Assignment 2

Numerical Optimization / Optimization for CS WS2019

Erich Kobler, erich.kobler@icg.tugraz.at
Sarah Schneider, sarah.a.schneider@student.tugraz.at
Tarik Softic, tarik.softic@student.tugraz.at

November 26, 2019

Submission: Upload your implementation as a single py file and your report as a single pdf file to the TU-Graz TeachCenter.
Deadline: December 11$^{\text{th}}$, 2019 at 18:00h.

1 Multi-Class Classification with Neural Networks

The task of this exercise is to learn a neural network that classifies the Swiss roll dataset. The dataset consists of tuples $(\mathbf{x}^s, y^s)$, where $\mathbf{x}^s = (x_1^s, x_2^s, x_3^s)^\top \in \mathbb{R}^3$ is the input and $y^s \in \{0, 1, 2, 3\}$ is the target label of the s^{th} sample. Figure 1 visualizes a subset of the Swiss roll dataset.

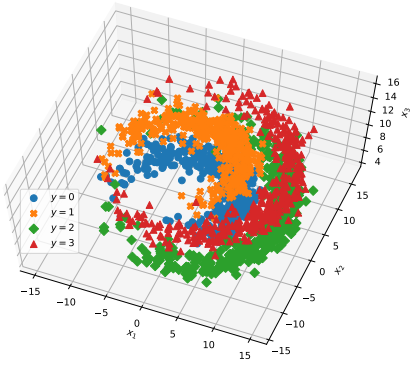

Figure 1: Samples from the swiss roll dataset.

2 Network Architecture

To achieve this task, we use a fully connected neural network with 1 hidden layer. In neural networks, the term fully connected means that every neuron from the previous layer is connected to every neuron of the subsequent layer. Using this approach, classifying multiple classes is typically performed in a lifted space that represents discrete probability distributions over the labels. Thus, the network predicts for *each* label how likely it is. To achieve this goal, we setup a neural network f that maps an input $\mathbf{x} \in \mathbb{R}^{N_I}$ (N_I is the input dimension) and the learnable network parameters $(\mathbf{W}^0, \mathbf{b}^0, \mathbf{W}^1, \mathbf{b}^1)$ to the probability simplex $S^{N_O-1} = \{\mathbf{y} \in \mathbb{R}^{N_O} : y_i \geq 0, \sum_{j=0}^{N_O} y_i = 1\}$ (N_O is the output dimension), i.e.

$$f(\mathbf{x}, \mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^0, \mathbf{b}^0) \coloneqq g\left(\mathbf{W}^1 h\left(\mathbf{W}^0 \mathbf{x} + \mathbf{b}^0\right) + \mathbf{b}^1\right) .$$

The activation function h applies the so called softpuls function to each element of the pre-activation of the input layer $\mathbf{z}^1 = \mathbf{W}^0 \mathbf{x} + \mathbf{b}^0 \in \mathbb{R}^{N_H}$ (N_H is the hidden dimension)

$$\mathbf{a}^1 = h(\mathbf{z}^1) \iff a_i^1 = \ln(1 + \exp(z_i^1)) .$$

The output of the network $\widetilde{\mathbf{y}} \in \mathbb{R}^{N_O}$ is eventually computed by applying the softmax function to the pre-activation of the output layer $\mathbf{z}^2 = \mathbf{W}^1\mathbf{a}^1 + \mathbf{b}^1 \in \mathbb{R}^{N_O}$

$$\widetilde{\mathbf{y}} = g(\mathbf{z}^2) \iff \widetilde{y}_i = \frac{\exp(z_i^2)}{\sum_{j=1}^{N_O} \exp(z_j^2)} .$$

The elements of the output of the softmax function are all positive and sum up to one. Thus, the output of the neural network $\widetilde{\mathbf{y}}$ can indeed be interpreted as a discrete probability distribution over all labels. The actual prediction of the network is then defined as

$$\widetilde{y} = \arg\max_i \widetilde{y}_i,$$

i.e. the most likely label.

3 Optimization

Assume we have given a set of S input-output samples $\{(\mathbf{x}^s, y^s) : s = 1 \dots S\}$, where $\mathbf{x}^s \in \mathbb{R}^3$ is an input image and $y^s \in \{0, 1, 2, 3\}$ is the corresponding ground-truth label. First, we convert each ground-truth label y^s into a discrete target probability distribution $\mathbf{y}^s$. For example let $y^s = 2$, then $\mathbf{y}^s = (0, 0, 1, 0)^\top$. Then, the training process consists of adapting the parameters of the network $\{\mathbf{W}^0, \mathbf{b}^0, \mathbf{W}^1, \mathbf{b}^1\}$ such that the output of the network $\widetilde{\mathbf{y}}^s = f(\mathbf{x}^s, \mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^0, \mathbf{b}^0)$ is as close as possible to the ground-truth distribution $\mathbf{y}^s$ for all samples. A common loss function in classification is the cross-entropy loss which is in our case defined as

$$l(\widetilde{\mathbf{y}}^s, \mathbf{y}^s) = -\sum_{i=1}^{N_O} y_i^s \ln(\widetilde{y}_i^s) .$$

Finally, the total loss is simply the average of the individual loss across the dataset, i.e.

$$\mathcal{L} = \frac{1}{S} \sum_{s=1}^{S} l(\widetilde{\mathbf{y}}^s, \mathbf{y}^s) .$$

In order to apply the steepest descent algorithm, you need to compute the gradient of the total loss w.r.t. to each learnable network parameter as presented in the exercise lecture. Let $\mathbf{p} \in \{\mathbf{W}^0, \mathbf{b}^0, \mathbf{W}^1, \mathbf{b}^1\}$ represent a network parameter. The k^{th} update step of the steepest descent algorithm for the parameter $\mathbf{p}$ reads as

$$\mathbf{p}^{k+1} = \mathbf{p}^k + t^k \mathbf{d}^k ,$$

where $\mathbf{d}^k$ is the negative gradient of the total loss w.r.t. $\mathbf{p}$ at the position $\mathbf{p}^k$

$$\mathbf{d}^k = -\frac{\partial \mathcal{L}}{\partial \mathbf{p}}\bigg|_{\mathbf{p}^k} .$$

4 Tasks

- For the Swiss roll dataset at hand, what is the size of the matrices $\mathbf{W}^0$, $\mathbf{W}^1$, and the bias terms $\mathbf{b}^0$ and $\mathbf{b}^1$? Specify the number of learnable parameters of the neural network as a function of N_H.
- Compute the derivative of the total loss w.r.t. to all model parameters using the results from the practical exercise session.
- Initialize the parameters using a normal distribution with $\mu = 0$ and $\sigma = 0.05$.
- Implement the forward pass of the network.
- Implement the backward pass using the back-propagation algorithm.
- Train the neural network using the provided *training data* for 1000 iterations with the steepest descent algorithm for $N_H \in \{4, 32, 512\}$.
  - Select a suitable step size $t^k \in [0.1, 0.001]$ and keep it constant over the iterations.

  - Select the step size with the Armijo rule in each iteration.

- Compare the training error and the accuracy of both trainings over the iterations in a plot. The accuracy is defined as the average number of correctly classified samples

$$\mathcal{A} = \frac{1}{S}\sum_{s=1}^{S} \delta(y^s, \widetilde{y}^s) \quad \text{with } \delta(u,v) = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{else,} \end{cases}$$

  where δ is the indicator function.

- Apply the learned network to the *test data* and report the loss and the accuracy.

- Create a 3D scatter plot showing the samples $\mathbf{x}^s$ with the true label y^s and your predicted label $\widetilde{y}^s$ for the training and test dataset. Discuss the obtained model performance.

- **Bonus Task: Weight decay (+2 points)**
  - In weight decay, the squared norm of the parameters $\mathbf{W}^i$ is included in the loss to prevent over-fitting. Thus, the updated total loss read as

$$\mathcal{L}_w = \frac{1}{S}\sum_{s=1}^{S} l(\widetilde{\mathbf{y}}^s, \mathbf{y}^s) + \frac{\lambda}{2}\left(\|\mathbf{W}^0\|_F^2 + \|\mathbf{W}^1\|_F^2\right).$$

  - Compute the derivative of the updated loss w.r.t. the learnable parameters and adapt the gradient update in your code.
  - The parameter λ is a tradeoff parameter between the classification objective and the regularization. Try two different values for λ and compare the results.
  - Report the influence of the weight regularization and additionally compare $\|\mathbf{W}^0\|_F^2 + \|\mathbf{W}^1\|_F^2$ with and without regularization over the training steps in a plot.

5 Framework

Use the provided python file for your implementation. It already provides the functionality to load the train and test data. For this task, you are only allowed to use the `numpy`, `scipy`, and `matplotlib` packages.

Show and describe all your results in the report!