

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Machine Learning Basics</b>                               | <b>3</b> |
| 1.1      | Un-/Supervised Learning                                      | 3        |
| <b>2</b> | <b>Linear Models</b>   | <b>4</b> |
| 2.1      | Regression, Classification                                   | 4        |
| 2.2      | Obtaining the model  | 4        |
| 2.3      | Linear Regression  | 4        |
| 2.3.1    | Linear Model   | 4        |
| 2.3.2    | Loss function  | 4        |
| 2.3.3    | Optimization   | 4        |
| 2.4      | Logistic Regression  | 5        |
| 2.4.1    | Model  | 5        |
| 2.4.2    | Loss function  | 5        |
| 2.4.3    | Cost function  | 5        |
| 2.4.4    | Optimization   | 5        |
| <b>3</b> | <b>Computational Graphs</b>                                  | <b>5</b> |
| 3.1      | Graphical representation                                     | 5        |
| <b>4</b> | <b>Neural Network</b>  | <b>6</b> |
| 4.1      | Activation Functions   | 6        |
| 4.1.1    | Sigmoid  | 6        |
| 4.1.2    | Softmax  | 7        |
| 4.1.3    | Tanh (Hyperbolic Tanjant Function)                           | 7        |
| 4.1.4    | ReLU (Rectified Linear Units)                                | 7        |
| 4.1.5    | Leaky ReLU   | 8        |
| 4.1.6    | Parametric ReLU  | 8        |
| 4.1.7    | ELU  | 8        |
| 4.1.8    | Maxout   | 9        |
| 4.1.9    | Step Function  | 9        |
| 4.2      | Loss Function  | 9        |
| 4.2.1    | Description  | 9        |
| 4.2.2    | Parameters   | 9        |
| 4.2.3    | L1 Loss  | 9        |
| 4.2.4    | L2/MSE Loss  | 10       |
| 4.2.5    | Binary Cross Entropy   | 10       |
| 4.2.6    | Cross Entropy  | 10       |
| 4.2.7    | Hinge Loss (SVM Loss)  | 10       |
| 4.3      | Optimization Functions                                       | 10       |
| 4.3.1    | General Optimization   | 10       |
| 4.3.2    | Gradient Descent   | 10       |
| 4.3.3    | Stochastic Gradient Descent                                  | 11       |
| 4.3.4    | Gradient Descent with Momentum                               | 11       |
| 4.3.5    | Nesterov Momentum  | 12       |
| 4.3.6    | Root Mean Squared Prop (RMSProp)                             | 12       |
| 4.3.7    | Adaptive Momemt Esimation (Adam)                             | 12       |
| 4.3.8    | Newton's Method  | 13       |
| 4.3.9    | Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS and L-BFGS) | 13       |
| 4.3.10   | Gauss-Newton   | 13       |
| 4.3.11   | Levenberg  | 13       |
| 4.3.12   | Levenberg-Marquardt  | 14       |
| 4.4      | Learning rate  | 14       |
| 4.4.1    | Learning Rate Decay  | 14       |
| 4.4.2    | Training Schedule  | 14       |
| 4.5      | Regularization Techniques                                    | 14       |
| 4.5.1    | L1/L2 Regularization   | 14       |
| 4.5.2    | Weight Decay   | 15       |
| 4.5.3    | Early Stopping   | 15       |

|          |  |           |
|----------|--|-----------|
| 4.5.4    | Bagging and Ensemble Methods . . . . .       | 15        |
| 4.5.5    | Dropout . . . . .                            | 15        |
| 4.5.6    | Batch Normalization (BN) . . . . .           | 15        |
| 4.5.7    | Data Augmentation . . . . .                  | 16        |
| <b>5</b> | <b>Fully Connected Neural Network</b>        | <b>16</b> |
| 5.1      | Structure . . . . .                          | 16        |
| 5.2      | Number of weights . . . . .                  | 17        |
| 5.3      | Forward and Backward Pass . . . . .          | 17        |
| 5.3.1    | Forward Pass/ Forward Propagation . . . . .  | 17        |
| 5.4      | Backward Pass/Backward Propagation . . . . . | 17        |
| <b>6</b> | <b>Training</b>                              | <b>19</b> |
| 6.1      | Learning . . . . .                           | 19        |
| 6.2      | Dataset . . . . .                            | 19        |
| 6.3      | Weight initialization . . . . .              | 19        |
| 6.3.1    | Xavier Initialization . . . . .              | 19        |
| 6.4      | Errors . . . . .                             | 19        |
| 6.5      | Hyperparameters . . . . .                    | 19        |
| 6.5.1    | Hyperparameter Tuning Methods . . . . .      | 20        |
| 6.6      | Learning Curves . . . . .                    | 20        |
| 6.6.1    | Ideal Training . . . . .                     | 20        |
| 6.6.2    | Underfitting . . . . .                       | 20        |
| 6.6.3    | Overfitting . . . . .                        | 21        |
| 6.6.4    | Other Examples . . . . .                     | 21        |
| 6.7      | How To . . . . .                             | 23        |
| 6.7.1    | Network Architecture . . . . .               | 23        |
| 6.7.2    | Training samples . . . . .                   | 23        |
| 6.7.3    | Learning rate . . . . .                      | 23        |
| 6.7.4    | Timings . . . . .                            | 23        |
| 6.7.5    | Basic Recipe . . . . .                       | 24        |
| 6.7.6    | Bad Signs . . . . .                          | 24        |
| 6.7.7    | Good/Bad Practice . . . . .                  | 24        |

# 1 Machine Learning Basics

## 1.1 Un-/Supervised Learning

- Unsupervised Learning
  - No label or target class
  - Find out properties of the structure of the data
  - clustering (k-means, PCA, etc.)
- Supervised Learning
  - Labels or target classes
- Reinforcement Learning

## 2 Linear Models

### 2.1 Regression, Classification

- **Regression:** Predicts a continuous output value
- **Classification:** Predicts a discrete value
  - **Binary Classification:** Output either 0 or 1
  - **Multi-class classification:** Set of N classes

### 2.2 Obtaining the model

1. Estimating using current model
2. Calculating loss
3. Optimizing the model

### 2.3 Linear Regression

#### 2.3.1 Linear Model

- $i$ : index of current sample  
 $j$ : index of current weight  
 $d$ : input dimension/number of weights  
 $x_{ij}$ :  $i$ -th input data/feature of the  $j$ -th weight  
 $\theta_0$ : bias  
 $\theta_j$ : weights  
 $\hat{y}_i$ :  $i$ -th Prediction/Estimation (predicted label)

$$\hat{y}_i = \theta_0 + \sum_{j=1}^d x_{ij}\theta_j = \theta_0 + x_{i1}\theta_1 + \dots + x_{id}\theta_d$$

**Matrix Notation:**

$$\hat{y} = X\theta$$

#### 2.3.2 Loss function

Measures how good my estimation is and tells the optimization method how to make it better

##### 2.3.2.1 Linear Least Squares:

- $n$ : number of training samples  
 $y$ : Ground truth labels  
 $\hat{y}$ : Estimated labels

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

**Matrix Notation:**

$$J(\theta) = (X\theta - y)^T(X\theta - y) = (\hat{y} - y)^T(\hat{y} - y)$$

#### 2.3.3 Optimization

Changes the model in order to improve the loss function/estimation

$$\theta = (X^T X)^{-1} X^T y$$

## 2.4 Logistic Regression

### 2.4.1 Model

- $i$ : index of current sample
- $j$ : index of current weight
- $d$ : input dimension/number of weights
- $x_{ij}$ :  $i$ -th input data/feature of the  $j$ -th weight
- $\theta_j$ : model parameters
- $\hat{y}_i$ :  $i$ -th Prediction/Estimation (predicted label)

$$\hat{y}_i = \sigma(x_i \theta) = \sigma \left( \sum_{j=1}^d x_{ij} \theta_j \right)$$

with

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

### 2.4.2 Loss function

Measures how good my estimation is and tells the optimization method how to make it better

#### 2.4.2.1 Binary Cross-Entropy:

- $y$ : Ground truth labels
- $\hat{y}$ : Estimated labels

$$\mathcal{L}(\hat{y}_i, y_i) = y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

### 2.4.3 Cost function

- $n$ : number of labels

$$C(\theta) = -\frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}_i, y_i)$$

### 2.4.4 Optimization

Changes the model in order to improve the loss function/estimation

- No closed-form solution
- Make use of an iterative method e.g. Gradient Descent

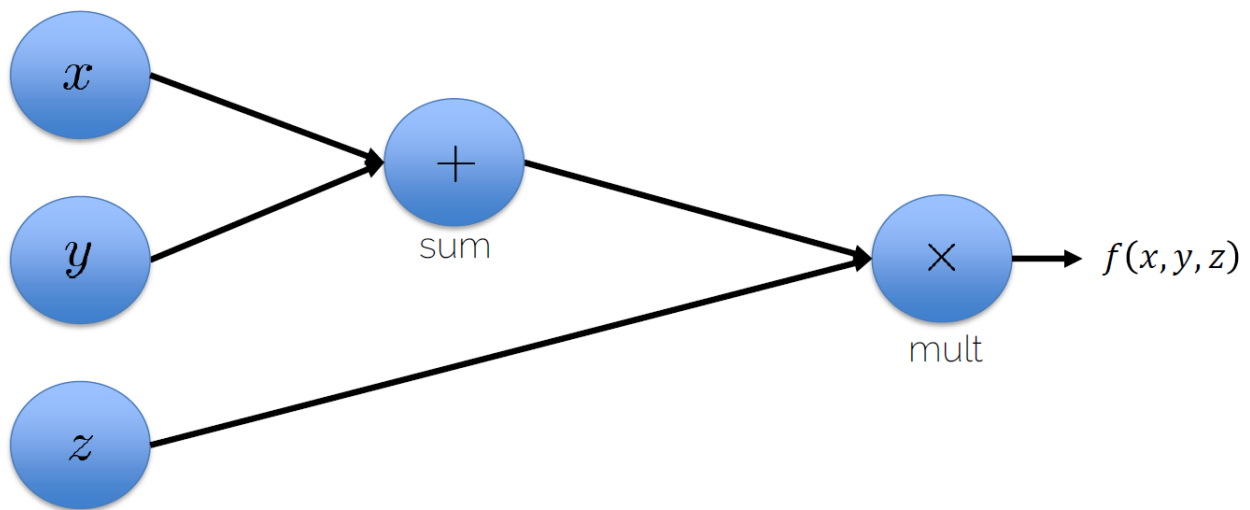
## 3 Computational Graphs

- Directional graph
- Matrix operations are represented as compute graphs
- Vertex nodes are variables or operators like  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\log()$ ,  $\exp()$ ,  $\dots$
- Directional edges show flow of inputs to vertices
- Neural network can be represented as computational graph

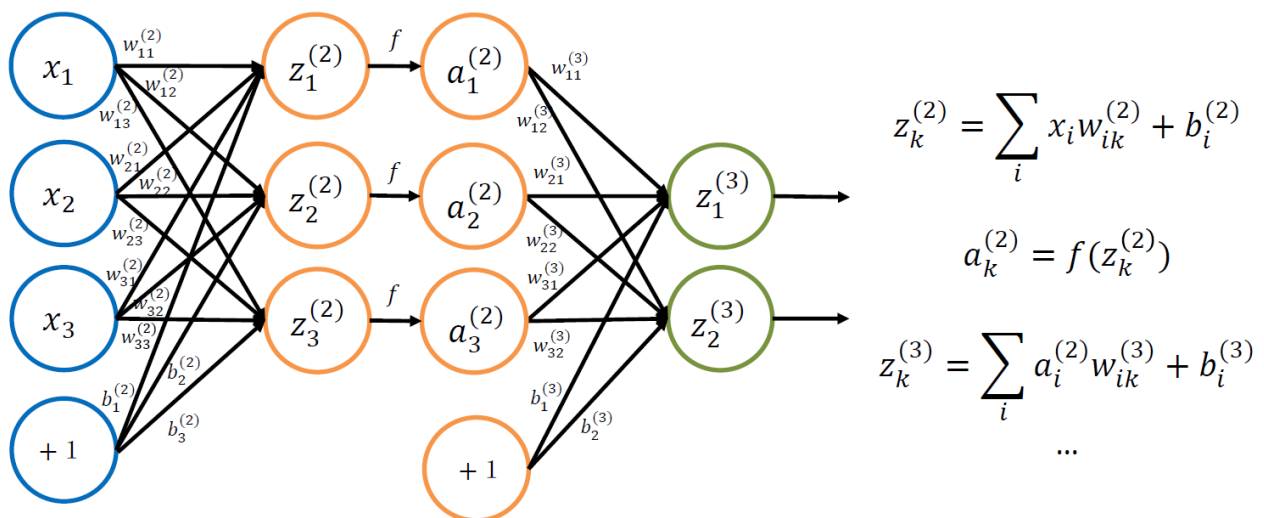
### 3.1 Graphical representation

Example

$$f(x, y, z) = (x + y) \cdot z$$



### Neural Network as Computational Graph



## 4 Neural Network

### 4.1 Activation Functions

#### Description

##### Parameters:

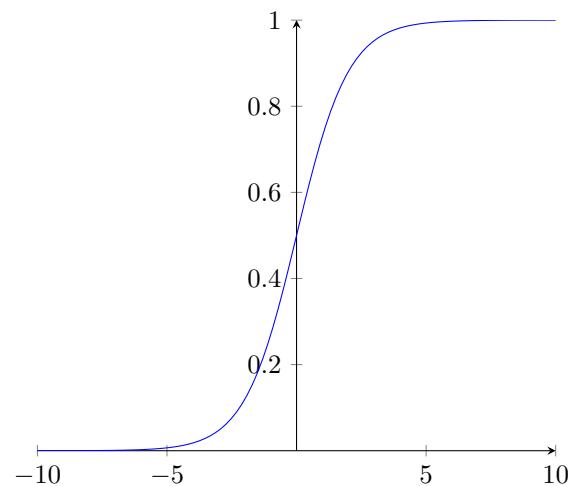
- $\hat{y}$ : Prediction
- $\hat{y}_j$ : Prediction of  $j$ -th output ( $j$ -th Neuron of output layer)
- $h$ : Outputs of hidden layer
- $s$ : Output of layer (before activation)
- $s_j$ : Output of the  $j$ -th neuron in layer (before activation)
- $C$ : Number of classes (number of neurons in output layer)

#### 4.1.1 Sigmoid

- Used for Binary Classification to output a probability matching first or second class
- **Output:**  $\hat{y} = (0, 1)$
- If used as activation function in hidden layer (typically it is not)
  - Output is always positive

- Saturates for high positive or low negative values → kills the gradient flow

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



#### 4.1.2 Softmax

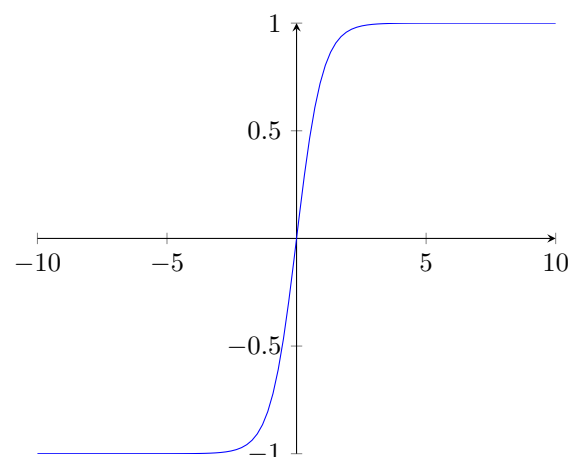
- Used for Multiclass Classification to output a probability matching the  $i$ -th class
- **Output:**  $\hat{y}_j = (0, 1)$

$$\hat{y}_j = \frac{e^{s_j}}{\sum_{k=1, k \neq j}^C e^{s_k}}$$

#### 4.1.3 Tanh (Hyperbolic Tanjant Function)

- Saturates for high positive or low negative values → kills the gradient flow

$$h = \tanh(s)$$

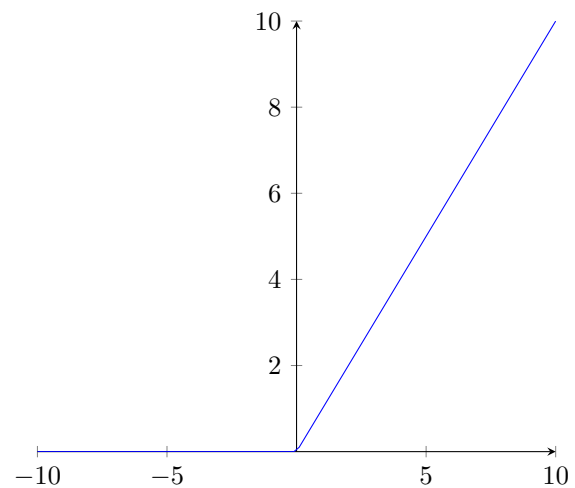


#### 4.1.4 ReLU (Rectified Linear Units)

- Standard choice for activation function
- + Does not saturate
- + Large and consistent gradients

- + Fast convergence
- Dead ReLU if output is zero
- Initialization of ReLU neurons with slightly positive biases (e.g. 0.01) → Likely to stay active for most inputs

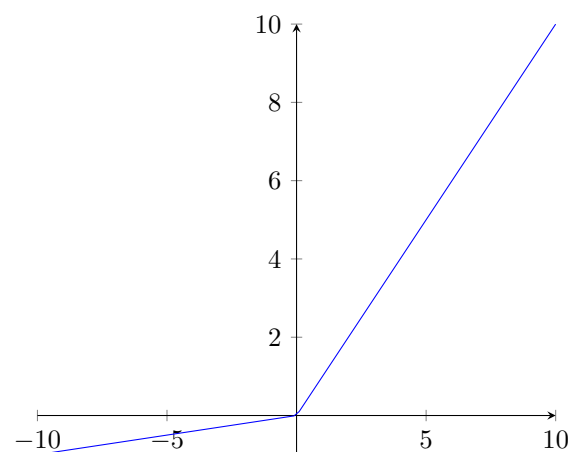
$$h = \max(0, s)$$



#### 4.1.5 Leaky ReLU

- + Does not die

$$h = \max(0.1s, s)$$



#### 4.1.6 Parametric ReLU

- + Does not die

$$h = \max(\alpha s, s)$$

#### 4.1.7 ELU

$$f(s) = \begin{cases} s & \text{if } s > 0 \\ \alpha(e^s - 1) & \text{if } s \leq 0 \end{cases}$$



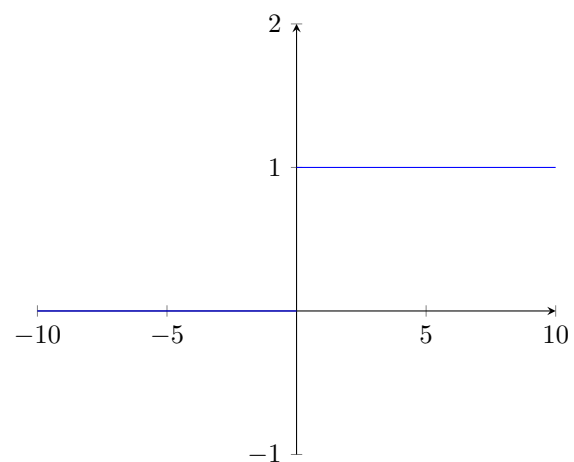
### 4.1.8 Maxout

- + Generalization of ReLUs
- + Linear regimes
- + Does not die
- + Does not saturate
- Increases the number of parameters

$$h = \max(w_1^T s + b_1, w_2^T s + b_2)$$

### 4.1.9 Step Function

$$h = \begin{cases} 0 & \text{if } s < 0 \\ 1 & \text{if } s \geq 0 \end{cases}$$



## 4.2 Loss Function

### 4.2.1 Description

A function to measure the goodness of the predictions

- Large loss  $\implies$  bad predictions
- Goal: Minimize the loss  $\iff$  Find better predictions
- Choice of the loss function depends on the concrete problem or the distribution of the target variable

### 4.2.2 Parameters

- $y_i$ : Ground truth of  $i$ -th sample
- $\hat{y}_i$ : Prediction of  $i$ -th sample
- $n$ : number of training samples
- $k$ : number of classes
- $\hat{y}_{i,gt}$ : Prediction of ground truth class of  $i$ -th sample (where  $y_{ij} = 1$ )

### 4.2.3 L1 Loss

- Sum of absolute differences
- Robust (cost of outliers is linear)
- Costly to optimize
- Optimum is median

$$\mathcal{L}(y, \hat{y}; \theta) = \frac{1}{n} \sum_i^n \|y_i - \hat{y}_i\|_1$$

#### 4.2.4 L2/MSE Loss

- Sum of squared differences
- Prone to outliers
- Compute-efficient optimization
- Optimum is mean

$$\mathcal{L}(y, \hat{y}; \theta) = \frac{1}{n} \sum_i^n \|y_i - \hat{y}_i\|_2^2$$

#### 4.2.5 Binary Cross Entropy

$$\mathcal{L}(y, \hat{y}; \theta) = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i))$$

#### 4.2.6 Cross Entropy

- Loss is typically always  $> 0 \rightarrow$  Always improvement

$$\mathcal{L}(y, \hat{y}; \theta) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k (y_{ij} \log \hat{y}_{ij})$$

#### 4.2.7 Hinge Loss (SVM Loss)

- Loss can become  $= 0 \rightarrow$  Saturation

$$\mathcal{L}(y, \hat{y}; \theta) = \frac{1}{n} \sum_{i=1}^n \sum_{\substack{j=1, \\ j \neq gt}}^k \max(0, \hat{y}_{ij} - \hat{y}_{i,gt} + 1)$$

### 4.3 Optimization Functions

#### 4.3.1 General Optimization

- **Goal:**  $\theta^* = \arg \min f(\theta, x, y)$
- **Linear Systems ( $Ax = b$ )**
  - LU, QU, Cholesky, Jacobi, Gauss-Seidel, CG, PCG, ...
- **Non-linear systems** (Gradient based methods):
  - First order methods:
    - \* Gradient Descent, SGD, SGD with Momentum, RMSProp, Adam (Standard)
  - Second order methods (faster than first order methods, but only for full batch updates):
    - \* Newton, Gauss-Newton, Levenberg-Marquardt, (L)BFGS

#### 4.3.2 Gradient Descent

- Finds local minimum
- Does not guarantee to find global optimum
- Does gradient steps in direction of negative gradient
- Requires a lot of memory  $\implies$  extremely expensive

Parameters:

$f(\theta, x_{1..n}, y_{1..n})$ : Function describing the neural network (including loss function)

$x_{1..n}$ : Input vectors for all  $n$  training samples

$y_{1..n}$ : Ground truth for all  $n$  training samples

$\theta^k = \{W, b\}$ : Model Parameters at step  $k$

$\alpha$ : Learning rate

Gradient Step:

$$\theta^{k+1} = \theta^k - \alpha \nabla_{\theta} f(\theta^k, x_{1..n}, y_{1..n})$$

### 4.3.3 Stochastic Gradient Descent

- Split training set into several minibatches
- Minibatch size:
  - Is a hyperparameter
  - Is typically a power of 2
  - Smaller batch size  $\implies$  Greater variance in the gradients
  - Is mostly limited by GPU memory
- Epoch: Complete pass through training set
- Cannot independently scale directions
- Need to have conservative min learning rate to avoid divergence
- Is slower than necessary

Parameters:

|                                   |  |
|-----------------------------------|--|
| $n$ :                             | Number of total training samples                                 |
| $m$ :                             | Minibatch size (number of training samples per minibatch)        |
| $n/m$ :                           | Number of minibatches  |
| $f(\theta, x_{1..m}, y_{1..m})$ : | Function describing the neural network (including loss function) |
| $x_{1..m}$ :                      | Input vectors for one minibatch                                  |
| $y_{1..m}$ :                      | Ground truth for one minibatch                                   |
| $\theta^k = \{W, b\}$ :           | Model Parameters at step $k$                                     |
| $k$ :                             | iteration in current epoch                                       |
| $\alpha$ :                        | Learning rate  |

Gradient Step:

$$\theta^{k+1} = \theta^k - \alpha \nabla_{\theta} f(\theta^k, x_{1..m}, y_{1..m})$$

#### 4.3.3.1 Convergence of Stochastic Gradient Descent

$f(\theta, x, y)$  converges to a local (global) minimum if:

1.  $\alpha_n \geq 0, \forall n \geq 0$
2.  $\sum_{n=1}^{\infty} \alpha_n = \infty$
3.  $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$
4.  $f(\theta, x, y)$  is strictly convex

where  $\alpha_1, \dots, \alpha_n$  is a sequence of positive step-sizes

### 4.3.4 Gradient Descent with Momentum

- Step will be largest when a sequence of gradients all point to the same direction

Parameters:

|                         |  |
|-------------------------|--|
| $f(\theta, x, y)$ :     | Function describing the neural network (including loss function) |
| $x$ :                   | Input vectors  |
| $y$ :                   | Ground truth   |
| $\theta^k = \{W, b\}$ : | Model Parameters at step $k$                                     |
| $\alpha$ :              | Learning rate  |
| $\beta$ :               | Accumulation rate (friction, momentum), default: 0.9             |
| $v^k$ :                 | velocity at step $k$   |

Gradient step:

$$v^{k+1} = \beta \cdot v^k - \alpha \cdot \nabla_{\theta} f(\theta^k, x, y)$$

$$\theta^{k+1} = \theta^k + v^{k+1}$$

### 4.3.5 Nesterov Momentum

- Look-ahead momentum
- Steps:
  1. Make a big jump in the direction of the previous accumulated gradient
  2. Measure the gradient where you end up
  3. Make a correction

Parameters:

|                         |  |
|-------------------------|--|
| $f(\theta, x, y)$ :     | Function describing the neural network (including loss function) |
| $x$ :                   | Input vectors  |
| $y$ :                   | Ground truth   |
| $\theta^k = \{W, b\}$ : | Model Parameters at step $k$                                     |
| $\alpha$ :              | Learning rate  |
| $\beta$ :               | Accumulation rate (friction, momentum), default: 0.9             |
| $v^k$ :                 | velocity at step $k$   |

Gradient step:

$$\begin{aligned}\tilde{\theta}^{k+1} &= \theta^k + \beta \cdot v^k \\ v^{k+1} &= \beta \cdot v^k - \alpha \cdot \nabla_{\theta} f(\tilde{\theta}^{k+1}, x, y) \\ \theta^{k+1} &= \theta^k + v^{k+1}\end{aligned}$$

### 4.3.6 Root Mean Squared Prop (RMSProp)

- Divides the learning rate by an exponentially-decaying average of squared gradients
- Damps the oscillations for high-variance directions
- + Can increase learning rate because it is less likely to diverge → Speeds up learning speed

Parameters:

|                         |  |
|-------------------------|--|
| $f(\theta, x, y)$ :     | Function describing the neural network (including loss function) |
| $x$ :                   | Input vectors  |
| $y$ :                   | Ground truth   |
| $\theta^k = \{W, b\}$ : | Model Parameters at step $k$                                     |
| $\alpha$ :              | Learning rate  |
| $\beta$ :               | Accumulation rate (friction, momentum), default: 0.9             |
| $\epsilon$ :            | Prevents division by zero, default: $10^{-8}$                    |
| $s^k$ :                 | Second momentum (uncentered variance of gradients)               |

Gradient step:

$$\begin{aligned}s^{k+1} &= \beta \cdot s^k + (1 - \beta)(\nabla_{\theta} f(\theta^k, x, y) \circ \nabla_{\theta} f(\theta^k, x, y)) \\ \theta^{k+1} &= \theta^k - \alpha \cdot \frac{\nabla_{\theta} f(\theta^k, x, y)}{\sqrt{s^{k+1}} + \epsilon}\end{aligned}$$

where  $a \circ b$  is an element-wise multiplication

### 4.3.7 Adaptive Moment Estimation (Adam)

- Combines Momentum and RMSProp
- Combines first and second order momentum

Parameters:

|                         |  |
|-------------------------|--|
| $f(\theta, x, y)$ :     | Function describing the neural network (including loss function) |
| $x$ :                   | Input vectors  |
| $y$ :                   | Ground truth   |
| $\theta^k = \{W, b\}$ : | Model Parameters at step $k$                                     |
| $\alpha$ :              | Learning rate  |
| $\beta_1$ :             | Accumulation rate 1, default: 0.9                                |
| $\beta_2$ :             | Accumulation rate 2, default: 0.999                              |
| $\epsilon$ :            | Prevents division by zero, default: $10^{-8}$                    |
| $s^k$ :                 | Second momentum (uncentered variance of gradients)               |

Gradient step:

$$\begin{aligned}\hat{m}^{k+1} &= \frac{\beta_1 \cdot m^k + (1 - \beta_1) \cdot \nabla_{\theta} f(\theta^k, x, y)}{1 - \beta_1^{k+1}} \\ \hat{v}^{k+1} &= \frac{\beta_2 \cdot v^k + (1 - \beta_2)(\nabla_{\theta} f(\theta^k, x, y) \circ \nabla_{\theta} f(\theta^k, x, y))}{1 - \beta_2^{k+1}} \\ \theta^{k+1} &= \theta^k - \alpha \cdot \frac{\hat{m}^{k+1}}{\sqrt{\hat{v}^{k+1} + \epsilon}}\end{aligned}$$

where  $m^0 = v^0 = 0$

#### 4.3.8 Newton's Method

- Computation complexity of inversion per iteration:  $\mathcal{O}(k^3)$

Parameters:

$f(\theta)$ : Function describing the neural network (including loss function)  
 $\theta = \{W, b\}$ : Model Parameters  
 $\nabla_{\theta} f(\theta)$ : Gradient (first derivative)  
 $H(\theta)$ : Hessian matrix (second derivative)

Approximate the function by a second-order Taylor series expansion

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0)^T \nabla_{\theta} f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$$

Update step:

$$\theta^* = \theta_0 - H^{-1} \nabla_{\theta} f(\theta)$$

#### 4.3.9 Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS and L-BFGS)

- Belongs to the family of quasi-Newton methods
- Have an approximation of the inverse of the Hessian
- Computation complexity of inversion per iteration:
  - BFGS:  $\mathcal{O}(n^2)$
  - L-BFGS:  $\mathcal{O}(n)$

Update step:

$$\theta^* = \theta_0 - H^{-1} \nabla_{\theta} f(\theta)$$

#### 4.3.10 Gauss-Newton

- True 2nd derivatives are often hard to obtain → Approximate

Parameters:

$\theta = \{W, b\}$ : Model Parameters  
 $\nabla_{\theta} f(\theta)$ : Gradient (first derivative)  
 $\mathcal{J}(\theta)$ : Jacobian matrix

$$H(\theta) \approx 2\mathcal{J}^T(\theta)\mathcal{J}(\theta)$$

Linear equation:

$$2(\mathcal{J}^T(\theta_k)\mathcal{J}(\theta_k)) \cdot (\theta_k - \theta_{k+1}) = \nabla_{\theta} f(\theta)$$

#### 4.3.11 Levenberg

- Damped version of Gauss-Newton
- Damping factor is adjusted in each iteration, so that:  $f(\theta_k) > f(\theta_{k+1})$

Parameters:

$\theta = \{W, b\}$ : Model Parameters  
 $\nabla_{\theta} f(\theta)$ : Gradient (first derivative)  
 $\mathcal{J}(\theta)$ : Jacobian matrix  
 $\lambda$ : Damping factor

Linear equation:

$$(\mathcal{J}^T(\theta_k)\mathcal{J}(\theta_k) + \lambda I) \cdot (\theta_k - \theta_{k+1}) = \nabla_{\theta} f(\theta)$$

### 4.3.12 Levenberg-Marquardt

- Avoids slow convergence in components with a small gradient

Parameters:

$\theta = \{W, b\}$ : Model Parameters  
 $\nabla_{\theta} f(\theta)$ : Gradient (first derivative)  
 $\mathcal{J}(\theta)$ : Jacobian matrix  
 $\lambda$ : Damping factor

Linear equation:

$$(\mathcal{J}^T(\theta_k)\mathcal{J}(\theta_k) + \lambda \cdot \text{diag}(\mathcal{J}^T(\theta_k)\mathcal{J}(\theta_k))) \cdot (\theta_k - \theta_{k+1}) = \nabla_{\theta} f(\theta)$$

## 4.4 Learning rate

- Goal: High learning rate in the beginning, then low learning rate

### 4.4.1 Learning Rate Decay

$\alpha_0$ : Initial learning rate (e.g. 0.1)  
 $t$ : factor by which the learning rate is decayed  
 $epoch$ : epoch of current run

Learning rate decays:

$$\alpha = \frac{1}{1 + t \cdot epoch} \cdot \alpha_0, \quad \text{default: } t = 0.1$$

Step decay:

$$\alpha = \alpha - t \cdot \alpha, \quad \text{default: } t = 0.5$$

Exponential decay:

$$\alpha = t^{epoch} \cdot \alpha_0, \quad t < 1.0$$

$$\alpha = \frac{t}{\sqrt{epoch}} \cdot \alpha_0$$

### 4.4.2 Training Schedule

- Manually set learning rate every n epochs

## 4.5 Regularization Techniques

- Increasing training error
- Lower validation error

### 4.5.1 L1/L2 Regularization

$L$ : Loss  
 $\mathcal{L}(y, \hat{y}, \theta)$ : Loss function (without generalization)  
 $\lambda$ : Regularization rate  
 $\theta = \{W, b\}$ : Model parameters

Add regularization term to loss function:

$$L = \mathcal{L}(y, \hat{y}, \theta) + \lambda R(\theta)$$

#### L1 Regularization

Enforces sparsity

$$R(\theta) = \sum_{i=1}^n |\theta_i|$$

#### L2 Regularization

Enforces that the weights have similar values

$$R(\theta) = \sum_{i=1}^n \theta_i^2$$

### 4.5.2 Weight Decay

Parameters:

|                                   |  |
|-----------------------------------|--|
| $f(\theta, x_{1..n}, y_{1..n})$ : | Function describing the neural network (including loss function) |
| $x_{1..n}$ :                      | Input vectors for all $n$ training samples                       |
| $y_{1..n}$ :                      | Ground truth for all $n$ training samples                        |
| $\theta^k = \{W, b\}$ :           | Model Parameters at step $k$                                     |
| $\alpha$ :                        | Learning rate  |
| $\lambda$ :                       | Regularization rate  |
| $R(\theta^k)$ :                   | L2 Regularization  |

Add regularization term to Gradient step:

$$\theta^{k+1} = \theta^k - \alpha \nabla_{\theta} f(\theta^k, x_{1..n}, y_{1..n}) - \lambda R(\theta^k)$$

### 4.5.3 Early Stopping

Stop training as soon as the model begins overfitting

### 4.5.4 Bagging and Ensemble Methods

- Train multiple models and average their results
- Use different optimization functions, loss functions, ... for each model
- + If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size

### 4.5.5 Dropout

- Disable a random set of neurons (typically 50%)
  - Intuition: Use half of the network:
    - Redundant representations
    - Base your scores on more features
    - Reducing co-adaptations between neurons
- Testing:
  - Disable dropout
  - Multiply with dropout probability:  $s'_j = s_j \cdot p$
- Reduces the effective capacity of a model → More training time

### 4.5.6 Batch Normalization (BN)

- Goal: Activations do not die out
- Normalizes the mean and the variance of the inputs to the activation functions
- Is applied after Fully Connected (or Convolutional) Layers and before non-linear activation functions
- + Very deep nets are much easier to train → more stable gradients
  - Can be undone by the network with:  $\gamma^{(k)} = \sqrt{\text{Var}[s^{(k)}]}$ ,  $\beta^{(k)} = \mathbb{E}[s^{(k)}]$
  - All biases of the layers before BN can be set to zero, since they will be canceled out by BN anyway
- Testing:
  - Compute mean  $\mu_{test}$  and variance  $\sigma_{test}^2$  by running an exponentially weighted averages across training minibatches:
 
$$\text{Var}_{running} = \beta_m \cdot \text{Var}_{running} + (1 - \beta_m) \cdot \text{Var}_{minibatch}$$

$$\mu_{running} = \beta_m \cdot \mu_{running} + (1 - \beta_m) \cdot \mu_{minibatch}$$

**Parameters:**

- $s^k$ : Output of the Fully Connected or Convolutional Layer before the Batch Normalization Layer  
 $\mathbb{E}[s^k]$ : Mean of the mini-batch examples over feature k  
 $Var[s^k]$ : Variance of the mini-batch examples over feature k  
 $\gamma^{(k)}, \beta^{(k)}$ : Parameters optimized during Backpropagation

1. Normalize:

$$\hat{s}^{(k)} = \frac{s^{(k)} - \mathbb{E}[s^{(k)}]}{\sqrt{Var[s^{(k)}]}}$$

2. Allow the network to change the range:

$$s'^{(k)} = \gamma^{(k)} \hat{s}^{(k)} + \beta^{(k)}$$

**4.5.7 Other Normalizations**

- Layer Norm
- Instance Norm
- Group Norm

**4.5.8 Data Augmentation**

- Classifier has to be invariant to a wide variety of transformations → Synthesize data simulating plausible transformations
  - Training: Random Crops, Testing: Fixed Set of Crops
  - Use same data augmentation when comparing two networks
  - Consider data augmentation a part of your network design
- Augmentations:
  - Flip
  - Crop
  - Brightness and contrast changes
  - ...

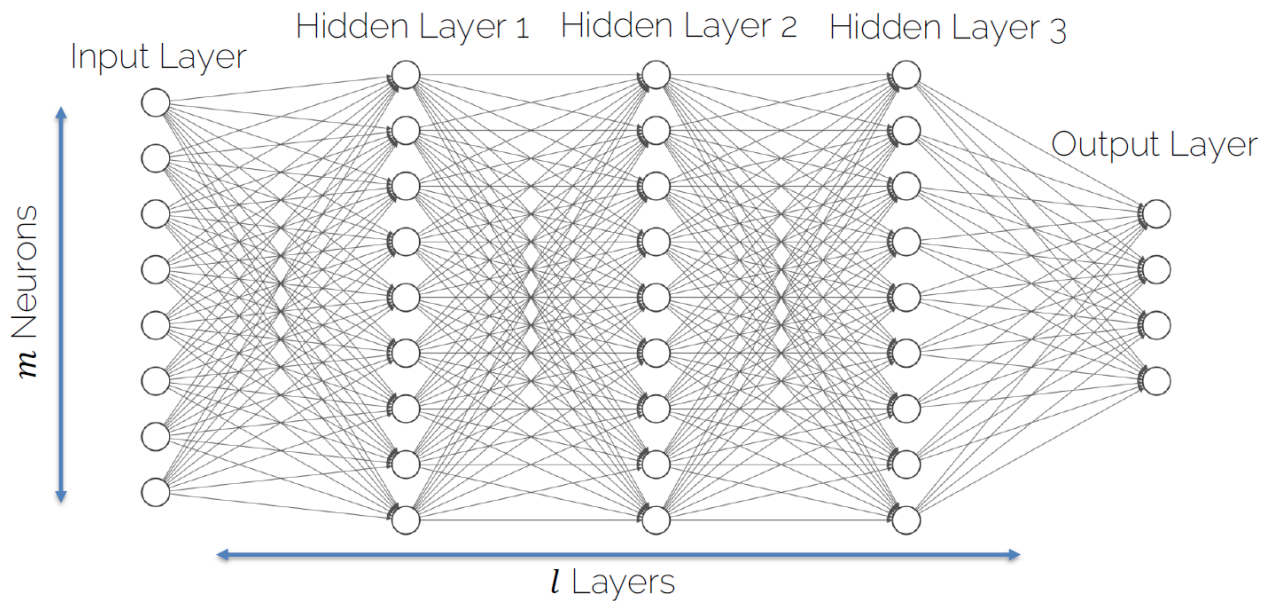
## 5 Fully Connected Neural Network

**5.1 Structure****Parameters:**

- $x_k$ : Input variables  
 $\theta = \{W, b\}$ : Model parameters  
 $w_{i,j,k}$ : Network weights  
 $b_{i,j}$ : Network biases  
     *i*: Index of layer  
     *j*: Index of neuron in layer (neuron of next layer)  
     *k*: Index of weight in neuron (neuron of previous layer)  
*l*: Depth: number of layers (All hidden and the output layer - no input layer)  
*m*: Width: number of neurons in layer (Can be different for each layer)  
*n*: Number of weights in neuron  
 $\hat{y}_i$ : Computed output  
 $y_i$ : Ground truth targets  
 $\mathcal{L}(y, \hat{y}, \theta)$ : Loss function

**Graphical Representation**





### Mathematical Representation

$$\begin{aligned}
 L &= \mathcal{L}(y_j, \hat{y}_j, \theta) \\
 \hat{y}_j &= a(s_{l,j}) \\
 h_{i,j} &= a(s_{i,j}) \\
 s_{i,j} &= b_{i,j} + \sum_{k=1}^m h_{i-1,k} \cdot w_{i,j,k} \\
 s_{1,j} &= b_{1,j} + \sum_{k=1}^m x_k \cdot w_{1,j,k}
 \end{aligned}$$

## 5.2 Number of weights

$l$ : Depth: number of layers

$m_i$ : Width: number of neurons in layer  $i$  (Here: layer 0 is the input layer)

Number of weights is defined as:

$$\sum_{i=1}^l m_i \cdot m_{i-1} + m_i$$

## 5.3 Forward and Backward Pass

### 5.3.1 Forward Pass/ Forward Propagation

Use formulas to calculate loss:

$$s_{1,1} = b_{1,1} + \sum_{k=1}^m x_k \cdot w_{1,1,k} \quad \dots \quad L = \mathcal{L}(y_j, \hat{y}_j)$$

## 5.4 Backward Pass/Backward Propagation

Weights of last layer:

$$\frac{\partial L}{\partial w_{l,j,k}} = \frac{\partial L}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial s_{l,j}} \cdot \frac{\partial s_{l,j}}{\partial w_{l,j,k}}$$

Weights of second last layer:

$$\frac{\partial L}{\partial w_{l-1,j,k}} = \sum_{o=1}^m \frac{\partial L}{\partial \hat{y}_o} \cdot \frac{\partial \hat{y}_o}{\partial s_{l,o}} \cdot \frac{\partial s_{l,o}}{\partial h_{l-1,j}} \cdot \frac{\partial h_{l-1,j}}{\partial s_{l-1,j}} \cdot \frac{\partial s_{l-1,j}}{\partial w_{l-1,j,k}}$$

**General:**

$$\frac{\partial L}{\partial w_{i,j,k}} = \sum_{o_1=1}^m \dots \sum_{o_p=1}^m \frac{\partial L}{\partial \hat{y}_{o_1}} \cdot \frac{\partial \hat{y}_{o_1}}{\partial s_{l,o_1}} \cdot \frac{\partial s_{l,o_1}}{\partial h_{l-1,o_2}} \cdot \frac{\partial h_{l-1,o_2}}{\partial s_{l-1,o_2}} \cdot \dots \cdot \frac{\partial s_{i+1,o_p}}{\partial h_{i,j}} \cdot \frac{\partial h_{i,j}}{\partial s_{i,j}} \cdot \frac{\partial s_{i,j}}{\partial w_{i,j,k}}$$

## 6 Training

### 6.1 Learning

- Learning means generalization to unknown dataset, i.e. train on known dataset, test with optimized parameters on unknown dataset

### 6.2 Dataset

- Split dataset into
  - Training data (e.g. 60%, 80%) - Used to train the model
  - Validation data (e.g. 20%, 10%) - Validate the current model to find the best hyperparameters
  - Test data (e.g. 20%, 10%) - Is only used once in the end

### 6.3 Weight initialization

Bad choice:

- All weights = 0 → No symmetry breaking
- Small Random Numbers → Output becomes zero using tanh as activation function → Vanishing gradient
- Big Random Numbers → Output saturates to -1 and 1 using tanh as activation function → Vanishing gradient

#### 6.3.1 Xavier Initialization

- Gaussian with zero mean and  $\text{var}(w) = \frac{1}{n}$  ( $n$ : number of weight per neuron)
- For ReLU:  $\text{Var}(w) = \frac{2}{n}$

### 6.4 Errors

- Ground truth error
  - Faults in dataset, e.g. wrong classification of sample image
  - Underfitting
- Training set error
  - Underfitting
- Validation/test set error
  - Overfitting

### 6.5 Hyperparameters

- Hyperparameters = Learning Setup + Optimization, i.e.,
  - Network architecture (number of layers, number of weights, ...)
  - Number of iterations
  - Learning rate(s)
  - Regularization
  - Batch size
  - ...

### 6.5.1 Hyperparameter Tuning Methods

- Manual search:
  - Find out the optimal hyperparameters manually
  - most common
- Grid search:
  - Define ranges for all parameters spaces and select points
  - Iterates over all possible configurations
- Random search:
  - Like grid search but one picks points at random in the predefined ranges

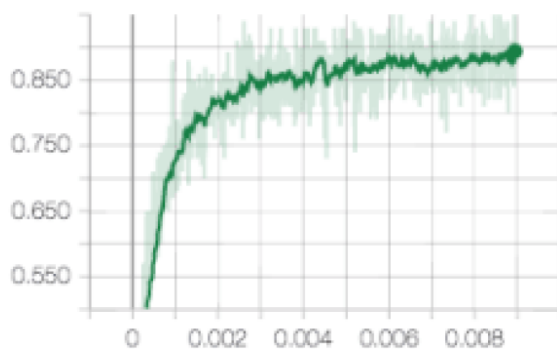
## 6.6 Learning Curves

### 6.6.1 Ideal Training

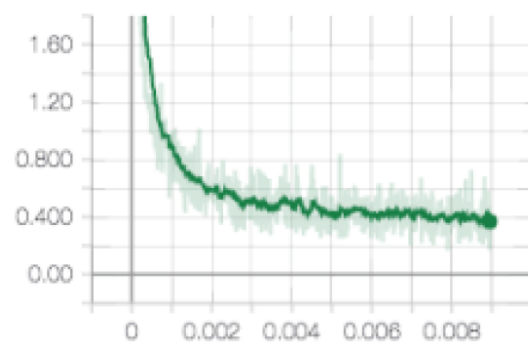
- Small gap between training and validation loss
- Training and validation loss go down at the same rate (stable without fluctuations)

Example:

- Accuracy



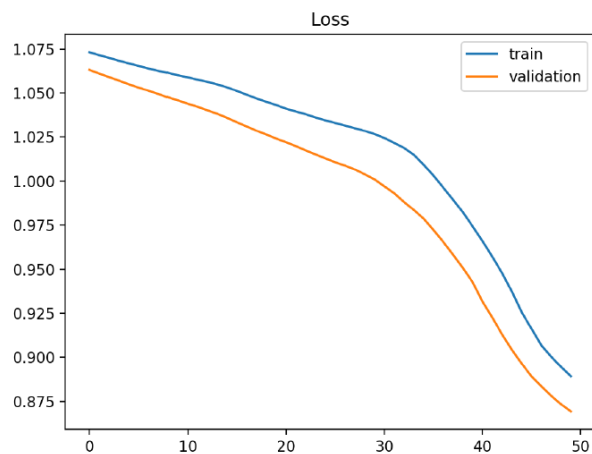
- Loss



### 6.6.2 Underfitting

- Training and validation losses decreases even at the end of training
- Reasons:
  - Model is still learning

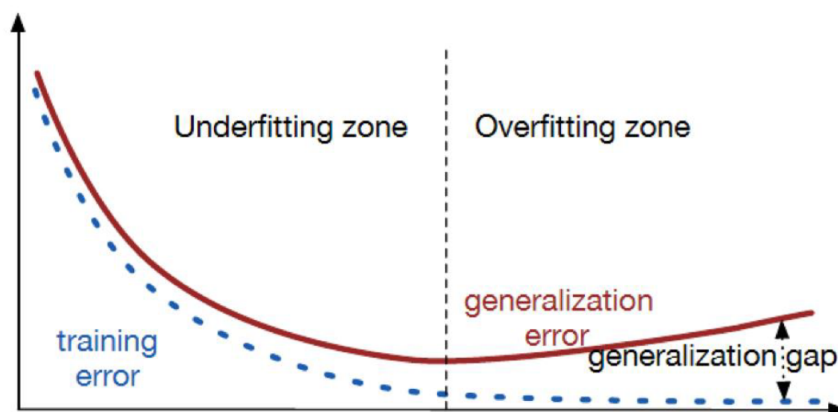
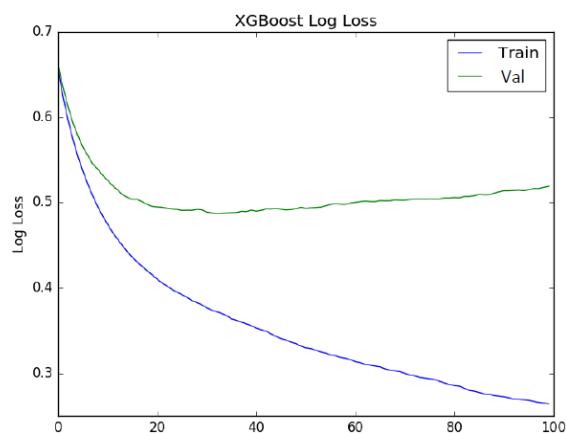
Example:



### 6.6.3 Overfitting

- Training loss decreases and validation loss increases
- Reasons:
  - Model is memorizing the training samples instead of generalizing

Example:

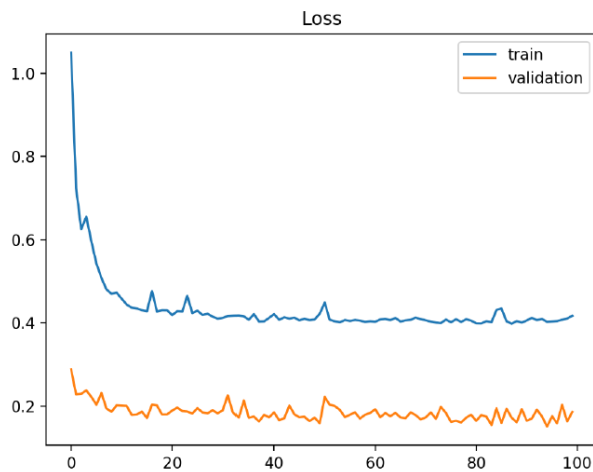


### 6.6.4 Other Examples

Validation set easier than training set:

- Validation loss is lower than training loss
- Reasons:
  - Validation set is easier than the training set
  - Bug in the implementation

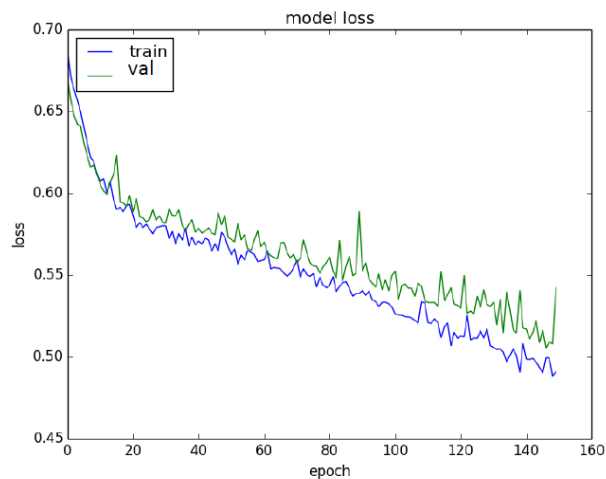
Example:



### Learning rate to low:

- Loss curves decrease almost linearly
- Reasons:
  - The initial Learning rate is too low

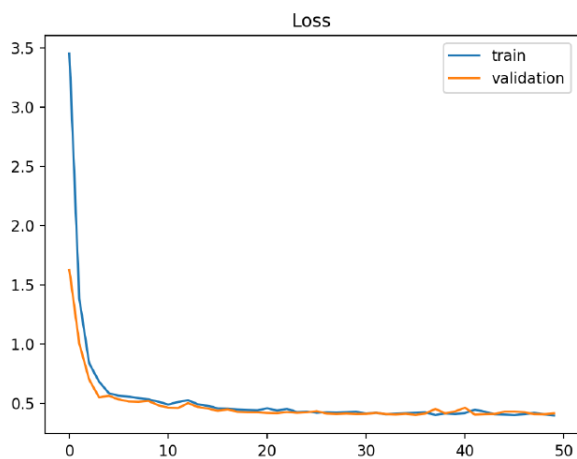
Example:



### Learning rate to high:

- Loss curves decrease very quickly at the beginning and then remain on plateaus
- Reasons:
  - Learning rate is too big
  - Inconsistent dataset

Example:



## 6.7 How To

### 6.7.1 Network Architecture

- Start with the simplest network possible

### 6.7.2 Training samples

1. Start with a single training sample
  - Check if output is correct
  - Should overfit
  - Train accuracy should be 100%
2. Increase to handful of samples (e.g., 4)
  - Check if input is handled correctly
3. Move to more samples
  - 5, 10, 100, 1000, ...
  - At some point, you should see generalization

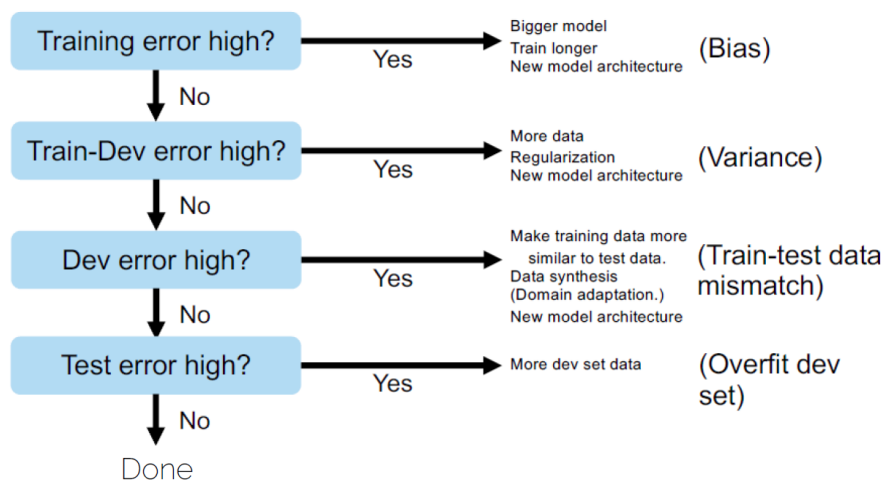
### 6.7.3 Learning rate

- Find a learning rate that makes the loss drop significantly within 100 iterations
- Good learning rates to try:  $1e-1, 1e-2, 1e-3, 1e-4$
- Good weight decay to try:  $1e-4, 1e-5, 0$
- Use Grid/Random search

### 6.7.4 Timings

- Measure how long each iteration takes (should be  $< 500ms$ )
- Look for bottlenecks (e.g. Dataloading, Backpropagation)
- Estimate total time

### 6.7.5 Basic Recipe



### 6.7.6 Bad Signs

- Training error not going down
- Validation error not going down
- Performance on validation better than on training set
- Tests on train set different than during training

### 6.7.7 Good/Bad Practice

#### Good Practice:

- Use train/validation/test curves
  - Evaluation needs to be consistent
  - Numbers need to be comparable
- Only make one change at a time

#### Bad Practice/Common Mistakes

- Using single batch it did not overfit
- Forgot to toggle train/eval mode for network
- Forgot to call `.zero_grad()` (in PyTorch) before calling `.backward()`
- Passed softmaxed outputs to a loss function that expects raw logits
- Training set contains test data
- Debug algorithm on test data



## Notes

This is a summary of the lecture Introduction to Deep Learning of the Technical University Munich. This lecture was presented by Nießner M. in the summer semester 2020. This summary was created by Gaida B. All provided information is without guarantee.