

Assignment (1)

24789 - Special Topics: Deep Learning for Engineers (Spring 2020)

Author: Yuyang Wang, Guilin Zhang, Amir Barati

Out Date: 2020/1/27 (Mon)

Due Date: 2020/2/16 (Sun) @ 11:59 pm EST

All exercises should be submitted to [Gradescope](#). There are 2 assignment sections to submit in Gradescope. The first is **HW1**, where you submit a pdf file containing your answers to 4 theory exercises. The second is **HW1-programming**, where you submit your codes, saved model as well as a brief report for 2 programming exercises. The report should include the structure of your model, the hyper-parameters, visualization of your training process and the final accuracy/performance you achieve. For **HW1-programming**, you should submit a zip file as the following structure:

Submission file structure

```
andrewID-HW1
├── p1
│   ├── p1.py
│   ├── p1_model.npz
│   └── p1_report.pdf
└── p2
    ├── p2.py (you may include other .py files)
    ├── p2_model.pkl
    └── p2_report.pdf
```

You can refer to [Python3 tutorial](#), [Numpy documentation](#) and [PyTorch documentation](#) while working on this assignment. Any deviations from the submission structure shown below would attract penalty to the assignment score. Please use [Pizza](#) for any questions on the assignment.

Theory Exercises (50 points)

PROBLEM 1

Gradient Descent (12 points)

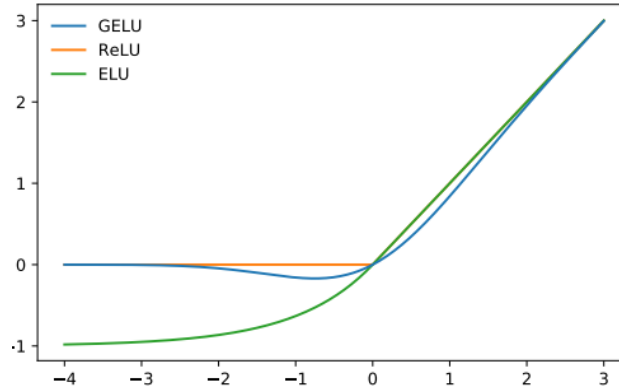


Fig. 1: ReLU, ELU and GELU activation functions

We now introduce GELU (Gaussian Error Linear Units), a high-performing neural network activation function. Unlike ReLU function, GELU non-linearity weights inputs by their magnitude rather than gates inputs purely by their sign, which can be approximates by function given in Eq. 2. As shown in Fig. 1, the blue line indicates GELU function and how it is different from ReLU (Rectified Linear Unit) and ELU (Exponential Linear Unit) function as given in Eq. 3 and 4.

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

$$\text{GELU}(x) \approx x\sigma(1.702x) \quad (2)$$

$$\text{ReLU}(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad (3)$$

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad (4)$$

And now you want to find the minimal of GELU function using gradient descent (GD).

(hint: the derivative of sigmoid function (Eq. 1) is given as $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$)

a) Suppose your initial guess is $x_0 = 0$ where $\text{GELU}(x_0) = 0$. With step size $\eta = 0.1$, perform one-round of GD to get x_1 . Continually, using GD to calculate x_2 and x_3 . And calculate $\text{GELU}(x_1)$, $\text{GELU}(x_2)$ and $\text{GELU}(x_3)$. (3 points)

b) This time, you use step size $\eta = 1$. With the same initial guess $x_0 = 0$, perform GD to get x_1 , x_2 and x_3 . How is the performance different with using $\eta = 0.1$? (3 points)

c) Gradient Descent with Momentum considers the past gradients to smooth out the update. It computes an exponentially weighted average of your gradients, and then use that gradient to update your weights instead. Generally it works faster than the standard gradient descent algorithm. The update rule of GD with Momentum is given in Eq. 5.

$$\begin{aligned}
 v_0 &= \nabla f_{x_0} \\
 v_{t+1} &= \beta v_t + (1 - \beta) \nabla f_{x_t} \\
 x_{t+1} &= x_t - \eta v
 \end{aligned}
 \tag{5}$$

where f is your objective function and in our case is GELU function.

Now, with initial guess $x_0 = -3$ and step size $\eta = 0.1$,

- Calculate 3 updates using Gradient Descent. Namely, you are supposed to calculate x_1, x_2, x_3 and corresponding $\text{GELU}(x_1), \text{GELU}(x_2), \text{GELU}(x_3)$.
- Calculate 3 updates using Gradient Descent with Momentum ($\beta = 0.9$). Namely, you are supposed to calculate x_1, x_2, x_3 and corresponding $\text{GELU}(x_1), \text{GELU}(x_2), \text{GELU}(x_3)$.
- How do the performance of these 2 methods differ?

(6 points)

PROBLEM 2

Neural Network Concepts (12 points)








Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\overline{A}	AB	\overline{AB}	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

Fig. 2: Logic Gates

Electronic circuits in computers are made up of thousands of logic gates, which takes binary inputs (0 and 1 only) and produce a binary output. Shown in Fig. 2 are common logic gates. And now you are asked to design a MLP (Multi-layer Perceptron) to achieve the function of different logic gates. To this end, the binary operations are considered as classification problems: given A (and B), outputs the corresponding 0 or 1 as shown in Truth Table of Fig. 2. Sigmoid function as given in Eq. 1 is used as activation function. And your MLP will classify the A (and B) as 0 if the output is smaller than 0 and 1 if greater or equals to 0.

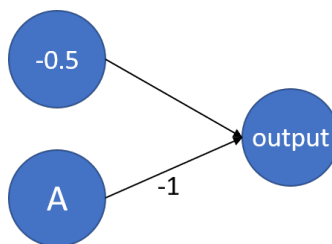


Fig. 3: Example of MLP to achieve NOT Gate

Please design the smallest MLP structures to accomplish **AND**, **OR**, **XOR** and **XNOR** gates. You should consider how many layers are at least required and how many neurons are needed in each layer. An example of NOT gate is shown in Fig. 3, where the -1 on the edge represents multiplying with A and -0.5 is a bias added to $-1 \times A$.

PROBLEM 3

Back Propagation (10 points)

Consider a 2-layer neural network as shown in Fig. 4. The configurations of the neural network are as follows:

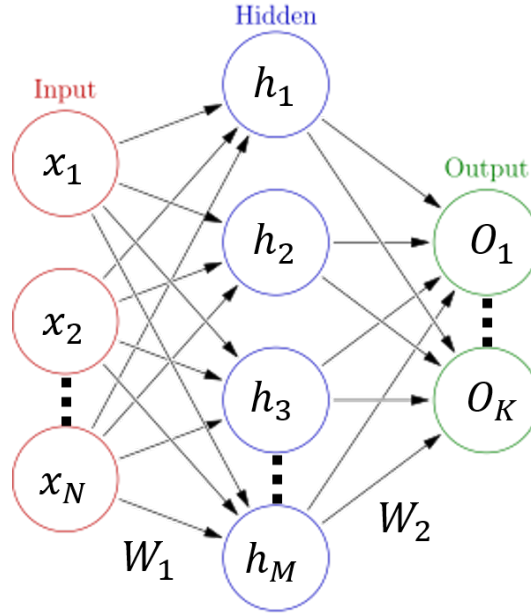


Fig. 4: A MLP with one hidden layer

- x is the input data with dimension $1 \times N$.
- h is the hidden layer with dimension $1 \times M$.
- o is the output layer with dimension $1 \times K$.
- Sigmoid function σ as given in Eq. 1 is used as activation for the hidden layer.
- Softmax function S as given in Eq. 6 is used as activation for the output layer.
- W_1, W_2 are weight matrix with shape $N \times M$ and $M \times K$ respectively.
- b_1, b_2 are biases with shape $1 \times M$ and $1 \times K$ respectively.

$$S(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (6)$$

Further, you want to use this neural network to address classification problems. And cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. The cross-entropy loss of our output o is given as

$$E(o) = - \sum_1^K y_i \log o_i \quad (7)$$

where $y \in \{0, 1\}$ is the ground truth label.

Please use the notions given below in your derivation:

$$f_1 = xW_1 + b_1$$

$$a = \sigma(f_1)$$

$$f_2 = aW_2 + b_2$$

$$o = S(f_2)$$

a) Use back propagation, derive the gradient with respect to f_2 . (5 points)

(hint: You should simplify the results using o and y)

b) Use back propagation and results from question a), derive the gradient with respect to x . (5 points)

PROBLEM 4

Convolutional Neural Network (16 points)

Suppose in one layer of a CNN (Convolutional Neural Network), you get a feature map F and want to conduct convolution using the filter f as given in Fig. 5.

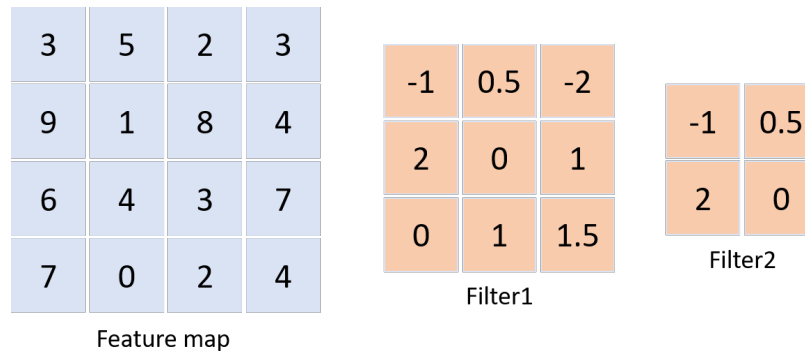


Fig. 5: Feature map and filters

a) Now use Filter1, you want to get a new feature map F' with the same size as the original one F after convolution with zero padding. With stride $s = 1$, what is the padding size? And calculate the new feature map. (4 points)

b) This time use Filter2 with padding size $p = 1$, is it possible to get a new feature map with the same size as the original. If yes, what is the stride s and calculate the new feature map? If no, briefly explain the reason. (4 points)

c) With the new feature map F' calculated in question a), calculate the feature map after average pooling. The pooling size is 2 and stride $s = 2$. (4 points)

d) Pooling as a down-sampling operation can be achieved with purely convolutional layers as well. For average pooling, design the filter that has the same performance as question c) and specify stride and padding size if needed. (4 points)

Programming Exercises (50 points)

PROBLEM 1

MLP and Back Propagation (25 points)

In this section, you are asked to implement a Multi-Layer Perceptron with an API similar to PyTorch. Please follow the given template `p1_template.py` to build your own codes. Remember no third-party deep learning package (PyTorch, Tensorflow, MXNet, etc.) is allowed in this section. The only three python libraries to import are `os`, `numpy` and `matplotlib`.

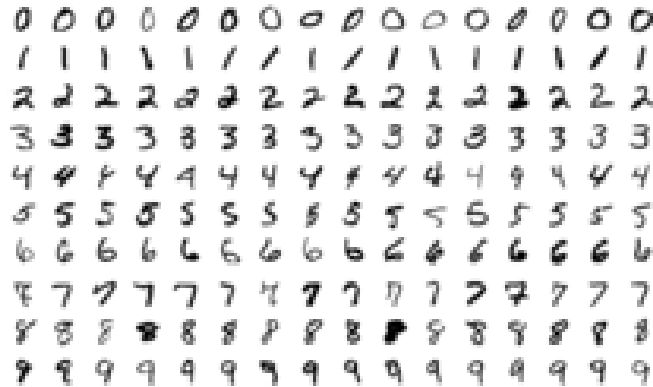


Fig. 6: MNIST dataset

The problem you are asked to solve is MNIST hand-written digits classification as shown in Fig. 6. Namely, given a 28 by 28 image (784 dimensional vector), your model is supposed to correctly classify it. Please download the training set: https://www.python-course.eu/data/mnist/mnist_train.csv and the test set: https://www.python-course.eu/data/mnist/mnist_test.csv. You are able to read the data into `numpy.array` using the code given in the template.

The classes/functions your are supposed to implement are as follows:

- a) Activation function (3 points): the forward pass and back propagation of different activation non-linearities:
 - **Sigmoid** non-linearity
 - **Tanh** non-linearity
 - **ReLU** non-linearity
- b) Softmax Cross-entropy Loss (5 points): the forward pass and back propagation. You can refer to <https://deepnotes.io/softmax-crossentropy#derivative-of-cross-entropy-loss-with-softmax> for definition and derivation of Softmax Cross-entropy Loss. Also, Problem 3 in Theory Exercise can help.
- c) Weight initialization (2 points)
 - `random_normal_weight_init`: initialize weight matrix W from standard normal distribution.
 - `zeros_bias_init`: initial bias vector b from zeros.
- d) Multi-layer Perceptron (10 points): the class which implements the Multi-layer Perceptron. The class functions you need to implement is given:
 - `forward`: forward pass of input to get the output of the model.

- **backward**: back-propagation given label, namely calculate the derivatives of all parameters.
 - **step**: update the parameters (weight matrices and bias) based on gradient calculated in **backward**.
 - **zero_grads**: set all gradients to zero.
 - **get_loss**: return value of loss functions given labels.
 - **get_error**: return the number of incorrect predictions given labels.
- e) Train your MLP (5 points): Here's the most exciting section! After you finish the codes concerning building MLP model and defining loss criterion, you are asked to actually train and test your MLP model using **get_training_stats** function. You should report your final test error in your submission and credits for this section will be graded on this.

- Error < 5%: 10 points (including 5 extra points)
- $5 \leq \text{Error} < 10\%$: 5 points
- $10 \leq \text{Error} < 20\%$: 3 points
- Error $\geq 20\%$: 0 points

And Fig. 7 shows an example of what you should get after your codes successfully work and train on MNIST dataset. Note that you are not required to get the same number of loss/error during your training. The figure is only to illustrate how the plot looks like.

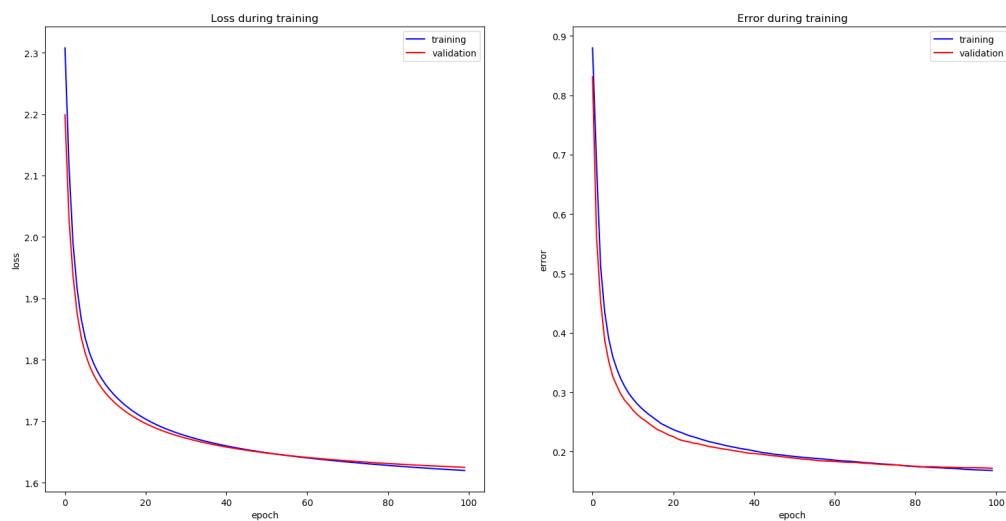


Fig. 7: Example of training plot

Some hyper-parameters to tune to improve the accuracy of your model:

- batch size
- learning rate
- number of epochs
- number of neurons/hidden layers
- use GD with momentum
- ...

You should submit a report **p1_report.pdf**, your code **p1.py** and your trained model saved as **p1_model.npz**

PROBLEM 2

CIFAR-10 Classification via CNN (25 points)

The CIFAR-10 dataset: <https://www.cs.toronto.edu/~kriz/cifar.html> is a popular image classification dataset for algorithm benchmark. It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class, where are split into 50000 training images and 10000 test images. All categories and some samples from each category are shown in Fig. 8.

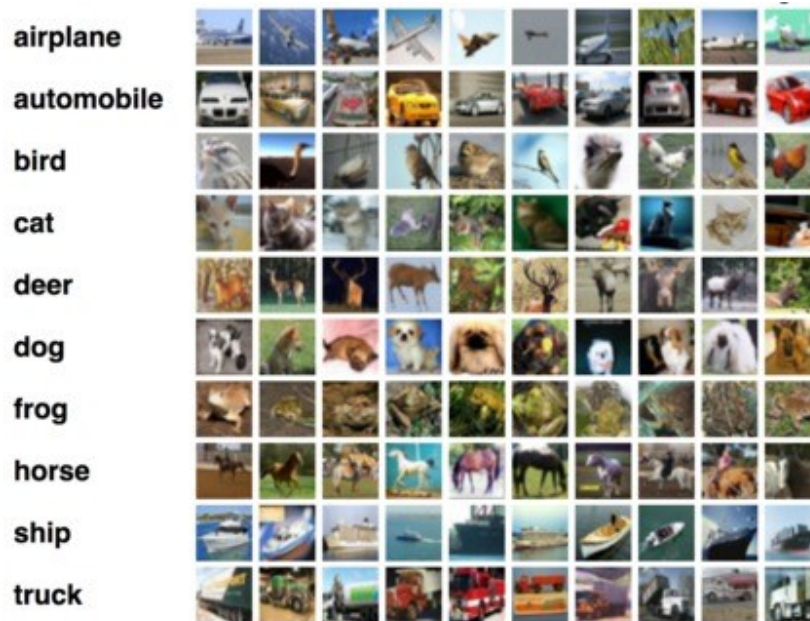


Fig. 8: CIFAR-10

And your task in this section is to build a CNN for image classification and train your model on CIFAR-10 dataset. You are recommended to use **PyTorch** as the deep learning API. Other frameworks like **Tensorflow** or **MXNet** are allowed, but it is likely that there will be little support from TA if frameworks other than **PyTorch** are used. Please refer to the tutorial from **PyTorch** official: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html to get familiar with **PyTorch** and learn how to load dataset. If you are new to **PyTorch**, it is recommended to follow the structure in the tutorial to build your CNN model, loss function and training function.

For this sections, 20 out of 25 points are graded on your code and report. And 5 points on the performance of your model. Tentative cutoffs for test accuracy:

- > 90%: 15 points (including 10 extra points)
- > 85%: 10 points (including 5 extra points)
- > 80%: 5 points
- > 75%: 3 points
- $\leq 75\%$: 0 points

You should submit a report **p2_report.pdf**, your codes **p2.py** (your are encouraged to split your codes into different modules, but make sure to submit all your **.py** files to Gradescope) and your trained model saved as **p2_model.pkl** (you can refer to this documentation for saving pytorch model: https://pytorch.org/tutorials/beginner/saving_loading_models.html).