Chapter 8

Optimization for Trainin Models

Deep learning algorithms involve optimization in many c
performing inference in models such as PCA involves so
problem. We often use analytical optimization to write proo
Of all of the many optimization problems involved in de
difficult is neural network training. It is quite common to in
time on hundreds of machines in order to solve even a single
network training problem. Because this problem is so impo
a specialized set of optimization techniques have been de
This chapter presents these optimization techniques for ne

If you are unfamiliar with the basic principles of gradie
we suggest reviewing chapter 4. That chapter includes a brie

the second derivatives of the cost function. Finally, we con
several optimization strategies that are formed by combini
algorithms into higher-level procedures.

8.1 How Learning Differs from Pure O

Optimization algorithms used for training of deep models
optimization algorithms in several ways. Machine learning
In most machine learning scenarios, we care about some
P, that is defined with respect to the test set and may al
therefore optimize P only indirectly. We reduce a differen
the hope that doing so will improve P. This is in contras
where minimizing J is a goal in and of itself. Optimization
deep models also typically include some specialization on t
machine learning objective functions.

Typically, the cost function can be written as an averag
such as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y),$$

where L is the per-example loss function, $f(\boldsymbol{x};\boldsymbol{\theta})$ is the p
the input is $\boldsymbol{x}$, $\hat{p}_{\text{data}}$ is the empirical distribution. In the su
y is the target output. Throughout this chapter, we dev
supervised case, where the arguments to L are $f(\boldsymbol{x};\boldsymbol{\theta})$ and
to extend this development, for example, to include $\boldsymbol{\theta}$ or
exclude y as arguments, in order to develop various form
unsupervised learning.

Equation 8.1 defines an objective function with respect

solvable by an optimization algorithm. However, when we c
but only have a training set of samples, we have a machine

The simplest way to convert a machine learning prol
timization problem is to minimize the expected loss on t
means replacing the true distribution $p(\boldsymbol{x}, y)$ with the empiri
defined by the training set. We now minimize the **empiric**

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}$$

where m is the number of training examples.

The training process based on minimizing this average t
as **empirical risk minimization**. In this setting, machin
similar to straightforward optimization. Rather than optin
we optimize the empirical risk, and hope that the risk dec
well. A variety of theoretical results establish conditions un
can be expected to decrease by various amounts.

However, empirical risk minimization is prone to ove
high capacity can simply memorize the training set. In
risk minimization is not really feasible. The most effective
algorithms are based on gradient descent, but many usef
as 0-1 loss, have no useful derivatives (the derivative is ei
everywhere). These two problems mean that, in the conte
rarely use empirical risk minimization. Instead, we must
approach, in which the quantity that we actually optimize
from the quantity that we truly want to optimize.

In some cases, a surrogate loss function actually results
more. For example, the test set 0-1 loss often continues
time after the training set 0-1 loss has reached zero, wh
log-likelihood surrogate. This is because even when the ex
one can improve the robustness of the classifier by further pu
from each other, obtaining a more confident and reliable cla
more information from the training data than would have b
minimizing the average 0-1 loss on the training set.

A very important difference between optimization in ge
as we use it for training algorithms is that training algorith
at a local minimum. Instead, a machine learning algorit
a surrogate loss function but halts when a convergence cr
stopping (section 7.8) is satisfied. Typically the early stop
on the true underlying loss function, such as 0-1 loss measu
and is designed to cause the algorithm to halt whenever ove
Training often halts while the surrogate loss function still
which is very different from the pure optimization setting,
algorithm is considered to have converged when the gradie

8.1.3 Batch and Minibatch Algorithms

One aspect of machine learning algorithms that separat
optimization algorithms is that the objective function usuall
over the training examples. Optimization algorithms for ma
compute each update to the parameters based on an expe
function estimated using only a subset of the terms of the

For example, maximum likelihood estimation problem most commonly used property is the gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y} \sim \hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{model}}(\boldsymbol{x}, y;$$

Computing this expectation exactly is very expensiv
evaluating the model on every example in the entire datas
compute these expectations by randomly sampling a sma
from the dataset, then taking the average over only those

Recall that the standard error of the mean (equation 5
samples is given by $\sigma/\sqrt{n}$, where σ is the true standard de
the samples. The denominator of $\sqrt{n}$ shows that there are
to using more examples to estimate the gradient. Com
estimates of the gradient, one based on 100 examples and a
examples. The latter requires 100 times more computatio
reduces the standard error of the mean only by a factor of
algorithms converge much faster (in terms of total compu
number of updates) if they are allowed to rapidly compute
of the gradient rather than slowly computing the exact gra

Another consideration motivating statistical estimation
small number of samples is redundancy in the training set
m samples in the training set could be identical copies of e
based estimate of the gradient could compute the correct
sample, using m times less computation than the naive ap
are unlikely to truly encounter this worst-case situation,
numbers of examples that all make very similar contributi

Optimization algorithms that use the entire training s
deterministic gradient methods, because they process all o

than one but less than all of the training examples. These w
minibatch or **minibatch stochastic** methods and it is n
call them **stochastic** methods.

The canonical example of a stochastic method is stoch
presented in detail in section 8.3.1.

Minibatch sizes are generally driven by the following fa

- Larger batches provide a more accurate estimate of
  less than linear returns.

- Multicore architectures are usually underutilized by e
  This motivates using some absolute minimum batch s
  is no reduction in the time to process a minibatch.

- If all examples in the batch are to be processed in p
  the case), then the amount of memory scales with th
  hardware setups this is the limiting factor in batch si

- Some kinds of hardware achieve better runtime with
  Especially when using GPUs, it is common for power
  better runtime. Typical power of 2 batch sizes range f
  sometimes being attempted for large models.

- Small batches can offer a regularizing effect (Wilson
  perhaps due to the noise they add to the learning p
  error is often best for a batch size of 1. Training wi
  size might require a small learning rate to maintain st
  variance in the estimate of the gradient. The total ru
  due to the need to make more steps, both because o

$\boldsymbol{H}$ or its inverse amplifies pre-existing errors, in this case,
Very small changes in the estimate of $\boldsymbol{g}$ can thus cause large
$\boldsymbol{H}^{-1}\boldsymbol{g}$, even if $\boldsymbol{H}$ were estimated perfectly. Of course, $\boldsymbol{H}$
approximately, so the update $\boldsymbol{H}^{-1}\boldsymbol{g}$ will contain even mor
predict from applying a poorly conditioned operation to th

It is also crucial that the minibatches be selected ran
unbiased estimate of the expected gradient from a set of sam
samples be independent. We also wish for two subsequent g
independent from each other, so two subsequent minibatch
also be independent from each other. Many datasets are m
in a way where successive examples are highly correlated.
have a dataset of medical data with a long list of blood sa
list might be arranged so that first we have five blood sam
times from the first patient, then we have three blood sa
second patient, then the blood samples from the third pa
were to draw examples in order from this list, then each of
be extremely biased, because it would represent primarily
many patients in the dataset. In cases such as these where t
holds some significance, it is necessary to shuffle the exa
minibatches. For very large datasets, for example datasets
examples in a data center, it can be impractical to sample ex
at random every time we want to construct a minibatch. F
it is usually sufficient to shuffle the order of the dataset on
shuffled fashion. This will impose a fixed set of possible min
examples that all models trained thereafter will use, and
will be forced to reuse this ordering every time it passes
data. However, this deviation from true random selection d

descent shuffle the dataset once and then pass through it r
first pass, each minibatch is used to compute an unbiase
generalization error. On the second pass, the estimate becor
formed by re-sampling values that have already been used,
new fair samples from the data generating distribution.

The fact that stochastic gradient descent minimizes
easiest to see in the online learning case, where examples or
from a **stream** of data. In other words, instead of receivir
set, the learner is similar to a living being who sees a new e
with every example $(\boldsymbol{x}, y)$ coming from the data generating
In this scenario, examples are never repeated; every expe
from p_{data}.

The equivalence is easiest to derive when both $\boldsymbol{x}$ and
case, the generalization error (equation 8.2) can be writter

$$J^*(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) L(f(\boldsymbol{x}; \boldsymbol{\theta}), y$$

with the exact gradient

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J^*(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}$$

We have already seen the same fact demonstrated for the
tion 8.5 and equation 8.6; we observe now that this holds
besides the likelihood. A similar result can be derived when
under mild assumptions regarding p_{data} and L.

Hence, we can obtain an unbiased estimator of the

of course, the additional epochs usually provide enough be
training error to offset the harm they cause by increasing th
error and test error.

With some datasets growing rapidly in size, faster tha
is becoming more common for machine learning applicatio
example only once or even to make an incomplete pass
set. When using an extremely large training set, overfitt
underfitting and computational efficiency become the pred
also Bottou and Bousquet (2008) for a discussion of the e
bottlenecks on generalization error, as the number of train

8.2 Challenges in Neural Network Opti

Optimization in general is an extremely difficult task.
learning has avoided the difficulty of general optimization
the objective function and constraints to ensure that the o
convex. When training neural networks, we must confront
case. Even convex optimization is not without its complic
we summarize several of the most prominent challenges in
for training deep models.

8.2.1 Ill-Conditioning

Some challenges arise even when optimizing convex functio
prominent is ill-conditioning of the Hessian matrix $\boldsymbol{H}$.
problem in most numerical optimization, convex or otherw

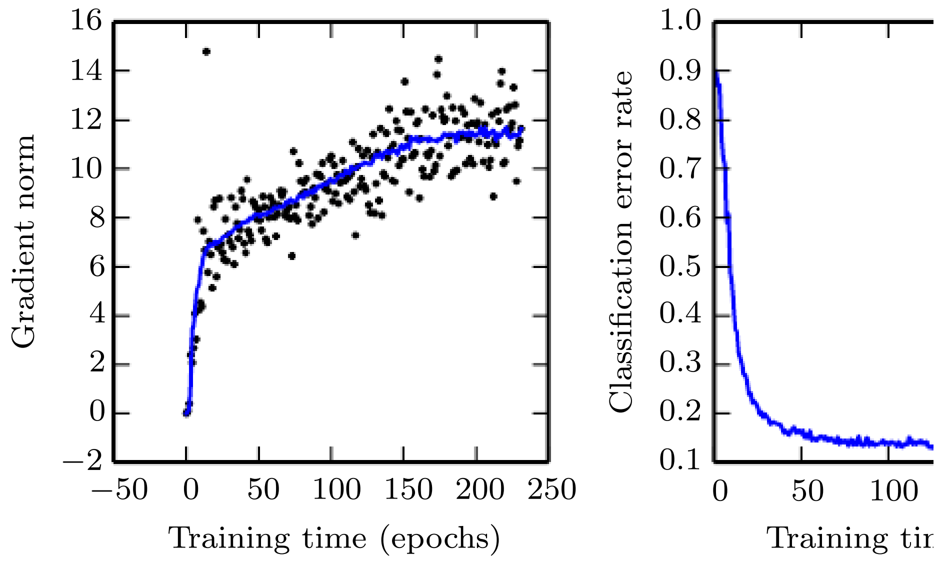


Figure 8.1: Gradient descent often does not arrive at a critical p
example, the gradient norm increases throughout training of a co
for object detection. *(Left)*A scatterplot showing how the norn
evaluations are distributed over time. To improve legibility,
is plotted per epoch. The running average of all gradient nor
curve. The gradient norm clearly increases over time, rather thar
expect if the training process converged to a critical point. *(Rig*
gradient, the training process is reasonably successful. The val
error decreases to a low level.

the $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ term. In many cases, the gradient norm does r
throughout learning, but the $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ term grows by more thar
The result is that learning becomes very slow despite the
gradient because the learning rate must be shrunk to compe
curvature. Figure 8.1 shows an example of the gradient i
during the successful training of a neural network.

guaranteed to be a global minimum. Some convex functior
the bottom rather than a single global minimum point, but
a flat region is an acceptable solution. When optimizing
know that we have reached a good solution if we find a cri

With non-convex functions, such as neural nets, it is
local minima. Indeed, nearly any deep model is essential
an extremely large number of local minima. However, as
necessarily a major problem.

Neural networks and any models with multiple equivalen
variables all have multiple local minima because of the **r**
problem. A model is said to be identifiable if a sufficiently
rule out all but one setting of the model's parameters. Mode
are often not identifiable because we can obtain equivalent
latent variables with each other. For example, we could tak
modify layer 1 by swapping the incoming weight vector for u
weight vector for unit j, then doing the same for the outgoin
have m layers with n units each, then there are $n!^m$ ways o
units. This kind of non-identifiability is known as **weight**

In addition to weight space symmetry, many kinds of
additional causes of non-identifiability. For example, in
maxout network, we can scale all of the incoming weights
α if we also scale all of its outgoing weights by $\frac{1}{\alpha}$. This m
function does not include terms such as weight decay that
weights rather than the models' outputs—every local minim
or maxout network lies on an $(m \times n)$-dimensional hyperb
minima.

These model identifiability issues mean that there can

for networks of practical interest and whether optimizatio
them. For many years, most practitioners believed that
common problem plaguing neural network optimization.
appear to be the case. The problem remains an active area
now suspect that, for sufficiently large neural networks, mo
low cost function value, and that it is not important to find
rather than to find a point in parameter space that has lo
(Saxe *et al.*, 2013; Dauphin *et al.*, 2014; Goodfellow *et a*
et al., 2014).

Many practitioners attribute nearly all difficulty with n
tion to local minima. We encourage practitioners to ca
problems. A test that can rule out local minima as the
norm of the gradient over time. If the norm of the gradi
insignificant size, the problem is neither local minima nor a
point. This kind of negative test can rule out local minim
spaces, it can be very difficult to positively establish that
problem. Many structures other than local minima also ha

8.2.3 Plateaus, Saddle Points and Other Flat

For many high-dimensional non-convex functions, local
are in fact rare compared to another kind of point with z
point. Some points around a saddle point have greater cost
while others have a lower cost. At a saddle point, the He
positive and negative eigenvalues. Points lying along eigen
positive eigenvalues have greater cost than the saddle po
along negative eigenvalues have lower value. We can thin

be heads. See Dauphin *et al.* (2014) for a review of the rel

An amazing property of many random functions is that
Hessian become more likely to be positive as we reach re
our coin tossing analogy, this means we are more likely to
heads n times if we are at a critical point with low cost.
minima are much more likely to have low cost than high co
high cost are far more likely to be saddle points. Critical
high cost are more likely to be local maxima.

This happens for many classes of random functions. Do
networks? Baldi and Hornik (1989) showed theoretically tha
(feedforward networks trained to copy their input to thei
chapter 14) with no nonlinearities have global minima an
local minima with higher cost than the global minimum.
proof that these results extend to deeper networks witho
output of such networks is a linear function of their inpu
to study as a model of nonlinear neural networks becaus
a non-convex function of their parameters. Such netwo
multiple matrices composed together. Saxe *et al.* (2013) p
to the complete learning dynamics in such networks and s
these models captures many of the qualitative features obs
deep models with nonlinear activation functions. Dauphi
experimentally that real neural networks also have loss func
many high-cost saddle points. Choromanska *et al.* (201
theoretical arguments, showing that another class of hig
functions related to neural networks does so as well.

What are the implications of the proliferation of saddle
rithms? For first-order optimization algorithms that use onl

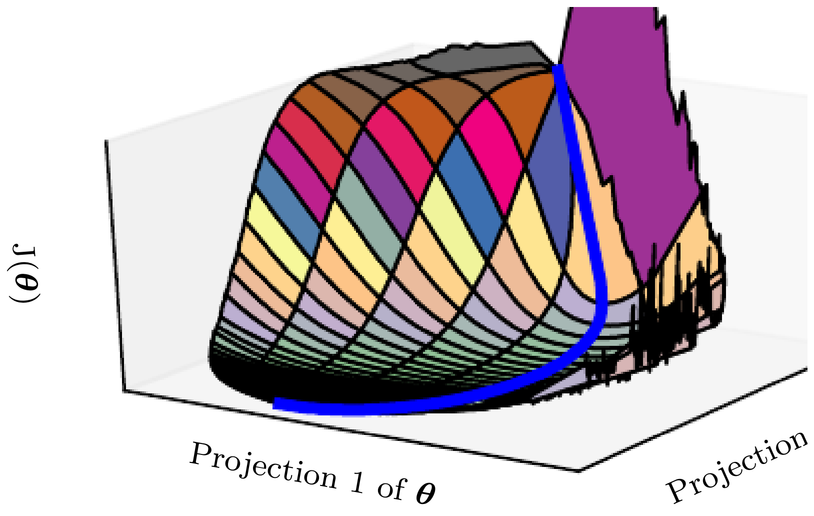


Figure 8.2: A visualization of the cost function of a neural n
with permission from Goodfellow *et al.* (2015). These visualiza
feedforward neural networks, convolutional networks, and rec
to real object recognition and natural language processing ta
visualizations usually do not show many conspicuous obstacles.
stochastic gradient descent for training very large models beg
neural net cost function surfaces were generally believed to hav
structure than is revealed by these projections. The primary c
projection is a saddle point of high cost near where the paramete

Gradient descent is designed to move "downhill" and is r
to seek a critical point. Newton's method, however, is d
point where the gradient is zero. Without appropriate mo
to a saddle point. The proliferation of saddle points in hi
presumably explains why second-order methods have not
gradient descent for neural network training. Dauphin *et*
saddle-free Newton method for second-order optimizat
improves significantly over the traditional version. Second-
difficult to scale to large neural networks, but this saddl
promise if it could be scaled.

There are other kinds of points with zero gradient besi
points. There are also maxima, which are much like sa
perspective of optimization—many algorithms are not a
unmodified Newton's method is. Maxima of many classe
become exponentially rare in high dimensional space, just

There may also be wide, flat regions of constant value.
gradient and also the Hessian are all zero. Such degenerat
problems for all numerical optimization algorithms. In a c
flat region must consist entirely of global minima, but in
problem, such a region could correspond to a high value of

8.2.4 Cliffs and Exploding Gradients

Neural networks with many layers often have extremely st
cliffs, as illustrated in figure 8.3. These result from the m
large weights together. On the face of an extremely ste
gradient update step can move the parameters extremely f

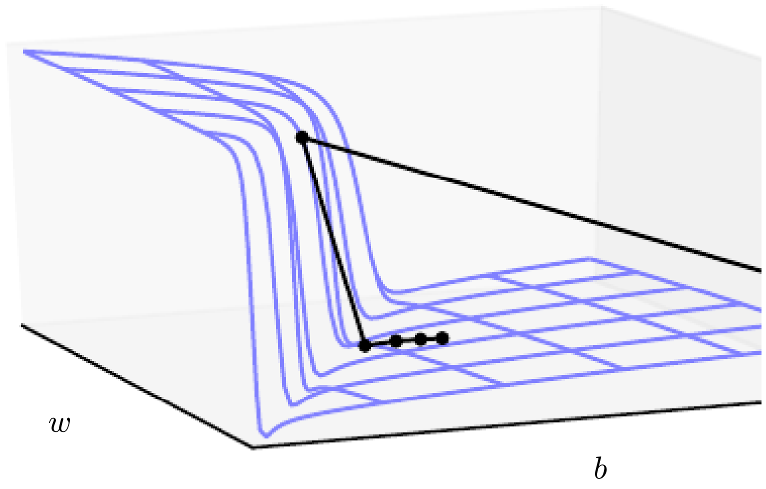


Figure 8.3: The objective function for highly nonlinear deep
recurrent neural networks often contains sharp nonlinearities in p
from the multiplication of several parameters. These nonline
high derivatives in some places. When the parameters get close
gradient descent update can catapult the parameters very far, po
optimization work that had been done. Figure adapted with p
et al. (2013).

The cliff can be dangerous whether we approach it fron
but fortunately its most serious consequences can be avoid
clipping heuristic described in section 10.11.1. The basi
the gradient does not specify the optimal step size, but only
within an infinitesimal region. When the traditional gradi
proposes to make a very large step, the gradient clipping
reduce the step size to be small enough that it is less likely t
where the gradient indicates the direction of approximately

by repeatedly applying the same operation at each time s
sequence. Repeated application of the same parameters g
pronounced difficulties.

For example, suppose that a computational graph contai
of repeatedly multiplying by a matrix $\boldsymbol{W}$. After t steps, th
tiplying by $\boldsymbol{W}^t$. Suppose that $\boldsymbol{W}$ has an eigendecompositio
In this simple case, it is straightforward to see that

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}$$

Any eigenvalues λ_i that are not near an absolute value of 1 wi
are greater than 1 in magnitude or vanish if they are less tha
vanishing and exploding gradient problem refers to t
through such a graph are also scaled according to diag($\boldsymbol{\lambda}$)
make it difficult to know which direction the parameters sh
the cost function, while exploding gradients can make learn
structures described earlier that motivate gradient clipping
exploding gradient phenomenon.

The repeated multiplication by $\boldsymbol{W}$ at each time step
similar to the **power method** algorithm used to find the
a matrix $\boldsymbol{W}$ and the corresponding eigenvector. From t
not surprising that $\boldsymbol{x}^\top \boldsymbol{W}^t$ will eventually discard all com
orthogonal to the principal eigenvector of $\boldsymbol{W}$.

Recurrent networks use the same matrix $\boldsymbol{W}$ at each tim
networks do not, so even very deep feedforward networks
vanishing and exploding gradient problem (Sussillo, 2014).

We defer a further discussion of the challenges of traini

with the more advanced models in part III. For example, c
gives a technique for approximating the gradient of the int
of a Boltzmann machine.

Various neural network optimization algorithms are d
imperfections in the gradient estimate. One can also avoid th
a surrogate loss function that is easier to approximate than

8.2.7 Poor Correspondence between Local and

Many of the problems we have discussed so far correspon
loss function at a single point—it can be difficult to make
poorly conditioned at the current point $\boldsymbol{\theta}$, or if $\boldsymbol{\theta}$ lies on a
point hiding the opportunity to make progress downhill fro

It is possible to overcome all of these problems at a
perform poorly if the direction that results in the most imp
not point toward distant regions of much lower cost.

Goodfellow *et al.* (2015) argue that much of the runtim
the length of the trajectory needed to arrive at the solution.
the learning trajectory spends most of its time tracing ou
mountain-shaped structure.

Much of research into the difficulties of optimization h
training arrives at a global minimum, a local minimum, or
practice neural networks do not arrive at a critical point o
shows that neural networks often do not arrive at a region of
such critical points do not even necessarily exist. For exam
$-\log p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$ can lack a global minimum point and i

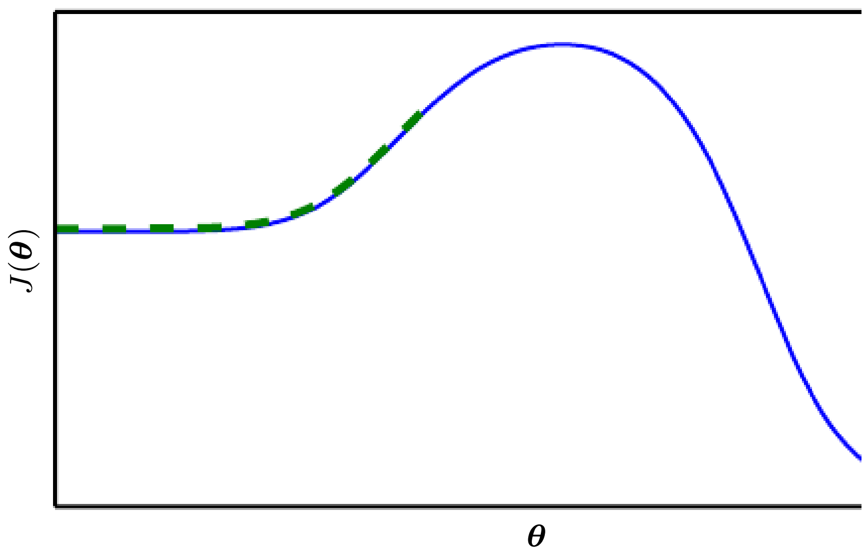


Figure 8.4: Optimization based on local downhill moves can fail
not point toward the global solution. Here we provide an examp
even if there are no saddle points and no local minima. Thi
contains only asymptotes toward low values, not minima. The m
this case is being initialized on the wrong side of the "mountai
traverse it. In higher dimensional space, learning algorithms c
such mountains but the trajectory associated with doing so m
excessive training time, as illustrated in figure 8.2.

of the process.

Many existing research directions are aimed at finding
problems that have difficult global structure, rather than
that use non-local moves.

Gradient descent and essentially all learning algorithm
training neural networks are based on making small, local

high computational cost. Sometimes local information prov
the function has a wide flat region, or if we manage to lan
point (usually this latter scenario only happens to methoc
for critical points, such as Newton's method). In these ca
not define a path to a solution at all. In other cases, local m
and lead us along a path that moves downhill but away fr
figure 8.4, or along an unnecessarily long trajectory to the s
Currently, we do not understand which of these problems
making neural network optimization difficult, and this is an

Regardless of which of these problems are most significar
avoided if there exists a region of space connected reasonabl
by a path that local descent can follow, and if we are abl
within that well-behaved region. This last view suggests r
good initial points for traditional optimization algorithms

8.2.8 Theoretical Limits of Optimization

Several theoretical results show that there are limits on tl
optimization algorithm we might design for neural netwo
1992; Judd, 1989; Wolpert and MacReady, 1997). Typica
little bearing on the use of neural networks in practice.

Some theoretical results apply only to the case where
network output discrete values. However, most neural
smoothly increasing values that make optimization via loca
theoretical results show that there exist problem classes th
it can be difficult to tell whether a particular problem falls
results show that finding a solution for a network of a given

8.3 Basic Algorithms

We have previously introduced the gradient descent (secti
follows the gradient of an entire training set downhill. Tl
considerably by using stochastic gradient descent to follow tl
selected minibatches downhill, as discussed in section 5.9 a

8.3.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) and its variants are pı
optimization algorithms for machine learning in general
in particular. As discussed in section 8.1.3, it is possible
estimate of the gradient by taking the average gradient
examples drawn i.i.d from the data generating distribution

Algorithm 8.1 shows how to follow this estimate of the

Algorithm 8.1 Stochastic gradient descent (SGD) update

```
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training se
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while
```

A crucial parameter for the SGD algorithm is the learni

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$$

In practice, it is common to decay the learning rate lin

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

with $\alpha = \frac{k}{\tau}$. After iteration τ, it is common to leave ϵ cons

The learning rate may be chosen by trial and error, to choose it by monitoring learning curves that plot the c function of time. This is more of an art than a science, and subject should be regarded with some skepticism. When us the parameters to choose are ϵ_0, ϵ_τ, and τ. Usually τ may b iterations required to make a few hundred passes through th ϵ_τ should be set to roughly 1% the value of ϵ_0. The main qu If it is too large, the learning curve will show violent osci function often increasing significantly. Gentle oscillations training with a stochastic cost function such as the cost fun use of dropout. If the learning rate is too low, learning proc initial learning rate is too low, learning may become stuck Typically, the optimal initial learning rate, in terms of total final cost value, is higher than the learning rate that yields after the first 100 iterations or so. Therefore, it is usually b several iterations and use a learning rate that is higher tha learning rate at this time, but not so high that it causes se

The most important property of SGD and related minib based optimization is that computation time per update d

and Bousquet (2008) argue that it therefore may not be
an optimization algorithm that converges faster than $O(\frac{1}{k}$
tasks—faster convergence presumably corresponds to over
asymptotic analysis obscures many advantages that stoch
has after a small number of steps. With large datasets, the
rapid initial progress while evaluating the gradient for o
outweighs its slow asymptotic convergence. Most of the a
the remainder of this chapter achieve benefits that matter
in the constant factors obscured by the $O(\frac{1}{k})$ asymptotic
trade off the benefits of both batch and stochastic gradien
increasing the minibatch size during the course of learning

For more information on SGD, see Bottou (1998).

8.3.2 Momentum

While stochastic gradient descent remains a very popular
learning with it can sometimes be slow. The method of mo
is designed to accelerate learning, especially in the face of hi
consistent gradients, or noisy gradients. The momentum
an exponentially decaying moving average of past gradients
in their direction. The effect of momentum is illustrated in

Formally, the momentum algorithm introduces a variabl
of velocity—it is the direction and speed at which the par
parameter space. The velocity is set to an exponentially d
negative gradient. The name **momentum** derives from
which the negative gradient is a force moving a particle thr
according to Newton's laws of motion. Momentum in physic

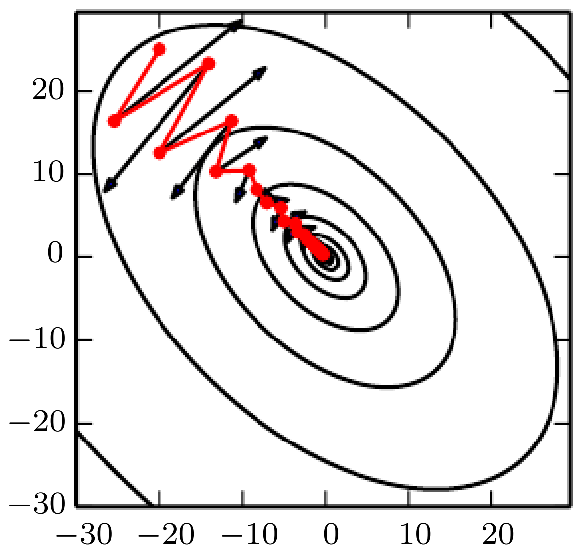


Figure 8.5: Momentum aims primarily to solve two problems:
Hessian matrix and variance in the stochastic gradient. Here, we i
overcomes the first of these two problems. The contour lines
function with a poorly conditioned Hessian matrix. The red
contours indicates the path followed by the momentum learning
function. At each step along the way, we draw an arrow indicati
descent would take at that point. We can see that a poorly condit
looks like a long, narrow valley or canyon with steep sides. Mom

Previously, the size of the step was simply the norm of t
by the learning rate. Now, the size of the step depends o
aligned a *sequence* of gradients are. The step size is largest
gradients point in exactly the same direction. If the momen
observes gradient $\boldsymbol{g}$, then it will accelerate in the direction o
terminal velocity where the size of each step is

$$\frac{\epsilon||\boldsymbol{g}||}{1-\alpha}.$$

It is thus helpful to think of the momentum hyperparamete
example, $\alpha = .9$ corresponds to multiplying the maximum s
the gradient descent algorithm.

Common values of α used in practice include .5, .9, and
rate, α may also be adapted over time. Typically it begins
is later raised. It is less important to adapt α over time tha

Algorithm 8.2 Stochastic gradient descent (SGD) with m

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training se
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

We can view the momentum algorithm as simulating

$$\boldsymbol{f}(t) = \frac{\partial}{\partial t}\boldsymbol{v}(t).$$

The momentum algorithm then consists of solving the diffe
numerical simulation. A simple numerical method for solvin
is Euler's method, which simply consists of simulating th
the equation by taking small, finite steps in the direction c

This explains the basic form of the momentum update, b
the forces? One force is proportional to the negative gradie
$-\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. This force pushes the particle downhill along th
The gradient descent algorithm would simply take a sing
gradient, but the Newtonian scenario used by the momen
uses this force to alter the velocity of the particle. We ca
as being like a hockey puck sliding down an icy surface. W
steep part of the surface, it gathers speed and continues sl
until it begins to go uphill again.

One other force is necessary. If the only force is the gradi
then the particle might never come to rest. Imagine a hoc
one side of a valley and straight up the other side, oscillating
assuming the ice is perfectly frictionless. To resolve this
other force, proportional to $-\boldsymbol{v}(t)$. In physics terminology,
to viscous drag, as if the particle must push through a res
syrup. This causes the particle to gradually lose energy ov
converge to a local minimum.

Why do we use $-\boldsymbol{v}(t)$ and viscous drag in particular?
use $-\boldsymbol{v}(t)$ is mathematical convenience—an integer power
to work with. However, other physical systems have othe
on other integer powers of the velocity. For example, a par

that the gradient can continue to cause motion until a mi
strong enough to prevent motion if the gradient does not j

8.3.3 Nesterov Momentum

Sutskever *et al.* (2013) introduced a variant of the moment
inspired by Nesterov's accelerated gradient method (Neste
update rules in this case are given by:

$$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}}\left[\frac{1}{m}\sum_{i=1}^{m} L\Big(\boldsymbol{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}+\alpha\boldsymbol{v}),\right.$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

where the parameters α and ϵ play a similar role as in the
method. The difference between Nesterov momentum and s
where the gradient is evaluated. With Nesterov momentum th
after the current velocity is applied. Thus one can interpre
as attempting to add a *correction factor* to the standard
The complete Nesterov momentum algorithm is presented

In the convex batch gradient case, Nesterov moment
convergence of the excess error from $O(1/k)$ (after k steps
by Nesterov (1983). Unfortunately, in the stochastic gr
momentum does not improve the rate of convergence.

Algorithm 8.3 Stochastic gradient descent (SGD) with N

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

8.4 Parameter Initialization Strategies

Some optimization algorithms are not iterative by nature
solution point. Other optimization algorithms are iterativ
applied to the right class of optimization problems, converge
in an acceptable amount of time regardless of initialization.
algorithms usually do not have either of these luxuries. Train
learning models are usually iterative in nature and thus req
some initial point from which to begin the iterations. M
models is a sufficiently difficult task that most algorithms a
the choice of initialization. The initial point can determine
converges at all, with some initial points being so unstab
encounters numerical difficulties and fails altogether. When
the initial point can determine how quickly learning con
converges to a point with high or low cost. Also, poin
can have wildly varying generalization error, and the initi
generalization as well.

Modern initialization strategies are simple and heuristi
initialization strategies is a difficult task because neural n
not yet well understood. Most initialization strategies are b
nice properties when the network is initialized. However,
understanding of which of these properties are preserved und
after learning begins to proceed. A further difficulty is th
may be beneficial from the viewpoint of optimization but
viewpoint of generalization. Our understanding of how th
generalization is especially primitive, offering little to no gui
the initial point.

motivates random initialization of the parameters. We c
for a large set of basis functions that are all mutually diff
but this often incurs a noticeable computational cost. For
most as many outputs as inputs, we could use Gram-Schr
on an initial weight matrix, and be guaranteed that each
different function from each other unit. Random initializati
distribution over a high-dimensional space is computationall
to assign any units to compute the same function as each c

Typically, we set the biases for each unit to heuristically
initialize only the weights randomly. Extra parameters, fo
encoding the conditional variance of a prediction, are usua
chosen constants much like the biases are.

We almost always initialize all the weights in the m
randomly from a Gaussian or uniform distribution. Th
or uniform distribution does not seem to matter very mu
exhaustively studied. The scale of the initial distribution,
large effect on both the outcome of the optimization proced
of the network to generalize.

Larger initial weights will yield a stronger symmetry b
to avoid redundant units. They also help to avoid losing si
back-propagation through the linear component of each laye
matrix result in larger outputs of matrix multiplication. I
too large may, however, result in exploding values during f
back-propagation. In recurrent networks, large weights ca
(such extreme sensitivity to small perturbations of the in
of the deterministic forward propagation procedure appea
extent, the exploding gradient problem can be mitigated

due to triggering some early stopping criterion based on c
prior that the final parameters should be close to the init
from section 7.8 that gradient descent with early stopping
decay for some models. In the general case, gradient descent
not the same as weight decay, but does provide a loose anal
the effect of initialization. We can think of initializing the
being similar to imposing a Gaussian prior $p(\boldsymbol{\theta})$ with mea
of view, it makes sense to choose $\boldsymbol{\theta}_0$ to be near 0. This pri
likely that units do not interact with each other than that t
interact only if the likelihood term of the objective funct
preference for them to interact. On the other hand, if w
values, then our prior specifies which units should interact
how they should interact.

Some heuristics are available for choosing the initial sca
heuristic is to initialize the weights of a fully connected la
n outputs by sampling each weight from $U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$, wh
(2010) suggest using the **normalized initialization**

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right).$$

This latter heuristic is designed to compromise between t
all layers to have the same activation variance and the
layers to have the same gradient variance. The formula
assumption that the network consists only of a chain of r
with no nonlinearities. Real neural networks obviously vi
but many strategies designed for the linear model perform
nonlinear counterparts

1,000 layers, without needing to use orthogonal initializat
this approach is that in feedforward networks, activations a
or shrink on each step of forward or back-propagation, fol
behavior. This is because feedforward networks use a diffe
each layer. If this random walk is tuned to preserve nor
networks can mostly avoid the vanishing and exploding g
arises when the same weight matrix is used at each step, de

Unfortunately, these optimal criteria for initial weight
optimal performance. This may be for three different re
be using the wrong criteria—it may not actually be ben
norm of a signal throughout the entire network. Second, t
at initialization may not persist after learning has begun t
criteria might succeed at improving the speed of optimiza
increase generalization error. In practice, we usually need t
weights as a hyperparameter whose optimal value lies somev
not exactly equal to the theoretical predictions.

One drawback to scaling rules that set all of the initia
same standard deviation, such as $\frac{1}{\sqrt{m}}$, is that every indiv
extremely small when the layers become large. Martens
alternative initialization scheme called **sparse initializatio**
initialized to have exactly k non-zero weights. The idea is to
of input to the unit independent from the number of inputs
magnitude of individual weight elements shrink with m. Spa
to achieve more diversity among the units at initialization
imposes a very strong prior on the weights that are chosen
values. Because it takes a long time for gradient descent to s
values, this initialization scheme can cause problems for unit

increasing its weights, it is possible to eventually obtain a n
initial activations throughout. If learning is still too slow a
useful to look at the range or standard deviation of the g
activations. This procedure can in principle be automate
computationally costly than hyperparameter optimization b
error because it is based on feedback from the behavior of
single batch of data, rather than on feedback from a trained r
set. While long used heuristically, this protocol has recent
formally and studied by Mishkin and Matas (2015).

So far we have focused on the initialization of the
initialization of other parameters is typically easier.

The approach for setting the biases must be coordinat
for settings the weights. Setting the biases to zero is compa
initialization schemes. There are a few situations where we
non-zero values:

- If a bias is for an output unit, then it is often beneficial
obtain the right marginal statistics of the output. To
the initial weights are small enough that the output of
only by the bias. This justifies setting the bias to the i
function applied to the marginal statistics of the out
For example, if the output is a distribution over classe
is a highly skewed distribution with the marginal prol
by element c_i of some vector $\boldsymbol{c}$, then we can set the b
the equation $\text{softmax}(\boldsymbol{b}) = \boldsymbol{c}$. This applies not only t
models we will encounter in Part III, such as autoen
machines. These models have layers whose output sho

can view h as a gate that determines whether $uh \approx$
situations, we want to set the bias for h so that $h \approx$
initialization. Otherwise u does not have a chance
Jozefowicz *et al.* (2015) advocate setting the bias to
the LSTM model, described in section 10.10.

Another common type of parameter is a variance or pr
example, we can perform linear regression with a conditi
using the model

$$p(y \mid \boldsymbol{x}) = \mathcal{N}(y \mid \boldsymbol{w}^T \boldsymbol{x} + b, 1/\beta)$$

where β is a precision parameter. We can usually initializ
parameters to 1 safely. Another approach is to assume the i
enough to zero that the biases may be set while ignoring th
then set the biases to produce the correct marginal mean
the variance parameters to the marginal variance of the out

Besides these simple constant or random methods of init
ters, it is possible to initialize model parameters using machi
strategy discussed in part III of this book is to initialize a
the parameters learned by an unsupervised model traine
One can also perform supervised training on a related ta
supervised training on an unrelated task can sometimes yiel
offers faster convergence than a random initialization. Som
strategies may yield faster convergence and better genera
encode information about the distribution in the initial pa
Others apparently perform well primarily because they set
the right scale or set different units to compute different fun

rate for each parameter, and automatically adapt these lea
the course of learning.

The **delta-bar-delta** algorithm (Jacobs, 1988) is an ea
to adapting individual learning rates for model parameter
approach is based on a simple idea: if the partial derivative c
to a given model parameter, remains the same sign, then tl
increase. If the partial derivative with respect to that pa
then the learning rate should decrease. Of course, this ki
applied to full batch optimization.

More recently, a number of incremental (or mini-batch
been introduced that adapt the learning rates of model pa
will briefly review a few of these algorithms.

8.5.1 AdaGrad

The **AdaGrad** algorithm, shown in algorithm 8.4, individua
rates of all model parameters by scaling them inversely pro
root of the sum of all of their historical squared values (D
parameters with the largest partial derivative of the loss l
rapid decrease in their learning rate, while parameters with s
have a relatively small decrease in their learning rate. Tl
progress in the more gently sloped directions of parameter

In the context of convex optimization, the AdaGrad a
desirable theoretical properties. However, empirically it ha
training deep neural network models—the accumulation of
the beginning of training can result in a premature and ex
effective learning rate. AdaGrad performs well for some bu

Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical s
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training se
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.    (Division an
    element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

have made the learning rate too small before arriving at su
RMSProp uses an exponentially decaying average to dis
extreme past so that it can converge rapidly after finding
were an instance of the AdaGrad algorithm initialized with

RMSProp is shown in its standard form in algorithm 8
Nesterov momentum in algorithm 8.6. Compared to Ad
moving average introduces a new hyperparameter, ρ, that c
of the moving average.

Empirically, RMSProp has been shown to be an effec
timization algorithm for deep neural networks. It is curr
optimization methods being employed routinely by deep le

Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabi
   numbers.
   Initialize accumulation variables $\boldsymbol{r} = 0$
   while stopping criterion not met do
      Sample a minibatch of $m$ examples from the training se
      corresponding targets $\boldsymbol{y}^{(i)}$.
      Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
      Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
      Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta+\boldsymbol{r}}} \odot \boldsymbol{g}$.   $(\frac{1}{\sqrt{\delta+}}$
      Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
   end while
```

bias corrections to the estimates of both the first-order mo
term) and the (uncentered) second-order moments to accoun
at the origin (see algorithm 8.7). RMSProp also incorpora
(uncentered) second-order moment, however it lacks the co
unlike in Adam, the RMSProp second-order moment estima
early in training. Adam is generally regarded as being fair
of hyperparameters, though the learning rate sometimes nee
the suggested default.

8.5.4 Choosing the Right Optimization Algori

In this section, we discussed a series of related algorithms th

Algorithm 8.6 RMSProp algorithm with Nesterov mome

```
Require: Global learning rate $\epsilon$, decay rate $\rho$, momentum
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training se
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.    ($\frac{1}{\sqrt{\boldsymbol{r}}}$ a
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

largely on the user's familiarity with the algorithm (for e
tuning).

8.6 Approximate Second-Order Metho

In this section we discuss the application of second-order m
of deep networks. See LeCun *et al.* (1998a) for an earlier tre
For simplicity of exposition, the only objective function we e
risk:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y} \sim \hat{p}_{\text{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)] = \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i}$$

Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimate
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilizatic
  $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training se
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 -$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$   (operations applied
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Newton's method is an optimization scheme based on us
lor series expansion to approximate $J(\boldsymbol{\theta})$ near some point $\boldsymbol{\theta}$
of higher order:

$$\top \qquad 1 \qquad \top$$

Algorithm 8.8 Newton's method with ob $\frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.

```
Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of $m$ examples
  while stopping criterion not met do
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian inverse: $\boldsymbol{H}^{-1}$
    Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

For surfaces that are not quadratic, as long as the He
definite, Newton's method can be applied iteratively. Tl
iterative procedure. First, update or compute the inverse l
ing the quadratic approximation). Second, update the pa
equation 8.27.

In section 8.2.3, we discussed how Newton's method is
the Hessian is positive definite. In deep learning, the su
function is typically non-convex with many features, such
are problematic for Newton's method. If the eigenvalues
all positive, for example, near a saddle point, then Newton
cause updates to move in the wrong direction. This situ
by regularizing the Hessian. Common regularization strat
constant, α, along the diagonal of the Hessian. The regula

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - [H(f(\boldsymbol{\theta}_0)) + \alpha\boldsymbol{I}]^{-1}\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0$$

such as saddle points, the application of Newton's method fo
networks is limited by the significant computational bu
number of elements in the Hessian is squared in the number
k parameters (and for even very small neural networks the
k can be in the millions), Newton's method would require t
matrix—with computational complexity of $O(k^3)$. Also, si
change with every update, the inverse Hessian has to be com
iteration. As a consequence, only networks with a very small
can be practically trained via Newton's method. In the rem
we will discuss alternatives that attempt to gain some of the a
method while side-stepping the computational hurdles.

8.6.2 Conjugate Gradients

Conjugate gradients is a method to efficiently avoid the cal
Hessian by iteratively descending **conjugate directions**.
approach follows from a careful study of the weakness of t
descent (see section 4.3 for details), where line searches are
the direction associated with the gradient. Figure 8.6 illustra
steepest descent, when applied in a quadratic bowl, progresse
back-and-forth, zig-zag pattern. This happens because eac
when given by the gradient, is guaranteed to be orthogon
search direction.

Let the previous search direction be $\boldsymbol{d}_{t-1}$. At the min
search terminates, the directional derivative is zero in dir
$\boldsymbol{d}_{t-1} = 0$. Since the gradient at this point defines the cu
$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ will have no contribution in the direction $\boldsymbol{d}_{t-1}$

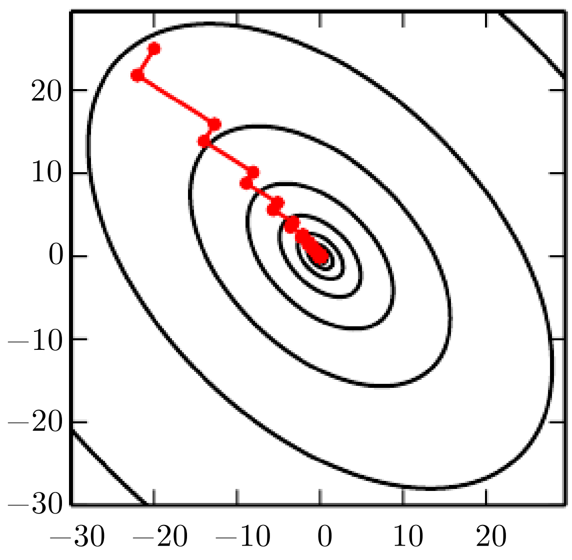


Figure 8.6: The method of steepest descent applied to a quac
method of steepest descent involves jumping to the point of lc
defined by the gradient at the initial point on each step. This resol
seen with using a fixed learning rate in figure 4.6, but even wi
the algorithm still makes back-and-forth progress toward the op
the minimum of the objective along a given direction, the grad
orthogonal to that direction.

the form:

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1}$$

where β_t is a coefficient whose magnitude controls how mucl
we should add back to the current search direction.

Two directions, $\boldsymbol{d}_t$ and $\boldsymbol{d}_{t-1}$, are defined as conjugate i
$\boldsymbol{H}$ is the Hessian matrix.

The straightforward way to impose conjugacy would inv

2. Polak-Ribière:

$$\beta_t = \frac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^\top \nabla_{\boldsymbol{\theta}} J}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

For a quadratic surface, the conjugate directions ensure t
the previous direction does not increase in magnitude. W
minimum along the previous directions. As a consequen
parameter space, the conjugate gradient method requires at
achieve the minimum. The conjugate gradient algorithm is

Algorithm 8.9 The conjugate gradient method

```
Require: Initial parameters $\boldsymbol{\theta}_0$
Require: Training set of $m$ examples
  Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$
  Initialize $g_0 = 0$
  Initialize $t = 1$
  while stopping criterion not met do
    Initialize the gradient $\boldsymbol{g}_t = \mathbf{0}$
    Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)
    (Nonlinear conjugate gradient: optionally reset $\beta_t$ to z
    a multiple of some constant $k$, such as $k = 5$)
    Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$
    Perform line search to find: $\epsilon^* = \text{argmin}_\epsilon \frac{1}{m}\sum_{i=1}^m L(f($
    (On a truly quadratic cost function, analytically so
    explicitly searching for it)
```

are no longer assured to remain at the minimum of the
directions. As a result, the **nonlinear conjugate gradie**
occasional resets where the method of conjugate gradients
search along the unaltered gradient.

Practitioners report reasonable results in applications of t
gradients algorithm to training neural networks, though i
initialize the optimization with a few iterations of stochastic
commencing nonlinear conjugate gradients. Also, while the
gradients algorithm has traditionally been cast as a batc
versions have been used successfully for the training of neu
2011). Adaptations of conjugate gradients specifically for
been proposed earlier, such as the scaled conjugate gradie
1993).

8.6.3 BFGS

The **Broyden–Fletcher–Goldfarb–Shanno (BFGS) a**
bring some of the advantages of Newton's method witho
burden. In that respect, BFGS is similar to the conju
However, BFGS takes a more direct approach to the appro
update. Recall that Newton's update is given by

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0),$$

where $\boldsymbol{H}$ is the Hessian of J with respect to $\boldsymbol{\theta}$ evaluate
computational difficulty in applying Newton's update is
inverse Hessian $\boldsymbol{H}^{-1}$. The approach adopted by quasi-New
the BFGS algorithm is the most prominent) is to approxi

unlike conjugate gradients, the success of the approach is
on the line search finding a point very close to the true m
Thus, relative to conjugate gradients, BFGS has the advan
less time refining each line search. On the other hand, the
store the inverse Hessian matrix, $\boldsymbol{M}$, that requires $O(n^2)$ n
impractical for most modern deep learning models that tyı
parameters.

Limited Memory BFGS (or L-BFGS) The memoı
algorithm can be significantly decreased by avoiding storin
Hessian approximation $\boldsymbol{M}$. The L-BFGS algorithm compute
using the same method as the BFGS algorithm, but beginni
that $\boldsymbol{M}^{(t-1)}$ is the identity matrix, rather than storing the a
step to the next. If used with exact line searches, the directi
are mutually conjugate. However, unlike the method of co
procedure remains well behaved when the minimum of the
only approximately. The L-BFGS strategy with no storage
generalized to include more information about the Hessian
vectors used to update $\boldsymbol{M}$ at each time step, which costs o

8.7 Optimization Strategies and Meta-

Many optimization techniques are not exactly algorithm
templates that can be specialized to yield algorithms, or sı
incorporated into many different algorithms.

example, suppose we have a deep neural network that has
and does not use an activation function at each hidden laye
Here, w_i provides the weight used by layer i. The output o
The output $\hat{y}$ is a linear function of the input x, but a no
weights w_i. Suppose our cost function has put a gradient o
decrease $\hat{y}$ slightly. The back-propagation algorithm can th
$\boldsymbol{g} = \nabla_{\boldsymbol{w}}\hat{y}$. Consider what happens when we make an upd
first-order Taylor series approximation of $\hat{y}$ predicts that the
by $\epsilon\boldsymbol{g}^{\top}\boldsymbol{g}$. If we wanted to decrease $\hat{y}$ by .1, this first-order i
the gradient suggests we could set the learning rate ϵ to $\frac{.1}{\boldsymbol{g}^{\top}}$
update will include second-order and third-order effects, on
The new value of $\hat{y}$ is given by

$$x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)\dots(w_l - \epsilon g_l).$$

An example of one second-order term arising from this up
This term might be negligible if $\prod_{i=3}^{l} w_i$ is small, or might
if the weights on layers 3 through l are greater than 1. T
to choose an appropriate learning rate, because the effec
parameters for one layer depends so strongly on all of the oth
optimization algorithms address this issue by computing an
second-order interactions into account, but we can see that
even higher-order interactions can be significant. Even sec
algorithms are expensive and usually require numerous appr
them from truly accounting for all significant second-order
an n-th order optimization algorithm for $n > 2$ thus seems
do instead?

Batch normalization provides an elegant way of reparame

and dividing by σ_j. The rest of the network then operate
same way that the original network operated on $\boldsymbol{H}$.

At training time,

$$\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{H}_{i,:}$$

and

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2},$$

where δ is a small positive value such as 10^{-8} imposed
the undefined gradient of $\sqrt{z}$ at $z = 0$. Crucially, *we*
these operations for computing the mean and the stand
applying them to normalize $\boldsymbol{H}$. This means that the gradi
an operation that acts simply to increase the standard
h_i; the normalization operations remove the effect of su
out its component in the gradient. This was a major in
normalization approach. Previous approaches had involv
the cost function to encourage units to have normalized
involved intervening to renormalize unit statistics after eac
The former approach usually resulted in imperfect norma
usually resulted in significant wasted time as the learning
proposed changing the mean and variance and the normali
undid this change. Batch normalization reparametrizes th
units always be standardized by definition, deftly sidestep

At test time, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ may be replaced by running avera
during training time. This allows the model to be evaluat
without needing to use definitions of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ that depend

of one of the lower weights can flip the relationship betw
situations are very rare. Without normalization, nearly ev
an extreme effect on the statistics of h_{l-1}. Batch normal
this model significantly easier to learn. In this example,
course came at the cost of making the lower layers useless.
the lower layers no longer have any harmful effect, but th
any beneficial effect. This is because we have normalized o
order statistics, which is all that a linear network can influ
network with nonlinear activation functions, the lower layers
transformations of the data, so they remain useful. Batch
standardize only the mean and variance of each unit in ord
but allows the relationships between units and the nonline
unit to change.

Because the final layer of the network is able to learn a
we may actually wish to remove all linear relationships b
layer. Indeed, this is the approach taken by Desjardins *et al*
the inspiration for batch normalization. Unfortunately,
interactions is much more expensive than standardizing t
deviation of each individual unit, and so far batch normaliz
practical approach.

Normalizing the mean and standard deviation of a unit ca
power of the neural network containing that unit. In
expressive power of the network, it is common to replace th
activations $\boldsymbol{H}$ with $\boldsymbol{\gamma}\boldsymbol{H}' + \boldsymbol{\beta}$ rather than simply the normal
$\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned parameters that allow the new vari
and standard deviation. At first glance, this may seem us
the mean to $\mathbf{0}$, and then introduce a parameter that allo

the latter. More specifically, $\boldsymbol{XW} + \boldsymbol{b}$ should be replaced b
of $\boldsymbol{XW}$. The bias term should be omitted because it be
the β parameter applied by the batch normalization reparar
to a layer is usually the output of a nonlinear activatior
rectified linear function in a previous layer. The statistics
more non-Gaussian and less amenable to standardization b

In convolutional networks, described in chapter 9, it is i
same normalizing μ and σ at every spatial location within
the statistics of the feature map remain the same regardles

8.7.2 Coordinate Descent

In some cases, it may be possible to solve an optimizatio
breaking it into separate pieces. If we minimize $f(\boldsymbol{x})$ wi
variable x_i, then minimize it with respect to another v
repeatedly cycling through all variables, we are guarantee
minimum. This practice is known as **coordinate descent**
one coordinate at a time. More generally, **block coordin**
minimizing with respect to a subset of the variables simu
"coordinate descent" is often used to refer to block coordir
the strictly individual coordinate descent.

Coordinate descent makes the most sense when the dif
optimization problem can be clearly separated into grou
isolated roles, or when optimization with respect to one
significantly more efficient than optimization with respect
For example, consider the cost function

us an optimization strategy that allows us to use efficien
algorithms, by alternating between optimizing $\boldsymbol{W}$ with $\boldsymbol{H}$
$\boldsymbol{H}$ with $\boldsymbol{W}$ fixed.

Coordinate descent is not a very good strategy when th
strongly influences the optimal value of another variable, as
$(x_1 - x_2)^2 + \alpha \left(x_1^2 + x_2^2\right)$ where α is a positive constant. Th
the two variables to have similar value, while the second
to be near zero. The solution is to set both to zero. Newt
the problem in a single step because it is a positive defin
However, for small α, coordinate descent will make very slo
first term does not allow a single variable to be changed
significantly from the current value of the other variable.

8.7.3 Polyak Averaging

Polyak averaging (Polyak and Juditsky, 1992) consists of ave
points in the trajectory through parameter space visite
algorithm. If t iterations of gradient descent visit points
output of the Polyak averaging algorithm is $\hat{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_i \boldsymbol{\theta}$
classes, such as gradient descent applied to convex proble
strong convergence guarantees. When applied to neural ne
is more heuristic, but it performs well in practice. The
optimization algorithm may leap back and forth across
without ever visiting a point near the bottom of the valley
the locations on either side should be close to the bottom

In non-convex problems, the path taken by the optimiz
very complicated and visit many different regions. Includi

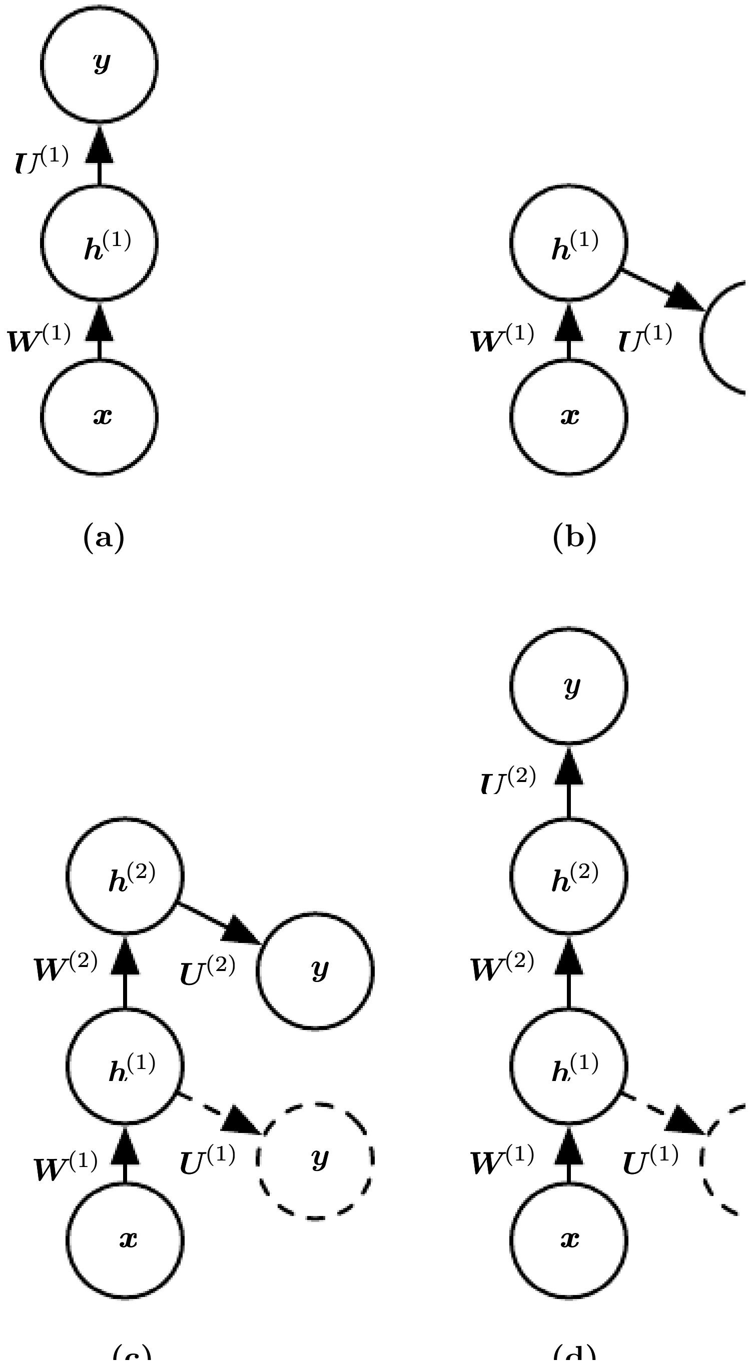
8.7.4 Supervised Pretraining

Sometimes, directly training a model to solve a specific tasl
if the model is complex and hard to optimize or if the tasl
sometimes more effective to train a simpler model to solv
the model more complex. It can also be more effective to tı
a simpler task, then move on to confront the final task.
involve training simple models on simple tasks before confr
training the desired model to perform the desired task are
pretraining.

Greedy algorithms break a problem into many comp
the optimal version of each component in isolation. Unfort
individually optimal components is not guaranteed to yiel
solution. However, greedy algorithms can be computationa
algorithms that solve for the best joint solution, and the qual
is often acceptable if not optimal. Greedy algorithms may
fine-tuning stage in which a joint optimization algorithm s
solution to the full problem. Initializing the joint optimiza
greedy solution can greatly speed it up and improve the qu
finds.

Pretraining, and especially greedy pretraining, algorit
deep learning. In this section, we describe specifically those
that break supervised learning problems into other simpl
problems. This approach is known as **greedy supervised**

In the original (Bengio *et al.*, 2007) version of greedy s
each stage consists of a supervised learning training task inv
the layers in the final neural network. An example of greedy


$\boldsymbol{y}$
$\boldsymbol{U}^{(1)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{W}^{(1)}$
$\boldsymbol{x}$
(a)
$\boldsymbol{h}^{(1)}$
$\boldsymbol{W}^{(1)}$
$\boldsymbol{U}^{(1)}$
$\boldsymbol{x}$
(b)
$\boldsymbol{h}^{(2)}$
$\boldsymbol{W}^{(2)}$
$\boldsymbol{U}^{(2)}$
$\boldsymbol{y}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{W}^{(1)}$
$\boldsymbol{U}^{(1)}$
$\boldsymbol{y}$
$\boldsymbol{x}$
$\boldsymbol{y}$
$\boldsymbol{U}^{(2)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{W}^{(2)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{W}^{(1)}$
$\boldsymbol{U}^{(1)}$
$\boldsymbol{x}$

intermediate levels of a deep hierarchy. In general, pretrai
terms of optimization and in terms of generalization.

An approach related to supervised pretraining extends
of transfer learning: Yosinski *et al.* (2014) pretrain a deep c
layers of weights on a set of tasks (a subset of the 1000 Imag
and then initialize a same-size network with the first k lay
the layers of the second network (with the upper layers ini
then jointly trained to perform a different set of tasks (ano
ImageNet object categories), with fewer training examples
tasks. Other approaches to transfer learning with neural ne
section 15.2.

Another related line of work is the **FitNets** (Romero
This approach begins by training a network that has low e
enough width (number of units per layer) to be easy to tr
becomes a **teacher** for a second network, designated the
network is much deeper and thinner (eleven to nineteen
difficult to train with SGD under normal circumstances
student network is made easier by training the student netw
the output for the original task, but also to predict the va
of the teacher network. This extra task provides a set o
hidden layers should be used and can simplify the optimizati
parameters are introduced to regress the middle layer of the
from the middle layer of the deeper student network. Howeve
the final classification target, the objective is to predict t
of the teacher network. The lower layers of the student n
objectives: to help the outputs of the student network ac
well as to predict the intermediate layer of the teacher net

8.7.5 Designing Models to Aid Optimization

To improve optimization, the best strategy is not always to im
algorithm. Instead, many improvements in the optimizatio
come from designing the models to be easier to optimize.

In principle, we could use activation functions that in
jagged non-monotonic patterns. However, this would make
difficult. In practice, *it is more important to choose a mode
optimize than to use a powerful optimization algorithm.* M
neural network learning over the past 30 years have been
the model family rather than changing the optimization
gradient descent with momentum, which was used to train
1980s, remains in use in modern state of the art neural net

Specifically, modern neural networks reflect a *design ch*
formations between layers and activation functions that ar
everywhere and have significant slope in large portions of
ticular, model innovations like the LSTM, rectified linear u
have all moved toward using more linear functions than pre
networks based on sigmoidal units. These models have nic
optimization easier. The gradient flows through many la
Jacobian of the linear transformation has reasonable sing
linear functions consistently increase in a single direction,
output is very far from correct, it is clear simply from co
which direction its output should move to reduce the loss fu
modern neural nets have been designed so that their *loca*
corresponds reasonably well to moving toward a distant so

Other model design strategies can help to make opt should do, via a shorter path. These hints provide an erroı

8.7.6 Continuation Methods and Curriculum

As argued in section 8.2.7, many of the challenges in optin
global structure of the cost function and cannot be resolved n
estimates of local update directions. The predominant strat
problem is to attempt to initialize the parameters in a re
to the solution by a short path through parameter space
discover.

Continuation methods are a family of strategies that
easier by choosing initial points to ensure that local optim
its time in well-behaved regions of space. The idea behind c
to construct a series of objective functions over the same p
minimize a cost function $J(\boldsymbol{\theta})$, we will construct new cost fur
These cost functions are designed to be increasingly difficult
easy to minimize, and $J^{(n)}$, the most difficult, being $J(\boldsymbol{\theta})$,
motivating the entire process. When we say that $J^{(i)}$ is
mean that it is well behaved over more of $\boldsymbol{\theta}$ space. A rando
likely to land in the region where local descent can minin
successfully because this region is larger. The series of cost
so that a solution to one is a good initial point of the ne
solving an easy problem then refine the solution to solve
problems until we arrive at a solution to the true underlyi

Traditional continuation methods (predating the use of
for neural network training) are usually based on smoothing
See Wu (1997) for an example of such a method and a r

become approximately convex when blurred. In many cases,
enough information about the location of a global minimu
global minimum by solving progressively less blurred versio
approach can break down in three different ways. First, it m
a series of cost functions where the first is convex and the
one function to the next arriving at the global minimum, l
many incremental cost functions that the cost of the entire p
NP-hard optimization problems remain NP-hard, even wher
are applicable. The other two ways that continuation metho
to the method not being applicable. First, the function mig
no matter how much it is blurred. Consider for example the
Second, the function may become convex as a result of blur
of this blurred function may track to a local rather than a
original cost function.

Though continuation methods were mostly originally de
problem of local minima, local minima are no longer belie
problem for neural network optimization. Fortunately, con
still help. The easier objective functions introduced by the c
eliminate flat regions, decrease variance in gradient estimate
of the Hessian matrix, or do anything else that will eith
easier to compute or improve the correspondence between
and progress toward a global solution.

Bengio *et al.* (2009) observed that an approach called
or **shaping** can be interpreted as a continuation method.
based on the idea of planning a learning process to begin by le
and progress to learning more complex concepts that de
concepts. This basic strategy was previously known to accele

more prototypical examples and then help the learner refi
with the less obvious cases. Curriculum-based strategies
teaching humans than strategies based on uniform samplin
also increase the effectiveness of other teaching strategies (
2013).

Another important contribution to research on curriculu
context of training recurrent neural networks to capture lo
Zaremba and Sutskever (2014) found that much better resul
stochastic curriculum, in which a random mix of easy and diffi
presented to the learner, but where the average proportio
examples (here, those with longer-term dependencies) is gra
a deterministic curriculum, no improvement over the base
from the full training set) was observed.

We have now described the basic family of neural netw
regularize and optimize them. In the chapters ahead, we tu
the neural network family, that allow neural networks to scal
process input data that has special structure. The optimiza
in this chapter are often directly applicable to these special
little or no modification.