CIS680: Assignment 1: Deep Learning Basics

Due: Part (a) Sept. 8 at 11:59 PM
Part (b) Sept. 15 at 11:59 PM

Instructions

- Part (a) consists of parts 1, 2 and 3 and it is due on September 8 at 11:59 PM EDT.
- Part (b) consists of part 4 and it is due on September 15 at 11:59 PM EDT.
- 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.
- There is no single answer to most problems in deep learning, therefore the questions will often be underspecified. You need to fill in the blanks and submit a solution that solves the (practical) problem. Document the choices (hyperparameters, features, neural network architectures, etc.) you made in the write-up.
- All the code should be written in Python. You should use PyTorch only to complete this homework.

1 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.

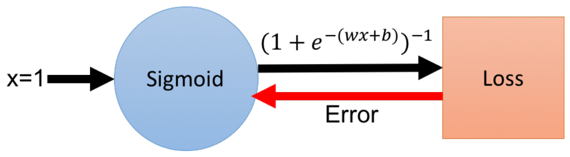


Figure 1: Network diagram for part 1.

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.

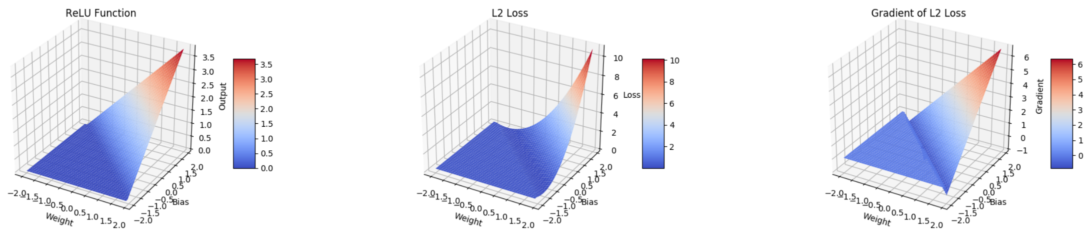


Figure 2: Example plots with ReLU activation and L2 loss. Left: Output of ReLU function. Middle: Loss plot with L2 loss. Right: Gradient plot.

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.

2 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 3. 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 $(x_1, x_2) \rightarrow (h_1, h_2)$ 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.

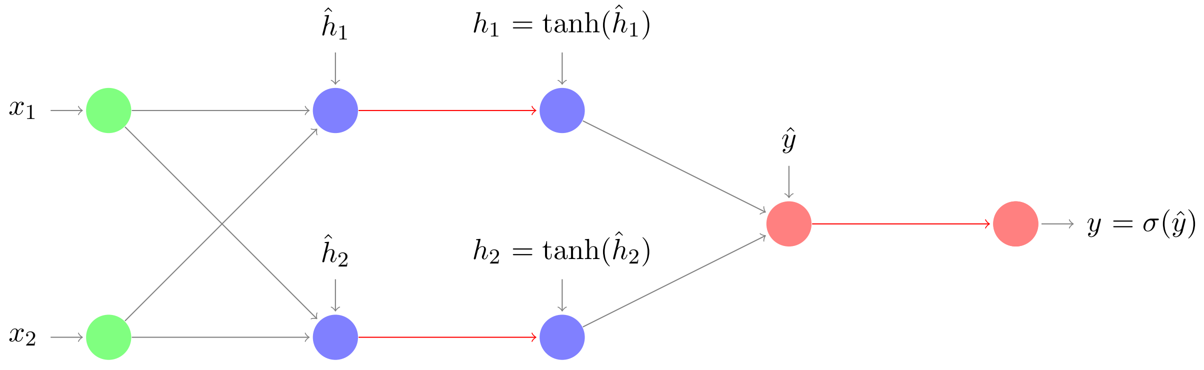


Figure 3: Graphical representation of the 2-layer Perceptron

Layers	**Hyper-parameters**
Convolution 1	Kernel size = (5, 5, 32), SAME padding. Followed by BatchNorm and ReLU.
Pooling 1	Average operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 2	Kernel size = (5, 5, 32), SAME padding Followed by BatchNorm and ReLU.
Pooling 2	Average operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 3	Kernel size = (5, 5, 64), SAME padding Followed by BatchNorm and ReLU.
Pooling 3	Average operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Fully Connected	Output channels = 64. Followed by BatchNorm and ReLU.
Fully Connected	Output channels = 10. Followed by Softmax.

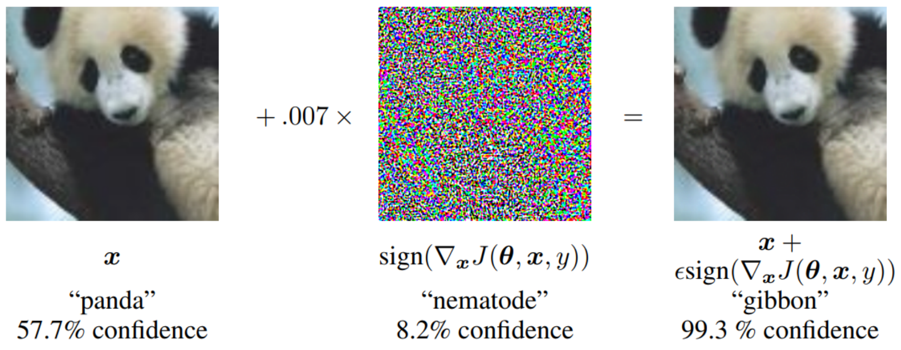
Table 1: Network architecture for part 1.

3 Train a Convolutional Neural Network (30%)

In this part you will be asked to train a convolutional neural network on the MNIST dataset.

1. (10%) Build a Convolutional Neural Network with architecture as shown in Table 1.
2. (20%) Train the CNN on the MNIST dataset using the *Cross Entropy* loss. Report training and testing curves. Your model should reach 99% accuracy on the test dataset. (Hint: Normalize the images in the (-1,1) range and use the Adam optimizer).

Figure 4: An adversarial example demonstrated in [1].


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

4 Adversarial Images (30%)

In this part you will see how you can use the gradients of the network to generate adversarial images. Using these images that look almost identical the original you will be able to fool different neural networks. You will also see that these images also affect different neural networks and expose a security issue of CNNs that malicious users can take advantage of. An example is shown in Figure 4. You are encouraged to read the relevant papers [1, 2] before solving this part.

1. (10%) Use the trained network from question 2 to generate adversarial images with constraints. The constraints that you have are:

   (a) You are not allowed to erase parts of the image, i.e. $I_{pert} \geq I$ at each pixel location.

   (b) The perturbed image has to take valid values, i.e. $-1 \leq I_{pert} \leq 1$.

   The algorithm works as follows:

   (a) Let I be a test image of your dataset that you want to perturb that is classified correctly by the network. Let I_ϵ be the perturbation that you should initialize with zeros.

   (b) Feed $I_{pert} = I + I_\epsilon$ in the network.

   (c) Calculate the loss given the ground truth label (y_{gt}). Let the loss be $L(\mathbf{x}, y \mid \theta)$ where θ are the learned weights.

   (d) Compute the gradients with respect to I_{pert}, i.e., $\nabla_{I_{pert}} L(I_{pert}, y_{gt} \mid \theta)$. Using backpropagation, compute $\nabla_{I_\epsilon} L(I_\epsilon, y_{gt} \mid \theta)$, i.e. the gradients with respect to the perturbation.

(e) Use the Fast Gradient Sign method to update the perturbation, i.e., $I_\epsilon = I_\epsilon + \epsilon sign\left(\nabla_{I_\epsilon} L(I_\epsilon, y_{gt})\right)$, where ϵ is a small constant of your choice.

(f) Repeat (a)-(d) until the network classify the input image I_{pert} as an arbitrary wrong category with confidence (probability) at least 90%.

Generate 2 examples of adversarial images. Describe the difference between adversarial images and original images.

2. (10%) For a test image from the dataset, choose a target label y_t that you want the network to classify your image as and compute a perturbed image. Note that this is different from what you are asked in part 1, because you want your network to believe that the image has a particular label, not just misclassify the image. You need to modify appropriately the loss function and then perform gradient descent as before. You should still use the constraints from part 1.

3. (10%) Retrain the network from the previous problem. Use some of the adversarial images you generated in parts (1) and (2) and feed them in the retrained network. What do you observe?

References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[2] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016.