AI Workshop – Lab 3

Sine Curve Prediction – Neural Network models

Lab Summary

- Developing & Training Neural Networks
- Comparison with ARIMA and Persistence

Introduction

Neural Networks (NNs), especially multi-layer NNs are a very flexible way to model data.

Unlike ARIMA (and other purely extrapolative models), NNs are best used as pattern classifiers with some “feature engineering” rather than directly on input data as predictors, as we've done in Labs 2 & 3.

For “easy” prediction problems like the sine curve, both approaches should yield good results. For more difficult problems, the feature engineering approach excels. But the direct extrapolation method is simpler. We'll try the direct method in this lab and continue with feature engineering in Lab 4 with real-world data.

Lab 3: Direct Extrapolation using NNs

This method is very similar to the one in Lab 2, except that we exchange the ARIMA model for a Neural Network.

Step 1: Prepare the dataset. Prepare a fresh sine train/test datasets by uncommenting and running the appropriate lines.

Step 2: Open network.m and select the network to use – we have 3 different networks for you to try:

- **single-layer-network** – represents a single layer “perceptron” neural network.
- **double-layer-network-v1** – represents a 2-layer “perceptron” neural network, each of the same size.
- **double-layer-network-v2** – Similar to v1, but a larger 1-layer followed by a smaller layer.

Quiz:

- What is the network size of the single-layer-network?
- What are the sizes of the larger and smaller layers in v2?
- Which network would you expect to perform the best?
- How many configurations are defined?

Step 3 : Train the network – Train your selected network and pick the best performing configuration.

Quiz: Was the outcome what you expected?

Step 4: Compare your answer with someone else who tried a different network.

Quiz: Which configuration performed the best? Why?

Before you move on to Lab 3b, be sure to:

a) Record the best performances by your networks.

b) Record the corresponding configuration that produce this result.

Quiz: Why do you need to do this?

Lab 3b: Different Solver configurations

Fine tuning the solver configuration can produce better results. Some questions you need to ask yourself after :

1. Was there enough iterations or did we stop training too early?
2. Was the network overtrained?

Quiz: *For each of these issues, how to determine if this is the case and how to fix?

Experiment 1: Open up config.m. Some basic solver configuration settings (the solver type and maximum iterations) are defined. Increase the maximum iteration steps. What would be an appropriate value? Train your network and carefully record the best results.

Quiz:

- Did this change result in an improvement?
- How much (%) improvement did you get if so?

- Was the same configuration the best performer? Did you expect this?
- *Note that in your solver, “early stopping” is enabled with a length of 25 steps and “skip” of 20. How does this affect training?

Experiment 2:

Take a close look at the debug-**solver**.prototext and try to make sense of the settings. Many of these can be overridden by your **solver** word. (The few that can't don't affect training at all).

Change the solver used from “Adam” to “SGD”. “Adam” is a modern gradient-descent solver that aggressively locates minima. It's main disadvantage is that it can get easily stuck in local minima of the Risk function. The SGD solver is much less optimal, but can better deal with local minima.

Try changing the solver to SGD and see if you can see improvement. Usually, SGD needs a longer training period (why?) compared to Adam.

Again, carefully record your best performing configuration.

Quiz:

- Did the SGD solver help improve performance? Or was Adam better? What can you deduce about the sine curve problem?
- *Especially with solvers like SGD, since they are stochastic, it may be better to retry the same configuration many times. (Why?). How would you repeat each configuration 5 times with an SGD solver?

Lab 3c: Automation

In many instances, you'd want to try different network configurations to get the best network. Consider that:

- In the network.m file, we have 3 network architectures.
- There are two possible solver configurations (Adam and SGD),
- 4 different Neural Network sizes, and
- 5 “repeats” for each network,

That gives us 3 x 2 x 4 x 5 = 120 combinations. That's an awful lot to try manually!

We can use Autocaffe to help us automate this trial-and-error process. For example, to automate the Network architecture, all you need to do is to add into config.m the line:

```
{{ “single-layer-network”
   “double-layer-network-v1”
   “double-layer-network-v2” }} := network-arch
```

And replace the model used in the word **network** to:

```
“${network-arch}” eval
```

Autocaffe will replace the network architecture with the named one. Note you must use the double-quotes “...”.

WARNING!!! – Don't copy and paste from this PDF Lab Notes, because PDF uses “smart” quotes instead of the standard double quote. This will give you lot of errors. Type it out instead into the Smojo editor.

Experiment 3: Try this with NO repeats (so you've only got 24 combinations). How can you determine the best performing configuration? You can use the “min-loss” word along with a loop through all the configurations:

```
1 2 … 24 take :> dup . test-loss min-loss . cr ; reduce
```

This will display the minimum test loss of each combination. (Be sure you understand how this program works before moving on.)

To determine the details of the best performing configuration, you can use the program:

```
3 config
```

to display the configuration of configuration #3.

Quiz:

- **Can you combine these programs to just display the configuration of the best network? You need to convert the first program into a sequence using **map**, then find the minimum loss configuration using the word:

```
: minimum ( seq – n ) { ss }
  ss head
  ss tail ['] min reduce
;
```

- Re-write the word “min-loss” using the “minimum” word above. NOTE: Smojo has an integrated tutorial and help. Take advantage of these resources to help you answer this quiz.

Lab 3d: Comparison with ARIMA & Persistence

When developing practical prediction models, it is not enough to say that the model performs “well”. You must always compare the model against a *benchmark*, and try to beat that.

Quiz: *What is the use of benchmarks beyond the reason given above?

Step 1: Compare the best test-losses you've obtained and compare them against the best ARIMA (or persistence) models you obtained in Lab 2. Is your model beating them?

Another way to compare the effectiveness of 2 models, X and Y is to determine the increase in lead time one model has over another *for the same test loss*. For example, if an ARIMA model at T+3 gives similar losses to a NN model at T+5, then the improvement in lead time is 2 time steps.

Quiz: Is this gain dependent on the lead time used?

***Step 2**: Calculate the improvement in lead time of NN (fix the architecture) over ARIMA for the sine wave prediction problem. You'll need to retrain the ARIMA models for different lead times. What is the improvement (if any) of your NN if the target lead time is 5 steps?

Lab 3e: Deploying your NN model

In the real world, you will deploy your model. You will certainly have to do so if you wish to compete in this Data Science Competition.

“Deployment” means many things, but in the context of modeling, it means getting your model's predictions into the hands of users – people who will actually consume your model's output.

Do not underestimate this stage because it is crucial if your model is to gain acceptance. “Users” are just people, so it is important to understand their psychology:

- Users who consume predictions oftentimes come from “operations”.
- They already have a method for performing forecasts and are often deeply skeptical of changes / new methods because of the impact it has on their work.
- You need to work “extra hard” to convince them of the utility of your model.
- You must compare your model against the benchmarks they currently use to solve the problem.

In this lab, you will “deploy” your best performing model (1 model per project per user only). You can remove your model at any time. The goal is to see how your model ranks against some benchmarks – and models from other students!

Step 1: Select your best performing configuration.

Step 2: Edit and uncomment the appropriate line in lab-3.m, then run the program. Remember you should comment the other lines for cleaning, training, etc. so they don't get activated. So, if your best configuration is (say) config #3, you'd use:

```
3 deploy
```

Step 3: View the leaderboard for lab-3 by uncommenting the applicable line and running the script. You should see users listed by their username. Note: remember to comment out your deployment submission in Step 2.

Step 4: If you're proud of your model (!), you should consider putting your “nickname” on the leaderboard:

“Arnold Doray” nickname

will set up my nickname as “Arnold Doray”. The nickname is displayed along with your username in the leaderboard. Nicknames are applicable to each deployment project separately. Avoid using nicknames based on profanity.

Class Discussion

1. Are there other objectives of deployment besides the one outlined in the introduction?
2. Get the top 3 networks on the leaderboard to share their solutions with the class. Did they expect to top the competition?