Deep Learning with Python Ch. 4 Workflow

The universal workflow of machine learning

- **Defining the problem and assembling a dataset**
  - What will your input be? What are you trying to predict?
  - You can only learn to predict something if you have available training data (unless you have the resources to pay people to collect data for you, sometimes this takes the form of graduate students).
  - What variety is your problem? binary classification? multiclass classification? scalar regression? vector regression? something else?
- (often implicit) Hypothesis: Your outputs can be predicted given your inputs, meaning that your data is sufficiently informative to allow learning.

Reasons a problem might not be solvable

- *Truly inadequate data*: If all you have are the first names of all the students in a Truman class and want to predict their final grades, you are destined to fail.
- *Non-stationary problems*: You want to build a recommendation system for clothing and you are training on data from August. This is unlikely to successfully predict clothing purchases in January.

Choosing a measure of success

- You need to define what counts as success. Your metric will guide the choice of your loss function. It should align with the higher level goals you are trying to achieve.
- Possibilities include:
  - *accuracy* especially where every possible answer is roughly equally likely
  - *precision and recall* especially for class-imbalanced problems
  - *mean average precision* for ranking problems or multi-label classification
  - *your own metric* you may need to define your own custom metric for some problems

Choosing an evaluation protocol

- *a hold-out validation set*: Most of the examples are looking at do this. It is probably the best choice when you have ample amounts of data.
- *k-fold cross-validation*: We won't go into this in detail, but this and the next choice are ways to accommodate when you don't have enough data.
- *iterated k-fold validation*: Produces highly accurate model evaluation with little data.

Preparing your data

- Once you know what you're training on, what you're optimizing for and how to evaluate your approach, the next step is to format your data in a way that facilitates machine learning.
  - Your data should be in the form of *tensors*.
  - The values in these tensors should be scaled to small values, for example in the [-1,1] range or the [0,1] range.
  - If you have heterogeneous data values, then they should be normalized.
  - You may want to do some feature engineering for smaller data problems.

Developing a model that is better than random

- Your goal is to achieve *statistical power*, that is to develop a model that does better than random, for example, more than 10% accuracy for MNIST or more than 50% accuracy for the IMDB problem.
- If you fail in your attempt to do this, then the initial hypothesis that the output can be derived from the input is probably not true. You need to start over.

Building your layers

- To build your first working model you need to make three key choices:
  - *last-layer activation*: establishes constraints on the output. For example, IMDB used *sigmoid* in the last layer, MNIST used *softmax*.
  - *loss function*: needs to match the kind of problem you are solving, e.g., *binary_crossentropy* for IMDB and *mse* for the regression example
  - *optimization configuration*: what optimizer will you use? for our purposes, the answer is pretty much always *rmsprop* with its default learning rate.

Scaling up

- Once you have a model with statistical power, you need to confirm that your model is sufficiently powerful. Keep adding layers, adding parameters, and increasing the number of epochs until it starts to overfit. You need to monitor both the training loss and validation loss to do this.

Tuning your hyperparameters

- This step usually takes the most time. Once you know you have enough power to overfit, tweak your model repeatedly (using the validation data, not the testing data) to optimize performance.
- Do things such as:
  - Add dropout
  - Try to add or remove layers
  - Add L1 and L2 regularization.
  - Refine other hyperparameters such as, number of units per layer or the learning rate of the optimizer.