Lab course

Image Analysis I

ST 2023

Hubert Kanyamahanga
(kanyamahanga@ipi.uni-hannover.de)

Lab 2: Discriminative Probabilistic Models

Logistic Regression & Pytorch Basics

Content of the 2nd Lab

Goal:

- Application of Logistic regression for **classification** purposes
- Introduction to Neural Networks
- Fundamental concepts about how **Pytorch** works

Tasks:

1. Implement and train Logistic Regression classifiers by
   - ✓ first, using Scikit-learn and then Pytorch Frameworks
2. Implement and train Neural Networks
   - ✓ apply them to synthetic datasets and existing benchmark
3. Discussion and evaluation (**Visually** and **Quantitively**)

Recall from the 1st Lab

- **Generative classifiers:**
  - Wanted to compute $p(C|\mathbf{x})$ with the help of the theorem of Bayes $p(C\,|\,\mathbf{x}) = \dfrac{p(\mathbf{x}\,|\,C)\cdot p(C)}{p(\mathbf{x})}$
  - Determine the parameters of the likelihood $p(\mathbf{x}|C)$ (training)
  - Determine the prior $p(C)$
  - '**Generative**': Generate synthetic data sets by sampling from the joint distribution

$$\mathrm{p}(\mathbf{x},\mathrm{C}) = p(\mathbf{x}|C) \cdot p(C)$$

Now in the 2nd Lab

- **Discriminative classifiers:**
  - Direct modelling of $p(C|\mathbf{x})$
  - Focus on the separating surfaces in feature space
  - In general, this leads to simpler models and, therefore, requires fewer training samples

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

1. Binary classification: Logistic Sigmoid Function

- **Binary classification** with two classes L^1, L^2 (*object, background*)
- We start with a **Bayesian view** and express the posterior for L^1 using the Theorem of Bayes:

$$p(C=L^1|\mathbf{x}) = \frac{p(\mathbf{x}|C=L^1)\cdot p(C=L^1)}{p(\mathbf{x}|C=L^1)\cdot p(C=L^1) + p(\mathbf{x}|C=L^2)\cdot p(C=L^2)} = \frac{1}{1+\dfrac{p(\mathbf{x}|C=L^2)\cdot p(C=L^2)}{p(\mathbf{x}|C=L^1)\cdot p(C=L^1)}} = \frac{1}{1+e^{-a}}$$

with $$a(\mathbf{x}) = \ln\frac{p(\mathbf{x}|C=L^1)\cdot p(C=L^1)}{p(\mathbf{x}|C=L^2)\cdot p(C=L^2)} = \ln\frac{p(C=L^1|\mathbf{x})}{p(C=L^2|\mathbf{x})}$$

- **Result:** Formally, the posterior ***p*(*C*=*L*¹|x)** can be expressed using the logistic sigmoid function $\sigma(a) = \dfrac{1}{1+e^{-a}}$
- **Assumptions**: $\Sigma_1 = \Sigma_2 = \Sigma$ (Lecture slide 8 ch.8), $a(\mathbf{x})$ is a linear function of the features!

$$p(C=L^1|\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}^T\cdot\mathbf{x}+w_0)}} = \sigma(a(\mathbf{x})) = \sigma(\mathbf{w}^T\cdot\mathbf{x}+w_0)$$

```
#Pytorch:Binary task with 2 fts x0 and x1)
torch.sigmoid(w_0*X[:,0] + w_1*X[:,1] + b)
```

Complex models: Feature Space Mapping

- General equation for the posterior: $p(C{=}L^1|\mathbf{x}) = \sigma[a(\mathbf{x})] = \dfrac{1}{1 + e^{-a(\mathbf{x})}}$
- For **identical covariance matrices**, $a(\mathbf{x})$ was a linear function of the features **x:**

  $a(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + w_0$

- Problem: Increasing complexity of the models for the probability densities → $a(\mathbf{x})$ becomes more complex, too, e.g. a **quadratic form** for normal distributions
- Rather **than using more complex models** for probability distributions**, we expand the feature space**

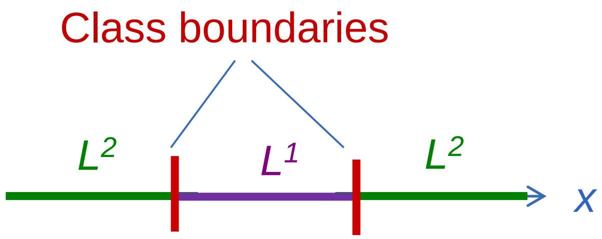
  → Feature Space Mapping $\boldsymbol{\Phi}(\mathbf{x}) = [\Phi_1(\mathbf{x}), \Phi_2(\mathbf{x}), \ldots, \Phi_N(\mathbf{x})]^T$

- In the expanded feature space, we can still work with linear models for the exponent
  - Example for 2D feature space, i.e. $\mathbf{x} = (x_1, x_2)^T$: $\boldsymbol{\Phi}(\mathbf{x}) = (1, x_1, x_2, x_1 \cdot x_2, x_1^2, x_2^2)^T$

  → "quadratic expansion" where $\boldsymbol{\Phi}(\mathbf{x})$ is frequently a polynomial function

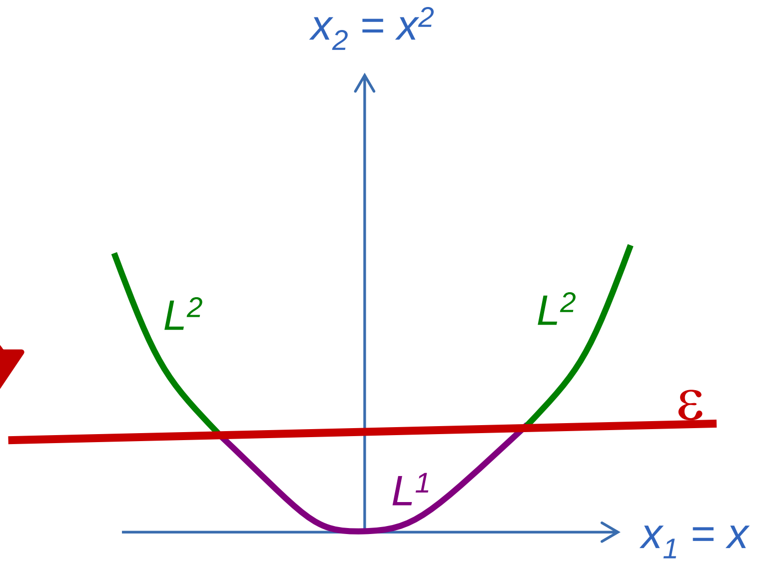
Example: Not linearly separable

- Transition to a higher-dimensional feature vector $\mathbf{\Phi}(\mathbf{x})$
- Example



Feature space transformation

$$\mathbf{\Phi}(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix}$$



1D feature space (x), 2 classes

Not linearly separable

After feature space transformation:

2D feature space (x, x^2)

Classes can be separated by a plane ε

Weights: [[0.549 0.603] <---- L^1
[0.549 0.603] <---- L^2]

Weights: [[0.549 0.715 0.603] <---- L^1
[0.549 0.715 0.603] <---- L^2]

2. Multi-class classification: Softmax Function

- The posterior $p(C = L^k \mid \mathbf{x})$ for each class L^k is modelled using the **softmax function**:

$$p(C=L^k|\mathbf{x}) = \frac{\exp\,[a_k(\mathbf{x})]}{\sum\limits_j \exp\,[a_j(\mathbf{x})]}$$

with $a_k(\mathbf{x}) = \ln\,[p(\mathbf{x}|C=L^k)] + \ln\,[p(C=L^k)]$

- Assumptions about $p(\mathbf{x}|C=L^k)$ and $p(C=L^k)$ lead to models for $a_k(\mathbf{x})$
- Again, feature space mapping can help to obtain linear models:
  $a_k(\mathbf{x}) = a_k(\mathbf{\Phi}(\mathbf{x})) = \mathbf{w}_k^T \cdot \mathbf{\Phi}(\mathbf{x})$
- In training, one parameter vector $\mathbf{w}_k$ per class has to be determined

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

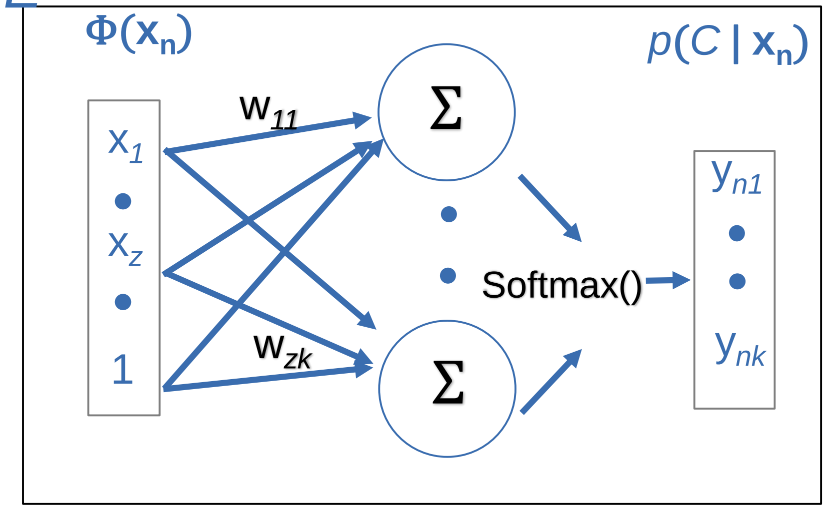
Multi-class classification: Training with ML

- Question: Which class label does a feature vector **x** belong to?
- Wanted: How to compute posteriors:

$$p\left(C = L^k \mid \mathbf{x}_n\right) = \frac{\exp\left[\mathbf{w}_k^T \cdot \mathbf{\Phi}(\mathbf{x}_n)\right]}{\sum_{j=1}^{M} \exp\left[\mathbf{w}_j^T \cdot \mathbf{\Phi}(\mathbf{x}_n)\right]} = y_{nk}$$

- Given: Class labels C_n for each training sample $\mathbf{x}_n$
- Approach: **Maximum Likelihood Training**

→ Similar to binary case minimize the negative log-likelihood E



$$E\left(\mathbf{w}_1, \ldots \mathbf{w}_M\right) = -\sum_{n=1}^{N} \sum_{k=1}^{M} t_{nk} \cdot \ln\left(y_{nk}\right) \rightarrow \min$$

with binary indicator variables $t_{nk} = \begin{cases} 1 & \text{if} \quad C_n = L^k \\ 0 & \text{otherwise} \end{cases}$

N: Number of samples M: Number of classes

Training (cont‘): Parameters Estimation

- **Procedure**: Compute the posteriors and take the class label to be the index of the maximum (np.argmax(posteriors, axis=1))

1. Compute the Gradient $\nabla E(\mathbf{w}^{\tau-1})$:

$$\nabla_{w_j} E(\mathbf{w}) = \sum_{n=1}^{N} \left(y_{nj} - t_{nj}\right) \cdot \boldsymbol{\Phi}(\boldsymbol{x}_n)$$

2. Compute the **Hessian Matrix** $\nabla\nabla E(\mathbf{w}^{\tau-1}) = \mathbf{H}\,(\mathbf{w}^{\tau-1})$:

$$\mathbf{H}_{jk} = \nabla_{\mathbf{w}_j} \nabla_{\mathbf{w}_k} E(\mathbf{w}) = \sum_{n=1}^{N} y_{nk} \cdot \left(\mathbf{I}_{kj} - y_{nj}\right) \cdot \boldsymbol{\Phi}(\mathbf{x}_n) \cdot \boldsymbol{\Phi}(\mathbf{x}_n)^T$$

$\mathbf{I}_{jk}$ … Elements of a unit matrix

3. Update the weights: Newton-Raphson

$$\mathbf{w}^{\tau} = \mathbf{w}^{\tau-1} - \mathbf{H}^{-1} \cdot \nabla E\left(\mathbf{w}^{\tau-1}\right)$$

(Ref. Ch. 8 Slide 11)

4. Increase iteration count τ and repeat until convergence and

5. Calculate posteriors:

$$p\left(C = L^k \mid \mathbf{x}_n\right) = \frac{\exp\left[\mathbf{w}_k^T \cdot \boldsymbol{\Phi}(\mathbf{x}_n)\right]}{\sum_{j=1}^{M} \exp\left[\mathbf{w}_j^T \cdot \boldsymbol{\Phi}(\mathbf{x}_n)\right]} = y_{nk}$$

Maximum Likelihood Training: Problem of Overfitting

- Maximum Likelihood training has the tendency to overfit the classifier to the training data → regularization with a prior for **w**
- Solution to keep the numerical values of **w** small:
  - Gaussian Prior $p(\mathbf{w})$ with expectation value **0** and Covariance Matrix $\sigma^2 \cdot \mathbf{I}$
- Requires hyper-parameter σ which is **either fixed by the user** or determined via a procedure such as cross-validation

- **Regularisation** (Gaussian prior) with exp. **0** and Covariance $\sigma \cdot \mathbf{I}$

- Negative logarithm of $p(\mathbf{w} \mid \mathbf{t}, \mathbf{x}_1, \dots \mathbf{x}_N)$ (excluding constant terms):

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\left[t_n \cdot \ln(y_n) + (1-t_n) \cdot \ln(1-y_n)\right] + \frac{\mathbf{w}^T \cdot \mathbf{w}}{2 \cdot \sigma^2} \rightarrow \min$$

Maximum Likelihood Training: Solution- Regularization

- Minimization of

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\left[t_n \cdot \ln(y_n) + (1-t_n)\cdot \ln(1-y_n)\right] + \frac{\mathbf{w}^T \cdot \mathbf{w}}{2\cdot\sigma^2} \rightarrow \min$$

leads to the numerical values of **w** that are as small as possible (indicated by σ)

- Gradient has to be extended compared to the ML method:

$$\nabla E(\mathbf{w}_1,\ldots\mathbf{w}_M) = \left[\nabla_{\mathbf{w}_2} E(\mathbf{w}_1,\ldots\mathbf{w}_M)^T, \ldots, \nabla_{\mathbf{w}_M} E(\mathbf{w}_1,\ldots\mathbf{w}_M)^T\right]^T \quad + \frac{1}{\sigma^2}\cdot\mathbf{w}$$

- This is also true for the Hesse Matrix:

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N}\left[y_n \cdot (1-y_n)\cdot \mathbf{\Phi}(\mathbf{x}_n)\cdot\mathbf{\Phi}(\mathbf{x}_n)^T\right] + \frac{1}{\sigma^2}\cdot\mathbf{I}$$

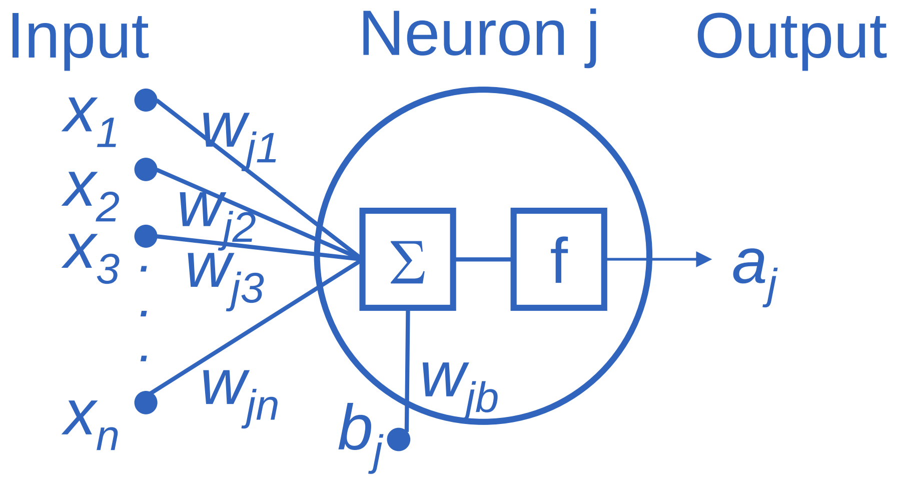
i.e. in the main diagonal, the weights of the direct observations for **w** are added

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

Artificial Model of a Neuron

- Input variables x_i: Components of the feature vector $\underline{\mathbf{x}}$
- Each input variable is multiplied with a weight;
  Determine **weighted sum** $z_j = \Sigma\, \underline{w_{ji}} \cdot \underline{x_i} + b_j = \underline{\mathbf{w}}_j^T \cdot \underline{\mathbf{x}} + b_j$
- b_j: Bias, considered to be a component of each feature vector
  → $\mathbf{x} = [\underline{\mathbf{x}}^T\ 1]^T$ and $\mathbf{w}_j = [\underline{\mathbf{w}}_j^T\ b_j]^T$
  → E.g. **For 1D feat. Space X, W** : [[0.549 0.603]]
  → Simplified notation : $l_j = \mathbf{w}_j^T \cdot \mathbf{x}$
- Output a_j of the neuron j:
  $a_j = f(l_j) = f(\mathbf{w}_j^T \cdot \mathbf{x})$ with $f(l_j)$ … activation function



Linear binary classifier based on a **Single neuron** → **Perceptron, logistic regression**

Neural Networks

- Networks consisting of **several layers** of neurons
  → More complex decision boundaries possible

- Example: two layers and “feed forward“ - architecture

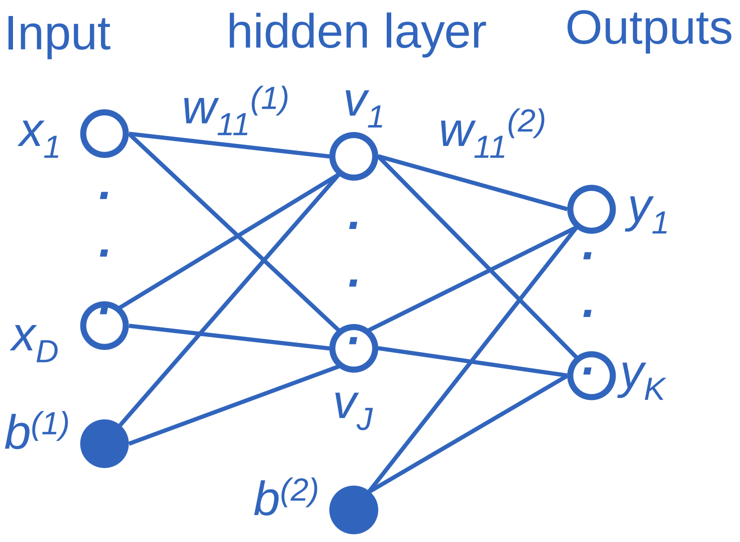
- Input: features x_i

- Hidden layer with neurons v_j:

  $v_j = f(\Sigma\, w_{ji}^{(1)} \cdot x_i)$

- Output layer: degree of membership to class C^k

  $y_k = f(\Sigma\, w_{kj}^{(2)} \cdot v_j)$



→ **Multilayer Perceptron (MLP)**

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

Neural Networks: Training with Gradient Descent

- **Goal: Minimizing the Loss Function** → Optimizer: Gradient Descent

- **Procedure**:

1. Compute the **posteriors** : $y_n = f(\mathbf{w}^T \cdot \mathbf{x} + w_b)$ → Sigmoid function

2. Compute the **BCELoss** : $E(\mathbf{w}) = -\sum_{n=1}^{N} \left[t_n \cdot \ln(y_n) + (1 - t_n) \cdot \ln(1 - y_n) \right] \rightarrow \min$

3. Compute the **gradients** : ∇E (w), *with respect to weights and bias*

4. Update the weights: Gradient Descent [Bishop, 2006]

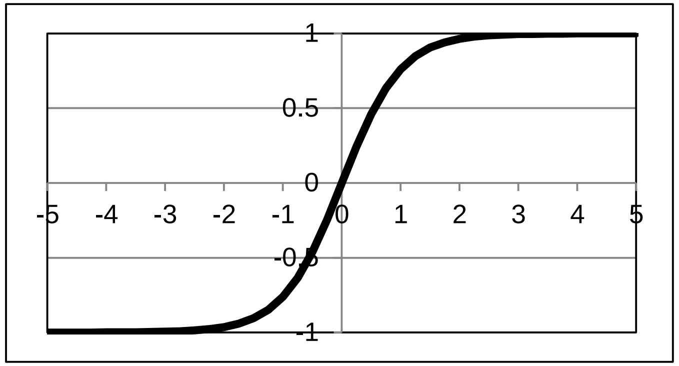
$$\mathbf{w}^{\tau} = \mathbf{w}^{\tau-1} - H^{-1} \cdot \nabla E(\mathbf{w}^{\tau-1}) \longrightarrow \mathbf{w}^{\tau} = \mathbf{w}^{\tau-1} - \eta \cdot \nabla E(\mathbf{w}^{\tau-1})$$ (ref. slide 27)

5. Run the training for a specific number of iterations/epochs until convergence

Activation Functions (non-linearity)

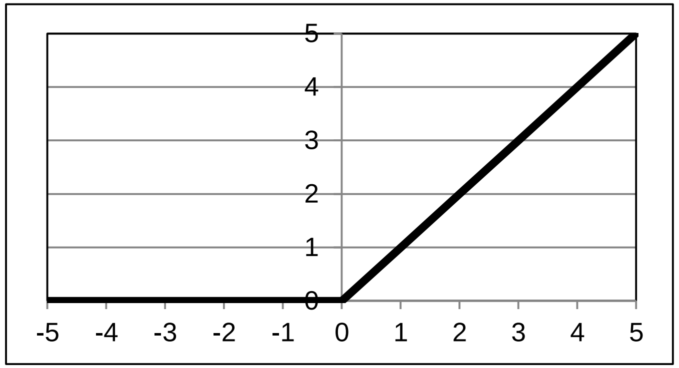
- Tangens Hyperbolicus (TanH):
  - Output is centered at 0.0
  - Small gradients for large / small inputs

  → Slow convergence in training!



$$f(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

- Rectified Linear Unit (ReLU):
  - Prevents vanishing gradients
  - Neurons can ‘die’
  - Mostly used for Deep Networks



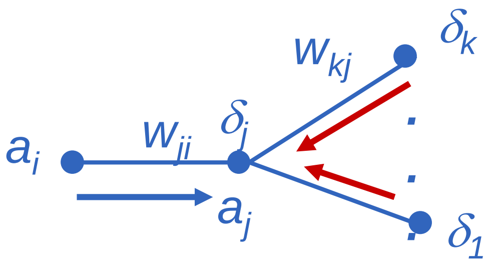
$$f(a) = \max(0, a)$$

- …

Optimization: Gradient Descent

- Back-propagation for computing the gradients:
  - Forward step:
    - Calculate output y_{nk} from $\mathbf{x}_n$ and the current values of $\mathbf{w}$
    - Save the output a_j as well as $f'(l_j)$ in every neuron j
    - The classification error and δ_k is calculated from y_{nk}
  - Actual back-propagation:
    - δ_j is calculated from δ_k and $f'(l_j)$ successively for each layer from δ_j and a_j:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j \cdot a_j$$



Stochastic Gradient Descent

- Stochastic Gradient Descent (SGD):
  - Gradients are averaged over Mini-Batch with B samples:

$$w'_{ji} = \frac{1}{B} \cdot \sum_{b=1}^{B} \frac{\partial E_{nb}}{\partial w_{jib}}$$

  - Weight update with learning rate η

$$w_{ji}^{(\tau+1)} = w_{ji}^{(\tau)} - \eta \cdot w'_{ji}$$

```
# Define the optimizer, here SGD: Stochastic Gradient Descent
optimizer = optim.SGD(net.parameters(), lr=Learning_rate)
```

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

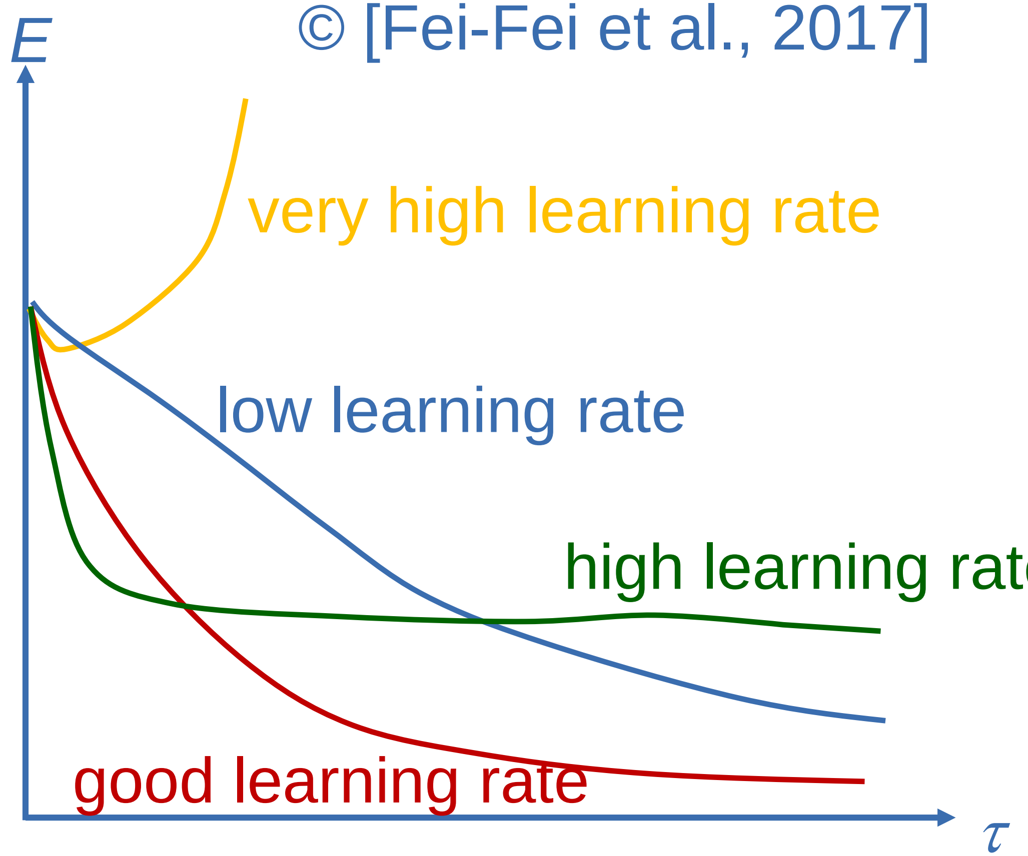
Neural Networks: Hyper-parameters

- Network architecture
  - Depth: Number of layers
  - Width: Number neurons per layer
  - Nonlinearity / activation function
  - …
- Training
  - Optimizer (SGD / Momentum / Adam / RMSProp, …)
  - Learning rate
  - Number of iterations
  - Mini-Batch size
  - …

Training Neural Networks: Learning rate

- Learning rate η in gradient descent is an important hyper-parameter
- Needs to be tuned carefully!
- Good η leads to …
  - Fast convergence
  - Strong minimum of E
- Adapt η in the iteration process
- Example: exponential decay with small ε

$$\eta = \eta_0 \cdot (1 - \varepsilon)^{k \cdot \tau}$$



© [Fei-Fei et al., 2017]

Overview: Lab 2

- Discriminative probabilistic classifiers:
  - Logistic Regression
  - Training
- Neural networks:
  - Model
  - Training
  - Hyper-parameters
- Frameworks
  - Overview

Frameworks overview: Pytorch

- Python package for machine learning, backed by Facebook
  - As of February 2022, PyTorch is the most used deep learning framework on Papers With Code
  - PyTorch also helps take care of many things such as GPU acceleration (making your code run faster) behind the scenes
- Pytorch has Five essential modules for building NN:

Module	What does it do?
torch.Tensor	Equivalent to Numpy arrays and matrices
torch.autograd	Automatic differentiation for all operations on tensors
torch.nn	Building Neural Networks
torch.optim	Various optimizations algorithms
torch.utils.data(.Dataset, DataLoader)	Data structures + transformation (augmentation)

Submission of Results

- Submission deadline: **July 3rd, 2023 before 11:00 am**

- Assignment → Jupyter Notebook (only digital)
  - Use meaningful variable names
  - Write comments if required
  - Answer concisely but completely
  - Consider acceptance rules
  - Consider IAI_23_Lab_Technical_Details.pdf about file naming scheme
- **Tutorial**: Neural Networks: **June 19th, 2023 at 11:30 am**
- Introduction to the 3rd Lab: **July 3rd, 2023 at 1:00 pm**