CIS680: Vision & Learning
Assignment 1: Deep Learning Basics
Due: Sep. 25, 2018 at 11:59 am

Instructions

- This is an **individual** assignment. "**Individual**" means each student must hand in their **own** answers, and each student must write their **own** code in the homework. It is admissible for students to collaborate in solving problems. To help you actually learn the material, what you write down must be your own work, not copied from any other individual. You must also list the names of students (maximum two) you collaborated with.
- All the code should be written in Python. You should use either Tensorflow or PyTorch to complete this homework.
- Four datasets are needed in this homework, namely, "random dataset", "line dataset", and "detection dataset" and "emoji dataset'. They can be downloaded from the course wiki website. Course wiki also has a template code for the first question.
- You must submit your solutions online on **Canvas**. Compress your files into a ZIP file named "1_<penn_key>.zip", which should contain **1 PDF report** and **6 Python files**, each of which contains the Python code for each part. Note that you should include all the figures in your report.

Overview

This homework aims at investigating the basics of neural networks, i.e., activation and loss functions (non-linearity and objectives). As discussed in class, changing activation and loss functions will highly influence the efficiency of learning through back-propagation. And even sometimes, they are coupled together. There are 6 parts of the homework:

1. The perceptron algorithm is arguably the basis of deep learning. Understand its dual form by implementing the learning algorithm on a toy task.

2. Build visualization tools to plot activations, losses, and gradients. It is the first step of understanding a function through its energy landscape of outputs and gradients. From the gradient response, it is possible to predict how it will work in a deep network.

3. Experiment with a fully connected network on a random (toy) dataset. In this part, we move one step forward: from theory to experiment. To start with, we use a rather simple network to experiment with different activation and loss functions.

4. Experiment with a convolutional network on two toy datasets, namely, line dataset and detection dataset. In this part, we experiment how the network can learn to interpret image patterns. Also, with little revision, the network can also predict attributes of an object.

5. After you are equipped with tools, it is time to try something new! Implement a new activation function. You can look for papers that propose novel activation functions, or just invent one by yourself. Analyze the activation function using previously developed tools.

6. To understand how the neural network is trained, it is helpful to visualize all of the aspects of the process. Re-produce the experiment that was shown in class. We will visualize the space into which the first layer of the network transforms the data to understand how the non-linear warping helps classification

7. Finally, try to download an run one of the popular deep learning packages. We will look at object detection and segmentation and try to use the package on everyday images.

1 Dual Perceptron for Learning to classify Emojis (5%)

In this question you are asked to build a single layer perceptron. Recall from class, the optimization objective of a binary perceptron classifier $f(\mathbf{x}) = 1$ if $\mathbf{w}^T\mathbf{x} + \mathbf{b} \geq 0$ else $f(\mathbf{x}) = -1$. The an online update algorithm is given on class (slide from lecture 0).

1. (5%) Now we want you to implement the dual form perceptron that optimizes over weights α_i, $i \in [0, n-1]$ for each of the n training samples. You need to implement the forward propagation and the backward parameter update rule by hand (without using any deep learning computation framework that does backprop for you). You will be working on a toy dataset with 20 emojis, classifying them as smiling ($y = 1$) or none-smiling ($y = -1$). After your algorithm converges, take a look at the training samples

that are assigned with the largest weight. What do you notice? You will have a template code in which you need to implement `def forward()` and `def backward()`. Details about I/O will be specified in the file. *Hint*: The weight and bias term of the separating hyperplane $(\mathbf{w}, \mathbf{b})$ can be represented as a linear combination of the training samples.

2 Plot Loss and Gradient (20%)

In this part, you will write code to plot the output and gradient for a single neuron with Sigmoid activation and two different loss functions. As shown in Figure 1, You should implement a single neuron with input 1, and calculate different losses and corresponding error.

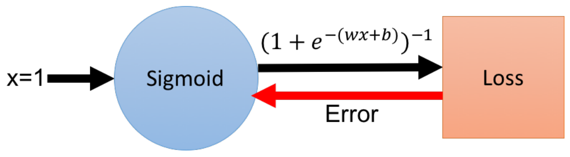

Figure 1: Network diagram for part 1.

All the figures plotted in this part should have the same range of x-axis and y-axis. The range should be centered at 0 but the extend should be picked so as to see the difference clearly.

A set of example plots are provided in Figure 2. Here we use ReLU (instead of Sigmoid) activation and L2 loss as an example.

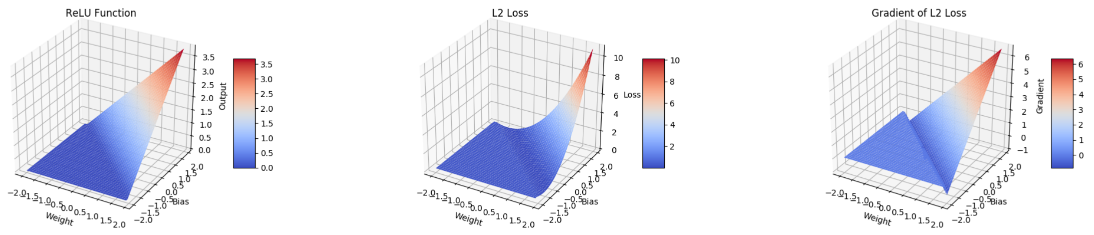

Figure 2: Example plots with ReLU activation and L2 loss. Left: Output of ReLU function. Middle: Loss plot with L2 loss. Right: Gradient plot.

1. (3%) Plot a 3D figure showing the relations of output of Sigmoid function and

weight/bias. To be specific, x-axis is weight, y-axis is bias, and z-axis is the output.

Hint: Use the Python package *matplotlib* and the function *plot_surface* from *mpl_toolkits.mplot3d* to draw 3D figures.

2. (3%) Experiment with L2 loss. The L2 loss is defined as $\mathcal{L}_{L2} = (\hat{y} - y)^2$, where y is the ground truth and $\hat{y}$ is the prediction. Let $y = 0.5$ and plot a 3D figure showing the relations of L2 loss and weight/bias. To be specific, x-axis is weight, y-axis is bias, and z-axis is the L2 loss.

3. (4%) Experiment with back-propagation with L2 loss. Compute $\frac{\partial \mathcal{L}_{L2}}{\partial weight}$ and plot 3D figure showing the relations of gradient and weight/bias. To be specific, x-axis is weight, y-axis is bias, and z-axis is the gradient w.r.t. weight.

4. (3%) Experiment with cross-entropy loss. The cross-entropy loss is defined as $\mathcal{L}_{CE} = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$, where y is the ground truth probability and $\hat{y}$ is the predicted probability. Let $y = 0.5$ and plot a 3D figure showing the relations of cross-entropy loss and weight/bias. To be specific, x-axis is weight, y-axis is bias, and z-axis is the cross-entropy loss.

5. (4%) Experiment with back-propagation with cross-entropy loss. Compute $\frac{\partial \mathcal{L}_{CE}}{\partial weight}$ and plot 3D figure showing the relations of gradient and weight/bias. To be specific, x-axis is weight, y-axis is bias, and z-axis is the gradient w.r.t. weight.

6. (3%) Explain what you observed from the above 5 plots. The explanation should include: 1) What's the difference between cross-entropy loss and L2 loss? 2) What's the difference between the gradients from cross-entropy loss and L2 loss? and 3) Predict how these differences will influence the efficiency of learning.

3 Experiment with Fully Connected Network (20%)

Starting from this question, you should use a deep learning framework. In this part, you will extend the theoretical analysis in the first part to quick experiments on a random toy dataset. By doing experiments, you will experience how different activation functions and loss functions influence the learning (or convergence) of a simple neural network. Construct a simple neural network as shown in Figure 3.

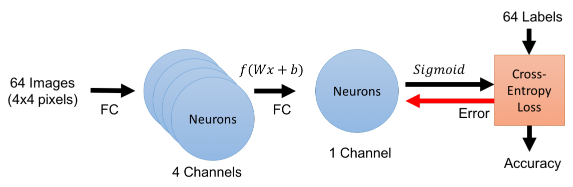

Figure 3: Network diagram for part2

The random toy dataset contains 64 images, each of which is of resolution 4×4. The labels is $\in \{0, 1\}$. For all the experiments in this part, use simple gradient descent method with learning rate 0.1. Weights are initialized with truncated normal distribution with mean 0 and standard deviation 0.1. Biases are initialized with constant of 0.1. Batch size is set to 64.

Note: The maximum number of training iterations is $10,000$.

1. (4%) Use Sigmoid function as nueron activation and L2 loss for the network. Plot two figures showing 1) loss vs training iterations, and 2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.

2. (4%) Use Sigmoid function as nueron activation and cross-entropy loss for the network. Plot two figures showing 1) loss vs training iterations, and 2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.

3. (4%) Use ReLU (Rectified Linear Units) function as nueron activation and L2 loss for the network. (Change the activation function of the first fully connected layer to ReLU.) layer Plot two figures showing 1) loss vs training iterations, and 2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.

4. (4%) Use ReLU function as nueron activation and cross-entropy loss for the network. Plot two figures showing 1) loss vs training iterations, and 2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.

5. (4%) Sort the settings according to the training iterations to reach 100% accuracy, and explain the reasons of different convergence rates.

4 Solving XOR with a 2-layer Perceptron (20%)

In this question you are asked to build and visualize a 2-layer perceptron that computes the XOR function. The network architecture is shown in Figure 4. The MLP has 1 hidden layer with 2 neurons. The activation function used for the hidden layer is the hyperbolic tangent function. Since we aim to model a boolean function the output of the last layer is passed through a sigmoid activation function to constrain it between 0 and 1.

1. (5%)Formulate the XOR approximation as an optimization problem using the cross entropy loss. *Hint: Your dataset consists of just 4 points,* $\mathbf{x}_1 = (0, 0)$*,* $\mathbf{x}_2 = (0, 1)$*,* $\mathbf{x}_3 = (1, 0)$ *and* $\mathbf{x}_4 = (1, 1)$ *with ground truth labels 0, 1, 1 and 0 respectively.*
2. (10%)Use gradient descent to learn the network weights that optimize the loss. Intuitively, the 2 layer perceptron first performs a nonlinear mapping from $(x_1, x_2) \to (h_1, h_2)$ and then learns a linear classifier in the (h_1, h_2) plane. For different steps during training visualize the image of each input point $\mathbf{x}_i$ in the (h_1, h_2) plane as well as the decision boundary (separating line) of the classifier.
3. (5%)What will happen if we don't use an activation function in the hidden layer? Is the network be able to learn the XOR function? Justify your answer.

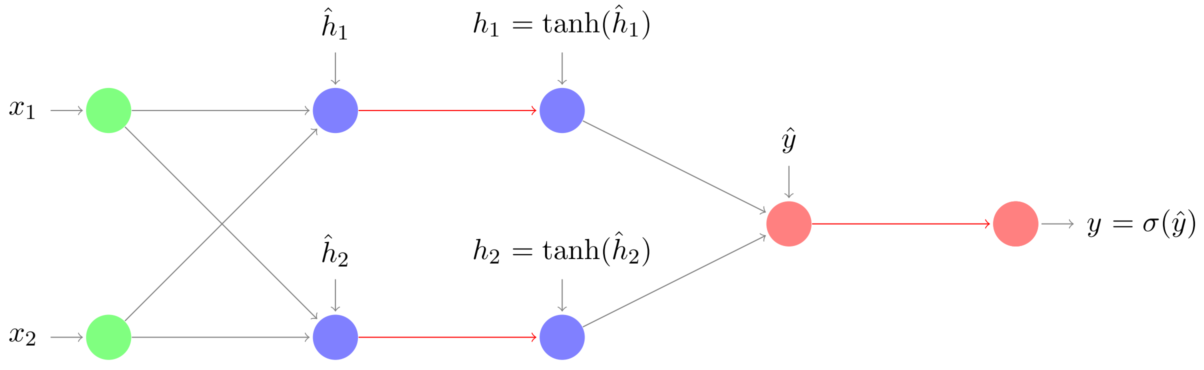

Figure 4: Graphical representation of the 2-layer Perceptron

5 Experiment with Convolutional Network (15%)

In this part, you will explore the convolutional networks. By doing experiments, you will experience how convolutional networks learn to interpret images and capture meaningful structure. Construct a convolutional network as shown in Figure 5.

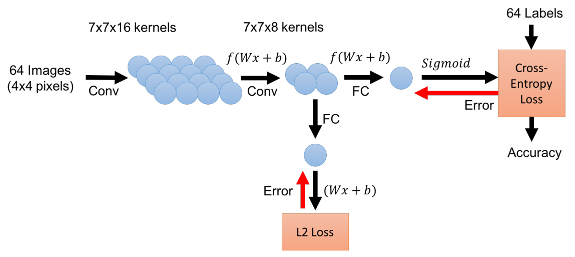

Figure 5: Network diagram for part3

The experiments will be done on two toy datasets (line dataset and detection dataset), each of which contains 64 images of resolution 8×8. The labels is $\in \{0, 1\}$. For all the experiments in this part, use simple gradient descent method with learning rate 0.1. Weights are initialized with truncated normal distribution with mean 0 and standard deviation 0.1. Biases are initialized with constant of 0.1. Batch size is set to 64.

Note: The maximum number of training iterations is $10,000$.

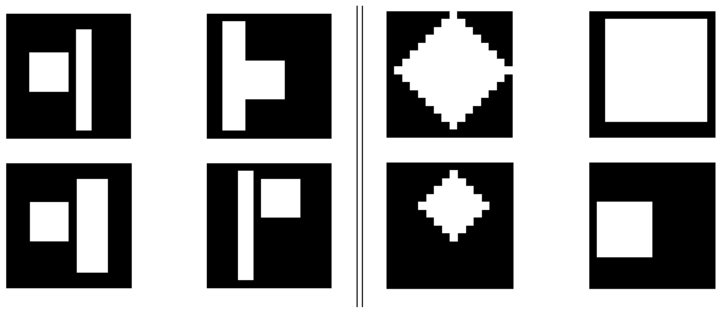

Figure 6: **Left**: 4 sample images from the line dataset. The left column corresponds to label 0, as the right column 1. **Right**: 4 sample images from the detection dataset. The left column corresponds to label 0, as the right column 1. The width of each sample is $7, 6, 4, 3$.

1. (10%) Experiment with the line dataset (Figure 6). You will construct a network with two convolutional layers and one fully connected layer, concatenated with a cross-entropy loss. (A standard classification network.)

   Use ReLU as the activation functions of convolutional layers. Plot two figures showing 1) loss vs training iterations, and 2) accuracy vs training iterations. Stop the training when accuracy reaches 100%. Compare the results with part 2.

2. (5%) Experiment with the detection dataset (Figure 6). Use the same network archi-

tecture from the previous questions. In addition, add one more fully connected layer on top of the convolutional layer. This fully connected layer (with linear activation) acts as a regressor, followed by an L2 loss. It is used to predict the width of the object. The learning rate for L2 loss is set to 0.001. We define the regression prediction is correct if the predicted width is within 0.5 of the ground truth width.

Use ReLU as the activation functions of convolutional layers. Plot four figures showing 1) cross-entropy loss vs training iterations, 2) classification accuracy vs training iterations, 3) L2 loss vs training iterations, and 4) regression accuracy vs training iterations. Stop the training when regression accuracy reaches 100%.

Compare the results with the previous question.

6 Customize Activation Function (10%)

In this part, implement one customize activation function and re-do the experiments of part 1 and part 3 using the exact same setting except activation functions.

You are welcome to invent a new activation function and test it out. Or you can implement one novel activation function that is proposed in the literature. Some candidate activation functions include variants of ReLU [5], e.g., leaky ReLU, parametric ReLU, randomized ReLU, or concatenated ReLU [4]. You are also welcome to investigate exponential linear units (ELU) [1] or scaled exponential linear units (SELU) [3].

1. (5%) Implement one customized activation function and re-do the analysis of part 1. Plot three 3D figures showing 1) output, 2) loss, and 3) gradient versus weight/bias.

2. (5%) Re-do the experiment in part 3. Plot all relevant figures (6 in total). Compare the customized activation function against ReLU. Explain the reasons of different convergence rates.

7 Using Mask-RCNN package (10%)

In this question you are asked to install and use the Mask-RCNN network. The recommended implementation is `https://github.com/matterport/Mask_RCNN`, which is in

TensorFlow. You are welcome to use any other implementation. Mask-RCNN [2] is a deep convolutional network for object detection and segmentation.

1. (5%) Install Mask-RCNN and its requirements. The recommended way is to use Conda or another virtual environment tool to create a clean environment for the requirements. For the recommended implementation, you'll need to:

   (a) (Optional) Install conda. After this, you can create and activate a new environment by typing:

   ```
   conda create -n mask_rcnn pip python=3
   source activate mask_rcnn
   ```

   (b) Follow the installation instructions 2, 1, 3 (in this order) from the github page. *Hint: if you use Conda, your* pip3 *and* python3 *are probably pointing to native installations. Use* pip *and* python *(without the 3) commands instead.*

   (c) Install pycocotools as in the step 5. *Hint: you will need to add the path to* pycocotools *utils to your* $PATH *variable.*

   (d) Test that everything is working e.g. by running the test notebook at samples/demo.ipynb.

2. (5%). Take a couple pictures with your phone and run in through the network. *Hint: you can use the* samples/demo.ipynb *notebook*. Find some objects that the network misclassifies with high confidence, as in Fig. 7. Why does it misclassify the objects?

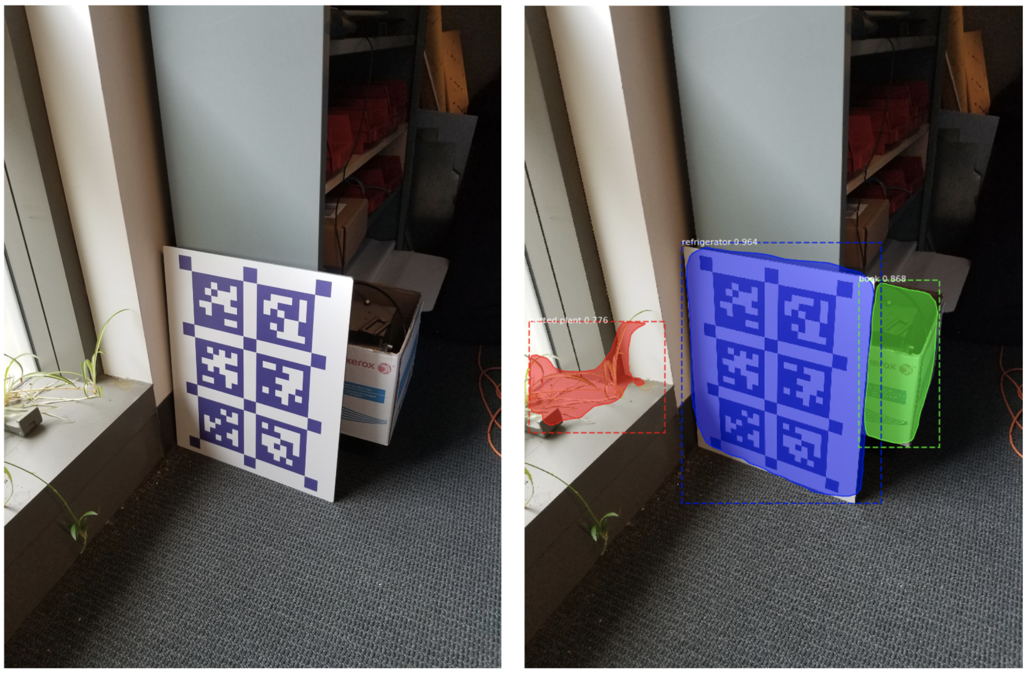

Figure 7: Misclassified objects from the GRASP lab. **Left**: the original photo. **Right**: the network output.

References

[1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[3] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

[4] W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning*, pages 2217–2225, 2016.

[5] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.