DD2424 - Project Advice

Based on and adapted from Practical Tips for Final Projects Notes from the Stanford course cs224n

1 Define your goals

At the beginning of your project, it is very important to do some planning and define the goals and parameters of your project and make sure everyone in your team is happy and aware of these. In particular:

- Clearly define the task. What are the inputs and outputs? If the task is not framed as input and output, what exactly are you trying to achieve?
- Is there a paper (or set of papers) on which you will base your project? We recommend this for custom projects as it gives some context for the level of the performance you can realistically expect to achieve and can spare you time-consuming searches for hyper-parameter settings if you use the ones used in the paper (though they might not be perfect if you need to use smaller batch sizes etc for training, they should at least be sufficiently good).
- What dataset(s) will you use for training and for testing? Are the dataset's input and output set-up in the formats you need? If not does your group need to do some coding and/or manual effort to get the data in the format you require and is it realistic to do in the timeframe of the project?
- What is your evaluation metric (or metrics)? This needs to be a well-defined numerical metric and not a vague idea. What is the standard ones used for your
- What does success look like for your project? For your chosen evaluation metrics, what numbers represent expected performance, based on research of the topic of your project? If you are doing an analysis or theoretical project, define your hypothesis and figure out how your experiments will confirm or negate your hypothesis.

2 No contamination of the test set

Many datasets have an official training and test split. From the training set you should also create a validation set if an official validation set does not exist. You should use the data splits as follows:

- **Training data**: Use this data to train the parameters for of the neural network. Don't use other data.
- **Validation data**: You need the validation as a proxy for the test set and use it to give an estimate of the "test performance" for networks trained on the

- **Test data**: At the end after you have performed network training, hyper-parameter tuning, and model selection then you should evaluate your best trained model(s) on the test data to compute the final performance metric. In the ideal world you should perform evaluations on the test set sparingly so that you don't overfit to the test data!

3 Build strong baselines

You have built and trained your potentially complex network, run it on some test data and then summarized the test results with an evaluation metric. That's great, but what have you learned from the number you have just computed? You need to put this number in context. Has your network just cracked a dataset that no other method or researcher has managed to come close to solving or would a simple linear classifier have performed just as well? The latter model is referred to a baseline and it is a simpler method you use to compare against your more complex system. Baselines are really important as one component in putting the performance of your method in context. They are also great for sanity checking when you are debugging your more complex model. Is your model doing better than an unbiased random guesser? If not then you almost definitely have a bug somewhere in your code.

You should, of course, build stronger baselines than a random guesser and what this baseline looks like will depend a lot on the problem you are working one. One common set of baseslines are simpler versions of your full model such as a single-layer network, or a network without attention. These options are also called *ablation experiments*. An ablation experiment removes some part of the full model and measures the performance. This is useful to quantify how much different parts of the network help performance. Ablation experiments are an excellent way to analyze your model.

Building strong baselines is very important. Too often, researchers and practitioners fall into the trap of making baselines that are too weak, or failing to define any baselines at all. In this case, we cannot tell whether our complex neural systems are adding any value at all. Sometimes, strong baselines perform much better than you expected, and this is important to know.

The other component in putting your performance in context is to compare to methods that achieve state-of-the-art (SOTA) performance on the dataset(s) you are using and adhere to the rules of training (type of labels used, number of training examples used etc) you are following with your method. But fair comparison to these SOTA methods is often not as trivial as fair comparisons to baseline models you have written and trained yourself, but more on this later in the document.

4 Training and debugging neural networks

Neural network code can be hard to debug especially as they are trained in a stochastic manner. Here are some tips:

- When debugging, train on a small fraction of your training data to sanity check. If you cannot overfit (achieve near-zero training loss), then you probably have a bug in your code.
- Hyper-parameters often impact results significantly. Try to identify the most critical/important hyoer-parameters and tune those. Remember only use the validation and training set when tuning hyper-parameters.

- Due to their high capacity, neural networks can easily overfit. Explicit regularization is important (weight decay, data-augmentation, dropout, etc) as is a stopping criteria based on performance on a validation set, to help with generalization.
- Ablation experiments are very important. Your network can appear to be working and get decent results. However, it may only be working because of some simple part of the network and the fancy, interesting thing is failing or not adding anything due to a bug (within the code implementation or the network design etc.) Do ablation experiments to figure out if the different components or training regimes are actually helping performance.
- During training shuffle the order of your data during training and do this for each epoch.
- Check out these online resources for some more tips, but remember some algorithmic developments have made some advice not so relevant as before.

5 Evaluation

Evaluation Metrics For the problem you choose to work on, there is usually one or more standard evaluation metric used in the literature to measure performance. The metric should have a numerical value and you should be able to compute it automatically from the labelled test data. Find out these metrics. For example, for semantic segmentation *precision*, *recall* and *F1* are mostly used as they are more informative metrics than the accuracy when there is a large class imbalance. You can, of course, define your own metrics as well, especially if you think it will highlight an interesting issue relevant to your method, but then you should also compute this metric on your baselines as well, but it could be hard to compare to other reported results where you do not have the code or resources to evaluate their results with your new metric. Thus it is still important to also use standard metrics to provide some context.

What to compare You should use your evaluation metric(s) to compare your model to

- prior work in the field,
- your baselines and
- different versions of your model.

When comparing against prior work make sure that you are comparing exactly the same metrics. Also check whether you used the same form and quantity of training data (number of labelled or unlabelled training points used etc), number of training epochs, degree of data-augmentation, how exhaustive the hyper-parameter tuning was, etc. In many cases you will not be able to standardize for these issues (due to the constraints on your computational resources and time) to allow you to make a fair direct comparison. However, you should still quote some of the most relevant comparable numbers in the literature to provide some context and then clearly state the advantages the published method (based purely on training regime and data) had over your trained model.

Qualitative evaluation Numerical performance metrics are great and a critical element when deciding if one method is better than another, but they summarise performance over many different inputs and this can hide a lot of useful information. Qualitative evaluation/analysis probes your network/model to better understand why and when your model succeeds or fails. Please include some qualitative evaluation in your final report. Here are some examples of qualitative analysis:

- Show some examples of input and then the model's output. You should show typical output and then cherry pick some examples that show interesting behaviour and support the report's arguments.
- Error analysis. Try to categorize the errors that the model makes.
- Break down the performance metric by some criteria. For example my object detector seems to work well on large objects but less well on smaller ones.
- Find examples where the baseline does poorly, but your model does better. Is there a discernible pattern for the inputs where this happens?