Assignment 1: Machine Learning Basics (updated)
MA-INF 2313: Deep Learning for Visual Recognition

Due Date: 02.11.2021

Leif Van Holland holland@uni-bonn.de
Jan Müller muellerj@uni-bonn.de

Infos regarding submission of solutions: To submit your solution, go to eCampus, choose the tutorial your signed up for and upload your solution using the respective exercise item of your tutorial. The upload has to have the following format:

- A **PDF** file containing solutions to the theoretical exercises. Please use tools like LaTeX for your submission and refrain from providing handwritten scans.
- For the practical part, you may
  - **either** provide a **Jupyter notebook** containing your code, answers to the questions posed in the exercises, and visualizations. All cells containing results should have already been executed.
  - **or** provide a **PDF with all visualizations & non-code solutions**, as well as a **ZIP-file** containing the source code to generate your solutions, with instructions how to run the code.

If you have any questions, do not hesitate to ask via the eCampus forum or contact your tutor via mail.

1 Theoretical Exercises *(15 points)*

a) Bias of an estimator ***(7 points)***
The bias of an estimator is defined as

$$\text{Bias}(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta,$$

where $E(\hat{\theta}_m)$ is the expectation over data and θ is the true value. When the difference between the expectation and true value is zero, the estimator is unbiased.

We define two types of sample variances $\hat{\sigma}_m^2$ and $\tilde{\sigma}_m^2$

$$\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2 \qquad \text{and} \qquad \tilde{\sigma}_m^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2.$$

For a 1-D Gaussian distribution, show whether these two sample variances are biased or unbiased. *Hint:* $\mathbb{E}\Big[\sum_{i=1}^{m}\left(x^{(i)}-\hat{\mu}_m\right)^2\Big] = (m-1)\cdot\sigma^2$.

b) Bias Variance Trade-off *(8 points)*
The *mean squared error* (MSE) of an estimator is defined as

$$\mathrm{MSE}(\hat{\theta}_m) = \mathbb{E}\Big[(\hat{\theta}-\theta)^2\Big].$$

The MSE measures the expected squared "deviation" between the estimator and the true parameter value and is a good measure for the generalization error. Derive the bias variance trade-off from the definition of MSE.

2 Programming Exercises *(15 points)*

In this assignment, you will implement, train, test and report the performance of a k-Nearest Neighbor (kNN) classifier. You will use the Fashion-MNIST dataset, which is an alternative to the popular but simplistic MNIST dataset set, which in turn has become a benchmark for testing a wide range of classification algorithms. Please download the following 4 files:

- `train-images-idx3-ubyte.gz`
- `train-labels-idx1-ubyte.gz`
- `t10k-images-idx3-ubyte.gz`
- `t10k-labels-idx1-ubyte.gz`

To access the dataset in your python script, you can use the `load_mnist`-function provided in supplementary material to this exercise. You can access the data as follows:

```
from load_mnist import load_mnist
images, label = load_mnist(dataset="training", path=".")
```

The functions returns a tensor with shape $(N, 28, 28)$ which contains N grey-scale images with values in the range of $[0, 255]$ and a tensor with shape (N) which contains labels $y_i \in [0, 9]$. By passing the argument `dataset="training"`, the functions returns the 60,000 training images and labels. The other option is to pass the argument `dataset="testing"` to obtain the 10,000 test images and labels.

a) K-Nearest Neighbors (kNN) Classifier ***(9 points)***

K-Nearest Neighbors (kNN) classifiers are very simple machine learning algorithms which are often used as a benchmark for more complex algorithms. kNN requires no work at training time, it only stores all training data and their labels. At test time, for a given test data, it finds the k closest training data points according to some distance metrics and predicts a class label based on majority voting. The kNN algorithm has two hyper-parameters, k and the distance metric. k is the number of closest training data points and the distance metric is a function that is used to compute the similarity between pairs of data points. In this exercise, you will use the Euclidean distance as the distance metric for your kNN classifiers and cross validate the value of k.

1. *(4 points)* Implement a Python class called `KNNClassifier` which **only uses PyTorch functions** to evaluate a kNN-classifier as described above. Furthermore, **your implementation must avoid loops and match the following interface**:

   - The constructor of your class takes an integer value k, which controls the number of closest training data points, a tensor of training images and a tensor of training labels.
   - Your class has to implement a member function `forward(self, x)`, which takes a tensor of images as its input and returns the labels obtained using the kNN-classifier algorithm.

2. *(5 points)* After you implemented your kNN-classifier it is time to test your implementation and to investigate the influence of the hyperparameter k. Use the Fashion-MNIST dataset and implement the following tasks:

   - Load the complete Fashion-MNIST dataset and construct a training set with 1000 examples by randomly selecting from the training set, 100 samples per class. Let y_i^* be a class label of the test set and y_i be the predicted class label. The accuracy of your classifier can be computed as

   $$\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbb{I}(y_i^*, y_i) \text{ where } \mathbb{I}(y_i^*, y_i) = \begin{cases} 1 & \text{if } y_i^* = y_i \\ 0 & \text{otherwise.} \end{cases}$$

     Implement a function to compute the accuracy and report the accuracy for the complete test set with $k = 1$ and $k = 5$ neighbours on your newly constructed training dataset.
   - Visualize 1 and 5 nearest neighbour images for the first 10 test samples. Plot your confusion matrix to see which classes are mixed up with each other. (You can use the sklearn.metrics library to compute the confusion matrix)
   - Implement a function which performs a 5-fold cross-validation on your trainings dataset. Use validation part to test the accuracy of your kNN-classifier. Test your functions for k neighbours $k = 1, 2, ..., 15$ and plot these accuracies. Which k is performing best?

b) Dataloading and Preprocessing *(6 points)*
Efficiently loading and processing the elements of a dataset is a fundamental task when training any kind of data-driven machine learning model, and especially if you are going to train a deep learning model. In this assignment, you will learn to load the Fashion-MNIST dataset using PyTorch's data loading utility called "DataLoader"

```
dataloader = torch.utils.data.DataLoader(dataset, batch_size=32,
                                         shuffle=True, drop_last=False)
```

The arguments in the constructor of the "DataLoader"-class have the following meaning:

- `dataset` - An instance of your "Dataset"-class.
- `batch_size` - The number of elements which are included in a single batch.
- `shuffle` - Tells the "dataloader" to randomize the order elements.
- `drop_last` - Allows the last batch to include fewer elements than the batch size.

A complete list of all arguments for the "DataLoader"-class can be found here. A key advantage of using the "DataLoader" class compared to a simple loop is that the frameworks allows for multithreaded prefetching of dataset elements. This ensures that your training is not bottlenecked by IO-operations.

1. *(4 points)* The image and label tensors cannot be passed to the "DataLoader" directly, and instead you have to implement a "Dataset"-Class first. Your dataset class must meet the following requirements:
   - The constructor of your class needs to unpack the **training data** from the Fashion-MNIST. **You dataset-class should only consider dataset elements with label 0 or 1 and discard all other elements.** Furthermore, you should compute the mean and variance of the remaining training images in the constructor.
   - Your "Dataset"-class has to implement a function `__len__(self)`, which returns the number of trainings examples with class label 0 or 1 in the dataset.
   - Your "Dataset"-class has to implement a function `__getitem__(self, index)`, which takes an index as its input and returns the processed image and label associated with this index. Data normalization is beneficial when addressing a classification problem. The processed image should be shifted and scaled, such that the resulting dataset has zero mean and standard variance. Furthermore, the processing step should ensure, that all pixel values all lie within an interval of $[-1, 1]$.

2. *(2 points)* Now that have you implemented a Dataset-Class to access the elements of the dataset, create an instance of the "DataLoader"-class and loop over batches and labels.