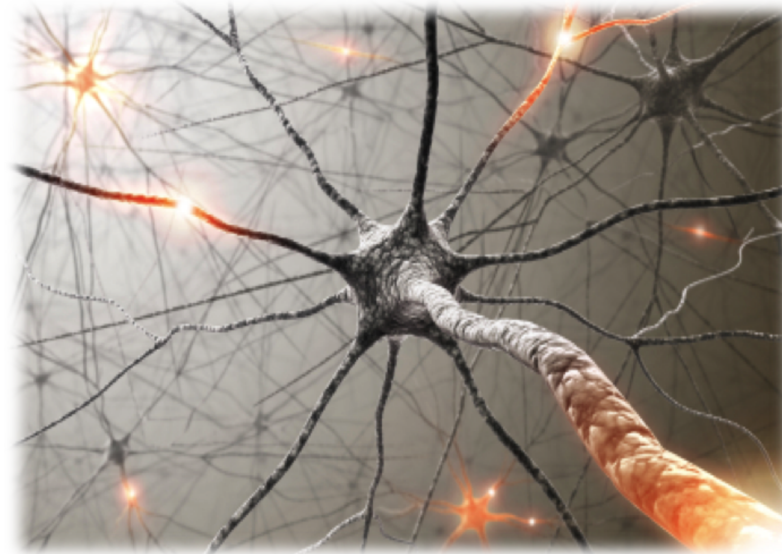
Machine learning: best practices



- We've spent a lot of talking about the formal principles of machine learning.
- In this module, I will discuss some of the more empirical aspects you encounter in practice.

Choose your own adventure

Hypothesis class:

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$

Feature extractor ϕ: linear, quadratic

Architecture: number of layers, number of hidden units

Training objective:

$$\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w}) + \text{Reg}(\mathbf{w})$$

Loss function: hinge, logistic

Regularization: none, L2

Optimization algorithm:

Algorithm: stochastic gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$
For $t = 1, \ldots, T$:
  For $(x, y) \in \mathcal{D}_{\text{train}}$:
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w})$

Number of epochs

Step size: constant, decreasing, adaptive

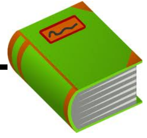
Initialization: amount of noise, pre-training

Batch size

Dropout

- Recall that there are three design decisions for setting up a machine learning algorithm: the hypothesis class, the training objective, and the optimization algorithm.
- For the hypothesis class, there are two knobs you can turn. The first is the feature extractor ϕ (linear features, quadratic features, indicator features on regions, etc. The second is the architecture of the predictor: linear (one layer) or neural network with layers, and in the case of neural networks, how many hidden units (k) do we have.
- The second design decision is to specify the training objective, which we do by specifying the loss function depending how we want the predictor to fit our data, and also whether we want to regularize the weights to guard against overfitting.
- The final design decision is how to optimize the predictor. Even the basic stochastic gradient descent algorithm has at least two knobs: how long to train (number of epochs) and how aggressively to update (the step size). On top of that are many enhancements and tweaks common to training deep neural networks: changing the step size over time, perhaps adaptively, how we initialize the weights, whether we update on batches (say of 16 examples) instead of 1, and whether we apply dropout to guard against overfitting.
- So it is really a choose your own machine learning adventure. Sometimes decisions can be made via prior knowledge and are thoughtful (e.g., features that capture periodic trends). But in many (even most) cases, we don't really know what the proper values should be. Instead, we want a way to have these just set automatically.

Hyperparameters



Definition: hyperparameters

Design decisions (hypothesis class, training objective, optimization algorithm) that need to be made before running the learning algorithm.

How do we choose hyperparameters?

Choose hyperparameters to minimize $\mathcal{D}_{\text{train}}$ error?

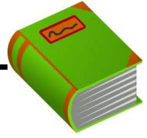
No - optimum would be to include all features, no regularization, train forever

Choose hyperparameters to minimize $\mathcal{D}_{\text{test}}$ error?

No - choosing based on $\mathcal{D}_{\text{test}}$ makes it an unreliable estimate of error!

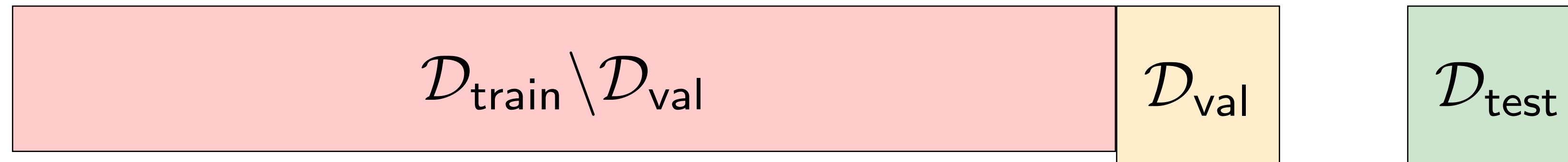
- Each of these many design decisions is a **hyperparameter**.
- We could choose the hyperparameters to minimize the training loss. However, this would lead to a degenerate solution. For example, by adding additional features, we can always decrease the training loss, so we would just end up adding all the features in the world, leading to a model that wouldn't generalize. We would turn off all regularization, because that just gets in the way of minimizing the training loss.
- What if we instead chose hyperparameters to minimize the test loss. This might lead to good hyperparameters, but is problematic because you then lose the ability to measure how well you're doing. Recall that the test set is supposed to be a surrogate for unseen examples, and the more you optimize over them, the less unseen they become.

Validation set



Definition: validation set

A **validation set** is taken out of the training set and used to optimize hyperparameters.



For each setting of hyperparameters, train on $\mathcal{D}_{\text{train}} \backslash \mathcal{D}_{\text{val}}$, evaluate on $\mathcal{D}_{\text{val}}$

- The solution is to invent something that looks like a test set. There's no other data lying around, so we'll have to steal it from the training set. The resulting set is called the **validation set**.
- The size of the validation set should be large enough to give you a reliable estimate, but you don't want to take away too many examples from the training set.
- With this validation set, now we can simply try out a bunch of different hyperparameters and choose the setting that yields the lowest error on the validation set. Which hyperparameter values should we try? Generally, you should start by getting the right order of magnitude (e.g., $\lambda = 0.0001, 0.001, 0.01, 0.1, 1, 10$) and then refining if necessary.
- In K-fold **cross-validation**, you divide the training set into K parts. Repeat K times: train on $K - 1$ of the parts and use the other part as a validation set. You then get K validation errors, from which you can compute and report both the mean and the variance, which gives you more reliable information.

Model development strategy

Algorithm: Model development strategy

- Split data into train, validation, test
- Look at data to get intuition
- Repeat:
  - Implement model/feature, adjust hyperparameters
  - Run learning algorithm
  - Sanity check train and validation error rates
  - Look at weights and prediction errors
- Evaluate on test set to get final error rates

- This slide represents the most important yet most overlooked part of machine learning: how to actually apply it in practice.
- We have so far talked about the mathematical foundation of machine learning (loss functions and optimization), and discussed some of the conceptual issues surrounding overfitting, generalization, and the size of hypothesis classes. But what actually takes most of your time is not writing new algorithms, but going through a **development cycle**, where you iteratively improve your system.
- The key is to stay connected with the data and the model, and have intuition about what's going on. Make sure to empirically examine the data before proceeding to the actual machine learning. It is imperative to understand the nature of your data in order to understand the nature of your problem.
- First, maintain data hygiene. Hold out a test set from your data that you don't look at until you're done. Start by looking at the (training or validation) data to get intuition. You can start to brainstorm what features / predictors you will need. You can compute some basic statistics.
- Then you enter a loop: implement a new model architecture or feature template. There are three things to look at: error rates, weights, and predictions. First, sanity check the error rates and weights to make sure you don't have an obvious bug. Then do an **error analysis** to see which examples your predictor is actually getting wrong. The art of practical machine learning is turning these observations into new features.
- Finally, run your system once on the test set and report the number you get. If your test error is much higher than your validation error, then you probably did too much tweaking and were **overfitting** (at a meta-level) the validation set.

Model development strategy example

Problem: simplified named-entity recognition

Input: a string x (e.g., *Governor Gavin Newsom in*)

Output: y, whether x (excluding first/last word) is a person or not (e.g., $+1$)

[code]

- Let's try out the model development strategy on the task of training a classifier to predict whether a string is person or not (excluding the first and last contxt words).
- First, let us look at the data (names.train). Starting simple, we define the empty feature template, which gets horrible error.
- Then we define a single feature template "entity is ___". Look at `weights` (person names have positive weight, city names have negative weight) and `error-analysis`.
- Let us add "left is ___" and "right is ___" feature templates based on the errors (e.g., the word "said" is indicative of a person). Look at `weights` ("the" showing up on the left indicates not a person) and `error-analysis`.
- Let us add feature templates "entity contains ___". Look at `weights` and `error-analysis`.
- Let us add feature templates "entity contains prefix ___" and "entity contains suffix ___". Look at `weights` and `error-analysis`.
- Finally we run it on the test set.

Tips

Start simple:

- Run on small subsets of your data or synthetic data
- Start with a simple baseline model
- Sanity check: can you overfit 5 examples

Log everything:

- Track training loss and validation loss over time
- Record hyperparameters, statistics of data, model, and predictions
- Organize experiments (each run goes in a separate folder)

Report your results:

- Run each experiment multiple times with different random seeds
- Compute multiple metrics (e.g., error rates for minority groups)

- There is more to be said about the practice of machine learning. Here are some pieces of advice. Note that many related to simply good software engineering practices.
- First, don't start out by coding up a large complex model and try running it on a million examples. Start simple, both with the data (small number of examples) and the model (e.g., linear classifier). Sanity check that things are working first before increasing the complexity. This will help you debug in a regime where things are more interpretable and also things run faster. One sanity check is to train a sufficiently expressive model on a very few examples and see if the model can overfit the examples (get zero training error). This does not produce a useful model, but is a diagnostic to see if the optimization is working. If you can't overfit on 5 examples, then you have a problem: maybe the hypothesis class is too small, the data is too noisy, or the optimization isn't working.
- Second, log everything so you can diagnose problems. Monitor the losses over epochs. It is also important to track the training loss so that if you get bad results, you can find out if it is due to bad optimization or overfitting. Record all the hyperparameters, so that you have a full record of how to reproduce the results.
- Third, when you report your results, you should be able to run an experiment multiple times with different randomness to see how stable the results are. Report error bars. And finally, if it makes sense for your application to report more than just a single test accuracy. Report the errors for minority groups and add if your model is treating every group fairly.

Summary

Don't look at the test set!

Understand the data!

Start simple!

Practice!

- To summarize, we've talked about the practice of machine learning.
- First, make sure you follow good data hygiene, separating out the test set and don't look at it.
- But you should look at the training or validation set to get intuition about your data before you start.
- Then, start simple and make sure you understand how things are working.
- Beyond that, there are a lot of design decisions to be made (hyperparameters). So the most important thing is to practice, so that you can start developing more intuition, and developing a set of best practices that works for you.