Machine Learning
Winter Semester 2019/2020

Exercise Sheet #8

Due: 2019-12-12 16:00

Fred Hamprecht
Manuel Haußmann

General Regulations.

- You should hand in your solutions in groups of at least two people (recommended are three).
- The theoretical exercises can be either handwritten notes (scanned), or typeset using LaTeX.
- Practical exercises should be implemented in python and submitted as jupyter notebooks (`.ipynb`). Always provide the (commented) code as well as the output, and don't forget to explain/interpret the latter.
- Submit all your files in a single `.zip` archive to `mlhd1920@gmail.com` using the following standardized format: The subject line should consist of the full names of all team members as well as the exercise, and the title of the zip archive the last names. I.e. assuming your group consists of Ada Lovelace, Geoffrey Hinton and Michael Jordan, this means

  Subject: [EX08] Michael Jordan, Geoffrey Hinton, Ada Lovelace
  Zip Archive: `ex08-jordan-hinton-lovelace.zip`

1 Reverse Mode Automatic Differentiation (10 pt)

As discussed in the lecture we can describe the flow of information through a "standard" neural network as falling into two phases. First we propagate information forward through the network from the input layer to the output (sometimes referred to as *forward propagation*). The second phase then consists of backpropagating the error we received from the scalar loss/error/cost function that we try to optimize. The function we consider in this exercise is

$$y(\mathbf{x}) = \left(\sin\frac{x_1}{x_2} + \frac{x_1}{x_2} - \exp(x_2)\right) \cdot \left(\frac{x_1}{x_2} - \exp(x_2)\right),$$

evaluated at $\mathbf{x} = (1.5, 0.5)$.

i) Give the computational graph of the function.

ii) Give the forward trace at the given $\mathbf{x}$.

iii) Give the backward/reverse trace.

Hint: For part i) *follow along with Figure 4 from the lecture and for* ii),iii) *with Table 3.*[1]

2 Getting to know Pytorch (10 pt)

There exist many deep learning libraries for python. A very popular one, which we will rely on throughout this sheet is `pytorch`[2]. See https://pytorch.org/get-started/locally/ for details on how to install it in your local environment. Although the current state of the art neural networks require GPUs to be trained efficiently

i) Go through the documentation to familiarize yourself with how to use the library[3].

[1] See also the original paper https://arxiv.org/abs/1502.05767 those the table and figure from the lecture were taken from.
[2] https://pytorch.org/
[3] See e.g. https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

ii) Implement a neural network with two hidden layers (the first one with 512 units, the second with 256) and ReLU nonlinearities that learns how to classify MNIST digits. Use Adam with a learning rate of 10^{-3} and a batch-size of 100. Train the network for at least 10 epochs and report the train/test set accuracies.

Hint: See `torchvision.datasets` for how to access the MNIST data, `torch.nn` for the functions necessary to define the network and `torch.optim` for the Adam optimizer. Adam is a very popular algorithm for stochastic optimization, see https://arxiv.org/abs/1412.6980 for more details.

3 Image classification with a deep network (10 pt +3pt)

In this exercise, we explore how to train a deep network on the GPU for image classification. The goal is to learn how to classify the CIFAR-10 data set, which consists of images showing one of ten categories (e.g. horse, car, dog,...).

i) Implement the VGG-16 we discussed in the lecture.[4] We have some modifications compared to the original architecture. As each CIFAR image has the shape of $32 \times 32 \times 3$, the spatial output shape of each convolutional layer will differ and after the fifth max-pooling you will end up with $1 \times 1 \times 512$, i.e. a 512 dimensional feature vector. As this gives us a much smaller latent space, and we only have ten categories, we reduce the number of units in the three fully connected layers to 512, 512, and 10 respectively.
Hint: It is often useful to normalize your input to the neural network. For an input vector, you would normalize each feature to have a zero mean and a standard deviation of one over all the data. For images it is common practice to normalize over each image channel, i.e. for CIFAR you would compute three means and three standard deviations for the normalization.

ii) Train the network with the Adam optimizer, a learning rate of 10^{-4}, a weight-decay regularization of 10^{-3}, and a batch-size of 256 for 10 epochs (feel free to experiment with these parameters and change them). Monitor the training loss throughout the training.

iii) Report your train/test set accuracies and plot three test images the network classified correctly as well as three images it failed to classify together with the probabilities for the ten classes it assigned.

iv) *(technical +3pt)* Two very popular methods for improving the training of neural networks are Dropout[5] and BatchNorm[6]. Implement Dropout (with $p = 0.5$) after your fully connected layers and batch normalization after the convolutional layers.
Hint: Dropout and Batchnorm should behave differently during training and during testing. You can use `model.train()` and `model.test()` to switch between these two settings.

For this exercise, you can rely either on your local GPU or if you do not possess one use the Google colab infrastructure[7] (`https://colab.research.google.com/`). It works similar to the jupyter notebooks we have been relying on throughout the exercises, just that it runs on Google servers. Per default this only uses CPUs, but if you go to *Edit→ Notebook Settings*, you can switch the "Hardware Accelerator" from *None* to *GPU*.[8] If you rely on colab, hand in the jupyter notebook you can get via *File→ Download .ipynb*.

[4] See also `https://arxiv.org/abs/1409.1556` for the original paper. Table 2 column D gives the basic setup for the architecture we rely on.

[5] See also `http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf`

[6] See also `https://arxiv.org/abs/1502.03167`

[7] Unfortunately this option requires a Google account.

[8] The other choice, the *TPU* is special hardware developed by Google and optimized for Deep Learning algorithms (see e.g. `https://en.wikipedia.org/wiki/Tensor_processing_unit`).