Deep learning - Practical Methodology - Notes

Cosmin G. Alexandru

March 27, 2017

This are notes on the 11th Chapter, Pracitcal Methodology, from the Deep Learning book[1].

The book[2] recommends the following procces for designing a machine learning system to solve a particular problem:

1. Choose your performance(error) metric.
2. Establish a working end-to-end pipeline as soon as possible.
3. Instrument the system to determine bottelnecks in performance and debugging.
4. Make incremental changes.

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`

Performance Metric

First thing to do when developing a machine learning system is to determine which error(performance) metric to use and improve.

For practical cases the error metric is lower bounded by the Bayes error. This is because the input features might have incomplete information, the processes modeled is stochastic, finite amount of data, etc.

Multiple performance metrics can be chosen. When multiple performance metrics are used, one can usually trade between them to achieve the desired global performance.

The performance metrics should be specific to the problem that is solved. Here are a few performance metrics to consider when designing a machine learning system:

- accuracy
- precision and recall
- coverage
- precision-recall curve
- $F - score = 2pr/(p + r)$

Baseline Models

The second step is to choose a model for your problem and establish a working end-to-end system. If the problem you are working on has

been solved or partialy solved before, it is a good idea to start from there.

Here are a few design choices that you can make but keep in mind that the field of machine learning is advancing quickly and better solution might be available (the book was published in 2016):

- Simple problem - simple model, e.g. logistic regression.
- For supervized learning on fixed size input vectors, use a feed forword network with fully connected layers.
- For input with topological structure, use a convolutional neural network.
- If the input or output is a squence, use a gated recurrent network (LSTM or GRU).

The most utilized methods for training (optimization) are Stochastic Gradient Descent with momentum (SGD) or Adam[3].

Popular learning rate decay schemes for SGD:

- Linear decay until fixed minimum.
- Exponential decay.
- Decrease learning rate by a factor of 2-10 each time validation error plateaus.

It is good practice to use some form of regularization in the optimization proccess, especialy for small number of examples. Some of the most successfuly and widely used methods of regularization are:

- Batch normalization.
- Early stopping (should be used almost universally[4]).
- Dropout is an excellent regularizer.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`

More Data

Often it is much better to gather more data than to improve the learning algorithm.

First thing to check when deciding wether to gather more data or not, is, the performance on the training set. If the performance on the training set is bad it is very probable that the model is not be using the training data that is already available. In this case, instead of gathering more data it is recomended to:

- Increase the model size(e.g: number of neurons per layer, number of layers, etc.)

- Tune the learning rate.
- Check the data quality.

When the performance on the training set is acceptable, check the performance on a test set. If the performance on the test set is good the there is nothing left to do, otherwise it is almost always a good idea to gather more data.

When deciding to gather more data you should always keep in mind the cost of gathering more data.

In order to decide how much more data to gather check to see how the performance of the system scales with the size of the training data. For this purpose one can plot curves of the performance depending on the training data size. Logarithmic steps in data size are recomended.

Sometimes the cost of gathering more data is too high. In this cases, the alternatives to gathering data are: reduce the size of the model, improve regularization (e.g: add drop out, adjust hpyperparameters such as weight decay coefficents).

Hyperparameters

Almost, if not, all models come with hyperparameters to be tuned in order to achieve the best performance for a specific problem. The hpyerparameters can affect from the time and memory cost of the running algorithm to the quality of the model.

Selection of the hyper parameters can be done manually or automatically.

The primary goal of hyperparameter search is to adjust the effective capacity of the model. Effective capacity of the model depends on three factors:

- Representation capacity of the model (more hidden layers / more units per hidden layer = greater representational capacity).
- Optimization algortihm to successfully minimize the cost function.
- Degree to which the cost function and training procedure regularizes the model.

The generalization error ploted as a founction of one of the hyperparameters follows a U-shaped curve.

At one end of the U-shaped curve there is the underfitting regime when the generalization error is high because the training error is high and the hyperparameter corresponds to low capacity. At the other end the hyperparameter corresponds to high capacity and the generalization error is high because the gap between the training

error and test error is high. Somewhere in the middle lies the optimal model capacity, which achieves the lowest possible generalization error, by adding a medium generalization gap to a medium amount of training error.

Manual hyperparameters tunning

When choosing to mannually tune the hyperparameters, one must have a good understanding of the relation between the hyperparameters, training error, generalization error and computational resources.

One of the most important hyperparameters to tune, especially when manual tunning is chosen, is the learning rate.

Automatic hyperparameters tunning

When tunning parameters automatically one has to choose the way to explore the U-shaped curve by choosign the appropiate hyperparameters ranges.

The direct approach is to choose the hyperparameters on a grid. This method is ussualy slow and the results are limited.

A better aproach is to choose hyperparameters randomly. This way achieves better search space coverage.

Model-based hyperparameter optimization

The search for good hyperparameters can be cast as and optimization problem. Current approaches to include Spearmint[5], TPE [6] and SMAC[7].

[5] Snoek J., Larochelle H., and Adams R.P. Practical bayesian optimization of machine learning algorithms. *NIPS 2012*, 2012

[6] Bergstra J., Bardenet R., Bengio Y., and Kegl B. Algorithms for hyper-parameter optimization. *NIPS 2011*, 2011

[7] Hutter F., Hoos H., and Layton-Brown K. Sequential model-based optimization for general algorithm configuration. *Lion-5 Extended version as UBC Tech report TR-2010-10.*, 2011

Debugging

Debuging machine learning systems can be difficult because a code problem can be hidden by the learning algorithm that manages to optimze despite the coding error.

Methods to debug software problems:

- Visualize the model in action.
- Visualize the worst mistake.
- Reason about software using train and test error.
- Fit a small dataset.
- Compare back-propagation derivatives to numerical derivatives.
- Monitor historgrams of activations and gradient.

References

Hutter F., Hoos H., and Layton-Brown K. Sequential model-based optimization for general algorithm configuration. *Lion-5 Extended version as UBC Tech report TR-2010-10.*, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Bergstra J., Bardenet R., Bengio Y., and Kegl B. Algorithms for hyper-parameter optimization. *NIPS 2011*, 2011.

Snoek J., Larochelle H., and Adams R.P. Practical bayesian optimization of machine learning algorithms. *NIPS 2012*, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.