Model Debugging

The first step in debugging your model is Data Debugging (/machine-learning/testing-debugging/common/data-errors). After debugging your data, follow these steps to continue debugging your model, detailed in the following sections:

1. Check that the data can predict the labels.
2. Establish a baseline.
3. Write and run tests.
4. Adjust your hyperparameter values.

Check that the Model can Predict Labels

Before debugging your model, try to determine whether your features encode predictive signals. You can find linear correlations between individual features and labels by using correlation matrices. For an example of using correlation matrices, see this Colab (https://colab.research.google.com/notebooks/mlcc/feature_sets.ipynb#scrollTo=hLvmkugKLany).

However, correlation matrices will not detect nonlinear correlations between features and labels. Instead, choose 10 examples from your dataset that your model can easily learn from. Alternatively, use synthetic data that is easily learnable. For instance, a classifier can easily learn linearly-separable examples while a regressor can easily learn labels that correlate highly with a feature cross (https://developers.google.com/machine-learning/glossary/#feature_cross). Then, ensure your model can achieve very small loss on these 10 easily-learnable examples.

Using a few examples that are easily learnable simplifies debugging by reducing the opportunities for bugs. You can further simplify your model by switching to the simpler gradient descent algorithm instead of a more advanced optimization algorithm.

Establish a Baseline

Comparing your model against a baseline is a quick test of the model's quality. When developing a new model, define a **baseline** by using a simple heuristic (https://developers.google.com/machine-learning/glossary/#heuristic) to predict the label. If your trained model performs worse than its baseline, you need to improve your model.

Examples of baselines are:

- Using a linear model trained solely on the most predictive feature.
- In classification, always predicting the most common label.
- In regression, always predicting the mean value.

Once you validate a version of your model in production, you can use that model version as a baseline for newer model versions. Therefore, you can have multiple baselines of different complexities. Testing against baselines helps justify adding complexity to your model. A more complex model should always perform better than a less complex model or baseline.

Implement Tests for ML Code

The testing process to catch bugs in ML code is similar to the testing process in traditional debugging. You'll write unit tests to detect bugs. Examples of code bugs in ML are:

- Hidden layers that are configured incorrectly.
- Data normalization code that returns NaNs.

A sanity check for the presence of code bugs is to include your label in your features and train your model. If your model does not work, then it definitely has a bug.

Adjust Hyperparameter Values

The table below explains how to adjust values for your hyperparameters (/machine-learning/glossary#hyperparameter).

Hyperparameter	Description
Learning Rate (/machine-learning/crash-course/reducing-loss/video-lecture)	Typically, ML libraries will automatically set the learning rate. F in TensorFlow, most TF Estimators use the AdagradOptimizer (https://www.tensorflow.org/api_docs/python/tf/train/Adagra , which sets the learning rate at 0.05 and then adaptively modif learning rate during training. The other popular optimizer, Adan (https://www.tensorflow.org/api_docs/python/tf/train/AdamO uses an initial learning rate of 0.001. However, if your model dc converge with the default values, then manually choose a value 0.0001 and 1.0, and increase or decrease the value on a logarit until your model converges. Remember that the more difficult y

	problem, the more epochs your model must train for before los decrease.
Regularization (https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/video-lecture)	First, ensure your model can predict without regularization on t data. Then add regularization only if your model is overfitting o data. Regularization methods differ for linear and nonlinear mc For linear models, choose L_1 regularization if you need to reduc model's size. Choose L_2 regularization if you prefer increased r stability. Increasing your model's stability makes your model tra reproducible. Find the correct value of the regularization rate, λ starting at 1e-5 and tuning that value through trial and error. To regularize a deep neural network model, use Dropout regula (https://developers.google.com/machine-learning/glossary/#dropout_regularization) . Dropout removes a random selection of a fixed percentage of neurons in a network layer for a single gradient step. Typically, will improve generalization at a dropout rate of between 10% a neurons.
Training epochs (https://developers.google.com/machine-learning/glossary/#epoch)	You should train for at least one epoch, and continue to train so you are not overfitting.
Batch size (https://developers.google.com/machine-learning/glossary/#batch_size))	Typically, the batch size of a mini-batch (https://developers.google.com/machine-learning/glossary/# is between 10 and 1000. For SGD (https://developers.google.com/machine-learning/glossary/# batch size is 1. The upper bound on your batch size is limited b amount of data that can fit in your machine's memory. The low on batch size depends on your data and algorithm. However, u smaller batch size lets your gradient update more often per ep can result in a larger decrease in loss per epoch. Furthermore, trained using smaller batches generalize better. For details, see batch training for deep learning: Generalization gap and sharp (https://arxiv.org/pdf/1609.04836.pdf) N. S. Keskar, D. Mudige Nocedal, M. Smelyanskiy, and P. T. P. Tang. ICLR, 2017. Prefer u smallest batch sizes that result in stable training.
Depth and width of layers	In a neural network, depth refers to the number of layers, and w to the number of neurons per layer. Increase depth and width a complexity of the corresponding problem increases. Adjust you and width by following these steps: 1. Start with 1 fully-connected hidden layer with the same widt input layer. 2. For regression, set the output layer's width to 1. For classific the output layer's width to the number of classes.

3. If your model does not work, and you think your model neec deeper to learn your problem, then increase depth linearly b fully-connected hidden layer at a time. The hidden layer's wi depends on your problem. A commonly-used approach is tc same width as the previous hidden layer, and then discover appropriate width through trial-and-error.

The change in width of successive layers also depends on you A practice drawn from common observation is to set a layer's \ to or less than the width of the previous layer. Remember, the c width don't have to be exactly right. You'll tune their values late optimize your model.

Previous
← Data Debugging (/machine-learning/testing-debugging/common/data-errors)

Next
Check Your Understanding →
(/machine-learning/testing-debugging/common/check-your-understanding)