Homework 5: Two machine learning approaches to handwritten digit recognition

Introduction

The *MNIST* (Modified National Institute of Standards and Technology) database is a database of handwritten digits. The database is frequently used in machine learning. Since the database contains 60000 handwritten digits for training and 10000 handwritten digits for testing it is enough for most machine learning applications.

One handwritten digit is represented as a 28×28 pixels image in gray scale. Thus mathematically it can be seen as an element in $\mathbb{R}^{28\times 28}$. The handwritten digits come with labels as well. The labels, which can be represented as so called *one-hot* vectors, tells which number the handwritten digit is supposed to represent. A one-hot vector is a vector with one component equal to 1 and the rest of the components equal to 0. Let $\boldsymbol{y} = (y_0, y_1, ..., y_9) \in \{0, 1\}^{10}$ be a one hot vector. Then $\boldsymbol{y}$ represents the digit i for which $y_i = 1$. E.g. $(0, 0, 0, 0, 0, 0, 1, 0, 0, 0)$ represents the number 6 and an example of a handwritten number 6 from the MNIST test set can be seen in Figure (1).

In this homework each handwritten digit will be seen as a vector $\boldsymbol{x} \in \mathbb{R}^{784}$, i.e. the two dimensional image is mapped onto a vector and thus the information of the two dimensionality of the handwritten digit is thrown away. The purpose is to keep this first example as simple as possible with still some degree of accuracy.

Part 1: Minimization with TensorFlow

Problem and algorithm descriptions

Consider the least squares problem

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{(10+784+1)\times K}} \mathbb{E}[|y(X) - \alpha_\theta(X)|^2] \tag{1}$$

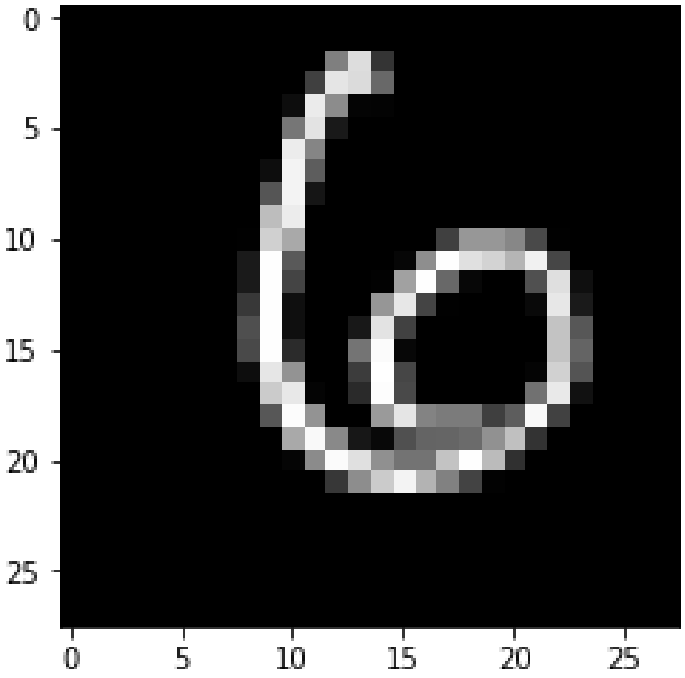

Figure 1: A handwritten number 6 with label $(0,0,0,0,0,0,1,0,0,0)$ from the MNIST test set.

where $X \in \mathbb{R}^{784}$ is a random handwritten digit with corresponding one-hot vector label $y(X)$,

$$\alpha_{\boldsymbol{\theta}}(x) = \left(\sum_{k=1}^{K} \theta_k^{1,1} s(\theta_k^2 \cdot x + \theta_k^3), \sum_{k=1}^{K} \theta_k^{1,2} s(\theta_k^2 \cdot x + \theta_k^3), ..., \sum_{k=1}^{K} \theta_k^{1,10} s(\theta_k^2 \cdot x + \theta_k^3) \right),$$
$$x \in \mathbb{R}^{784},\ \theta_k^{1,1}, \theta_k^{1,2}, ..., \theta_k^{1,10},\ \theta_k^3 \in \mathbb{R},\ \theta_k^2 \in \mathbb{R}^{784}, k = 1, 2, ..., K$$

and

$$s(x) = \frac{1}{1 + \exp(-x)}, \quad x \in \mathbb{R}.$$

A solution can be approximated using the following mini-batch Gradient Descent optimizer algorithm:

- Generate an initial guess $\boldsymbol{\theta}_0 \in \mathbb{R}^{(10+784+1)\times K}$ from a normal distribution.
- Take 60000 handwritten training digits $x_1, x_2, ..., x_{60000}$.
- Take $T \in (0, \infty)$ and denote by $\Delta t = \frac{T}{M}$ the step length.

- for $m = 1, 2, ..., M$ do
  - Take N random indices $i_1, i_2, ..., i_N \in \{1, 2, ..., 60000\}$.
  - $\boldsymbol{\theta}_{m+1} = \boldsymbol{\theta}_m - 2\Delta t \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} y(x_{i_j}) - \alpha_{\boldsymbol{\theta}_m}(x_{i_j})$.

An approximate solution to the minimization problem (1) is then given by $\boldsymbol{\theta}_M$. Here the positive hyper-parameter N corresponds to the batch size of our training algorithm. Note that for $N = 1$ the algorithm becomes a stochastic gradient descent optimizer algorithm and for $N = 60000$ the algorithm becomes a standard gradient descent optimizer algorithm.

Problems

1. Run the code `tf2_mnist_lsq.py`. What is the accuracy? Can you improve the accuracy by changing any hyper-parameters?

The softmax and cross-entropy functions

The neural network model will give a vector $\boldsymbol{z} = (z_i)_1^{10} \in \mathbb{R}^{10}$ as an output. That vector $\boldsymbol{z}$ is mapped to a vector $S(\boldsymbol{z})$ for which the components add up to 1 by the so called *softmax* function

$$S(\boldsymbol{z}) = \left(\frac{e^{z_1}}{\sum_{i=1}^{10} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{10} e^{z_i}}, ..., \frac{e^{z_{10}}}{\sum_{i=1}^{10} e^{z_i}} \right), \quad \boldsymbol{z} \in \mathbb{R}^{10}. \tag{2}$$

Thus each component of $S(\boldsymbol{z})$ can be seen as probability of $\boldsymbol{z}$ representing a specific number.

In the field of information theory one wants to quantify how much information is contained in some given data. Intuitively unlikely events contain more information than likely events. For two given probability distributions $p(X)$ and $q(X)$ over the same random variable X it is possible to measure how different $p(X)$ and $q(X)$ are with the so called *cross-entropy* function

$$H(p(X), q(X)) = \mathbb{E}_p[-\log q(X)]. \tag{3}$$

Note that the cross-entropy function is not a metric since it is not symmetric, $H(p(X), q(X)) \neq H(q(X), p(X))$. In the discrete case with vectors in $\mathbb{R}^{10}$ the cross entropy function is given by

$$H(\boldsymbol{y}, \boldsymbol{z}) = -\sum_{i=1}^{10} y_i \log(z_i), \quad \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^{10} \tag{4}$$

where $\boldsymbol{y}$ is the correct distribution or, in the handwritten digit recognition problem, the label and $\boldsymbol{z}$ is the output of the neural network model.

A cross-entropy minimization problem

We can now reformulate problem 1 to the problem of minimizing the cross-entropy instead as

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{(10+784+1)\times K}} \mathbb{E}[H(y(X), S(\alpha_\theta(X)))]. \tag{5}$$

Problems

Running the given code `tf_mnist_entropy.py` should give an accuracy of approximately 93%. That is really bad if compared to state of the art techniques that get more than 99.7% correct. Your task is to improve the given code to give a higher accuracy by following the steps below.

1. Run the code in `tf_mnist_entropy.py`. What is the accuracy?

2. The relu function $r(x) = \max(x, 0)$ is another activation function common in machine learning. Change the line

   ```
   hidden_activation_function = tf.keras.activations.sigmoid
   ```

   to

   ```
   hidden_activation_function = tf.keras.activations.relu
   ```

   in the code to start using the relu function as an activation function.

3. Another optimizer frequently used in machine learning is the so called Adam optimizer. A complete explanation of the Adam optimizer algorithm is outside the scope of this text. Change the line

   ```
   optimizer = tf.keras.optimizers.SGD(learning_rate=dt)
   ```

   to

   ```
   optimizer = tf.keras.optimizers.Adam()
   ```

   in the code to start using the Adam optimizer. Does the accuracy change? You might have to double M.

4. From the neural network model view $\alpha_{\boldsymbol{\theta}}$ can be seen as a neural network with an input layer, a hidden layer and an output layer. Now two more hidden layers, with the relu activation function $r(x)$ and K number of neurons, are added. Rewrite problem (5) with the new $\alpha_{\boldsymbol{\theta}}$. Extend the given code with the new hidden layers. To add one new hidden layer in the code can be simply accomplished with the sequential model. You can copy the definition of `hidden_layer_2` and name this copy `hidden_layer_2`:

```
nn_model.add(
layers.Dense(K,
             activation=hidden_activation_function,
             use_bias=True,
             kernel_initializer='random_normal',
             bias_initializer='zeros',
             name='Hidden_layer_2'))
```

Then paste this definition right after the definition the first hidden layer. To add several hidden layers is done analogously. How does the accuracy change? (Hint: maybe you can try to use different types of kernel initializers, e.g., `kernel_initializer='he_normal'`.)

5. Increase `K`. Warning: The computations becomes quite burdensome for the computer as `K` increases and the purpose is not for you to have to run very long time. It is enough that you can show some improvement in the accuracy.

6. Optional problem. Can you further improve the accuracy?

Part 2: Minimization with Random Fourier features

Random Fourier features

Consider the ten least squares problems

$$\min_{\hat{\boldsymbol{\beta}}^i \in \mathbb{C}^K} \left(N^{-1} |\mathbf{S}\hat{\boldsymbol{\beta}}^i - \mathbf{y}^i|^2 + \lambda |\hat{\boldsymbol{\beta}}^i|^2 \right), \; i = 0, 1, \ldots, 9\,, \tag{6}$$

where $\mathbf{S} \in \mathbb{C}^{N \times K}$ is the matrix with elements $\mathbf{S}_{n,k} = e^{\mathrm{i}\omega_k \cdot x_n}$, $n = 1, ..., N$, $k = 1, ..., K$ and $\mathbf{y}^i = (y_1^i, \ldots, y_N^i) \in \mathbb{R}^N$. The training data $\{(x_n; (y_n^0, y_n^1, ..., y_n^9))\}_{n=1}^N$ consist of handwritten digits represented by vectors $x_n \in \mathbb{R}^{784}$ and corresponding vector labels $(y_n^0, y_n^1, ..., y_n^9)$. Each vector label $(y_n^0, y_n^1, ..., y_n^9)$ has one component equal to one and the other components equal to zero. The index i of the component y_n^i that is equal to 1 is the number that the handwritten digit x_n represents. Here the frequencies ω_k, $k = 1, 2, ..., K$ are samples from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathrm{diag}([\sigma^2, \sigma^2, ..., \sigma^2]))$ where $\sigma = 0.1$.

Problems

Note that for the programming problems in this section we recommend you to use `Matlab`. You are not supposed to use any external machine learning package, like `TensorFlow` or `Matlabs` machine learning packages for example.

1. Write a function that maps any vector of digits $(0, 1, 2, \ldots, 9)$ to a matrix with the corresponding one-hot vectors as rows.

2. Write a program that finds approximate solutions to the linear least squares problems (6) using J digits from the set of training data and the function you created in the previous problem. You can solve the linear least squares problem with e.g. `Matlabs` backslash. Choose J reasonably large/small depending on the hardware you are using. Compute the percentage of mistaken digits on the test data.

3. Run your program for $K = 2, 4, 8, 16, 32, ..., K_{\max}$ where $K_{\max}$ is not larger than your computer can handle within reasonable time. Can you approximately replicate Figure 2?

4. Study for a fixed K the effect of varying λ on the generalization error.

5. The distribution of the frequencies ω_k, $k = 1, 2, ..., K$ is chosen somewhat arbitrary. Can you think of a distribution of frequencies that could give a smaller error?

Hint! You can use the function `readMNIST.m` to read the mnist-data into `Matlab`. By default `readMNIST.m` will remove a 4 pixel padding around each handwritten digit. To not remove the padding just make sure to not call `trimDigits`.

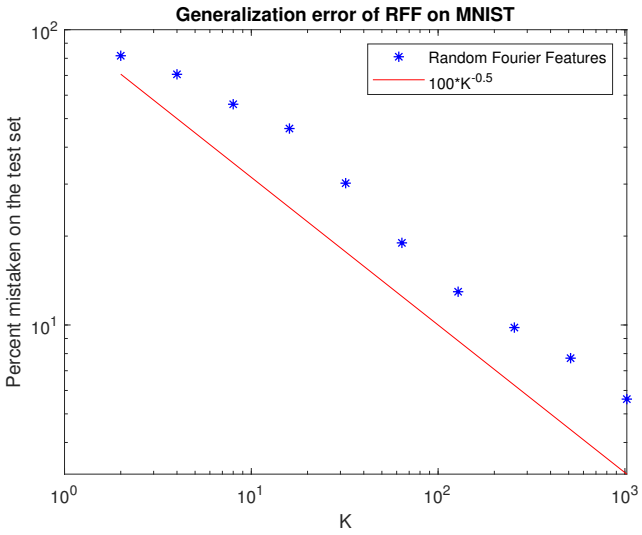

Figure 2: The dependence of K on the accuracy of β illustrated.