CHAPTER 11
PRACTICAL METHODOLOGY

Presenter: Sonia Thomas
27th March 2017
Applying deep learning techniques requires good knowledge of existing algorithms and the principles that explain how they work.

How to choose an algorithm for a particular application and how to monitor and respond to feedback obtained from experiments in order to improve a machine learning system.

A practical design process consists of determining goals, establishing system, instrument the model, and repeatedly make changes to improve model.

11.1 Performance Metric
11.2 Default Baseline Model
11.3 Determine whether to gather more data
11.4 Selecting hyper parameters
11.5 Debugging strategies
11.6 Eg: Multi-Digit Number Recognition
11.2 Default Baseline Models

- This section deals with recommendations for algorithms to use as the 1st baseline approach in different situations.

- Depending on the complexity of the problem the type of learning is selected. Eg.: If problem could be solved by few linear weights then a simple statistical model (logistic regression) could be used.

- Depending on category of problem: Eg: “AI-complete” category problems like object recognition, speech recognition, machine translation, and so on, then begin with an appropriate deep learning model.

Step 1: Determine the category of the model based on structure of data.

<table>
<thead>
<tr>
<th>Structure of Data</th>
<th>Type of model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Fixed size vector</td>
<td>Fully connected Feed-forward network(Supervised)</td>
</tr>
<tr>
<td>Input: Image or (topological structure)</td>
<td>Convolutional network</td>
</tr>
<tr>
<td>Input: Sequence</td>
<td>RNN, gated RNN, LSTM</td>
</tr>
</tbody>
</table>
Step 2: Determine optimization algorithm and regularization.

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Properties</th>
</tr>
</thead>
</table>
| Stochastic Gradient Decent (SGD) with momentum with a decaying learning rate | Popular decay schemes are (perform better or worse on different problems)  
  - decaying linearly until reaching a fixed minimum learning rate  
  - decaying exponentially,  
  - or decreasing the learning rate by a factor of 2-10 each time validation error plateaus |
| Batch normalization                           | • dramatic effect on optimization performance especially for CNN and networks with sigmoidal nonlinearities.  
  • omit batch normalization at 1st baseline, introduce if optimization appears to be problematic. |
| Regularization  
*Include some mild forms of regularization from the start, unless your training set contains tens of millions of examples or more* | • Early stopping: be used almost universally.  
  • Dropout is an excellent regularizer that is easy to implement and compatible with many models and training algorithms  
  *Batch normalization also sometimes reduces generalization error and allows dropout to be omitted, due to the noise in the estimate of the statistics used to normalize each variable* |
STEP 3: Copying the baseline model from previous study

- If your task is similar to another task that has been studied extensively, start by copying the model and algorithm that is already known to perform best. (copy a trained model)
- Eg: Use the features from a CNN trained on ImageNet as base model to solve other CV tasks.

STEP 4: Supervised or unsupervised learning?

- To begin by using unsupervised learning or not is domain specific.
- Some domains, such as NLP (natural language processing), benefit tremendously from unsupervised learning techniques such as learning unsupervised word embeddings.
- In other domains, such as computer vision, current unsupervised learning techniques do not benefit.
- If your application is in a context where unsupervised learning is known to be important, then include it in your first end-to-end baseline.
11.3 Determining Whether to Gather More Data?

- After the first end-to-end system is established, the performance of the algorithm is measured.
- How to improve it?
- Many ML techniques are tempted to make improvements by trying out many different algorithms. However, it is often much better to gather more data than to improve the learning algorithm.

<table>
<thead>
<tr>
<th>How does one decide whether to gather more data? Determine the performance on the training set</th>
</tr>
</thead>
</table>
| **If performance is poor** | • learning algorithm is not entirely using the training data that is already available, so there is no reason to gather more data.  
• increasing the size of the model by adding more layers or adding more hidden units to each layer.  
• or try improving the learning algorithm, Eg: tuning hyperparameters (learning rate) |
| **If large models and carefully tuned optimization algorithms do not work well** | • the problem might be the quality of the training data.  
• The data may be too noisy or may not include the right inputs needed to predict the desired outputs.  
• This suggests starting over, collecting cleaner data or collecting a richer set of features. |
### Determine the performance on the test set

<table>
<thead>
<tr>
<th>If performance is poor, compared to training set</th>
<th>then gathering more data is one of the most effective solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The key consideration are:</td>
</tr>
<tr>
<td></td>
<td>▪ cost and feasibility of gathering more data,</td>
</tr>
<tr>
<td></td>
<td>▪ cost and feasibility of reducing the test error by other means</td>
</tr>
<tr>
<td></td>
<td>▪ The amount of data that is expected to be necessary to improve test set performance significantly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gather more data? It is domain specific</th>
<th>For example, the development of large labeled datasets was one of the most important factors in solving object recognition.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In other contexts, such as medical applications, it may be costly or infeasible to gather more data.</td>
</tr>
</tbody>
</table>

#### Alternative to gathering more data?

- A simple alternative to gathering more data is to reduce the size of the model or improve regularization, by adjusting hyper parameters such as weight decay coefficients, or by adding regularization strategies such as dropout.
How much more to gather?

- Plot curves showing the relationship between training set size and generalization error.
- By extrapolating such curves, one can predict how much additional training data would be needed to achieve a certain level of performance.
- Adding a small fraction of the total number of examples will not have a noticeable impact on generalization error. It is therefore recommended to experiment with training set sizes on a logarithmic scale, for example doubling the number of examples between consecutive experiments.
- If gathering much more data is not feasible, the only other way to improve generalization error is to improve the learning algorithm itself.
11.4 Selecting Hyper parameters

- Most deep learning algorithms come with many hyper parameters that control many aspects of the algorithm’s behaviour.
- Some of these hyper parameters affect the time and memory cost of running the algorithm.
- Some affect the quality of the model recovered by the training process and its ability to infer correct results when deployed on new inputs.

<table>
<thead>
<tr>
<th>Manual Hyper parameter Tuning</th>
<th>• requires understanding what the hyper parameters do and how machine learning models achieve good generalization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Hyper parameter Tuning</td>
<td>• Automatic hyper parameter selection algorithms greatly reduce the need to understand these ideas, but they are often much more computationally costly.</td>
</tr>
</tbody>
</table>
11.4.1 Manual Hyper parameter Tuning

• To set hyper parameters manually, one must understand the relationship between hyper parameters, training error, generalization error and computational resources (memory and runtime). {chapter 5}

• The primary goal of manual hyper parameter search is to adjust the effective capacity of the model to match the complexity of the task.

• Effective capacity is constrained by three factors:
  ▪ the representational capacity of the model,
  ▪ The ability of the learning algorithm to successfully minimize the cost function used to train the model,
  ▪ the degree to which the cost function and training procedure regularize the model.
Generalization curve: The generalization error typically follows a U-shaped curve when plotted as a function of one of the hyper parameters.

- At one extreme, the hyper parameter value corresponds to low capacity, and generalization error is high because training error is high. This is the under fitting regime.
- At the other extreme, the hyper parameter value corresponds to high capacity, and the generalization error is high because the gap between training 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.
For some hyper parameters, over fitting occurs when the value of the hyper parameter is large.

eg: The number of hidden units in a layer, increasing the number of hidden units, increases the capacity of the model.

For some hyper parameters, over fitting occurs when the value of the hyper parameter is small.

eg: the smallest allowable weight decay coefficient of zero corresponds to the greatest effective capacity of the learning algorithm.

Not every hyper parameter will be able to explore the entire U-shaped curve.
<table>
<thead>
<tr>
<th>Hyper parameter</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>• discrete, eg as the number of units in a layer or the number of linear pieces in a maxout unit.</td>
<td>so it is only possible to visit a few points along the curve</td>
</tr>
<tr>
<td>• binary, usually these hyper parameters are switches that specify whether or not to use some optional component of the learning algorithm</td>
<td>Eg: a pre processing step that normalizes the input features by subtracting their mean and dividing by their SD, hence these hyper parameters can only explore two points on the curve.</td>
</tr>
<tr>
<td>• Other hyper parameters have some minimum or maximum value that prevents them from exploring some part of the curve</td>
<td>Eg: the minimum weight decay coefficient is zero. Thus if the model is underfitting when weight decay is zero, we cannot enter the overfitting region by modifying the weight decay coefficient. Hence, some hyper parameters have only subtract capacity.</td>
</tr>
</tbody>
</table>
• The learning rate is perhaps the most important hyper parameter.
• If you want to tune only one hyper parameter, tune the learning rate.
• It controls the effective capacity of the model in a more complicated way than other hyper parameters—the effective capacity of the model is highest when the learning rate is correct for the optimization problem, not when the learning rate is especially large or especially small.

Figure: Typical relationship between the learning rate and the training error
• When the learning rate is too large, gradient descent can inadvertently increase the training error rather than decreasing it.
• When the learning rate is too small, training is not only slower, but may become permanently stuck with a high training error.
• Tuning the parameters other than the learning rate requires monitoring both training and test error to diagnose whether your model is over fitting or under fitting, then adjusting its capacity appropriately.
• If error on the training set is higher than your target error rate, then increase model capacity by adding more layers to your network or add more hidden units,
• Unfortunately, this increases the computational costs associated with the model.
• If error on the test set is higher than your target error rate?
• The test error is the sum of the training error and the gap between training and test error. The optimal test error is found by trading off these quantities.
• Eg: Neural networks perform best when the training error is very low (and thus, when capacity is high) and the test error is primarily driven by the gap between train and test error.
• To reduce the gap, change regularization hyper parameters to reduce effective model capacity, such as by adding dropout or weight decay.
• Usually the best performance comes from a large model that is regularized well, for example by using dropout.
Most hyper parameters can be set by reasoning about whether they increase or decrease model capacity

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Increases capacity when...</th>
<th>Reason</th>
<th>Caveats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of hidden units</td>
<td>increased</td>
<td>Increasing the number of hidden units increases the representational capacity of the model.</td>
<td>Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.</td>
</tr>
<tr>
<td>Learning rate</td>
<td>tuned optimally</td>
<td>An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure</td>
<td></td>
</tr>
<tr>
<td>Convolution kernel width</td>
<td>increased</td>
<td>Increasing the kernel width increases the number of parameters in the model</td>
<td>A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.</td>
</tr>
<tr>
<td>Implicit zero padding</td>
<td>increased</td>
<td>Adding implicit zeros before convolution keeps the representation size large</td>
<td>Increased time and memory cost of most operations.</td>
</tr>
<tr>
<td>Weight decay coefficient</td>
<td>decreased</td>
<td>Decreasing the weight decay coefficient frees the model parameters to become larger</td>
<td></td>
</tr>
<tr>
<td>Dropout rate</td>
<td>decreased</td>
<td>Dropping units less often gives the units more opportunities to “conspire” with each other to fit the training set</td>
<td></td>
</tr>
</tbody>
</table>

Table 11.1: The effect of various hyperparameters on model capacity.
• While manually tuning hyper parameters, do not lose sight of your end goal: good performance on the test set.
• Adding regularization is only one way to achieve this goal.
• As long as you have low training error, you can always reduce generalization error by collecting more training data. The brute force way to practically guarantee success is to continually increase model capacity and training set size until the task is solved. This approach does of course increase the computational cost of training and inference, so it is only feasible given appropriate resources.
• In principle, this approach could fail due to optimization difficulties, but for many problems optimization does not seem to be a significant barrier, provided that model is chosen appropriately.
OUTLINE:

- 11.1 Performance Metric
- 11.2 Default Baseline Model
- 11.3 Determine whether to gather more data
- 11.4 Selecting hyper parameters
  - 11.4.1 Manual Hyperparameter Tuning
  - 11.4.2 Automatic Hyperparameter Optimization Algorithms
  - 11.4.3 Grid Search
  - 11.4.4 Random Search
  - 11.4.5 Model-Based Hyperparameter Optimization
- 11.5 Debugging strategies
- 11.6 Eg: Multi-Digit Number Recognition
THANK YOU