Ex1 - ConvNets

October 2019

Please submit a single tar file named "ex1_<your-id> *.tar*". 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.

Theoretical Questions (5 Points)

0.1 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.

0.2 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.

Practical Exercise (95 Points)

The goal of this Ex' is to be familiar with tensorflow and to understand and write some simple neural networks.

please write your code in the supplied files without changing the methods signatures, so we will be able to run your code.

Exercise Requirments

1. Implement and train linear regression from scratch using tensorflow
2. Implement and train multi layer perceptron and simple ConvNet for image classification task.
3. choose one: find an adversarial example, or plot SNR of your gradients.

Computation resources

In this exercise we will train small models on small datasets. your laptop should be fine, school computers are more then enough.

Environment

Recommended setup in school computers:

1. Create virtual environment (python 3.6)
2. Install tensorflow version 1.13.1, pandas, numpy and matplotlib using pip
3. You may also need to load tensorflow as a module. in the terminal, type "module load tensorflow/1.13.1"

Tensorflow api documentation

We are working with tensorflow version 1.13, in non-eager mode. The official tutorials in tensorflow website are for tensorflow 2 Beta.

Doc for 1.x are here: https://github.com/tensorflow/docs/tree/master/site/en/r1
Api reference is here: https://www.tensorflow.org/versions/r1.13/api_docs/python/tf
Relevant guide: https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/low_level_intro.md

0.3 Implement linear regression from scratch (35 Points)

In this part you are going to implement two tf-operations with there gradients: Matrix-multiplication and mean-squared-error, and use them to build linear regression model. You will train the model using batch-sgd with tf.gradients directly, and not tf.optimizer. You will train your model on the boston housing dataset.

0.3.1 operations and gradients

Please implement mean-square-error and matrix-multiplication with their gradients using numpy, and wrap them as tensorflow operations. Relu is implemented as example. Write your code inside operations.py file.

0.3.2 model and training loop

Implement the model and the training loop as described in linear_regression.py. Use tf.gradients (and not optimizer). Use the mse and matrix multiplication from the previous section. Useful operations: tf.assign_sub, tf.group (for the training). There is example of linear regression implementation in the recitation slides.

Please plot a graph of the training-loss at each epoch and attach it to your answers file.

0.4 Implement and train mlp and ConvNet (40 Points)

In this section you will implement multi-layer-percceptron and ConvNet. You will train them on Fashion-MNIST dataset. You can use optimizers, tf.nn and any tensorflow functionality you want. please don't use keras for more than data loading.
You are welcome to let tf manage your variables using the "TRAINABLE_VARIABLES" collection in the graph. If you do not specify which variables to train, the optimizer will update all the variables that participate in the computation, this is fine for us. Just set "trainable=True" in tf.get_variable.

The mlp and convnet functions should be implemented in the *models.py* file, and the training loop in *train.py*.

0.4.1 MLP

Implement the mlp as a function that get symbolic tensor as input (shape: [batch, 28, 28]) and return symbolic tensor as output (shape: [batch, 10]). The output of the model are the logits: the un-normalized probabilities that the model give to each one of the classes. The softmax operation will be added as part of the loss. Your mlp should have this architecture:

x − > linear layer − > relu − > linear layer

with shapes:

784 (x, flattened) − > 100 (after 1 linear layer) − > 10 (after the second linear layer)

the linear layer is $W \times x + b, W \in (dim_{in} \times dim_{out})$

0.4.2 ConvNet

Implement a ConvNet with a same API. use this architecture:

x − > conv − > maxpool − > conv − > maxpool − > linear

0.4.3 training

Implement the training loop in the *train*() function in *train.py* file and train the models. Train your models using sparse_softmax_cross_entropy_with_logits as loss. For each of the models, plot the following curves: average training loss at each step, average training accuracy at each step, test loss and accuracy at each step. You are welcome to use tensorboard for the plots.

Can you describe distinct stages in the training process? Do you have overfitting?

0.4.4 Regularization (15 Points)

choose one of the above models, add some regularization for the training and describe the change it cause the training process (if any). Note: you can add weight-regularization regularization in tf.get_variable, or dropout as an operation.

0.5 Choose one: (10 Points)

1. find adversarial example for one of your models. you should optimize the image (not the network!) to give you the "wrong" prediction, under some regularization (the new image should be close enough to some real image). weight-sharing mechanism (like variable-scope), or checkpoints, can be useful here. if you prefer to transfer your variables using numpy arrays it's also fine. you can also train a variable that is added to your image.

2. train one of the previous networks and plot the SNR (signal-to-noise ratio) of the gradients for each of the variables during training. think of informative way to do that. does the result make sense? any reasonable way to perform the calculation, together with the expected result and comparison of it to the actual result, will be accepted.

1 Notes

1. the default initializer is good for all what we need here.

2. Adam optimizer with default parameters should also be fine.

3. tf.nn is where all the complicated operations hide; tf.train is where the optimizers are; tf.summaries is for tensorboard stuff.

4. the maximal grade in this exercise is 105.

Good luck!
Omri