Lehrstuhl für Informatik VI **Thursday, November 29th, 2018**
Rheinisch–Westfälische Technische Hochschule Aachen
Prof. Dr.–Ing. H. Ney **{kim,kitza,gao}@cs.rwth-aachen.de**

6. Exercise Sheet **Statistical Classification and Machine Learning**

Solutions to the problems indicated by (*...) must be submitted by **08:00 of Thursday, December 6th, 2018** via L2P. Please form a group of up to **three** students! (Group Workspace in L2P)

1. **Empirical Distribution and Binary Divergence**

   Suppose we have a training dataset (x_n, c_n) for $n = 1, ..., N$, where $x_n \in \mathcal{X}$ is a discrete observation and $c_n \in \{1, ..., C\}$ is a class index. We would like to train a model $q(x, c)$ on this dataset.

   (a) Let $N(x, c)$ denote how many times a specific pair of observation and class index (x, c) appears in the training data. Express $N(x, c)$ using Kronecker delta $\delta(\cdot, \cdot)$. How can this quantity be normalized? (* 1P)

   (b) Let $pr(x, c)$ be the normalized quantity of (a). Express the marginal distributions $pr(x)$ and $pr(c)$ using the Kronecker delta. (* 1P)

   (c) Assume that the output of the model $q(x, c)$ is constrained to $[0, 1]$ using the Sigmoid function. The suitable training criterion would make the output large for the correct class and small for the other classes per training example: (* 3P)

$$
\begin{aligned}
E &= \frac{1}{N}\sum_{n=1}^{N}\left(\log q(x_n, c_n) + \sum_{c \neq c_n}\log[1 - q(x_n, c)]\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}\log[1 - |\delta(c_n, c) - q(x_n, c)|] \\
&= (...) \\
&= \sum_{x \in \mathcal{X}} pr(x) \sum_{c=1}^{C}\left(pr(c|x)\log q(x, c) + [1 - pr(c|x)]\log[1 - q(c, x)]\right),
\end{aligned}
$$

   which is called binary divergence. Show a detailed derivation for the missing steps (...) above.
   Hint: Introduce an empirical distribution using an index other than c for the class index, e.g. k. Consider switching two different sums over the class index at some point.

   (d) Prove that the optimal $q(x, c)$ which maximizes E in (c) is indeed $pr(c|x)$. (* 2P)

2. **Implemention of Neural Networks in Python**

   As attachment to this task you can find data for the Iris data set and a pre implemented structure of a basic neural network in file *network.py*. You will implement basic neural network functions and training using nothing but python and *numpy*.

   (a) Implement the **sigmoid** $sig(x)$ and **softmax**, $\sigma()$ activation functions as well as their derivatives. For an example look at the **tanh** function. (* 2P)

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

$$\sigma(x)_d = \frac{e^{x_d}}{\sum_{i=1}^{D} e^{x_i}} \quad \text{with } i = 1, \dots, D$$

(b) Implement the **inference** and **forward** functions. The **inference** computes of the output of a single layer y, using a bias term b, weights a_{ij} and a activation function $f(u)$. The **forward** function computes the output of a neural network $z^{(L)}$ by passing the output from one layer l to the next $l+1$. (* 1P)

$$z_i = b_i + \sum_j a_{ij} \cdot y_j$$

$$y_i^{(l)} = f(b_i + \sum_{j=1}^{J} a_{ij}^{(l)} \cdot y_j^{(l-1)})$$

(c) Implement the **backpropagation** function in the **Layer** class to calculate the gradient for the weights $\frac{\partial E_n}{\partial a_{ij}^{(l)}}$, $\frac{\partial E_n}{\partial b_i^{(l)}}$ as well as the error signal for the previous layer $\frac{\partial E_n}{\partial y_i^{(l-1)}}$. (* 4P)

$$\frac{\partial E_n}{\partial a_{ij}^{(l)}} = \frac{\partial E_n}{\partial y_i^{(l)}} \cdot f'(z_i^{(l)}) \cdot y_j^{(l-1)}$$

$$\frac{\partial E_n}{\partial y_i^{(l-1)}} = \sum_k \frac{\partial E_n}{\partial y_k^{(l)}} \cdot f'(z_k^{(l)}) \cdot a_{ki}^{(l)}$$

(d) Implement the **online_SGD**, **mini_batch_SGD** and **initializeWeights** functions. (* 3P)

- online_SDG: Train a neural network by updating the weights based on a single observation/feature vector.
- mini_batch_SDG: Train a neural network by updating the weights based on a accumulation of observations/feature vectors.
- initializeWeights: The weights of a **Layer** should not be initialized with 0. Try initializing them with noise.

(e) Run several tests on your neural network and try out different hyperparameter combinations. You should run each test serveral times and describe your observations: (* 3P)

i. Verify that your inference is working by using the **load** function of the network to read the provided parameter file *task_2a.sav* and run a classification on the evaluation data.

ii. Change the initialization of your weight matrix to 0, *unity*, $\mathcal{N}(\mu = 0, \sigma = \sqrt{2/N})$ with N beeing the number of weights.

iii. Vary the training settings: learning rate, mini batch size, layer size