IT3030 - Deep Learning

Assignment 1: Backprop with numpy

Jan. 2020

This documents describes the first assignment in IT3030, where you will implement gradient descent learning in deep learning ("backprop") without using any of the available libraries for automatic differentiation etc. The rules are as follows:

- The task should be solved individually
- It should be solved in Python, implement the functionalities as described below, using only packages that are explicitly listed as OK.
- Your solution will be given between 0 and 20 points; the rules for scoring are listed in the last section
- Deadline for **demonstration** is Feb 14th, during the announced lab-hours on that day. Upload the code to Blackboard before you demonstrate your code to an evaluator.
- Please note that grading depends on how you demonstrate your code to the evaluator, submission of code is just to lock your work for the event.
- Make sure code is commented properly and submitted to Blackboard before the demonstration starts.
- Lab hours/demonstration hours schedule will be published on Blackboard.

1 Setup

Before detailing what you will do in the assignment, we start by going through the setup externally to your Python-code:

1.1 Config-file

The parameters for your system will be set using a config-file, and the following structure can be expected:

```
[DATA]
# training and validation are pointers to the files used for learning
training=./DATA/training-file.csv
validation=./DATA/validation-file.csv

[MODEL]
# layers is a comma-separated list of integers telling us how many nodes in each
```

```
# hidden layer. Special case: If the value is only one element in the list, and
# its value is 0,  you should generate a net without a hidden layer
layers = 24,12,6

# activations is a comma-separated list of key-words. It will have as many
# elements as there are elements in the layers-list. Each keyword is a
# non-linearity function, and legal values are relu, linear, and tanh.
activations = relu, relu, tanh

# loss_type chooses between L2 loss (for regression) and
# cross_entropy (for classification).
loss_type = L2

[HYPER]
# Learning rate to use
learning_rate=1.E-3
# Number of epochs before finalizing
no_epochs=10
# What L2-regularization to use to avoid over fitting.
L2_regularization=1.E-2
```

You can for instance use ConfigParser from the configparser package to parse this. We will use a set of different config files during the evaluation of your code, so make sure this part works. You need to be able to parse a comma-separated list of integers to get the layers; here you can safely assume that first reading the full value ("24, 12, 6" in the example) as a string, then splitting based on ",", trim of white spaces, and finally cast as integers will not produce errors. The only special case is if layers has the value "0" which means there are no hidden layers, and the input-layer ports directly to the output.

For the activation functions you will get one function-type per layer, and the types you are expected to handle are "linear", "relu", and "tanh". Again, no error checking is required. Note that in addition to the mentioned functions, you will also need to handle the softmax as part of your implementation to handle the classification-loss (see below).

1.2 Data files

The data files are comma separated values, one line per data record. All entries can be assumed to be numeric. One record will start with the inputs, then ended by one output variable.

When doing classification, the classes are named as integers, so we use "0", "1", "2", ... If y_train is a numpy vector holding the classes from the training data, you can assume that you will have number of classes equal to no_classes = 1 + y_train.max(). You will have to transform this representation into one-hot encoding, something you can do as follows: First allocate the required space: one_hot = np.zeros((no_examples, no_classes)). Thereafter you need to set the correct element of each row to 1: one_hot[np.arange(no_examples), y_train] = 1; ensure that y_train is coded as integer before doing this transformation.

1.3 Packages

The following imports are OK, and not anything else: numpy, matplotlib.pyplot, configparser, enum, sys and softmax from scipy.special. Notice that tanh is available from numpy.

2 What to do

This section discusses what you need to implement in a step-by-step fashion. Each of the sub-sections will result in code that could give you points as described in the final section of the document. It is recommended to do the steps one-by-one, and for instance ensure that you are able to learn a simple model without hidden layers before starting to make more complex models. Still, read the full assignment before you start coding; you may save yourself some work if you design your code efficiently. You may use different set of config files for each section depending upon the requirements.

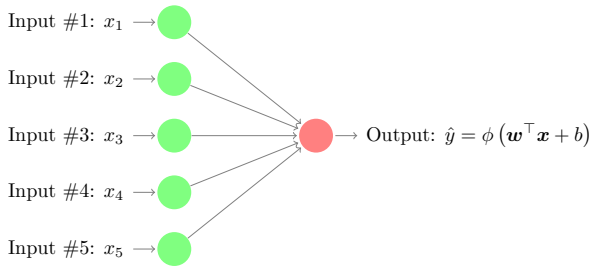

Figure 1: Simple neural net without hidden layers used for regression.

2.1 Minimalist example for regression

After setting up file-reader and parser for the config-file, the first task is to build a simple neural network as in Figure 1. In the figure there are 5 inputs, but this obviously depends on the input-data. As the the model has no hidden layers, the output based on an input $\boldsymbol{x}$ using weights $\boldsymbol{w}$ is simply $\phi\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right)$, with some nonlinearity ϕ. For now, use the ReLU, $\phi(z) = \max(0, z)$. Optimize the weights to minimize the L2-loss. Here and for all the other learning tasks, start from a random initialization after first setting the seed by using `np.random.seed(42)`.

2.2 Minimalist example for classification

In this section, we will change the output in Figure 1 to fit a classification. Do that by changing the nonlinearity to a softmax. A softmax for k classes will have k logits as outputs, see Figure 2. The logit-node for class i has its own weights $\boldsymbol{w}_i$ and bias b_i, hence is calculated as $\boldsymbol{w}_i^\top \boldsymbol{x} + b_i$. Thereafter, the logits are run through the softmax, and the loss-function to use is the cross entropy.

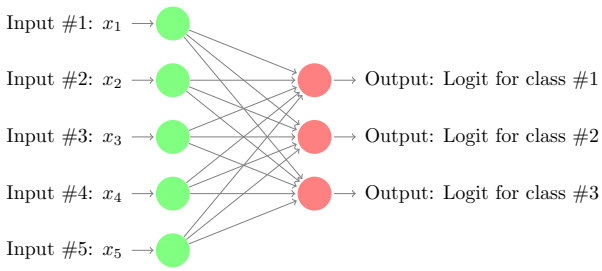

Figure 2: Simple neural net without hidden layers used for classification.

2.3 Models with a single hidden layer

This section requires attention since it has two parts.

Firstly, extend the model with a single hidden layer, where the number of nodes are read off from the layers list (which in this case will contain a single integer). The transfer functions in the hidden layer should first be relu functions, later (second part) to be read from the config-file. Recall that the loss gradient must be propagated back from the output layer and to the hidden layer to ensure that the weights of that layer are correctly updated; see your lecture notes for details. The implementation should support both regression (L2) and classification (cross_entropy).

Secondly, make the same model as above, but with the transfer-function read from the config-file (relu, linear, and tanh should be supported). Again, the implementation should support both regression (L2) and classification (cross_entropy).

2.4 Models with arbitrary structure

Finally, in this section, you shall build a deep feed forward model with an arbitrary number of layers, where the number of nodes and transfer function per layer are taken from the config file only. As always, your implementation is supposed to support both L2 and cross-entropy loss.

2.5 Regularization

Add L2 regularization to the learning. All layers should have the same regularization constant (found in the config file), and you can use the same regularization constant for both weights ($\boldsymbol{w}$-s) and offsets (b-s).

3 Earning points

The exercise can give you up to 20 points, and the table below lists which functionalities are required to get the different points. Start from the top, and notice, for instance, that if you fail to finalize the learning using classification loss ("MINIMAL CLASS"), you will not get more than maximum half score on the following challenges where the classification-loss is assumed to function (as in "SINGLE HIDDEN RELU").

Each of the points below will be tested by supplying your system with a specific config-file, e.g., one that has `layers = 0` and `loss_type = cross_entropy` to check your score on the "MINIMAL CLASS" task.

Item	Description	Points
DATA	Support to read data files into numpy arrays and parse the config file.	1
MINIMAL REGRESSION	Build a neural network without hidden layers (only input and output layer), and learn the weights that minimizes the L2-loss. The learning should run a predefined number of epochs using a fixed learning rate η as given in the config file. Dump loss for each to console, also for validation-data. Dump the learned weights to text-file.	3
MINIMAL CLASS	With the same structure, learn weights in the same way, but with classification-loss. Your output layer needs to have as many outputs as there are classes; see the description above. Same dump as above.	5

Continued on next page

Item	Description	Points
SINGLE HIDDEN RELU	Add a single hidden dense-layer of width specified in config-file using relu as non-linearity, optimize for the loss as specified in config-file.	3
SINGLE ARBI-TRARY	With the same structure having a single hidden layer, use the transfer-functions read from the config-file, optimize for the loss as specified in config-file.	2
ANY STRUC-TURE	Support of arbitrarily many hidden layers; widths and transfer functions from config-file.	4
REGULA-RIZATION	Support for L2 regularization; λ-parameter is from config-file. Should work for any configuration of the network, loss, and transfer functions.	2
Total		20

WARNING: Failure to properly explain *any* portion of your code (or to convince the reviewer that you wrote the code) can result in the loss of 5 to 20 points, depending upon the seriousness of the situation. This is an individual exercise in programming, not in downloading nor copying. A zip file containing your commented code must be uploaded to Blackboard prior to your demonstration. You will not get explicit credit for the code, but it is crucial that we have the code online in the event that you decide to register a formal complaint about your grade (for the entire course).

The 20 total points for this project are 20 of the 100 points that are available for the entire semester.