Document Version 1.0.0, released 2020-12-11

Bonus Point Assignment I

1 Introduction

This document gives all necessary information concerning the first bonus point assignment for the class CSME II in the winter semester 20/21. This assignment consists of two Jupyter notebooks. In the first (Assignment 1A), you will implement a simple neural network class from scratch. With this class, you will construct and train an autoencoder on images of handwritten digits. The second Jupyter notebook (Assignment 1B) treats building and training a neural network in PyTorch.

Formalia With this voluntary assignment, you can earn up to 5% of bonus points for the final exam. The bonus points will only be applied when passing the exam. A failing grade cannot be improved with bonus points. This assignment starts on December 11th and you have until January 22nd to complete the tasks. The assignment has to be handed in in groups of 3 to 4 students. All group members have to register in the same group on Moodle under *Groups Assignment 1*.
Please hand in your solution as a single zip file including both notebooks as well as the `solution` folder, which contains automatically generated export files. Do not clear the output of your submitted notebooks. Your solution has to be handed in via Moodle before the deadline.

Grading Your implementations will mostly be checked automatically via unit tests, so ensure that your notebooks run correctly from start to finish. Throughout the notebooks, there are short unit tests for many tasks which you can use to validate your implementation. However, note that these are not the unit tests which will be used to grade your assignment.

Jupyter To run the Jupyter Notebooks for this assignment, we recommend to use the RWTH Jupyter-Hub. For this, search for the `CSME2` profile and start the server. This profile was specifically created for this class and automatically comes with PyTorch. Once your server is started up you can upload the notebooks and datasets and start completing the tasks.
Alternatively, you can set up your environment locally. However, please note that we cannot give support in case your environment does not work. First, you should install Anaconda or Miniconda on your system. Then install the required dependencies by running

```
conda install -y pandas seaborn scikit-learn pytorch torchvision cpuonly -c pytorch
```

in your **Anaconda prompt**. Next you can launch Jupyter via the Anaconda GUI or, if you are using miniconda, via executing `jupyter notebook` in a terminal or the Anaconda prompt. When using miniconda, you additionally may have to install Jupyter via `conda install -y jupyter`.

2 Assignment Part A

In the first part of the bonus point assignment you will implement a simple neural network class from scratch in Python, including weight updates via backpropagation. This network class will then be used to train an autoencoder on the MNIST handwritten digits dataset.

MNIST Handwritten Digits The MNIST database [1] consists of 28x28 greyscale images of handwritten digits. One example for each digit is shown in fig. 1. The dataset is split into a training set of 60 000, and a validation set of 10 000 labeled instances. The MNIST dataset is common introductory dataset for starting with machine learning, as it is well researched but also builds on real-world data. The classical task for MNIST is classification, i.e. prediction of the digit given the image. However, MNIST has been utilized for a wide range of tasks, including image generation.

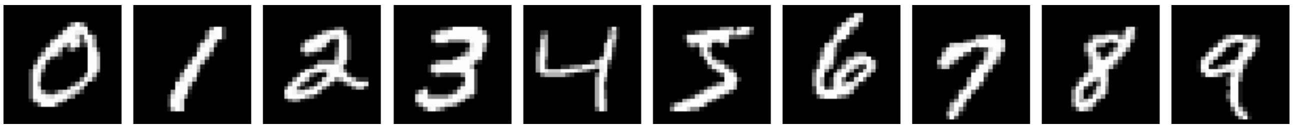

Figure 1: Example images from the MINST dataset.

Autoencoders Autoencoders are a type of artificial neural network used to learn efficient data encodings with unsupervised learning. An autoencoder aims to learn a representation (encoding) for a set of data, typically for dimensionality reduction. Along with the reduction side, a reconstructing side is learned, where the autoencoder tries to generate its original input from the reduced encoding, hence the name. An illustration of the autoencoder architecture can be found in fig. 2.
Besides dimensionality reduction, autoencoders can also be used as generators. To this end, the decoder is used to decode random input. This reconstruction typically resembles the training data. Thus the autoencoder can be used to generate unseen examples from the distribution of the training data.

Question A1 - Neural Network Class

In this task, you will implement the building blocks of a simple neural network class. The structure of this class resembles the architecture of standard machine learning frameworks like PyTorch and Tensorflow. For simplicity, the class in this assignment will only allow for feed-forward networks. However, all layers will support batched input. In this model, all individual operations, including activation functions, are represented as layers. The abstract `Layer` interface defines three methods:

`def forward(self, x):` This is the forward pass of the layer. It receives the output of a previous layer (or the input layer), applies its transformation to the data and returns the output of the layer.

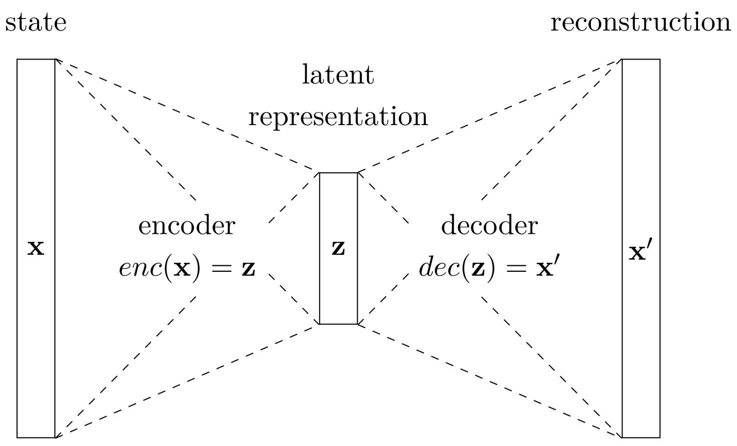
state
reconstruction
latent representation
x
encoder
$enc(\mathbf{x}) = \mathbf{z}$
z
decoder
$dec(\mathbf{z}) = \mathbf{x}'$
x'

Figure 2: Architecture of an autoencoder.

`def backward(self, gradient):` This is the backward pass of the layer. It receives the gradient of a following layer.

`def update(self, learn_rate):` This methods updates the weights of the layer according to the learning rate and the previous gradients.

a) The first layer to implement is a linear layer. The linear layer

$$\mathbf{x}_{\text{out}} = \mathbf{x}_{\text{in}}^T \mathbf{w} + \mathbf{b}$$

multiplies the input with its weights $\mathbf{w}$ and applies a bias $\mathbf{b}$. Complete the implementations of the `forward`, `backward` and `update` functions. Note: $\mathbf{x}_{\text{in}}$ and $\mathbf{x}_{\text{out}}$ are two-dimensional, as they contain the batch dimension as first dimension.

b) The next layer applies a ReLU activation

$$\mathbf{x}_{\text{out}}^i = \begin{cases} \mathbf{x}_{\text{in}}^i & : \mathbf{x}_{\text{in}}^i > 0 \\ 0 & : \text{otherwise} \end{cases}$$

on all inputs. Here, $\mathbf{x}^i$ denotes the i-th element of $\mathbf{x}$. Complete the implementations of `forward` and `backward`. As ReLU has no weights, there is no weight update for this layer.

Hint: The derivative of ReLU is not defined at zero. Choose an appropriate value.

c) The third layer implements a softmax activation function. The softmax is defined as

$$\mathbf{x}_{\text{out}}^i = \frac{\exp(\mathbf{x}_{\text{in}}^i)}{\sum_j \exp\left(\mathbf{x}_{\text{in}}^j\right)},$$

where x^i again denotes the i-th element of $\mathbf{x}$. Implement only the `forward` pass.

Hint: The exponentiation can result in huge numerical values. Think about how you can prevent numerical overflow.

Question A2

In this task you will build an autoencoder and train it on the MNIST dataset.

a) Using the layers you implemented previously, you can now build an autoencoder. This autoencoder gets a batch of MNIST flattened images as input. Flattened means that the 28x28 images are reshaped to (784,) vectors.

Choose an appropriate architecture and make sure to

- use a bottleneck of 16 neurons,
- apply ReLU activations after each layer, except for the bottle neck and output.

Hint: MNIST can be effectively learned with very small networks, larger networks are more prone to overfitting.

b) The next step is to actually train the autoencoder on the MNIST dataset via minibatch gradient descent. Most of the training loop is already implemented, your task is to implement the training of a batch.

The incoming batches of images have a shape of `(batch_size, 28, 28)`. To process them in a linear layer, you first need to reshape them to `(batch_size, 28 * 28)`. Then, call the `train` method of the `FeedForwardNetwork` to train the batch.

Tune your architecture (number and size of layers) and hyperparameters (e.g. batch size, learn rate, number of epochs until you reach a mean square error loss below 11 on the validation set .

Hint: Choose a learn rate in the order of 1×10^{-4} to 1×10^{-2}. Lower learn rates converge slower, higher learn rates can be unstable.

c) After successfully training the autoencoder you can utilize the decoder to generate new images. Split the autoencoder model at the bottleneck into two models, for encoder and decoder, respectively. The last layer of the encoder is the bottleneck layer, and the first layer of the decoder is the first layer after the bottleneck.

Then, implement the linear interpolation between the latent states $\mathbf{z}_a, \mathbf{z}_b$ of two images. Use `f` as mixing factor:

$$\mathbf{z} = (1 - f)\mathbf{z}_a + (f)\mathbf{z}_b$$

The generated images are exported to the `solution/` folder. Include them in your submission.

d) Finally, you can use the latent space to sample completely new images. The latent representations of the training samples are approximated with a Gaussian distribution. From this distribution you can draw new latent space samples and reconstruct images from them.

Implement the random sampling from the latent space.

Again, the generated images are exported to the `solution/` folder. Include them in your submission.

3 Assignment Part B

This part of the assignment gets you familiar with pandas, scikit-learn and PyTorch, three very common libraries for machine learning and data science. Pandas provides data frames, which are very useful for handling and working with tabular data. Scikit-learn implements many common data preprocessing, clustering and modelling methods. Lastly, PyTorch is a framework for neural networks.

Weather-Energy

For this task, we will use the dataset from the file `energy-weather.csv`. It consists of weather records and solar energy generation in the Province of València in Spain over a span of four years.

Question B1 - Data Preprocessing

Before the dataset can be used for machine learning, it requires some prepossessing.

a) Load the `energy-weather.csv` dataset into a Pandas DataFrame. The variable should be called `df`. Show the first five rows of the dataset.

Hint: You may use `pd.read_csv`.

b) Some rows in the dataset contain outlier values, which can negatively impact the fitted model. For example, the `pressure` contains a small isolated island at around 1080 as shown in fig. 3. Usually, detecting outliers requires a detailed analysis of each feature, but for this exercise they

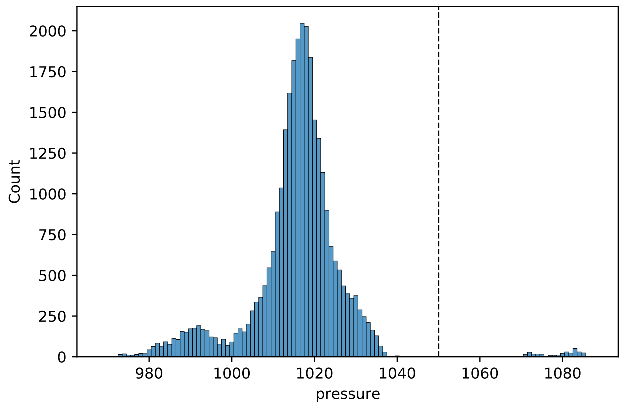

Figure 3: Histogram of `pressure`.

are given to you. Consider any row with

- a `pressure` greater than 1050,
- a `wind_speed` greater than 30,

to be an outlier. Filter out the outliers and name the resulting DataFrame `df_filtered`.

c) Timestamps are usually not suited as predictors. Extract the day of the year from `time` and store it as a `day` feature in the dataframe.

For classification purposes, a categorical feature is required. Create an additional column `generation_solar_categorical` by mapping `generation_solar` to the following class labels:

$$
\begin{aligned}
(-\infty, 160) &\mapsto \text{low} \\
[160, 1600) &\mapsto \text{medium} \\
[1600, \infty) &\mapsto \text{high}
\end{aligned}
$$

Hint: You may use `pandas.cut`.

d) For many machine learning algorithms it is important that the input is normalized. Otherwise, features with large numerical values can have a higher impact than features with smaller numerical values. Furthermore, large differences in magnitude can lead to numerical instabilities. Standardize the dataset, so that each numerical feature has a mean of 0 and a standard deviation of 1. Numerical features are all features except for `time`, which is a timestamp, and `generation_solar_categorical`, which is a categorical feature.

Hint: You may use the `StandardScaler` from scikit-learn.

e) Apply a 90/10 train-validation split to the data. Randomly choose 90% of the rows as the training set and use the others as a validation set. The test set was already separated from the data you received and will be used to evaluate your model accuracy.

Question B2 - Modelling with scikit-learn

We want to predict `generation_solar` from the weather features.

a) Fit a linear model to predict `generation_solar` from the predictors `day`, `temperature`, `pressure`, `humidity`, `wind_speed`, `wind_deg`, `rain_1h`, and `clouds_all`. Use `LinearRegression` from scikit-learn as the model. Train the model only on data from `df_train`.

b) Improve the regression by applying a polynomial basis transformation to the data. Tune the degree of the polynomials by choosing that degree, which leads to the lowest error on the validation set.

Hint: Use `PolynomialFeatures` from scikit-learn.

c) Use a `RidgeClassifier` to predict `generation_solar_categorical`, using the same predictors as in a). If done correctly, the resulting classifier should have an accuracy of at least 55 % of the validation set.

Question B3 - Modelling with PyTorch

a) As a warm-up task, construct a simple feed-forward neural network with PyTorch. A model in PyTorch is a subclass of `torch.nn.Module`. In the `__init__` method, all layers and parameters are created. The `forward` function defines how the network processes its input, similar to the `FeedForwardNet` class from part A. However, it is not necessary to define a `backward` method as PyTorch computes the gradients automatically.

Complete the implementation of the `__init__` and `forward` methods. This model should have

- 8 neurons at the input layer,
- two dense hidden layers with 10 neurons each,
- 1 neuron at the output layer,
- ReLU non-linearities after each layer, except for the output.

Hint: You can use this model for guidance on how a torch model looks like.

b) Now it's time to build a PyTorch neural network to predict `generation_solar`. Tune the hyperparameters (e.g. learn rate, epochs, batch size, optimizer) and architecture (e.g. number/size of layers, activation function) until you reach a validation error below 0.5.

Hint: You can start with the network from a) and then expand it. Possible further options to look into, besides the hyperparameters, are LeakyReLU activations, dropout layers or adding input noise.

c) Next, build a network to predict `generation_solar_categorical`. This target feature is categorical, so instead of just one output, the classifier has one output for each label. Also, it utilizes a cross-entropy loss instead of the MSE loss as with the regression.

Tune learn rate and architecture until you reach an accuracy on the validation set of at least 0.62.

d) How do the accuracies of the different models fitted in Question B2 and Question B3 compare? What model do you think is best suited for the prediction of `generation_solar`?

References

[1] Y. LeCun and C. Cortes, "The mnist database of handwritten digits," 2005.