Convolutional Neural Networks Lab – Lab 4

In-depth testing of your LeNet-5

CSSE 313 Artificial Intelligence · Lab Manual · Lab 4 (65 points)

Name: _____ Date: _____

How This Lab Works

In Lab 3 you showed that your network computes correct gradients and learns. In this lab you train it fully, look inside it, and find out how sensitive its training is. Every part follows the same four steps:

1. **Predict.** Before running anything, write down what you expect and why. Date it, or commit it, and do not edit it afterwards.
2. **Run** the experiment and record the result.
3. **Explain** the result in terms of how the network works: kernels, pooling, receptive fields, gradients.
4. **Reconcile.** If the result differs from your prediction, say what you expected, what happened, and what your mental model was missing.

Predictions are graded on the quality of the reasoning, not on whether they turned out right. A wrong prediction with good reasoning and a careful reconciliation earns full marks. A "prediction" written after the run earns none.

AI use policy

- **Permitted:** using AI to write and debug code (for example, a script that plots `learning_curve.csv`) and to explain concepts to you.
- **Not permitted for credit:** AI-written predictions, explanations or reconciliations.

What you need

- Your working `LeNet5` from Lab 3, and your results from the feedforward lab (for one comparison in 4A).
- The provided `MNISTLoader.java`, `MNISTTrainer.java`, `NetworkVisualizer.java` and `PerturbedMNIST.java` (all four must be in the same folder). `MNISTTrainer` also prints results for shifted and noisy test digits; this lab does not use them.
- About 20 minutes of computer time in total. A 2-epoch run of `MNISTTrainer` takes roughly 5–6 minutes on a laptop, and a 1-epoch run about 3. You can work on other parts while a run finishes.

Part 4A — Training to 97%

25 points

`MNISTTrainer` trains on the first 50,000 training images and every 5,000 examples reports the accuracy on the last 10,000 training images (the **validation** set). At the end it reports the accuracy on the 10,000 **test** images once. It also writes `learning_curve.csv`.

1. (5) **Predict** how many training examples your CNN will need before validation accuracy first reaches 97%, and how many your FFNet needed (count examples, not epochs). Explain your numbers. (Recall from Lab 1 how many times each C1 weight is used for one image.)
2. (7) Run `java MNISTTrainer 2 1` (2 epochs, seed 1). Plot training accuracy and validation accuracy against examples seen from `learning_curve.csv`, and paste the plot. Report the number of examples to 97% and the test accuracy.
3. (6) The validation curve is jagged: sometimes it drops by a percentage point between two checkpoints. Explain why training one example at a time with a fixed learning rate produces this. What does it mean for "examples to reach 97%" as a measurement, and how would you make that measurement trustworthy?
4. (7) An earlier version of this assignment stopped training as soon as *test* accuracy reached 97% and reported that number. What is wrong with the reported number? Why does this lab watch a separate validation set and touch the test set only once? Compare your final training and validation accuracy: is there any sign of overfitting after 2 epochs?

Part 4B — Looking inside

15 points

1. (4) **Predict** what the six C1 feature maps of the trained network will show for the first training image (a 5). Then look: at the end of the run, `MNISTTrainer` opens the `NetworkVisualizer` for that image. Paste a screenshot.
2. (6) For at least three C1 maps, describe what they respond to: which strokes, which orientation, and which side of a stroke. Why are the C3 and S4 maps much harder to interpret?
3. (5) `MNISTTrainer` also reports *dead* units: feature maps or units that never produce a positive output on 1,000 validation images. Explain what a dead unit is, and why a ReLU unit that is dead never recovers under gradient descent.

Part 4C — How sensitive is training?

25 points

Your Lab 3 run `java MNISTTrainer 1 1` (learning rate 0.01, initial weights in [−0.1, 0.1)) is the baseline. You now change one constant at a time in `LeNet5`, recompile, and run one epoch with seed 1:

Run	Learning rate	Initial weights
baseline (from Lab 3)	0.01	[−0.1, 0.1)
slow	0.001	[−0.1, 0.1)
fast	0.1	[−0.1, 0.1)

Run	Learning rate	Initial weights
small start	0.01	[−0.05, 0.05)
large start	0.01	[−0.3, 0.3)

1. (6) **Predict** for each of the four new runs: the shape of the training-loss curve, the validation accuracy after one epoch, and whether the dead-unit counts will change.
2. (6) Run them. Make a table with, for each run, the examples to 97% (if reached), the validation accuracy after one epoch, the test accuracy and the dead-unit counts. Plot the mean training loss of the three learning rates on one graph.
3. (13) **Explain and reconcile** each result. In particular, what happened at learning rate 0.1: did the loss explode, or did something else happen? Use the dead-unit report and the test accuracy as evidence: if the network gave the same answer for every image, which accuracy would you expect? (The test set contains between 892 and 1,135 images of each digit.) How does the small start relate to what you saw with `TrainingExample` in Lab 3?

Experiment Record (use one per experiment)

```
Experiment: ________          Prediction written on (date/commit): ________
I predict: _____________________________________________________________
Because: _______________________________________________________________
Result: ________________________________________________________________
Explanation: ___________________________________________________________
Reconciliation (what my mental model was missing): ____________________
```

Points

Part	Points
4A Training to 97%	25
4B Looking inside	15
4C How sensitive is training?	25
Total	65