Assignment 2: Solutions

1 Softmax for Multi-Class Classification

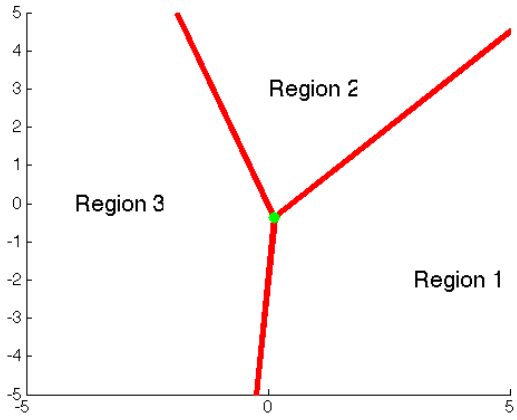
The *softmax function* is a multi-class generalization of the logistic sigmoid:

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

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?

Solutions

1. At the green point, the activation functions are equal, so the probabilities will be equal: $p(\mathcal{C}_k|\boldsymbol{x}) = \frac{1}{3}\ \forall k$.

2. On each red line, the values of two of the activation functions are equal. As we move along one of the red lines away from the green point, these two activations functions become much larger than the third one (since they are linear activations). Hence, the softmax function will have two probabilities that are close to $\frac{1}{2}$, with the remaining one approaching 0.

3. In this case, one activation function will be much larger than the other two. The probability for this one class will approach 1, with the other two approaching 0.

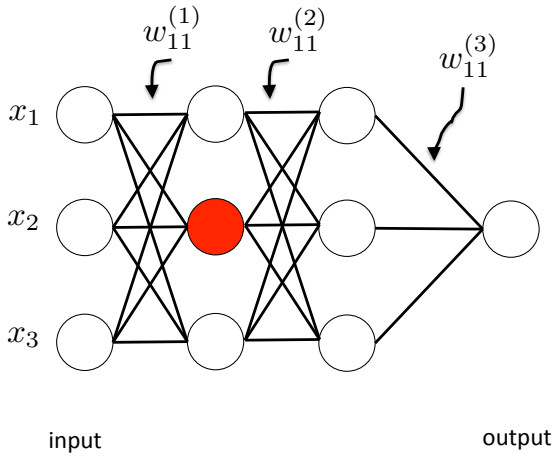
2 Error Backpropagation

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 $a_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3$ and output $z_2^{(1)} = h(a_2^{(1)})$.

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 $\frac{\partial E_n(w)}{\partial a_1^{(3)}}$. Note that $a_1^{(3)}$ is the activation of the output node, and that $\frac{\partial E_n(w)}{\partial a_1^{(3)}} \equiv \delta_1^{(3)}$.
- 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 $\frac{\partial E_n(w)}{\partial a_1^{(2)}}$. Use $\delta_1^{(3)}$ 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 $\frac{\partial E_n(w)}{\partial a_1^{(1)}}$. Use the set of $\delta_k^{(2)}$ in this expression.
- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(1)}}$.

Solutions

- *Calculate $\frac{\partial E_n(w)}{\partial a_1^{(4)}}$. Note that $a_1^{(4)}$ is the activation of the output node, and that $\frac{\partial E_n(w)}{\partial a_1^{(4)}} \equiv \delta_1^{(4)}$.*

$$\begin{aligned} \delta_1^{(4)} &= \frac{\partial E_n(w)}{\partial a_1^{(4)}} && (2)\\ &= \frac{\partial}{\partial a_1^{(4)}} \frac{1}{2}(a_1^{(4)} - t_n)^2 && (3)\\ &= (a_1^{(4)} - t_n) && (4) \end{aligned}$$

- *Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{12}^{(3)}}$.*

$$\frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = z_2^{(3)} \cdot (a_1^{(4)} - t_n) \quad (5)$$

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.*

$$\begin{aligned} \delta_1^{(3)} &= \frac{\partial E_n(w)}{\partial a_1^{(3)}} && (6)\\ &= \frac{\partial a_1^{(4)}}{\partial a_1^{(3)}} \delta_1^{(4)} && (7)\\ &= h'(a_1^{(3)}) w_{11}^{(3)} \delta_1^{(4)} && (8) \end{aligned}$$

- *Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(2)}}$.*

$$\frac{\partial E_n(w)}{\partial w_{11}^{(2)}} = z_1^{(2)} \cdot h'(a_1^{(3)}) w_{11}^{(3)} \delta_1^{(4)} \quad (9)$$

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.*

$$\begin{aligned} \delta_1^{(2)} &= \frac{\partial E_n(w)}{\partial a_1^{(2)}} && (10)\\ &= \sum_{k=1}^{3} \frac{\partial a_k^{(3)}}{\partial a_1^{(2)}} \delta_k^{(3)} && (11)\\ &= h'(a_1^{(2)}) \sum_{k=1}^{3} w_{k1}^{(2)} \delta_k^{(3)} && (12) \end{aligned}$$

- *Use this result to calculate* $\frac{\partial E_n(w)}{\partial w_{11}^{(1)}}$.

$$\frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = x_1 \cdot h'(a_1^{(2)}) \cdot \sum_{k=1}^{3} w_{k1}^{(2)} \delta_k^{(3)} \tag{13}$$

3 Logistic Regression

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 P3 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.

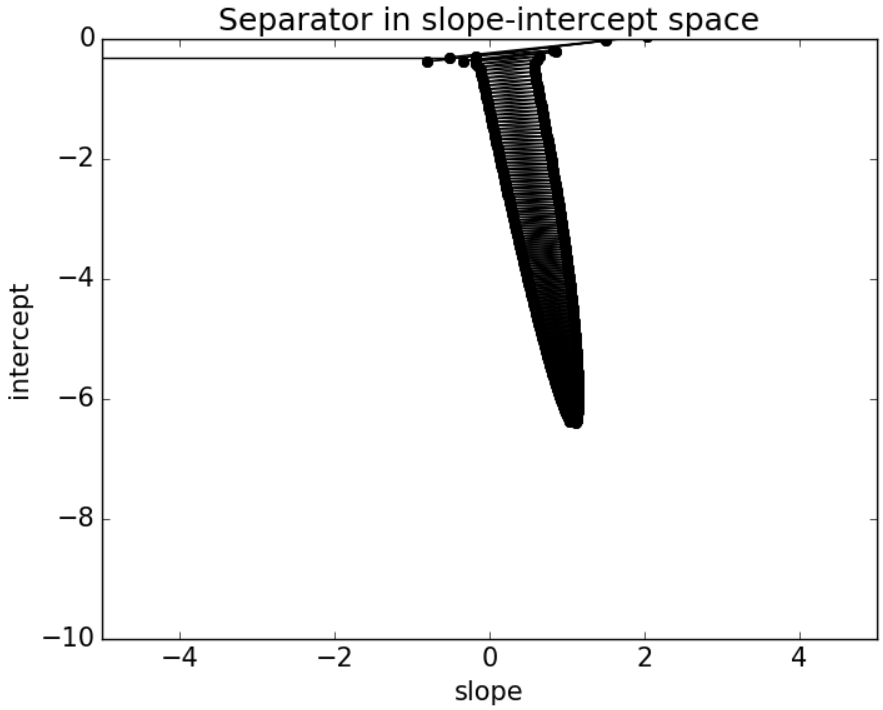
   Modify this code to do stochastic gradient descent. Use the parameters $\eta = 0.5, 0.3, 0.1, 0.05, 0.01$.

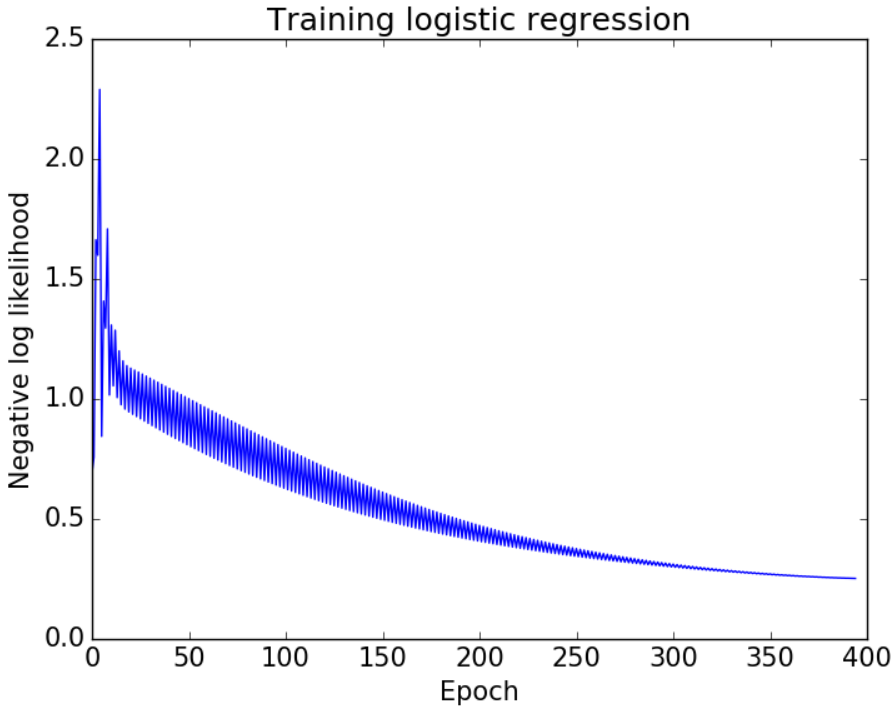
   Include in your report a new plot comparing negative log-likelihood versus iteration using stochastic gradient descent.

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

Solutions

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

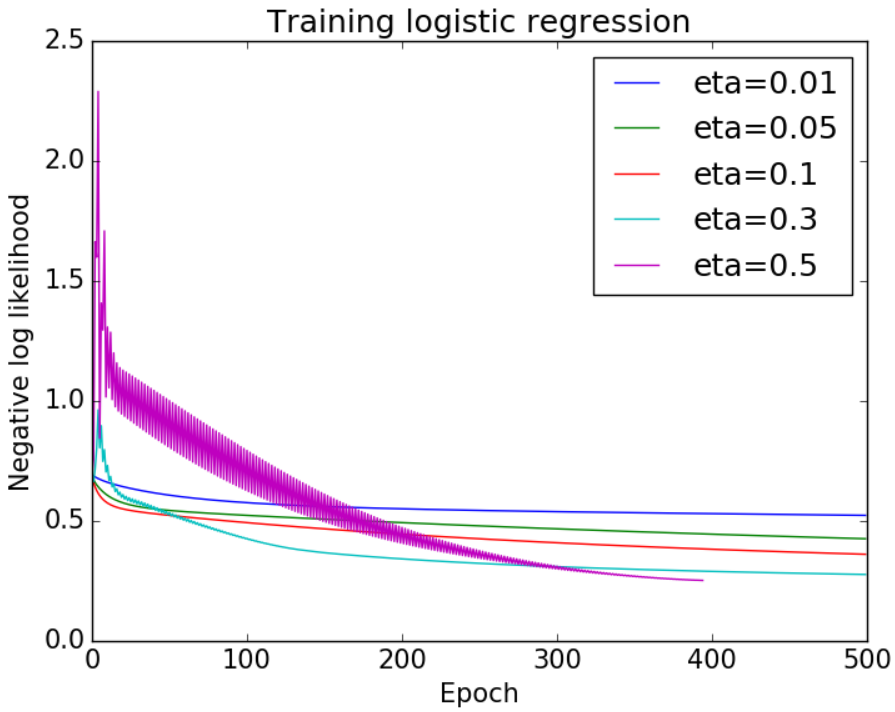




Why are these plots oscillating? Briefly explain why in your report.

The learning rate η is "too large" in the sense that it jumps across a depression in the error surface. While the error decreases in the direction of the gradient, with $\eta = 0.5$ the value of the error a step that large in the gradient direction results in a higher error value in some instances.

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

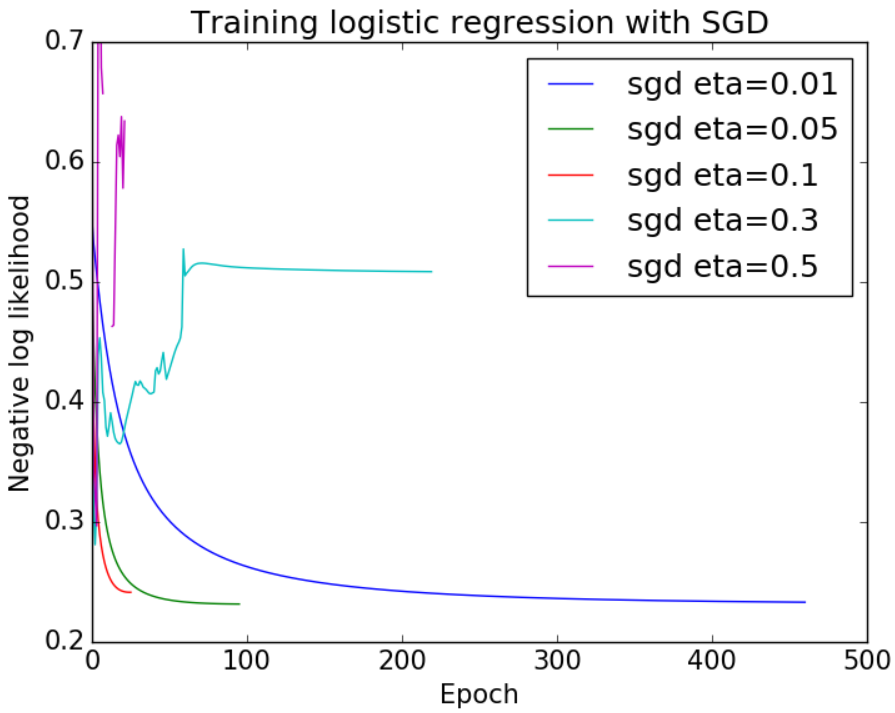


Compare these results. What are the relative advantages of the different rates?

The smaller learning rates ($\eta = 0.1, 0.05, 0.01$) lead to less oscillation. $\eta = 0.1$ obtains relatively low error (negative log likelihood) quickly. However, the larger learning rate $\eta = 0.5$ achieves lower error overall, by the end of the iterations it has a slightly lower error value. Overall, the picture isn't clear, and both of these values have relative advantages.

3. *Modify this code to do stochastic gradient descent. Use the parameters* $\eta = 0.1, 0.05, 0.03, 0.02, 0.01, 0.001, 0.0001$.

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



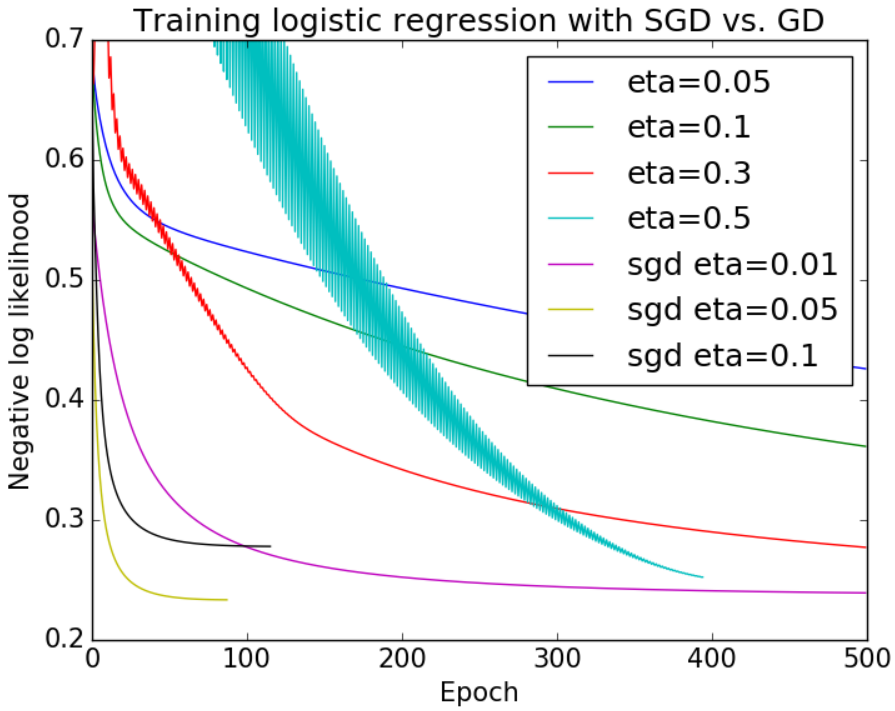
Note that for large values of η poor behaviour is exhibited – the oscillation and increases in error are symptomatic of a large error rate.

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

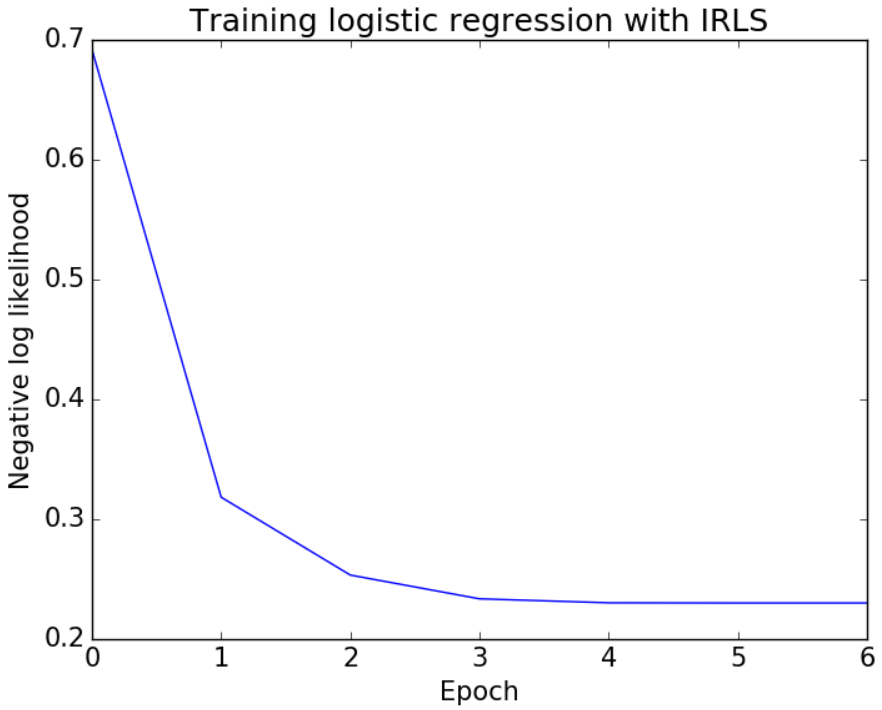
Stochastic gradient descent is much faster – it obtains a low error value in fewer iterations than gradient descent.

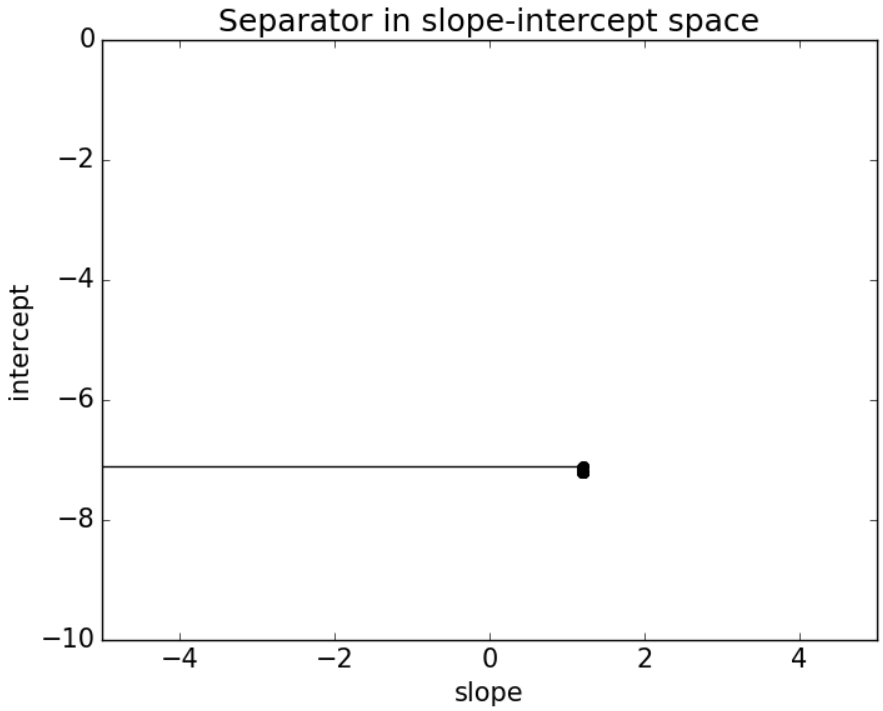
Note that it is important to fairly compare iterations between the two methods. One iteration of gradient descent is equivalent to N iterations of stochastic gradient descent.

The below graph shows results from the two methods for particular learning rates. One iteration in stochastic gradient descent is one run over all N data points. This is also known as one "epoch" in learning.



4. *Include new plots of Figures 2 and 3 using IRLS in your report.*




Separator in slope-intercept space
intercept
slope
0
−2
−4
−6
−8
−10
−4
−2
0
2
4

4 Fine-Tuning a Pre-Trained Network

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 4 under a separate directory named P4 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.

Grading criteria

A. 5 marks for organizing code and `readme` file as requested if the student finished at least one of the options. Code and `readme` files should be in a separate directory after unzipping `code.zip`.

B. 15 marks for correctly finishing one of the options in code files, depending on completeness and correctness. Some of the frequently seen coding problems are listed below.

C. 5 marks for including a `readme` file describing what is implemented and including figures in the report if necessary.

D. 3 marks for describing what each code file does in the `readme` file. If the directory only contains one or two code files, this requirement may overlap with the previous.

E. 2 marks for including the correct commands or instructions to run the code in the `readme` file.

Common issues

A. For the second option, the standard way to save a model in PyTorch is to use `torch.save`. We cannot give full marks for using other ways to save model weights or for saving model weights in other file formats. To assign model to `best_model` or `best_state_dict`, use deep copy. Otherwise `best_model` and `best_state_dict` may be updated during training. Full credit is also not given if only the model of the last epoch is saved.

B. For the third option, the simplest way to add L2 regularization is to use weight decay parameter in `torch.optim.SGD`. If L2 regularization is done manually, please make sure all the trainable parameters are regularized. For example, the linear layer has two sets of parameters, weights and bias.

C. For the first option, please show the labels of each class in text instead of numbers. The scores for each class should be the probabilities of an input image being predicted to be the class.