NEURAL NETWORKS FROM SCRATCH IN PYTHON

By Harrison Kinsley & Daniel Kukiela

Feel free to post comments/questions by highlighting the text and clicking the comment button to the right. Looks like this:

Do not resolve comments that are not yours.

Links to chapters:

Neural Networks from Scratch in Python

Harrison Kinsley & Daniel Kukieła

Copyright

License for Code

Readme

The objective of this book is to break down an extremely complex topic, neural networks, into small pieces, consumable by anyone wishing to embark on this journey. Beyond breaking down this topic, the hope is to dramatically demystify neural networks. As you will soon see, this subject, when explored from scratch, can be an educational and engaging experience. This book is for anyone willing to put in the time to sit down and work through it. In return, you will gain a far deeper understanding than most when it comes to neural networks and deep learning.

This book will be easier to understand if you already have an understanding of Python or another programming language. Python is one of the most clear and understandable programming languages; we have no real interest in padding page counts and exhausting an entire first chapter with a basics of Python tutorial. If you need one, we suggest you start here: *https://pythonprogrammingnet/python-fundamental-tutorials/* To cite this material:

Harrison Kinsley & Daniel Kukieła Neural Networks from Scratch (NNFS) https://nnfs.io

Chapter 12

Validation Data

In the chapter on optimization, we used hyperparameter tuning to select hyperparameters that lead to better results, but one more thing requires clarification. We *should not* check different hyperparameters using the test dataset; if we do that, we're going to be manually optimizing the model to the test dataset, biasing it towards overfitting these data, and these data are supposed to be used only to perform the last check if the model trains and generalizes well. In other words, if we're tuning our network's parameters to fit the testing data, then we're essentially optimizing our network on the testing data, which is another way for overfitting on these data.

Thus, hyperparameter tuning using the test dataset is a mistake. The test dataset should only be used as unseen data, not informing the model in any way, which hyperparameter tuning is, other than to test performance.

Hyperparameter tuning can be performed using yet another dataset called **validation data**. The test dataset needs to contain real out-of-sample data, but with a validation dataset, we have more freedom with choosing data. If we have a lot of training data and can afford to use some for validation purposes, we can take it as an out-of-sample dataset, similar to a test dataset. We can now search for parameters that work best using this new validation dataset and test our model

at the end using the test dataset to see if we really tuned the model or just overfitted it to the validation data.

There are situations when we'll be short on data and cannot afford to create yet another dataset from the training data. In those situations, we have two options:

The first is to temporarily split the training data into a smaller training dataset and validation dataset for hyperparameter tuning. Afterward, with the final hyperparameter set, train the model on all the training data. We allow ourselves to do that as we tune the model to the part of training data that we put aside as validation data. Keep in mind that we still have a test dataset to check the model's performance after training.

The second possibility in situations where we are short on data is a process called **cross-validation**. Cross-validation is primarily used when we have a small training dataset and cannot afford any data for validation purposes. How it works is we split the training dataset into a given number of parts, let's say 5. We now train the model on the first 4 chunks and validate it on the last. So far, this is similar to the case described previously — we are also only using the training dataset and can validate on data that was not used for training. What makes cross-validation different is that we then swap samples. For example, if we have 5 chunks, we can call them chunks A, B, C, D, and E. We may first train on A, B, C, and D, then validate on E. We'll then train on A, B, C, E, and validate on D, doing this until we've validated on each of the 5 sample groups. This way, we do not lose any training data. We validate using the data that was not used for training during any given iteration and validate on more data than if we just temporarily split the training dataset and train on all of the samples. This validation method is often called k-fold cross-validation; here, our k is 5. Here's an example of 2 steps of cross-validation:

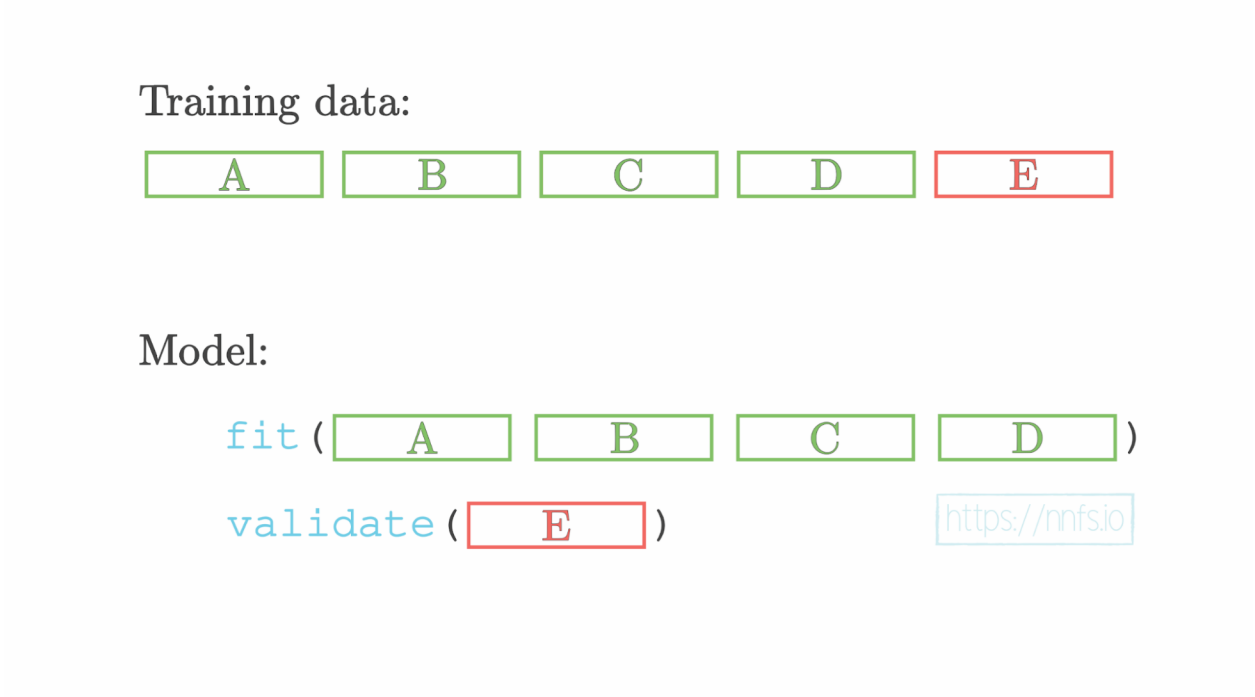

Fig 12.01: Cross-validation, first step.

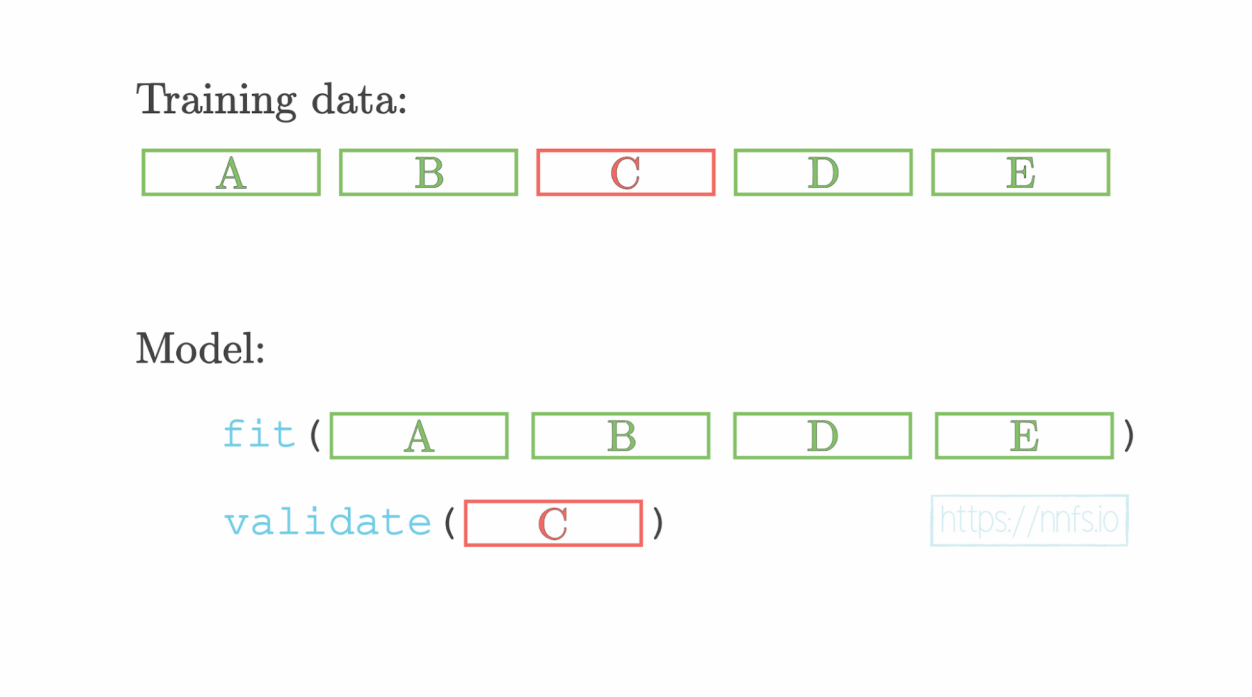

Fig 12.02: Cross-validation, third step.

Anim 12.01-12.02: https://nnfs.io/lho

When using a validation dataset and cross-validation, it is common to loop over different hyperparameter sets, leaving the code to run training multiple times, applying different settings each run, and reviewing the results to choose the best set of hyperparameters. In general, we should not loop over *all* possible setting combinations that we would like to check unless training is exceptionally fast. It's usually better to check some settings that we suspect will work well, pick the best combination of those settings, tweak them to create the next list of setting sets, and train the model on new sets. We can repeat this process as many times as we'd like.

Supplementary Material: https://nnfs.io/ch12 Chapter code, further resources, and errata for this chapter.