Chapter 11

Practical Methodology

Successfully applying deep learning techniques requires more th
knowledge of what algorithms exist and the principles that ex
work. A good machine learning practitioner also needs to know h
algorithm for a particular application and how to monitor and resp
obtained from experiments in order to improve a machine learning
day-to-day development of machine learning systems, practitioners
whether to gather more data, increase or decrease model capacity,
regularizing features, improve the optimization of a model, impro
inference in a model, or debug the software implementation of the m
operations are at the very least time consuming to try out, so it
be able to determine the right course of action rather than blindly

Most of this book is about different machine learning models
rithms, and objective functions. This may give the impression
important ingredient to being a machine learning expert is knowing
of machine learning techniques and being good at different kinds of
tice, one can usually do much better with a correct application of

algorithm than by sloppily applying an obscure algorithm. Correc
an algorithm depends on mastering some fairly simple methodolog
recommendations in this chapter are adapted from Ng (2015).

We recommend the following practical design process:

- Determine your goals—what error metric to use, and your t
  this error metric. These goals and error metrics should be
  problem that the application is intended to solve.
- Establish a working end-to-end pipeline as soon as possible

estimation of the appropriate performance metrics.

- Instrument the system well to determine bottlenecks in perf
nose which components are performing worse than expecte
poor performance is due to overfitting, underfitting, or a de
or software.

- Repeatedly make incremental changes such as gathering new
hyperparameters, or changing algorithms, based on specifi
your instrumentation.

As a running example, we will use the Street View address numb
system (Goodfellow *et al.*, 2014d). The purpose of this applica
buildings to Google Maps. Street View cars photograph the build
the GPS coordinates associated with each photograph. A convolu
recognizes the address number in each photograph, allowing the
database to add that address in the correct location. The sto
commercial application was developed gives an example of how to fo
methodology we advocate.

We now describe each of the steps in this process.

11.1 Performance Metrics

Determining your goals, in terms of which error metric to use, is a
step because your error metric will guide all your future actions. Y
have an idea of what level of performance you desire.

Keep in mind that for most applications, it is impossible to a
zero error. The Bayes error defines the minimum error rate that y
achieve, even if you have infinite training data and can recover the
distribution. This is because your input features may not co

information about the output variable, or because the system might
stochastic. You will also be limited by having a finite amount of t

The amount of training data can be limited for a variety of reaso
goal is to build the best possible real-world product or service, yo
collect more data but must determine the value of reducing error fu
this against the cost of collecting more data. Data collection ca
money, or human suffering (for example, if your data collection p
performing invasive medical tests). When your goal is to answer a sc
about which algorithm performs better on a fixed benchmark,

specification usually determines the training set, and you are not al
more data.

How can one determine a reasonable level of performance to exp
in the academic setting, we have some estimate of the error rate th
based on previously published benchmark results. In the real-w
have some idea of the error rate that is necessary for an applica
cost-effective, or appealing to consumers. Once you have determine
desired error rate, your design decisions will be guided by reaching

Another important consideration besides the target value of th
metric is the choice of which metric to use. Several different perfo
may be used to measure the effectiveness of a complete applicatio
machine learning components. These performance metrics are u
from the cost function used to train the model. As described in se
common to measure the accuracy, or equivalently, the error rate, o

However, many applications require more advanced metrics.

Sometimes it is much more costly to make one kind of a mistak
For example, an e-mail spam detection system can make two kin
incorrectly classifying a legitimate message as spam, and incorre
spam message to appear in the inbox. It is much worse to blo
message than to allow a questionable message to pass through
measuring the error rate of a spam classifier, we may wish to mea
of total cost, where the cost of blocking legitimate messages is high
of allowing spam messages.

Sometimes we wish to train a binary classifier that is intended
rare event. For example, we might design a medical test for a rare d
that only one in every million people has this disease. We can
99.9999 percent accuracy on the detection task, by simply hard codi
to always report that the disease is absent. Clearly, accuracy is
characterize the performance of such a system. One way to solve

to instead measure **precision** and **recall**. Precision is the fractio
reported by the model that were correct, while recall is the fractio
that were detected. A detector that says no one has the disease
perfect precision, but zero recall. A detector that says everyone
would achieve perfect recall, but precision equal to the percentage
have the disease (0.0001 percent in our example of a disease that on
a million have). When using precision and recall, it is common to pl
with precision on the y-axis and recall on the x-axis. The classifier g
that is higher if the event to be detected occurred. For example

network designed to detect a disease outputs $\hat{y} = P(y = 1 \mid \boldsymbol{x})$, probability that a person whose medical results are described by the disease. We choose to report a detection whenever this score threshold. By varying the threshold, we can trade precision for r cases, we wish to summarize the performance of the classifier with a rather than a curve. To do so, we can convert precision p and r **F-score** given by

$$F = \frac{2pr}{p + r}.$$

Another option is to report the total area lying beneath the PR cu

In some applications, it is possible for the machine learning syst make a decision. This is useful when the machine learning algorith how confident it should be about a decision, especially if a wron be harmful and if a human operator is able to occasionally take o View transcription system provides an example of this situation. transcribe the address number from a photograph to associate the the photo was taken with the correct address in a map. Because t map degrades considerably if the map is inaccurate, it is impor address only if the transcription is correct. If the machine learning that it is less likely than a human being to obtain the correct transcr best course of action is to allow a human to transcribe the photo ins the machine learning system is only useful if it is able to dramatic amount of photos that the human operators must process. A natur metric to use in this situation is **coverage**. Coverage is the fracti for which the machine learning system is able to produce a respons to trade coverage for accuracy. One can always obtain 100 perce refusing to process any example, but this reduces the coverage to 0 p Street View task, the goal for the project was to reach human-lev accuracy while maintaining 95 percent coverage. Human-level perfo task is 98 percent accuracy.

Many other metrics are possible. We can, for example, measur
rates, collect user satisfaction surveys, and so on. Many specialis
areas have application-specific criteria as well.

What is important is to determine which performance metric to
of time, then concentrate on improving this metric. Without clearl
it can be difficult to tell whether changes to a machine learning
progress or not.

11.2 Default Baseline Models

After choosing performance metrics and goals, the next step in application is to establish a reasonable end-to-end system as soon this section, we provide recommendations for which algorithms to baseline approach in various situations. Keep in mind that deep le progresses quickly, so better default algorithms are likely to becom after this writing.

Depending on the complexity of your problem, you may even without using deep learning. If your problem has a chance of b just choosing a few linear weights correctly, you may want to begi statistical model like logistic regression.

If you know that your problem falls into an "AI-complete" cate recognition, speech recognition, machine translation, and so on, the to do well by beginning with an appropriate deep learning model.

First, choose the general category of model based on the str data. If you want to perform supervised learning with fixed-size v use a feedforward network with fully connected layers. If the in topological structure (for example, if the input is an image), use network. In these cases, you should begin by using some kind of unit (ReLUs or their generalizations, such as Leaky ReLUs, PreLus your input or output is a sequence, use a gated recurrent net (LST

A reasonable choice of optimization algorithm is SGD with m a decaying learning rate (popular decay schemes that perform b on different problems include decaying linearly until reaching a learning rate, decaying exponentially, or decreasing the learning rat 2–10 each time validation error plateaus). Another reasonable alter Batch normalization can have a dramatic effect on optimizatio especially for convolutional networks and networks with sigmoida

While it is reasonable to omit batch normalization from the very f
should be introduced quickly if optimization appears to be proble

Unless your training set contains tens of millions of example
should include some mild forms of regularization from the start.
should be used almost universally. Dropout is an excellent regulari
to implement and compatible with many models and training alg
normalization also sometimes reduces generalization error and all
be omitted, because of the noise in the estimate of the statistics us
each variable.

If your task is similar to another task that has been studied e
will probably do well by first copying the model and algorithm
known to perform best on the previously studied task. You may ev
a trained model from that task. For example, it is common to u
from a convolutional network trained on ImageNet to solve other c
tasks (Girshick *et al.*, 2015).

A common question is whether to begin by using unsupervis
scribed further in part III. This is somewhat domain specific. Som
as natural language processing, are known to benefit tremendously
vised learning techniques, such as learning unsupervised word e
other domains, such as computer vision, current unsupervised lear
do not bring a benefit, except in the semi-supervised setting, when
labeled examples is very small (Kingma *et al.*, 2014; Rasmus *et al.*
application is in a context where unsupervised learning is known t
then include it in your first end-to-end baseline. Otherwise, only us
learning in your first attempt if the task you want to solve is uns
can always try adding unsupervised learning later if you observe t
baseline overfits.

11.3 Determining Whether to Gather More

After the first end-to-end system is established, it is time to meas
mance of the algorithm and determine how to improve it. Many m
novices are tempted to make improvements by trying out many diffe
Yet, it is often much better to gather more data than to impro
algorithm.

How does one decide whether to gather more data? First, dete
the performance on the training set is acceptable. If performance
set is poor, the learning algorithm is not using the training data

available, so there is no reason to gather more data. Instead, try
size of the model by adding more layers or adding more hidden uni
Also, try improving the learning algorithm, for example by tunin
rate hyperparameter. If large models and carefully tuned optimiza
do not work well, then the problem might be the *quality* of the trai
data may be too noisy or may not include the right inputs needed
desired outputs. This suggests starting over, collecting cleaner dat
a richer set of features.

If the performance on the training set is acceptable, then m

formance on a test set. If the performance on the test set is a
then there is nothing left to be done. If test set performance is m
training set performance, then gathering more data is one of the
solutions. The key considerations are the cost and feasibility of g
data, the cost and feasibility of reducing the test error by other
amount of data that is expected to be necessary to improve test s
significantly. At large internet companies with millions or billion
feasible to gather large datasets, and the expense of doing so can
less than that of the alternatives, so the answer is almost always
training data. For example, the development of large labeled data
the most important factors in solving object recognition. In other c
medical applications, it may be costly or infeasible to gather more
alternative to gathering more data is to reduce the size of the mo
regularization, by adjusting hyperparameters such as weight dec
or by adding regularization strategies such as dropout. If you fin
between train and test performance is still unacceptable even a
regularization hyperparameters, then gathering more data is advis

When deciding whether to gather more data, it is also neces
how much to gather. It is helpful to plot curves showing the relati
training set size and generalization error, as in figure 5.4. By extr
curves, one can predict how much additional training data woul
achieve a certain level of performance. Usually, adding a small fract
number of examples will not have a noticeable effect on generaliza
therefore recommended to experiment with training set sizes on a lo
for example, doubling the number of examples between consecutiv

If gathering much more data is not feasible, the only other w
generalization error is to improve the learning algorithm itself. Th
domain of research and not the domain of advice for applied pract

11.4 Selecting Hyperparameters

Most deep learning algorithms come with several hyperparamete
many aspects of the algorithm's behavior. Some of these hyperparam
time and memory cost of running the algorithm. Some of these hy
affect the quality of the model recovered by the training process a
infer correct results when deployed on new inputs.

There are two basic approaches to choosing these hyperparam
them manually and choosing them automatically. Choosing the hy

manually requires understanding what the hyperparameters do an
learning models achieve good generalization. Automatic hyperpara
algorithms greatly reduce the need to understand these ideas, but
much more computationally costly.

11.4.1 Manual Hyperparameter Tuning

To set hyperparameters manually, one must understand the relati
hyperparameters, training error, generalization error and computa
(memory and runtime). This means establishing a solid foundatio
mental ideas concerning the effective capacity of a learning algorith
in chapter 5.

The goal of manual hyperparameter search is usually to find the
ization error subject to some runtime and memory budget. We do
to determine the runtime and memory impact of various hyperp
because this is highly platform dependent.

The primary goal of manual hyperparameter search is to adju
capacity of the model to match the complexity of the task. Eff
is constrained by three factors: the representational capacity of
ability of the learning algorithm to successfully minimize the cost f
train the model, and the degree to which the cost function and tra
regularize the model. A model with more layers and more hidden un
higher representational capacity—it is capable of representing mo
functions. It cannot necessarily learn all these functions though,
algorithm cannot discover that certain functions do a good job of
training cost, or if regularization terms such as weight decay forbi
functions.

The generalization error typically follows a U-shaped curve w
a function of one of the hyperparameters, as in figure 5.3. At on

hyperparameter value corresponds to low capacity, and generalizati
because training error is high. This is the underfitting regime. At th
the hyperparameter value corresponds to high capacity, and the
error is high because the gap between training and test error is hi
in the middle lies the optimal model capacity, which achieves the
generalization error, by adding a medium generalization gap to a n
of training error.

For some hyperparameters, overfitting occurs when the valu
parameter is large. The number of hidden units in a layer is one

because increasing the number of hidden units increases the capaci
For some hyperparameters, overfitting occurs when the value of th
ter is small. For example, the smallest allowable weight decay co
corresponds to the greatest effective capacity of the learning algor

Not every hyperparameter will be able to explore the entire U
Many hyperparameters are discrete, such as the number of units i
number of linear pieces in a maxout unit, so it is only possible to vi
along the curve. Some hyperparameters are binary. Usually these h
are switches that specify whether or not to use some optional
the learning algorithm, such as a preprocessing step that norma
features by subtracting their mean and dividing by their standard d
hyperparameters can explore only two points on the curve. Other h
have some minimum or maximum value that prevents them from
part of the curve. For example, the minimum weight decay coefficie
means that if the model is underfitting when weight decay is zero,
the overfitting region by modifying the weight decay coefficient.
some hyperparameters can only subtract capacity.

The learning rate is perhaps the most important hyperpara
have time to tune only one hyperparameter, tune the learning
trols the effective capacity of the model in a more complicated
hyperparameters—the effective capacity of the model is highest wh
rate is *correct* for the optimization problem, not when the learning r
large or especially small. The learning rate has a U-shaped curve fo
illustrated in figure 11.1. When the learning rate is too large, g
can inadvertently increase rather than decrease the training error.
quadratic case, this occurs if the learning rate is at least twice
optimal value (LeCun *et al.*, 1998a). When the learning rate is too
is not only slower but may become permanently stuck with a high
This effect is poorly understood (it would not happen for a convex

Tuning the parameters other than the learning rate requires m
training and test error to diagnose whether your model is overfitting
then adjusting its capacity appropriately.

If your error on the training set is higher than your target erro
no choice but to increase capacity. If you are not using regularizati
confident that your optimization algorithm is performing correctly,
add more layers to your network or add more hidden units. Unfo
increases the computational costs associated with the model.

If your error on the test set is higher than your target error rat

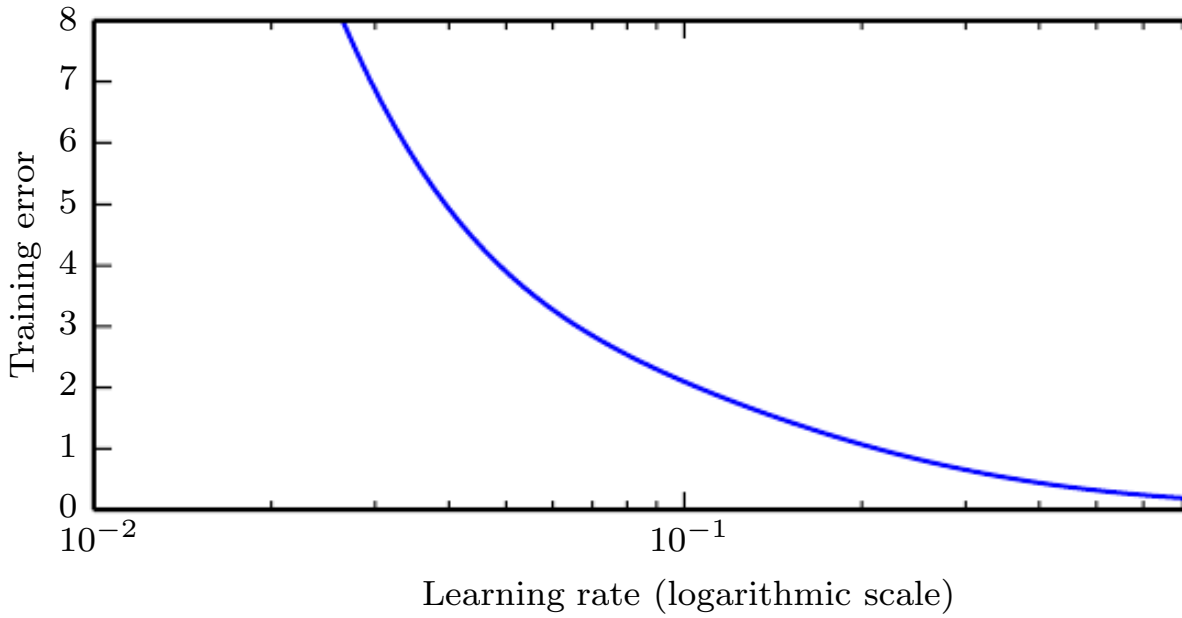

Figure 11.1: Typical relationship between the learning rate and the train
the sharp rise in error when the learning is above an optimal value. T
training time, as a smaller learning rate may sometimes only slow down tra
proportional to the learning rate reduction. Generalization error can
or be complicated by regularization effects arising out of having too la
learning rates, since poor optimization can, to some degree, reduce or pr
and even points with equivalent training error can have different genera

take two kinds of actions. The test error is the sum of the trainin
gap between training and test error. The optimal test error is fo
off these quantities. Neural networks typically perform best wh
error is very low (and thus, when capacity is high) and the test er
driven by the gap between training and test error. Your goal is to
without increasing training error faster than the gap decreases. To
change regularization hyperparameters to reduce effective model ca
by adding dropout or weight decay. Usually the best performanc
large model that is regularized well, for example, by using dropou

Most hyperparameters can be set by reasoning about whether t

decrease model capacity. Some examples are included in table 11.

While manually tuning hyperparameters, do not lose sight of good performance on the test set. Adding regularization is only one this goal. As long as you have low training error, you can always ization error by collecting more training data. The brute force wa guarantee success is to continually increase model capacity and t until the task is solved. This approach does of course increase the cost of training and inference, so it is only feasible given appropria principle, this approach could fail due to optimization difficulties

Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing of hidden both the t cost of ess eration on
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure.	
Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model.	A wider in a narr mension, capacity implicit z reduce th kernels re ory for pa and increa a narrowe memory c
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large.	Increases ory cost tions.
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger.	
Dropout rate	decreased	Dropping units less often gives the units more oppor-	

	tunities to "conspire" with each other to fit the training set.

Table 11.1: The effect of various hyperparameters on model ca

problems optimization does not seem to be a significant barrier, pr
model is chosen appropriately.

11.4.2 Automatic Hyperparameter Optimization Al

The ideal learning algorithm just takes a dataset and outputs a fu
requiring hand tuning of hyperparameters. The popularity of s
algorithms such as logistic regression and SVMs stems in part from
perform well with only one or two tuned hyperparameters. Neura
sometimes perform well with only a small number of tuned hyperp
often benefit significantly from tuning of forty or more. Manual h
tuning can work very well when the user has a good starting
one determined by others having worked on the same type of a
architecture, or when the user has months or years of experien
hyperparameter values for neural networks applied to similar ta
applications, however, these starting points are not available.
automated algorithms can find useful values of the hyperparamete

If we think about the way in which the user of a learning algorit
good values of the hyperparameters, we realize that an optimization
we are trying to find a value of the hyperparameters that optimiz
function, such as validation error, sometimes under constraints (su
for training time, memory or recognition time). It is therefore possi
to develop **hyperparameter optimization** algorithms that w
algorithm and choose its hyperparameters, thus hiding the hyperpa
learning algorithm from the user. Unfortunately, hyperparamete
algorithms often have their own hyperparameters, such as the rang
should be explored for each of the learning algorithm's hyperpara
secondary hyperparameters are usually easier to choose, however, i
acceptable performance may be achieved on a wide range of tasks
secondary hyperparameters for all tasks.

11.4.3 Grid Search

When there are three or fewer hyperparameters, the common practi
grid search. For each hyperparameter, the user selects a sma
values to explore. The grid search algorithm then trains a model
specification of hyperparameter values in the Cartesian product of t
for each individual hyperparameter. The experiment that yields the
set error is then chosen as having found the best hyperparameters
figure 11.2 for an illustration of a grid of hyperparameter values.

How should the lists of values to search over be chosen? In the ca
(ordered) hyperparameters, the smallest and largest element of eac

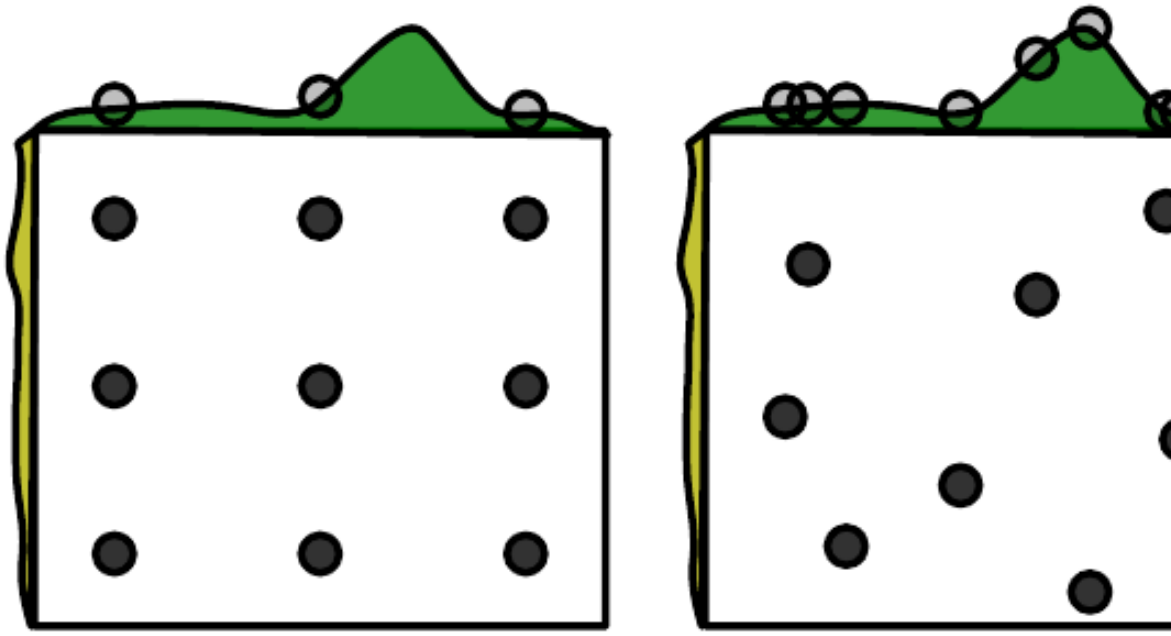

Figure 11.2: Comparison of grid search and random search. For illustrat
display two hyperparameters, but we are typically interested in having m
To perform grid search, we provide a set of values for each hyperparam
algorithm runs training for every joint hyperparameter setting in the cross
sets. *(Right)* To perform random search, we provide a probability distri
hyperparameter configurations. Usually most of these hyperparameters
from each other. Common choices for the distribution over a single hyperp
uniform and log-uniform (to sample from a log-uniform distribution, t
sample from a uniform distribution). The search algorithm then randon
hyperparameter configurations and runs training with each of them. I
and random search evaluate the validation set error and return the be
The figure illustrates the typical case where only some hyperparameters h
influence on the result. In this illustration, only the hyperparameter on th
has a significant effect. Grid search wastes an amount of computation th
in the number of noninfluential hyperparameters, while random searc
value of every influential hyperparameter on nearly every trial. Figure
permission from Bergstra and Bengio (2012).

conservatively, based on prior experience with similar experiment

that the optimal value is likely to be in the selected range. Typicall
involves picking values approximately on a *logarithmic scale*, e.g.,
taken within the set $\{0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$, or a number
taken with the set $\{50, 100, 200, 500, 1000, 2000\}$.

Grid search usually performs best when it is performed repeated
suppose that we ran a grid search over a hyperparameter α using valu
If the best value found is 1, then we underestimated the range in
α lies and should shift the grid and run another search with α i
$\{1, 2, 3\}$. If we find that the best value of α is 0, then we may wis

estimate by zooming in and running a grid search over $\{-0.1, 0, 0$

The obvious problem with grid search is that its computatio exponentially with the number of hyperparameters. If there are m h each taking at most n values, then the number of training and e required grows as $O(n^m)$. The trials may be run in parallel an parallelism (with almost no need for communication between diff carrying out the search). Unfortunately, because of the exponent search, even parallelization may not provide a satisfactory size of

11.4.4 Random Search

Fortunately, there is an alternative to grid search that is as simple to convenient to use, and converges much faster to good values of the h random search (Bergstra and Bengio, 2012).

A random search proceeds as follows. First we define a margi for each hyperparameter, for example, a Bernoulli or multinoul discrete hyperparameters, or a uniform distribution on a log-sc real-valued hyperparameters. For example,

$$\texttt{log\_learning\_rate} \sim u(-1, -5),$$
$$\texttt{learning\_rate} = 10^{\text{log\_learning\_rate}},$$

where $u(a, b)$ indicates a sample of the uniform distribution in th Similarly the `log_number_of_hidden_units` may be sampled log(2000)).

Unlike in a grid search, we *should not discretize* or bin the va perparameters, so that we can explore a larger set of values and a computational cost. In fact, as illustrated in figure 11.2, a random exponentially more efficient than a grid search, when there are s

rameters that do not strongly affect the performance measure. Th
length in Bergstra and Bengio (2012), who found that random sea
validation set error much faster than grid search, in terms of the n
run by each method.

As with grid search, we may often want to run repeated vers
search, to refine the search based on the results of the first run.

The main reason that random search finds good solutions fa
search is that it has no wasted experimental runs, unlike in the cas
when two values of a hyperparameter (given values of the other hy

would give the same result. In the case of grid search, the other hy
would have the same values for these two runs, whereas with rando
would usually have different values. Hence if the change between th
does not marginally make much difference in terms of validation
search will unnecessarily repeat two equivalent experiments while
will still give two independent explorations of the other hyperpara

11.4.5 Model-Based Hyperparameter Optimization

The search for good hyperparameters can be cast as an optimiz
The decision variables are the hyperparameters. The cost to be o
validation set error that results from training using these hyperp
simplified settings where it is feasible to compute the gradient of son
error measure on the validation set with respect to the hyperpara
simply follow this gradient (Bengio *et al.*, 1999; Bengio, 2000; M
2015). Unfortunately, in most practical settings, this gradient is una
because of its high computation and memory cost, or because of hy
that have intrinsically nondifferentiable interactions with the valid
as in the case of discrete-valued hyperparameters.

To compensate for this lack of a gradient, we can build a model o
set error, then propose new hyperparameter guesses by performin
within this model. Most model-based algorithms for hyperparamet
Bayesian regression model to estimate both the expected value of th
error for each hyperparameter and the uncertainty around this exp
mization thus involves a trade-off between exploration (proposing h
for that there is high uncertainty, which may lead to a large improv
also perform poorly) and exploitation (proposing hyperparameters
is confident will perform as well as any hyperparameters it has seen
hyperparameters that are very similar to ones it has seen before).
approaches to hyperparameter optimization include Spearmint (Sno

TPE (Bergstra *et al.*, 2011) and SMAC (Hutter *et al.*, 2011).

Currently, we cannot unambiguously recommend Bayesian h
optimization as an established tool for achieving better deep lea
for obtaining those results with less effort. Bayesian hyperparamet
sometimes performs comparably to human experts, sometimes b
catastrophically on other problems. It may be worth trying to see
particular problem but is not yet sufficiently mature or reliable. T
hyperparameter optimization is an important field of research th
driven primarily by the needs of deep learning, holds the potentia

only the entire field of machine learning but also the discipline of general.

One drawback common to most hyperparameter optimization a more sophistication than random search is that they require for periment to run to completion before they are able to extract a from the experiment. This is much less efficient, in the sense of h mation can be gleaned early in an experiment, than manual sear practitioner, since one can usually tell early on if some set of hyp completely pathological. Swersky *et al.* (2014) have introduced a of an algorithm that maintains a set of multiple experiments. A points, the hyperparameter optimization algorithm can choose t experiment, to "freeze" a running experiment that is not promisin and resume an experiment that was earlier frozen but now appears more information.

11.5 Debugging Strategies

When a machine learning system performs poorly, it is usually whether the poor performance is intrinsic to the algorithm itself or is a bug in the implementation of the algorithm. Machine learni difficult to debug for various reasons.

In most cases, we do not know a priori what the intended b algorithm is. In fact, the entire point of using machine learning discover useful behavior that we were not able to specify ourselve neural network on a *new* classification task and it achieves 5 per we have no straightforward way of knowing if this is the expect suboptimal behavior.

A further difficulty is that most machine learning models have

that are each adaptive. If one part is broken, the other parts can

that are each adaptive. If one part is broken, the other parts can
achieve roughly acceptable performance. For example, suppose that
a neural net with several layers parametrized by weights $\boldsymbol{W}$ and bi
further that we have manually implemented the gradient descen
parameter separately, and we made an error in the update for the

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha,$$

where α is the learning rate. This erroneous update does not use
all. It causes the biases to constantly become negative throughout

is clearly not a correct implementation of any reasonable learning
bug may not be apparent just from examining the output of the
Depending on the distribution of the input, the weights may be a
compensate for the negative biases.

Most debugging strategies for neural nets are designed to get
both of these two difficulties. Either we design a case that is so s
correct behavior actually can be predicted, or we design a test tha
part of the neural net implementation in isolation.

Some important debugging tests include the following.

Visualize the model in action: When training a model to de
images, view some images with the detections proposed by the m
superimposed on the image. When training a generative model of s
some of the speech samples it produces. This may seem obvious, b
fall into the practice of looking only at quantitative performance
like accuracy or log-likelihood. Directly observing the machine
performing its task will help to determine whether the quantitati
numbers it achieves seem reasonable. Evaluation bugs can be so
devastating bugs because they can mislead you into believing
performing well when it is not.

Visualize the worst mistakes: Most models are able to outpu
confidence measure for the task they perform. For example, classif
softmax output layer assign a probability to each class. The proba
to the most likely class thus gives an estimate of the confidence th
its classification decision. Typically, maximum likelihood training
values being overestimates rather than accurate probabilities of cor
but they are somewhat useful in the sense that examples that are ac
to be correctly labeled receive smaller probabilities under the mod
the training set examples that are the hardest to model correctly
discover problems with the way the data have been preprocessed

example, the Street View transcription system originally had a pro
address number detection system would crop the image too tightly
digits. The transcription network then assigned very low probabilit
answer on these images. Sorting the images to identify the most co
showed that there was a systematic problem with the cropping.
detection system to crop much wider images resulted in much bett
of the overall system, even though the transcription network neede
process greater variation in the position and scale of the address n

Reason about software using training and test error: It is of

determine whether the underlying software is correctly implement
can be obtained from the training and test errors. If training error
error is high, then it is likely that that the training procedure w
and the model is overfitting for fundamental algorithmic reasons.
possibility is that the test error is measured incorrectly because of
saving the model after training then reloading it for test set evaluat
the test data was prepared differently from the training data. If bo
test errors are high, then it is difficult to determine whether ther
defect or whether the model is underfitting due to fundamental algo
This scenario requires further tests, described next.

Fit a tiny dataset: If you have high error on the training set, det
it is due to genuine underfitting or due to a software defect. Usu
models can be guaranteed to be able fit a sufficiently small datase
a classification dataset with only one example can be fit just by se
of the output layer correctly. Usually if you cannot train a classif
label a single example, an autoencoder to successfully reproduce a
with high fidelity, or a generative model to consistently emit sampl
single example, there is a software defect preventing successful opti
training set. This test can be extended to a small dataset with fe

Compare back-propagated derivatives to numerical derivatives:
a software framework that requires you to implement your own
putations, or if you are adding a new operation to a differentiat
must define its `bprop` method, then a common source of error is im
gradient expression incorrectly. One way to verify that these deriva
is to compare the derivatives computed by your implementatio
differentiation to the derivatives computed by **finite differences**.

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon},$$

we can approximate the derivative by using a small, finite ϵ:

we can approximate the derivative by using a small, finite ϵ:

$$f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}.$$

We can improve the accuracy of the approximation by using the **ce**
ence:

$$f'(x) \approx \frac{f(x+\frac{1}{2}\epsilon) - f(x-\frac{1}{2}\epsilon)}{\epsilon}.$$

The perturbation size ϵ must be large enough to ensure that the
not rounded down too much by finite-precision numerical comput

Usually, we will want to test the gradient or Jacobian of a vector-
$g : \mathbb{R}^m \to \mathbb{R}^n$. Unfortunately, finite differencing only allows us t
derivative at a time. We can either run finite differencing mn time
the partial derivatives of g, or apply the test to a new function th
projections at both the input and the output of g. For example
our test of the implementation of the derivatives to $f(x)$, where f
and $\boldsymbol{u}$ and $\boldsymbol{v}$ are randomly chosen vectors. Computing $f'(x)$ co
being able to back-propagate through g correctly yet is efficient t
differences because f has only a single input and a single outpu
a good idea to repeat this test for more than one value of $\boldsymbol{u}$ a
the chance of the test overlooking mistakes that are orthogonal
projection.

If one has access to numerical computation on complex numbe
a very efficient way to numerically estimate the gradient by using co
as input to the function (Squire and Trapp, 1998). The method
observation that

$$f(x + i\epsilon) = f(x) + i\epsilon f'(x) + O(\epsilon^2),$$

$$\text{real}(f(x + i\epsilon)) = f(x) + O(\epsilon^2), \quad \text{imag}(\frac{f(x + i\epsilon)}{\epsilon}) = f'(x) + O$$

where $i = \sqrt{-1}$. Unlike in the real-valued case above, there is
effect because we take the difference between the value of f at
This allows the use of tiny values of ϵ, like $\epsilon = 10^{-150}$, which make
insignificant for all practical purposes.

Monitor histograms of activations and gradient: It is often use
statistics of neural network activations and gradients, collected over
of training iterations (maybe one epoch). The preactivation value
can tell us if the units saturate, or how often they do. For examp
how often are they off? Are there units that are always off?

the average of the absolute value of the preactivations tells us
the unit is. In a deep network where the propagated gradients c
quickly vanish, optimization may be hampered. Finally, it is useful
magnitude of parameter gradients to the magnitude of the parame
As suggested by Bottou (2015), we would like the magnitude of par
over a minibatch to represent something like 1 percent of the ma
parameter, not 50 percent or 0.001 percent (which would make
move too slowly). It may be that some groups of parameters are m
pace while others are stalled. When the data is sparse (like in nat

some parameters may be very rarely updated, and this should b
when monitoring their evolution.

Finally, many deep learning algorithms provide some sort of g
the results produced at each step. For example, in part III, we will s
imate inference algorithms that work by using algebraic solutions
problems. Typically these can be debugged by testing each of th
Some guarantees that some optimization algorithms offer include th
function will never increase after one step of the algorithm, that th
respect to some subset of variables will be zero after each step of
and that the gradient with respect to all variables will be zero
Usually due to rounding error, these conditions will not hold exac
computer, so the debugging test should include some tolerance pa

11.6 Example: Multi-Digit Number Recogn

To provide an end-to-end description of how to apply our desig
in practice, we present a brief account of the Street View transc
from the point of view of designing the deep learning componer
many other components of the complete system, such as the Street
database infrastructure, and so on, were of paramount importance

From the point of view of the machine learning task, the proc
data collection. The cars collected the raw data, and human oper
labels. The transcription task was preceded by a significant amo
curation, including using other machine learning techniques to *d*
numbers prior to transcribing them.

The transcription project began with a choice of performan
desired values for these metrics. An important general principle
choice of metric to the business goals for the project. Because maps

if they have high accuracy, it was important to set a high accuracy

if they have high accuracy, it was important to set a high accuracy
this project. Specifically, the goal was to obtain human-level, 98 p
This level of accuracy may not always be feasible to obtain. To rea
accuracy, the Street View transcription system sacrificed coverage.
became the main performance metric optimized during the project
held at 98 percent. As the convolutional network improved, it bec
reduce the confidence threshold below which the network refused to
input, eventually exceeding the goal of 95 percent coverage.

After choosing quantitative goals, the next step in our recomme

ogy is to rapidly establish a sensible baseline system. For vision tasl convolutional network with rectified linear units. The transcriptio with such a model. At the time, it was not common for a convolu to output a sequence of predictions. To begin with the simplest p the first implementation of the output layer of the model consiste softmax units to predict a sequence of n characters. These softn trained exactly the same as if the task were classification, with ea trained independently.

Our recommended methodology is to iteratively refine the ba whether each change makes an improvement. The first change to t transcription system was motivated by a theoretical understanding metric and the structure of the data. Specifically, the network ref an input $\boldsymbol{x}$ whenever the probability of the output sequence $p($ some threshold t. Initially, the definition of $p(\boldsymbol{y} \mid \boldsymbol{x})$ was ad-hoc, b multiplying all the softmax outputs together. This motivated the a specialized output layer and cost function that actually comput log-likelihood. This approach allowed the example rejection mechan much more effectively.

At this point, coverage was still below 90 percent, yet there w theoretical problems with the approach. Our methodology there instrumenting the training and test set performance to determi problem was underfitting or overfitting. In this case, training an were nearly identical. Indeed, the main reason this project proceed was the availability of a dataset with tens of millions of labeled exa training and test set error were so similar, this suggested that the p to either underfitting or a problem with the training data. One of strategies we recommend is to visualize the model's worst errors. In meant visualizing the incorrect training set transcriptions that the highest confidence. These proved to mostly consist of examples w image had been cropped too tightly, with some of the digits of th

removed by the cropping operation. For example, a photo of an
might be cropped too tightly, with only the "849" remaining visible
could have been resolved by spending weeks improving the accuracy
number detection system responsible for determining the cropping r
the team made a much more practical decision, to simply expand t
crop region to be systematically wider than the address number de
predicted. This single change added ten percentage points to th
system's coverage.

Finally, the last few percentage points of performance came
hyperparameters. This mostly consisted of making the model lar
taining some restrictions on its computational cost. Because trai
remained roughly equal, it was always clear that any performan
due to underfitting, as well as to a few remaining problems with th

Overall, the transcription project was a great success and allow
millions of addresses to be transcribed both faster and at lower c
have been possible via human effort.

We hope that the design principles described in this chapter wi
other similar successes.