Deep Learning Assignment II: Classification

Felix Wiewel

February 18, 2021

1 Introduction

A common problem that arises in machine learning is the classification of images. The goal of this exercise is to introduce you to the practical implementation of a complete pipeline for solving a classification task with neural networks in Tensorflow. For this we will make extended use of Keras, a specification of a high-level API for implementing powerful classes and functions to create and train neural networks. Tensorflow already comes with an implementation of the Keras specification so there is no need to install Keras if you have already installed Tensorflow. Using Keras instead of low level Tensorflow code to implement your neural networks and their training algorithms comes with some advantages but also some disadvantages. Since Keras makes it very easy to build and train neural networks, even people without strong programming skills can quickly develop solutions for given problems using neural networks. Keras essentially provides an abstract interface to the user and takes care of the low-level implementation details. This, however, comes with a cost. Since many technical details are hidden from the user, it is sometimes quite difficult to modify and change certain parts of your model/training algorithm. To a certain extend Keras provides you with ways to implement custom network architectures, layers and training algorithms but sometimes having to write code that is compliant with the Keras specification is more difficult than implementing it in low-level Tensorflow code. But for most standard applications, e.g. simple image classification like in this exercise, Keras is flexible enough and can save you a lot of time and frustration.

2 Mathematical Background

In classification we are interested in learning a function that maps an input $\mathbf{x} \in \mathbb{R}^M$ to one out of possibly many classes. In order to characterize such a mapping, we need a mathematical expression to uniquely identify different classes. This is typically done by assigning each class an integer value. So in the case of classifying images in the three classes Dog, Cat, Bird one could assign

these classes with the one-hot vectors $[1,0,0]^T, [0,1,0]^T, [0,0,1]^T$. Using one-hot vectors to represent labels of individual classes is very useful for training neural networks. It closely matches the mathematical model that we will be using for interpreting the networks output where each of its elements describes a probability. In classification using neural networks we are interested in learning the function

$$\mathbf{y} = f_{\boldsymbol{\theta}}(\mathbf{x}), \tag{1}$$

where f is a neural network with parameters $\boldsymbol{\theta}$ that maps our possibly high-dimensional input $\mathbf{x} \in \mathbb{R}^M$ to an output $\mathbf{y} \in \{0,1\}^k$ that represents one out of k possible classes. Similar to the regression exercise, we will use a probabilistic view on this problem in order to derive a suitable cost function we can use to train our model. This time, however, we will not introduce uncertainty through assuming some sort of noise acting on the prediction of the model as we did in the regression exercise. Instead we will treat both $\mathbf{x}$ and $\mathbf{y}$ as random variables with probability distribution functions (pdfs) $p(\mathbf{x})$ and $p(\mathbf{y})$. We already know that $\mathbf{y}$ can be one out of k classes, i.e. $p(\mathbf{y})$ is a categorical distribution. Instead of trying to learn a deterministic function that maps an input $\mathbf{x}$ to an output $\mathbf{y}$ we can instead learn a function that takes $\mathbf{x}$ as an input and outputs a pdf over our k different classes, i.e. $p(\mathbf{y}|\mathbf{x})$. In order to then classify an input we typically assign it with the class label that is most likely. In this way we can easily interpret the output of the model and also incorporate prior information into our decision. Using this probabilistic perspective on the classification problem we are interested in learning the conditional distribution

$$p(\mathbf{y}|\mathbf{x}) = \prod_{n=1}^{k} \mu_n(\mathbf{x})^{y_n} \tag{2}$$

where $y_n \in \{0,1\}$, $\sum_{n=1}^{k} y_n = 1$, $0 \leq \mu_n(\mathbf{x}) \leq 1$ and $\sum_{n=1}^{k} \mu_n(\mathbf{x}) = 1$. In other words, we want to learn the parameters of a neural network $\boldsymbol{\theta}$ that, given an input $\mathbf{x}$, outputs $\mu_n(\mathbf{x})$ for $1 \leq n \leq k$ satisfying the constraints $0 \leq \mu_n(\mathbf{x}) \leq 1$ and $\sum_{n=1}^{k} \mu_n(\mathbf{x}) = 1$. Practically, we can realize this by designing a neural network that accepts $\mathbf{x}$ as an input and has an output layer with k neurons and a softmax activation function. In order to derive a cost function for learning the parameters $\boldsymbol{\theta}$ of the neural network, we will again use the log-likelihood

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \ln p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \ln \prod_{n=1}^{k} \mu_n(\mathbf{x})^{y_n} = \sum_{n=1}^{k} y_n \ln \mu_n(\mathbf{x}) \tag{3}$$

where $\mu_n(\mathbf{x})$ depends on the parameters $\boldsymbol{\theta}$ of the neural network. The log-likelihood measures how likely $\mathbf{y}$ is given $\mathbf{x}$ and the parameters $\boldsymbol{\theta}$. Maximizing the expected log-likelihood over the complete data set yields the optimal pa-

(a) MNIST

(b) Fashion MNIST

Figure 1: Examples from dataset

rameters

$$\boldsymbol{\theta}^\star = \arg\max_{\boldsymbol{\theta}} \mathbb{E}\left[\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})\right] = \arg\min_{\boldsymbol{\theta}} -\mathbb{E}\left[\sum_{n=1}^{k} y_n \ln \mu_n(\mathbf{x})\right]$$
$$\approx \arg\min_{\boldsymbol{\theta}} -\frac{1}{N}\sum_{i=1}^{N}\sum_{n=1}^{k} y_{n,i} \ln \mu_n(\mathbf{x}_i), \tag{4}$$

where we have reformulated the maximization as a minimization of the negative objective function and approximated the expectation with a sample average. As you probably already noticed, the last term is the cross entropy loss that you are familiar with from the lecture. We can now again use SGD or some variant of it in order to minimize the cost function and obtain find some good parameters for our neural network. But keep in mind that while $\boldsymbol{\theta}^\star$ are the globally optimal parameters, we are usually only able to find locally optimal parameters due to the non-convex cost function that we are minimizing.

3 Data set

In this exercise we will use common data sets like the MNIST [1] or FashionMNIST [2] data sets. Examples from both are shown in Figure 1. Those and some other data sets are provided through simple functions in the Keras implementation of Tensorflow. For an overview of available data sets and how to use them click here. The MNIST data set is a simple and very popular data set for handwritten digit classification. It contains 70000 images with 28×28 pixels that are split into 60000 training and 10000 test samples. Since the task is to classify a handwritten digit, the labels y for each image are from the set $\{0, \ldots, 9\}$, which we will one-hot encode to $\{[1, 0, \ldots, 0], [0, 1, \ldots, 0], \ldots, [0, 0, \ldots, 0, 1]\}$. The FashionMNIST data set is very similar to MNIST. It also has approxi-

mately the same number of images with exactly the same dimensions as MNIST. It was provided by Zalando research and contains 10 classes of different fashion objects.

4 Transfer Learning

In many applications there is only a limited amount of annotated data available. In order to still train a neural network that generalizes well on such a data set, one can use transfer learning. In transfer learning the goal is to transfer knowledge from a source domain or task to a target domain or task. The hope is that this transfer will be positive, i.e. the performance on the target domain or task increases compared to only training on the target data set. A requirement for successful transfer learning is that the source and target domain have something in common, e.g. similar features. There is a broad literature on transfer learning methods [3] but in this exercise we will restrict ourselves to the most basic approach, fine tuning. Fine tuning can be implemented by using a part of a neural network, which was trained on the source domain or task, as a feature extractor. A common approach is to use neural network pretrained on the ImageNet [4] data set. In this exercise, we will also use a neural network pretrained on ImageNet in order to fine tune it for the Caltech 101 [5] data set. For a guide on transfer learning using fine tuning with Keras click here.

5 Catastrophic Forgetting

Despite many advances in better architectures and training algorithms for training, neural networks still suffer from a long known phenomenon called "catastrophic forgetting". In order to understand what this phenomenon is we can compare human learning with the way neural networks learn. Humans can quickly learn from few examples and most importantly, they can learn to solve tasks in a sequential way. This means one can learn a language, e.g. English, and after a certain period of time learn a second language, e.g. Chinese, without having to repeatedly refresh everything that was learned on the first language. Neural networks are currently not capable of learning on a sequence of different tasks. Overcoming catastrophic forgetting and enabling learning on a sequence of tasks is known as *continual learning*, *continuous learning* or *life long learning* in the literature. See [6] for a survey of recent publications and methods in this field.

6 Tasks

In order to complete this assignment, please fill out the code skeleton provided with this task description and solve the following tasks.

1. Determine the size of all feature maps of the first model created in the code skeleton, which is named `MyModel`.

2. Determine the ratio between the number of parameters used for convolutional and dense layers in `MyModel`.

3. Consider a weight matrix $\mathbf{W}$ of a dense layer and a random vector $x \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{I})$ as its input, i.e. the activation is $\mathbf{a} = \mathbf{W}\mathbf{x}$. How does a dropout layer applied before the dense layer affect the expected activation $\boldsymbol{\mu}_a = \mathbb{E}[\mathbf{a}]$? What is a simple way of ensuring that the expected activation is not effected if such a dropout layer is applied?

4. How could you quantify how similar two domains are?

References

[1] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[2] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[3] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[5] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer Vision and Pattern Recognition Workshop*, 2004.

[6] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *arXiv preprint arXiv:1909.08383*, 2019.