Assignment 2 (Advanced)

Note: This assignment is **mandatory** for Computing MSc and *optional* for Computing BEng/MEng.

Overview

In this assignment you will implement Neural Networks from scratch and practice writing backpropagation by yourself. The goals of this advanced assignment are as follows:

- understand **Neural Networks** and their layered architectures,
- understand and implement **backpropagation**,
- implement dropout to regularize network,
- effectively find the best hyperparameters for the Neural Network architecture.

You will implement a two-layer network in a modular way, i.e. implement the forward and backward pass of a linear layer (with ReLU activations), implement a Softmax classifier with gradients and add dropout as regularizer to the network. Finally, you will use the modules implemented to build multi-layer networks for object classification on CIFAR10 and emotion classification on FER-2013. All implementations should be completed in Python. We do **NOT** provide any code in Matlab or other programming languages.

Deliverables are the implementations to all questions and a report. Read this manual *THOROUGHLY*.

Deadline: 06/03/2018 midnight

Setup

We do recommend you work on the **Ubuntu** workstations in the lab. This assignment and all code was tested for Linux and Mac OS machines. We cannot guarantee compatability with Windows machine and cannot promise any support if you do choose to work on a Windows machine.

Working on DoC lab workstations (recommended)

You can also work from home and use the lab workstations. See this list of CSG workstations to ssh into one of the machines. You can create a shared project in `\vol\bitbucket` for your group and use GitHub/GitLab as version control.

Virtual Environemnt (recommended) We recommend using virtual environment for the project. You can use the script `install_lab_env.sh` to install the virtual environment and all needed dependencies. **Note**: This does **NOT** install TensorFlow or PyTorch. You will need to do this yourself. This is the easiest way to work either on the lab machines or locally.

Start the terminal and you can install the virtual environment as follow:

```
# Go to assignment folder
cd assignment2_advanced

# Change permissions to execute bash script
chmod 755 install_lab_env.sh

# Create virtual environment 'env' and install requirements
# This can take several minutes to finish
./install_lab_env.sh
```

You only need to execute `chmod 755 install_lab_env.sh` and `./install_lab_env.sh` once.

Once the lab environment is installed, you can work with it as follows:

```
# Go to assignment folder
cd assignment2_advanced

# Activate the virtual environment
source ./env/bin/activate

# Work on assignment ...
###############
# Work on Qx #
###############
# ... when you are done:

# Exit the virtual environment
deactivate
```

When your environment is activated, you can also install other packages with `pip`.

Setting up the CUDA environment This is only needed for the bonus `Q6`. CUDA is installed on the DoC network. You should add the following in the .bashrc file of your home folder (`~/.bashrc`).

```
export CUDA_HOME="/vol/cuda/9.0.176"
export PATH="$CUDA_HOME/bin:$PATH"
export CUDA_ROOT=${CUDA_HOME}/bin
export LD_LIBRARY_PATH="${CUDA_HOME}/lib64:$LD_LIBRARY_PATH"
export CPATH="${CUDA_HOME}/include:$CPATH"
```

Activate the settings by running `source ~/.bashrc`.

Tensorflow/PyTorch For bonus Q6, you will need to install either TensorFlow or PyTorch. Install it within your environment:

```
cd assignment2_advanced
source ./env/bin/activate

# Install Tensorflow (https://www.tensorflow.org/install/)
pip install --upgrade tensorflow-gpu

# OR

# Install PyTorch (http://pytorch.org/)
pip3 install http://download.pytorch.org/whl/cu90/torch-0.3.0.post4-
    cp35-cp35m-linux_x86_64.whl
pip3 install torchvision

deactivate
```

Once you installed everything, you can activate your environment and work on the questions as follows:

```
# Go to assignment folder
cd assignment2_advanced

# Activate the virtual environment
source ./env/bin/activate

# Work on assignment ...
###############
# Work on Qx #
###############
# ... when you are done:

# Exit the virtual environment
deactivate
```

Anaconda You are allowed to use Anaconda for Python package, dependency and environment management on the lab machines. Install Anaconda according to the installation instructions. If so, you need to give us instructions what requirements are needed to run your code.

Working locally

If you decide to work locally on your machine, then you will need to give us explicit instructions how to run your code. Anything we cannot run will result to your points of the question reduced by 30 %.

Python>=3.5: All provided code has been tested on Python versions 3.5 or 3.6. Make sure to install Python version 3.5 or 3.6 on your local machine. Otherwise, you might encounter errors and the tests might not work! If you are working on Mac OSX, you can use Homebrew to `brew install python3`.

Virtual Environment Similarly to the settings given for the CSG machines, you can create a virtual environment and install the requirements.

Anaconda You are allowed to use Anaconda for Python package, dependency and environment management. Install Anaconda according to the installation instructions.

Tensorflow/PyTorch For `Q6`, you will need to install either TensorFlow or PyTorch.

Working remotely on Google Cloud

We provide $75 credits for Google Cloud for each group intended for groups who would like to do `Q6`. We **do not** provide any instructions for setting up on Google Cloud, so it is your responsibility to set up everything. Please email Linh (linh.tran@imperial.ac.uk) for the credits and further instructions.

Working on assignment

Download the data

CIFAR-10 You can use `datasets/get_cifar10.sh` to download the dataset CIFAR10 (~ 163 MB). Alternatively, you can also download CIFAR10 at CIFAR-10 dataset. You will need it for `Q4`. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

```
cd assignment2_advanced/datasets
sh get_cifar10.sh
```

FER013 You can use `datasets/get_fer2013.sh` to download FER2013 (~ 40MB) or download it via https://www.doc.ic.ac.uk/~dlt10/Fer2013pu.zip. You will need it for `Q5`.

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

```
cd assignment2_advanced/datasets
sh get_fer2013.sh
```

Loading the data

We provide you with the function `get_CIFAR10_data()` from `src/utils/data_utils.py` to load the CIFAR data.

For FER2013 you should write your own code to load the labels and images in memory.

Q1: Linear and ReLU Layers (5 out of 100 points)

In `Q1`, you have to implement the forward and backward passes for linear layers and ReLU activations. For this question, open the file `src/layers.py` and implement the following:

1. Implement the linear function (function `linear_forward`).
2. Implement the linear backward pass (function `linear_backward`).
3. Implement the forward pass for the ReLU activation function (function `relu_forward`).
4. Implement the backward pass for the ReLU activation function (function `relu_backward`).

Note We mean by linear layer a layer which applies a linear transformation to the incoming data X ($y = Wx + b$) and ReLU activation as applying rectified linear unit function element-wise on input data X ($ReLU(X) = \max(0, X)$).

Test your implementation Once you have finished the implementations, you can test it with

```
# Go to assignment folder
cd assignment2_advanced
# activate environment
source ./env/bin/activate
# execute tests for Q1
python -m test.test_layers TestLinearLayer
python -m test.test_layers TestReLULayer
# deactivate environment
deactivate
```

These tests were written with python unittest and should help you verifying your implementations.

You should see the following output if all implementations are correct:

```
python -m test.test_layers TestLinearLayer
```

```
==========================================
Testing linear backward function:
==========================================
dX relative error:  3.97401818799e-11
dW relative error:  3.91757851635e-10
db realtive error:  4.88680423566e-12
.
==========================================
Testing linear forward function:
==========================================
Relative difference 1.08483901729e-08
.
----------------------------------------------------------------------
Ran 2 tests in 0.029s

OK
python -m test.test_layers TestReLULayer

==========================================
Testing relu backward function:
==========================================
dX relative difference: 3.27563658113e-12
.
==========================================
Testing relu forward function:
==========================================
Relative difference: 2.72727259342e-08
.
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

Otherwise, the tests will show you some failures:

```
==========================================
Testing linear backward function:
==========================================
dX relative error:  4.04756324381e-11
dW relative error:  3.91757851635e-10
db realtive error:  1.0
F
```

```
==========================================
Testing linear forward function:
==========================================
Relative difference 1.0
F
======================================================================
FAIL: test_output_linear_backward (__main__.TestLinearLayer)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/assignment2/test/test_layers.py", line 63, in
      test_output_linear_backward
    self.assertTrue(db_e <= 5e-10)
AssertionError: False is not true

======================================================================
FAIL: test_output_linear_forward (__main__.TestLinearLayer)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/assignment2/test/test_layers.py", line 36, in
      test_output_linear_forward
    self.assertTrue(e <= 5e-08)
AssertionError: False is not true

----------------------------------------------------------------------
Ran 2 tests in 0.006s

FAILED (failures=2)
```

Deliverable

In order to obtain 5 points, you must:

- Complete `linear_forward`, `linear_backward` (2.5 points)
- Complete `relu_forward`, `relu_backward` (2.5 points)

We will run further tests to verify the code you submitted is correct.

Q2: Dropout (5 out of 100 points)

In `Q2`, **inverted dropout** will be implemented. The forward (`dropout_forward`) and backward (`dropout_backward`) passes are to be implemented in `src/layers.py`.

Dropout As known from the lecture, dropout is used as regularisation in Neural Networks. Given probability p, neurons are dropped with probability p, therefore keeping neurons with probability $q = 1 - p$.

The principle of dropout implicates that every single neuron has the same probability to be turned off.

Given: * $h(X) = XW + b$ is a linear transformation of a d_i-dimensional input X in a d_h-dimensional output space.

- $a(h)$ is the activation function

the application of dropout on the i-th neuron is resulting in an output o_i:

$$o_i = m_i \cdot a(\sum_{k=1}^{d_i} w_k x_k + b) \tag{1}$$

where M = $m_1, \dots, m_{d_h}$ is a d_h-dimensional vector of Bernoulli distributed variables.

If we are applying dropout to the training phase, we need to perform a scaling of the inputs by q. This is crucial because all neurons see all their inputs at test time and we need to make sure the outputs of test neurons are identical to their expected outputs at training time. Thus:

- Train phase: $o_i = m_i \cdot a(\sum_{k=1}^{d_i} w_k x_k + b)$
- Test phase: $o_i = q \cdot a(\sum_{k=1}^{d_i} w_k x_k + b)$

Inverted Dropout As described above, the dropout scheme presented needs to scale the activations by q at test time. Since the test performance is critical, it is also preferable to leaving the forward pass unchanged at test time. Therefore, in most implementations inverted dropout is employed to overcome the undesirable property of the original dropout.

The scale factor is the inverse of the keep probability: $\frac{1}{1-p} = \frac{1}{q}$

and thus we have changed our passes for train and test to the following:

- Train phase: $o_i = \frac{1}{q} \cdot m_i \cdot a(\sum_{k=1}^{d_i} w_k x_k + b)$
- Test phase: $o_i = a(\sum_{k=1}^{d_i} w_k x_k + b)$

For a more detailed explanation of dropout and inverted dropout, have a read at Analysis of Dropout.

Test your implementation Once you have finished the implementations, you can test it with

```
# Go to assignment folder
cd assignment2_advanced
# activate environment
source ./env/bin/activate
```

```
# execute tests for Q2
python -m test.test_layers TestDropoutLayer
# deactivate environment
deactivate
```

Deliverables

In order to obtain 5 points, you must:

- Complete `dropout_forward`, `dropout_backward` in `src/layers.py`

We will run further tests to verify the code you submitted is correct.

Q3: Softmax Classifier (5 out of 100 points)

For this task, open the file `src/classifiers.py` and implement the softmax loss and gradients (function `softmax`). You are supposed to include an additional term for improving the numerical stability.

Notation The input is defined as $X = [x_i, \dots, x_N]$ with $x_i \in \mathbb{R}^d,\ i = 1, \dots, N$. The corresponding labels are defined by $Y = [y_1, \dots, y_N]$ with $y_i \in [0, \dots, C)$. Further, we define weights as $W \in \mathbb{R}^{Cx N}$ and bias as $b \in \mathbb{R}^C$.

Logits Assume $\hat{y}_i$ contains computed scores for each class, e.g. $\hat{y}_i = Wx_i + b$. In probability theory, $\hat{y}$ is interpreted as un-normalized log probabilities and are called logits. These logits are input for the softmax loss function for your implementation.

Softmax function The softmax function σ is defined by:

$$\sigma_j(\hat{y}_i) = \frac{e^{\hat{y}_i[j]}}{\sum_k e^{\hat{y}_i[k]}},\ j = 1, \dots, C \tag{2}$$

The softmax function "squashes" a C-dimensional vector $\hat{y}_i$ to a C-dimensional vector that add up to 1.

In probability theory, the output of the softmax function can be used to represent a categorical distribution and the predicted probability for the j-th class is given by:

$$P(y_i = j|\hat{y}_i) = \frac{e^{\hat{y}_i[j]}}{\sum_k e^{\hat{y}_i[k]}},\ j = 1, \dots, C \tag{3}$$

Looking at the equation, the softmax classifier can be interpreted as the (normalized) probability assigned to the correct label y_i given an input $\hat{y}$.

Softmax loss If you have a look at lecture slides 15-16 of second NN lecture, you will get the definition of the softmax function, as well as its usage as loss function: negative log likelihood cost.

Concretely, the negative log-likelihood for a single x_i is defined as:

$$L_i = -\log(\sigma_j(\hat{y}_i)[y_i]) \tag{4}$$

For a given batch X, the loss consist of the average negative log-likelihood of all samples:

$$L = \frac{\sum_{j=0}^{N-1} L_j}{N} \tag{5}$$

Why average negative log-likelihood? If you are using a first order optimisation algorithm, such as gradient ascent, using the average likelihood as you objective function stabilises the behaviour of algorithm as the sample size changes.

Numerical issues When calculating the Softmax function in practice, both $e^{\hat{y}_i[j]}$ and $\sum_k e^{\hat{y}_i[k]}$ can be very large due to the exponentials. Dividing large numbers can be numerically unstable, therefore a normalisation trick is used:

$$\sigma_j(\hat{y}_i) = \frac{e^{\hat{y}_i[j]}}{\sum_k e^{\hat{y}_i[k]}} = \frac{K \cdot e^{\hat{y}_i[j]}}{K \cdot \sum_k e^{\hat{y}_i[k]}} = \frac{e^{\hat{y}_i[j]+log(K)}}{\sum_k e^{\hat{y}_i[k]+log(K)}} \tag{6}$$

The value K can be chosen and will not change any of the results but can improve the numerical stability of the computation. A common choice for K is set to $log(K) = -\max \hat{y}_i$.

Note In the implementation you have to deal with one-hot encodings of labels instead of the labels described in this manual.

Test your implementation Once you have finished the implementations, you can test it with

```
# Go to assignment folder
cd assignment2_advanced
# activate environment
source ./env/bin/activate
# execute tests for Q3
python -m test.test_classifiers
# deactivate environment
deactivate
```

Deliverables

In order to obtain 5 points, you must:

- Complete `softmax` in `src/classifiers.py`

We will run further tests to verify the code you submitted is correct.

Q4: Fully-Connected Neural Network (10 out of 100 points)

In this task you will implement a fully-connected neural network with arbitrary number of hidden layers, ReLU activation, softmax classification and optional dropout. For this question, you will need to implement `FullyConnectedNet` in `src/fcnet.py`. This task is about reusing your implementations from `Q1` to `Q3`. In addition, you will add a L2 regularizer.

Sanity checks Once finished with the implementation, you can test your implementation with:

```
# Go to assignment folder
cd assignment2_advanced
# activate environment
source ./env/bin/activate
# execute tests for Q4
python -m test.test_fcnet
# deactivate environment
deactivate
```

This sanity check computes the initial loss for a two-layer fully-connected neural networks with different regularisation factors [0, 3.14].

Do the initial losses seem reasonable? For gradient checking, the relative errors should be around 1e-6 or less.

Solver We provide a solver for the `FullyConnectedNet` that you have implemented. Open the file `src/utils/solver.py` and read through it to familiarize yourself with the API. We also provide you SGD as optimiser to use for the solver in `src/utils/optim.py`.

Overfit the network Another sanity check is to try to overfit a small dataset (CIFAR10) of 50 images. Complete `src/overfit_fcnet.py` for this task. You will need to play with the learning rate and initialisation, but you should be able to overfit and achieve 100% training accuracy within 20 epochs.

You can run `overfit_fcnet.py` as follows:

```
cd assignment2_advanced
source ./env/bin/activate
python -m src.overfit_fcnet
```

Plotting The solver saves accuracy of training and validation which you can plot. Here is an example to do so. Include plotting of the models you train if applicable.

```
import matplotlib.pyplot as plt
# declare model and solver and train the model
# model = [...]
# solver = [...]
# solver.train()
# Run this cell to visualize training loss and train / val accuracy

plt.subplot(2, 1, 1)
plt.title(”Training loss”)
plt.plot(solver.loss_history, ”o”)
plt.xlabel('Iteration')

plt.subplot(2, 1, 2)
plt.title('Accuracy')
plt.plot(solver.train_acc_history, '-o', label='train')
plt.plot(solver.val_acc_history, '-o', label='val')
plt.plot([0.5] * len(solver.val_acc_history), 'k--')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.gcf().set_size_inches(15, 12)
plt.show()
```

Train a two-layer fully-connected networks for CIFAR10 Having finished the sanity checks, use `src/train_fcnet.py` to train a two-layered fully-connected network on CIFAR10 which achieves at least 45 % accuracy on the validation set. This should be easily achieved and you do not need to search extensively for the hyperparameters. Report the parameters used (update rule, learning rate, decay, epochs, batch size) and include the plots in your report. You can run `train_fcnet.py` as follows:

```
cd assignment2_advanced
source ./env/bin/activate
python -m src.train_fcnet
```

Deliverables

In order to obtain 10 points, you must:

- Complete `src/fcnet.py` (8 points)
- Complete `src/overfit_fcnet.py` (1 points)

- Complete `src/train_fcnet.py` (1 points)

We will run further tests to verify the code you submitted is correct.

Q5: Hyper-parameter Optimisation with FER2013 (10 out of 100 points)

In this question, you will optimise the hyper-parameters of a fully-connected neural network with FER2013.

Firstly, similarly to the given implementation of SGD, you should implement SGD with momentum. For that, open `src/utils/optim.py` and complete the function `sgd_momentum`.

Secondly, select a performance measure which will be used to compare the performance of the different parameters on the validation set.

You should use stochastic gradient descent with momentum for all experiments.

Optimise the hyper-parameters as follows:

- Select a reasonable network architecture as starting point and explain the motivation to choose so. Define a momentum and a learning rate. You can also extend the code to have a different stopping criterion or a different learning rate update schedule.
- Optimise the learning rate (disable regularisation). Explain how you found a good initial value for learning rate. Include the plot for training loss and report training and validation classification error.
- Use dropout and report if there is any improvement in the validation performance.
- You have implemented dropout and L2 as ways of regularisation. Use L2 and compare the performance with dropout.
- Optimise the topology of the network, i.e. the number of hidden layers and the number of neurons in each hidden layer.

Saving your model Save your best trained model for FER2013 in `assignment2` and provide a function to load and test the model, so that we can test the performance of your model on a secret test set. The performance of your model account to up to **10 points**.

Deliverables

- Complete `src/utils/optim.py`
- Saved best model
- Function to load and test your model

All points will be given according to the performance of your model.

The test function should be saved to `src/test.py` with the following structure:

```python
def test_fer_model(img_folder, model="/path/to/model"):
    """
    Given a folder with images, load the images (in lexico-graphical ordering
    according to the file name of the images) and your best model to predict
    the facial expression of each image.
    Args:
    - img_folder: Path to the images to be tested
    Returns:
    - preds: A numpy vector of size N with N being the number of images in
    img_folder.
    """
    preds = None
    ### Start your code here
    ### End of code
    return preds
```

Q6: Do something extra! (15 bonus points!)

Note Even though you can get up to 15 bonus points for this question, beware that the maximum number of points you can reach is *100*!

For this question, you have to train a Convolutional Neural Network (CNN) on the FER2013 image dataset. You can use either Tensorflow or PyTorch deep learning libraries. You can also use any higher level API with the above backend (e.g., Keras). You are expected to justify the choice of the hyperparameters (e.g., depth, filters, stride, zero-padding).

Tip: While searching for the "optimal" architecture, keep the sample complexity of your model in mind!

Deliverables

- Test the performance of the network with the optimal set of parameters on the test set and report the confusion matrix, classification rate and F1 measure per class.(2 points)
- Compare the performance of your CNN with the feedforward NN in the previous question. Justify the results. (7 points)
- You should provide a function to load and test your model, along with clear instructions on how to use it. You will be graded based on the performance of your model on our private test set (6

points).

The test function should be saved to `src/test.py` with the following structure:

```
def test_deep_fer_model(img_folder, model="/path/to/model"):
    """
    Given a folder with images, load the images (in lexico-graphical
       ordering
    according to the file name of the images) and your best model to
       predict
    the facial expression of each image.
    Args:
    - img_folder: Path to the images to be tested
    Returns:
    - preds: A numpy vector of size N with N being the number of
       images in
    img_folder.
    """
    preds = None
    ### Start your code here
    ### End of code
    return preds
```

In case your code does not run, 2 points will be deducted.

Report (55 out of 100 points)

We expect you to submit a report with the *theoretical* solution to the questions. Do not explain code, but rather explain how forward and backward passes of certain layers and classifiers work, how you put your network together (i.e. architecture) and what strategies you used for hyper-parameter optimisation. Further, all results should be reported and described in the report. Finally, please limit your report to **no more than 5 pages of text**.

The report should include:

- Q1: Explain the forward and backward pass of linear layers and relu activations. (2 points)
- Q2: Explain the forward and backward pass of dropout. (2 points)
- Q3: Explain the computation of softmax and its gradient (2 points)
- Q4:
  - Architecture + parameters used to overfit a small subset of the CIFAR-10 data, plots of loss and accuracy of train and test set (2 points)

  - Architecture + parameters used to achieve at least 50 % accuracy on CIFAR-10 as well as plots of loss and accuracy of train and test set (2 points)
- Q5
  - Explanation of how the network was fine-tuned, what strategies were employed. Include plots of accuracy, loss and F1 if appropriate. (6 points for each step , 30 points in total)
  - Test the performance of the network trained with the optimal set of parameters on the test set and report the confusion matrix, classification rate and F1 measure per class. (5 points)

Two additional questions need to be also answered in the report:

A1. Assume that you train a neural network classifier using the dataset of the previous coursework. You run cross-validation and you compute the classification error per fold. Let's also assume that you run a statistical test on the two sets of error observations (one from decision trees and one from neural networks) and you find that one algorithms results in better performance than the other.
Can we claim that this algorithm is a better learning algorithm than the other in general? Why? Why not? (5 points)

A2. Suppose that we want to add some new emotions to the existing dataset. What changes should be made in decision trees and neural networks classifiers in order to include new classes? (5 points)

Quality of Presentation (10 out of 100 points)

You can get up to 10 points for the quality and presentation of your report.

Summary of Grading

You can get a total of **100 points**:

- Q1 (5 points)
- Q2 (5 points)
- Q3 (5 points)
- Q4 (10 points)
- Q5 (10 points)
- Q6 (15 bonus points)
- Report (55 points)
- Quality of presentation (10 points)

Submit your work

Submit your code

You need to zip your entire folder `assigment2_advanced` and submit this and your report. Name it:

`assignment2_advanced_code_team_<#team>.zip`

Errors and Feedback

Did you find an error and/or typo in this document? Or you have any feedback, recommendations or comments on this new assignment? Please email Linh (Q1-Q5) or Markos (Q6), every feedback is welcomed!

Acknowledgement

We are grateful to entire Stanford Vision Lab for their course CS291n: Convolutional Neural Networks for Visual Recognition, all their notes and assignments. We reused some of their scripts for this course. Overall, it served as guidance and helped a lot to design this assignment.