Assignment 2: Classification / Deep learning

Due October 29 at 11:59pm

This assignment is to be done individually.

Important Note: The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the instructor and the TA if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use Google to find solutions for assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment

---

1 Softmax for Multi-Class Classification (10 marks)

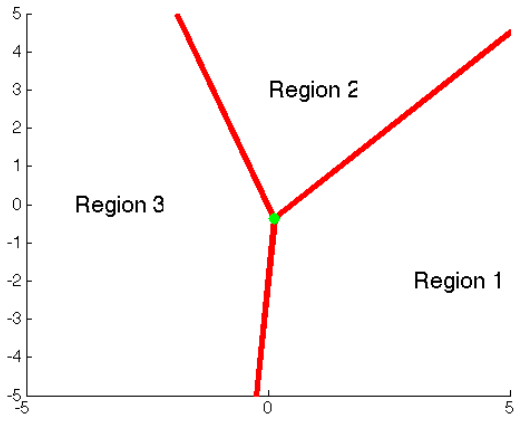
The *softmax function* is a multi-class generalization of the logistic sigmoid:

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{1}$$

Consider a case where the *activation functions* a_j are linear functions of the input. Assume there are 3 classes $(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$, and the input is $\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$

- $a_1 = 3x_1 + 1x_2 + 1$
- $a_2 = 1x_1 + 3x_2 + 2$
- $a_3 = -3x_1 + 1.5x_2 + 2$

The image below shows the 3 decision regions induced by these activation functions, their common point intersection point (in green) and decision boundaries (in red).



Answer the following questions. For 2 and 3, you may provide qualitative answers (i.e. no need to analyze limits).

1. (3 marks) What are the probabilities $p(\mathcal{C}_k|\boldsymbol{x})$ at the green point?
2. (3 marks) What happens to the probabilities along each of the red lines? What happens as we move along a red line (away from the green point)?
3. (4 marks) What happens to the probabilities as we move far away from the intersection point, staying in the middle of one region?

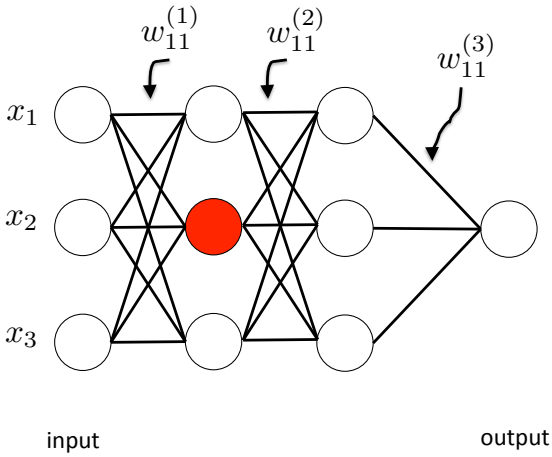
2 Error Backpropagation (30 marks)

We will derive error derivatives using back-propagation on the network below.

Notation: Please use notation following the examples of names for weights given in the figure. For activations/outputs, the red node would have activation $a_2^{(2)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3$ and output $z_2^{(2)} = h(a_2^{(2)})$.

Activation functions: Assume the activation functions $h(\cdot)$ for the hidden layers are logistics. For the final output node assume the activation function is an identity function $h(a) = a$.

Error function: Assume this network is doing regression, trained using the standard squared error so that $E_n(w) = \frac{1}{2}(y(\boldsymbol{x}_n, w) - t_n)^2$.



Consider the output layer.

- Calculate $\frac{\partial E_n(w)}{\partial a_1^{(4)}}$. Note that $a_1^{(4)}$ is the activation of the output node, and that $\frac{\partial E_n(w)}{\partial a_1^{(4)}} \equiv \delta_1^{(4)}$.
- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{12}^{(3)}}$.

Next, consider the penultimate layer of nodes.

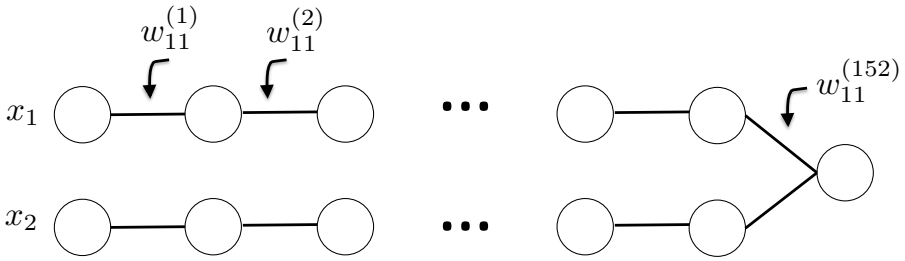
- Write an expression for $\frac{\partial E_n(w)}{\partial a_1^{(3)}}$. Use $\delta_1^{(4)}$ in this expression.
- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(2)}}$.

Finally, consider the weights connecting from the inputs.

- Write an expression for $\frac{\partial E_n(w)}{\partial a_1^{(2)}}$. Use the set of $\delta_k^{(3)}$ in this expression.
- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(1)}}$.

3 Vanishing Gradients (40 marks)

Consider the network below. Use the same notation as the previous question.



- Write an expression for $\frac{\partial E_n(w)}{\partial w_{11}^{(\ell)}}$ for all layers ℓ in the network.
- Suppose we use logistic sigmoid activation functions in this network. Describe what would happen to the gradient for weights early in the network, $\frac{\partial E_n(w)}{\partial w_{11}^{(\ell)}}$ for smaller ℓ. For example, when would these gradients be very small? When would they be reasonable in magnitude?
- Suppose we use rectified linear units (ReLU) in activation functions. When would the gradients be zero?
- Suppose we modify the graph to have complete bipartite connections at each layer, still using ReLU activation functions. When would the gradients be zero?

4 Logistic Regression (40 marks)

In this question you will examine optimization for logistic regression.

1. Download the assignment 2 code and data from the website. Run the script `logistic_regression.py` in the lr directory. This code performs gradient descent to find $\boldsymbol{w}$ which minimizes negative log-likelihood (i.e. maximizes likelihood).

   Include the final output of Figures 2 and 3 (plot of separator path in slope-intercept space; plot of neg. log likelihood over epochs) in your report.

   Why are these plots oscillating? Briefly explain why in your report.

2. Create a Python script `logistic_regression_mod.py` for the following.

   Modify `logistic_regression.py` to run gradient descent with the learning rates $\eta = 0.5, 0.3, 0.1, 0.05, 0.01$.

   Include in your report a single plot comparing negative log-likelihood versus epoch for these different learning rates.

   Compare these results. What are the relative advantages of the different rates?

3. Create a Python script `logistic_regression_sgd.py` for the following.

   Modify this code to do stochastic gradient descent. Use the parameters $\eta = 0.5, 0.3, 0.1, 0.05, 0.01$.

   Include in your report a new plot comparing negative log-likelihood versus iteration using stochastic gradient descent.

   Is stochastic gradient descent faster than gradient descent? Explain using your plots.

5 Fine-Tuning a Pre-Trained Network (30 marks)

In this question you will experiment with fine-tuning a pre-trained network. This is a standard workflow in adapting existing deep networks to a new task.

We will utilize PyTorch (`https://pytorch.org`) a machine learning library for python.

The provided code builds upon ResNet 50, a state of the art deep network for image classification. ResNet 50 has been designed for ImageNet image classification with 1000 output classes.

The ResNet 50 model has been adapted to solve a (simpler) different task, classifying an image as one of 10 classes on CIFAR10 dataset.

The code `imagenet_finetune.py` does the following:

- Constructs a deep network. This network starts with ResNet 50 up to its average pooling layer. Then, a small network with 32 hidden nodes then 10 output nodes (dense connections) is added on top.
- Initializes the weights of the ResNet 50 portion with the parameters from training on ImageNet.
- Performs training on only the new layers using CIFAR10 dataset – all other weights are fixed to their values learned on ImageNet.

The code and data can be found on the course website. For convenience, Anaconda (`https://www.anaconda.com`) environment config files with the latest stable release of PyTorch and torchvision are provided for Python 2.7 and Python 3.6 for Linux and macOs users. You can use one of the config files to create virtual environments and test your code. To set up the virtual environment, install Anaconda and run the following command

```
conda env create -f CONFIG_FILE.
```

Replace `CONFIG_FILE` with the path to the config files you downloaded. To activate the virtual environment, run the following command

```
source activate ENV_NAME
```

Replacing `ENV_NAME` with `cmpt419-pytorch-python27` or `cmpt419-pytorch-python36` depending on your Python version.

Windows users please follow the instructions on PyTorch website (`https://pytorch.org`) to install manually. PyTorch only supports Python3 on Windows!

If you wish to download and install PyTorch by yourself, you will need PyTorch (v 0.4.1), torchvision (v 0.2.1), and their dependencies.

What to do:

Start by running the code provided. It will be *very* slow to train since the code runs on a CPU. You can try figuring out how to change the code to train on a GPU if you have a good GPU and want to accelerate training. Try to do one of the following tasks:

- Write a Python function to be used at the end of training that generates HTML output showing each test image and its classification scores. You could produce an HTML table output for example.
- Run validation of the model every few training epochs on validation or test set of the dataset and save the model with the best validation error.
- Try applying $L2$ regularization to the coefficients in the small networks we added.
- Try running this code on one of the datasets in torchvision.datasets (`https://pytorch.org/docs/stable/torchvision/datasets.html`) except CIFAR100. You may need to change some layers in the network. Try creating a custom dataloader that loads data from your own dataset and run the code using your dataloader. (Hints: Your own dataset should not come from torchvision.datasets. A standard approach is to implement your own torch.utils.data.Dataset and wrap it with torch.utils.data.DataLoader)
- Try modifying the structure of the new layers that were added on top of ResNet 50.
- Try adding data augmentation for the training data using torchvision.transforms and then implementing your custom image transformation methods not available in torchvision.transforms, like gaussian blur.
- The current code is inefficient because it recomputes the output of ResNet 50 every time a training/validation example is seen, even though those layers aren't being trained. Change this by saving the output of ResNet 50 and using these as input rather than the dataloader currently used.
- The current code does not train the layers in ResNet 50. After training the new layers for a while (until good values have been obtained), turn on training for the ResNet 50 layers to see if better performance can be achieved.

Put your code and a readme file for Problem 5 under a separate directory named P5 in the code.zip file you submit for this assignment. The readme file should describe what you implemented for this problem and what each one of your code files does. It should also include the command to run your code. If you have any figures or tables to show, put them in your report for this assignment and mention them in your readme file.

Submitting Your Assignment

The assignment must be submitted online at `https://courses.cs.sfu.ca`. You must submit three files:

1. An assignment report in **PDF format**, called `report.pdf`. This report must contain the solutions to questions 1-3 as well as the figures / explanations requested for 4-5.
2. A .zip file of all your code, called `code.zip`.