\gamma^t}\left[\gamma\, m_{t-1} + (1-\gamma)\frac{\partial\mathcal{L}}{\partial W}\right]$$

$$v_t = \frac{1}{1-\beta^t}\left[\beta\, v_{t-1} + (1-\beta)\left(\frac{\partial\mathcal{L}}{\partial W}\right)^2\right]$$

- Proposed coefficient values: $\gamma = 0.9,\ \beta = 0.999$
- Norm factors loose influence for $t \gg 1$

D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv e-prints (2014)

Optimization of network parameters

- Learning strategies: methods for accelerating network training!

  4. **Adam** (adaptive moments): combines ideas of RMSprob and the momentum method

     - m_t: scales direction of next step in parameter space
     - v_t: adapts learning rates

$$\vec{u}_t = -\alpha \frac{m_t}{\sqrt{v_t} + \epsilon} \qquad \vec{\theta}_{t+1} = \vec{\theta}_t + \vec{u}_t$$

     - Adam is efficient & robust: good first choice as optimizer!

D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv e-prints (2014)

Summary

- Data must be appropriately preprocessed before inserted to neural network: especially ***scaling*** reduces high fluctuations in data
- During training, the data set is used multiple times – each time is called ***Epoch***
- Parameter optimization is done in smaller steps using only samples of the data set called ***minibatches***
- Weight coefficients are initialized using random numbers following a distribution, shoes variance depends on the number of nodes in a layer and the ***activation function***
- Common ***objective functions*** for regression are ***MAE***, ***MSE***, ***RMAE*** and for classification we use ***cross-entropy***
- With the help of the chain rule of partial derivatives (***backpropagation***), ***stochastic gradient descent*** minimizes ***objective function***
- ***Learning rate*** corresponds to the steps size of the optimization procedure
- Different optimization strategies can be chosen – during optimization the learning rate is dynamically adapt for efficiency and robust results