ECE4880J: Computer Vision July 14, 2022

Homework 5: Neural Networks and Object Detection

Instructor: Siheng Chen **YOUR NAME**

Instruction

- This homework is due at 11:59:59 p.m. on ** July 23th, 2022.
- The write-up must be an electronic version edited by LaTeX.

Python Environment. We are using Python 3.7 for this course. We will use the following packages in this course: Numpy, SciPy, Matplotlib, Pytorch.

Q1. Neuron and Back Propagation [35 points]

In this part, you will track how a neuron processes the input to the expected prediction for a typical supervised-learning task. As is shown in Figure 1, the input value first activates a neuron, which outputs a predicted value. The predicted value is compared with the ground truth, and then the error between these two values is used to calculate the loss. Finally, the loss is back propagated to the neuron to tune the weight and bias (denoted by w, b) in the neuron's activation function.

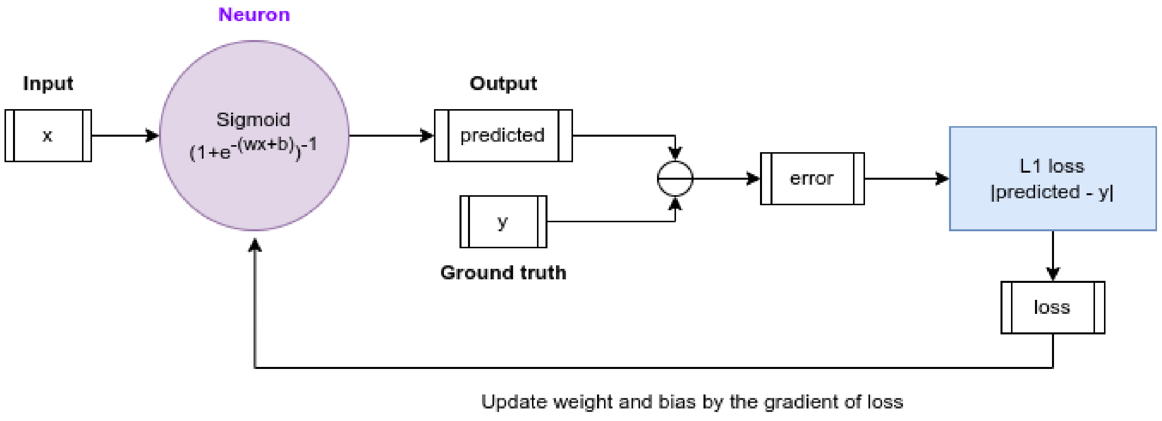

Figure 1: Network diagram.

Your task is to plot a series of 3D figures that reveals how the weight and bias of the activation function affect the predicted value, the loss, and the backpropagated gradient. **The activation function you are going to implement is the Sigmoid function**. All the plots are 3D, whose x-axis represents the activation function's weight, ranging in [-2.0, 2.0]; and the y-axis represents the activation function's bias, ranging in [-2.0, 2.0].

A set of sample plots is provided in Figure 2. Here we use ReLU activation and ℓ_1 loss.

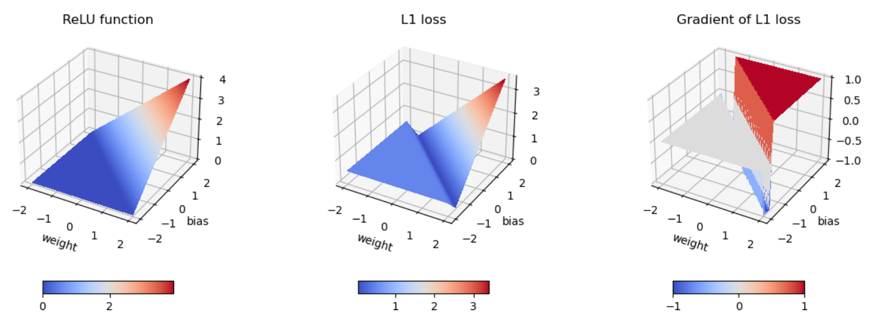

Figure 2: Example plots with ReLU activation and ℓ_1 loss. Left: Output of ReLU function. Middle: Loss plot with ℓ_1 loss. Right: Gradient plot.

1. Plot the output of the Sigmoid function versus the weight and bias.
2. Plot the ℓ_2 loss versus the weight and bias. The ℓ_2 loss is defined as $L_{\ell_2} = (\widehat{y} - y)^2$, where y is the ground truth and $\widehat{y}$ is the prediction. Let $y = 0.5$ in your calculation.
3. Experiment with back-propagation with ℓ_2 loss. Compute the gradient by $\frac{\partial L_{\ell_2}}{\partial weight}$. Then plot the gradient versus the weight and bias.
4. Plot the cross-entropy loss versus the weight and bias. The cross-entropy loss is defined as $L_{\text{CE}} = -(y \log \widehat{y} + (1-y)\log(1-\widehat{y}))$, where y is the ground truth and $\widehat{y}$ is the prediction. Let $y = 0.5$ in your calculation.
5. Experiment with back-propagation with cross-entropy loss. Compute the gradient by $\frac{\partial L_{\text{CE}}}{\partial weight}$. Then plot the gradient versus the weight and bias.
6. Explain what you observed from the above 5 plots. The explanation should include:
   a) What is the difference between cross-entropy loss and ℓ_2 loss?
   b) What is the difference between the gradients from cross-entropy loss and ℓ_2 loss?
   c) How will these differences influence the efficiency of learning?

Q2. Train a camera-only 3D object detector [65 points]

In this part, you will write code to train a camera-only 3D object detector CaDDN [1] on KITTI [2] benchmark. You could refer to the open source repository OpenPCDet and CaDDN.

Requirements. We are providing you with sample code *CaDDN_DADCN* which includes the basic CaDDN implementation and incomplete functions.

1. Refer to **docs/INSTALL.md** and run *python setup.py develop* to set up the environment.
2. Refer to **docs/GETTING_STARTED.md**. Download the KITTI dataset and preprocess the dataset.

Method	Car (IOU=0.7)		
	Easy	Moderate	Hard
CaDDN (Reported)	19.17	13.41	11.46
CaDDN (Implemented)	-	-	-
CaDDN_DADCN	-	-	-

Table 1: 3D detection results on the KITTI test set. Results are shown using the AP|R40 metric only for results that are readily available.

3. Refer to **docs/GETTING_STARTED.md**. Train the detector CaDDN with

   ```
   python train.py --cfg-file tools/cfgs/CaDDN.yaml
   ```

   and plot the training loss curve.

4. Refer to **docs/GETTING_STARTED.md**. Test the trained CADDN checkpoint with

   ```
   python test.py --cfg-file tools/cfgs/CaDDN.yaml \
       --batch-size YourBatchSize \
       --eval-all
   ```

   and report the evaluation metric average precision (AP|R40) of object class (car), including difficulty settings (Easy, Moderate, and Hard) in Tab. 1.

5. Implement the distance-aware deformable convolutional network (DADCN) in **pcdet/models/backbones_2d/map_to_bev/DCNv2/dcn_v2.py** and integrate DADCN with CaDDN.

6. Build the deformable convolutional network in **pcdet/models/backbones_2d/map_to_bev/DCNv2/** with

   ```
   python3 setup.py build
   ```

7. Refer to **docs/GETTING_STARTED.md**. Train the detector CaDDN_DADCN with

   ```
   python train.py --cfg-file tools/cfgs/CaDDN_DCN.yaml
   ```

   and plot the training loss curve.

8. Refer to **docs/GETTING_STARTED.md**. Test the trained CaDDN_DADCN checkpoint with

   ```
   python test.py --cfg-file tools/cfgs/CaDDN_DCN.yaml \
       --batch-size YourBatchSize \
       --eval-all
   ```

   and report the evaluation metric average precision (AP|R40) of object class (car), including difficulty settings (Easy, Moderate, and Hard) in Tab. 1.

Submission format. Please submit your filled *CaDDN_DADCN* and report your experimental results in the write-up.

References

[1] Cody Reading, Ali Harakeh, Julia Chae, and Steven L. Waslander. Categorical depth distributionnetwork for monocular 3d object detection. *CVPR*, 2021.

[2] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.