67750: Advanced Practical Machine Learning 2018/9

Exercise 3 — Convnets

Or Sharir *TA: Daniel Gissin*

1 Theoretical Questions (5 Points)

1.1 Parameterized ReLU

Define the following function:

$$f_i(o;t) = \max\{t, o_i\}$$

Will the incorporation of this function into a network define a larger hypothesis class? explain your answer and/or give an example.

1.2 Sigmoid Derivative

So far we talked mainly about the ReLU activation function, but another activation function which used to be very popular is the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

We saw in the recitation how we can easily calculate the derivative of the ReLU function by remembering which neuron had a positive activation, but calculating the derivative of the sigmoid function is potentially less pleasant.

Write the sigmoid activation as a computational graph with an input x and an output $\sigma(x)$.

Deriving the gradient of the sigmoid function means back-propagating through quite a few nodes. Derive the gradient of the sigmoid function and show that it can be expressed using the sigmoid function itself. This will show we can calculate the sigmoid derivative easily by remembering the activation value during the forward pass.

1.3 Network Initialization

Yossi didn't listen in class again, and decided to initialize our toy convnet with zeros for u, w_1 and w_2.

Calculate the gradient in this scenario and explain why this is a problem.

2 Practical Exercise (95 Points)

Please submit a single tar file named "ex3_<YOUR_ID>". This file should contain your code, and an "Answers.pdf" file in which you should write your answers and provide all figures/data to support your answers (and write your ID in there as well, just in case). Your code will be checked manually so please write readable, well documented code.

If you have any constructive remarks regarding the exercise (e.g. question X wasn't clear enough, we lacked the theoretical background to complete question Y, the exercise was too long and question Z felt like a lot of work with little knowledge gained...) we'll be happy to read them in your Answers file.

2.1 Exercise Requirements

1. Complete a quick and dirty code that learns a toy Convnet (35 points).
2. Transition to Tensorflow/Keras and build a simple linear classifier for MNIST (10 points).
3. Use Tensorflow/Keras to train a multi-layer perceptron and compare it to the linear model (10 points).
4. Build a convnet and compare it to the two previous models (10 points).
5. Choose a hyper-parameter and explore how it affects the learning process (10 point).
6. Build an autoencoder and compare it to PCA (20 point).

2.2 Dataset

In this exercise we will use the MNIST dataset, since training on it will be relatively fast on your personal computers and you won't need to use GPUs.

You can load the MNIST dataset, along with other datasets, directly from Keras. Code examples can be found here.

2.3 Python Environment

In case you're having problems with Keras on the CS computers, you're welcome to use the virtual-env we've set up for you. To activate it, run the following command:

```
>> source /cs/labs/shais/dsgissin/env/bin/activate.csh
```

To deactivate it, simply run:

```
>> deactivate
```

You may also need to load tensorflow if you get an error. To do that, run:

```
>> module load tensorflow
```

2.4 Toy Convnet (35 points)

We start off with the toy convnet we saw in the recitation, implementing the back propagation algorithm from scratch to understand the inner workings of computational graphs.

For an unknown matrix $W \in \mathbb{R}^{4x4}$, Define the following functions:

$$\begin{aligned} y_1(x) &= \sum_i (Wx)_i \\ y_2(x) &= \max_i(x_i) \\ y_3(x) &= \log(\sum_i e^{(Wx)_i}) \end{aligned}$$

We'll try to learn these functions with two models: a linear model and a toy Convnet (as described in the recitation). Our loss will be the squared error with l_2 regularization over all of the parameters:

$$\ell(p, y) = (p - y)^2 + \frac{\lambda}{2}(\|w\|_2^2)$$

Where p is the model's prediction. Complete the code that performs stochastic gradient descent for both the linear case and the convolutional network (found in the file *"Toy Convnet.py"*). Demonstrate that your algorithm is working and reducing the loss. Was it able to learn all of the functions?

Discuss the results and limitations of the two different models with respect to the different functions. When does the linear models perform better and when does the convnet? Why? Provide graphical comparisons of the results, and relate to them in your answers.

2.5 MNIST Classifier

After the last section, it should be obvious that building an arbitrarily complex neural network is trivial - it's simply a composition of functions. Furthermore, to learn the network parameters, one only has to be able to perform gradient descent, which is very simple as long as each layer is differentiable w.r.t. its input and the parameters. These insights led to the development of several packages that abstract the inner-workings of the back-propagation algorithm. There are two frameworks that are the most popular today:

1. Tensorflow - Google's framework. A simpler framework which uses Tensorflow as a backend and has easier abstractions and training procedures is called Keras.
2. PyTorch - Facebook's framework.

Since we aren't using neural networks for most of the course, we won't have time to learn all about Tensorflow and PyTorch, and instead we'll use Keras as our framework, which is the easiest to pick up quickly. Still, if you're interested in neural networks we recommend looking at some of the tutorials online for these frameworks, as they are very readable and have many code examples to help get you started.

To get acquainted with Keras, you can start with the Keras website. We will be using the Keras sequential model, so the best place to quickly understand the syntax is the sequential model guide.

Note that training a convnet until convergence may take a while on a laptop. In the classification part we don't really care about converging - we're mostly interested with understanding how the different models behave. This means you can stop training before convergence as long as you are able to support your claims. For instance, if you're saying that the convnet performs better than the MLP, you should train the two models long enough so that it's obvious that the convnet has better test accuracy...

Also note that many of the things you are about to implement have many code examples online. You can use these examples to get a quick understanding for Keras, but we recommend that you make an effort to solve the autoencoder part by yourself, using only the Keras documentation, as this will give you a much better understanding than simply copying from some blog post...

2.5.1 Linear Model (10 points)

Use Keras to build a simple linear model, and apply gradient descent on a cross-entropy loss to learn a model for MNIST. Does your model generalize? Plot the learning curves: train/test loss and accuracy as a function of the epoch and shortly discuss the results.

Relevant Keras layers are the "Dense" layer and the "softmax" activation function.

2.5.2 Multi-Layer Perceptron (10 points)

A multi-layer perceptron (MLP) is a neural net consisting of several fully-connected layers, interleaved with activation functions. Build and train a multi-layer perceptron on MNIST. You'll need to use fully connected layers ("Dense"), an activation function for each layer ("relu"/"sigmoid"...) and the "softmax" activation function for the final layer.

Select the model's depth, a loss function and an optimization algorithm, and perform gradient descent to learn the model. Discuss your choices and results.

2.5.3 Convnet (10 points)

Build and train a convnet, made from convolution layers, activation layers and pooling layers, ending with a fully connected layer just before the network's output. You'll need to use the "Conv2D" and "MaxPooling2D" layers, in addition to what you already used before.

Again, select a loss function and some optimization algorithm, and perform gradient descent to learn the model. Explain your choices.

Compare the convnet performance to the linear and MLP models.

2.5.4 Hyper-Parameter Exploration (10 point)

Choose one hyper-parameter out of the following:

1. Learning Rate

2. Dropout Probability
3. Batch Size

Train a neural network on MNIST over a range of values for your hyper-parameter. Plot the learning curves for all of the values in a single figure. Compare the different values and discuss how your hyper-parameter effects the training procedure. Make sure you use a large enough range for your hyper-parameter such that we can see how it affects the training.

2.6 Autoencoders (20 points)

Finally, we will implement an autoencoder for the MNIST dataset using Keras.

Build an autoencoder that reduces the dimensions of the MNIST images from 784 all the way down to 2 dimensions. Use the mean squared loss and show learning curves for your autoencoder. Explain which architecture you chose for the encoder and the decoder and why.

2.6.1 Comparison to PCA

Retrieve the final 2D embedding of 5000 random MNIST digits from the inner layer of your network. How to access intermediate representations of your model is explained in the Keras FAQ.

Plot the digit embedding in a scatter plot, color coded according to the labels. Compare your embedding to PCA embedding (you may use the scikit-learn implementation to save you some time). Does your autoencoder extract the structure of the data better than PCA?

To color-code your scatter plot according to the labels of the data, use the “c” argument. For example:

```
plt.scatter(data[:,0],data[:,1],c=labels)
```