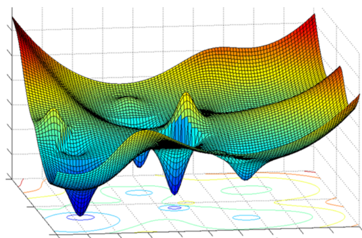
CSC321 Lecture 8
Optimization tricks

Roger Grosse and Nitish Srivastava

January 29, 2015

Overview

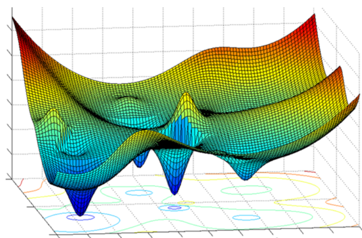
When training very deep networks, we are optimizing a non-convex function over millions of parameters.



Lots of saddle points and local minima!

Overview

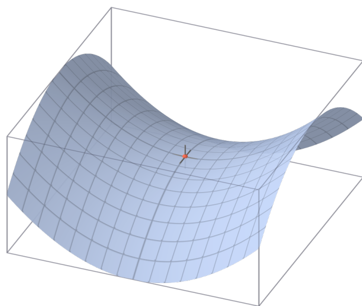
When training very deep networks, we are optimizing a non-convex function over millions of parameters.



Lots of saddle points and local minima!

Strangely, we still don't know if local minima are a problem for most neural nets!

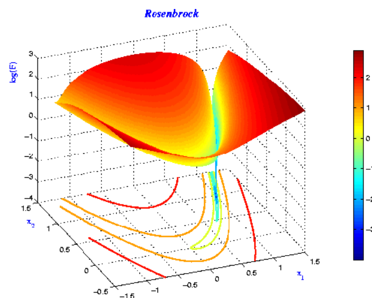
Saddle points



At a saddle point $\frac{\partial C}{\partial w} = 0$ along all dimensions, even though we are not at a minima.

E.g. symmetries: if two hidden units have identical weights, there's nothing to push them apart

Long narrow ravines



Lots of sloshing around the walls, only a small derivative along the slope of the ravine's floor.

This is like the situation we encountered where the level sets were elongated ellipses.

Tricks of the trade

Choices to be made -

- Initial Weight scale and distribution.
- Learning rate, decay schedule.
- Momentum + Nesterov.
- Stochastic (minibatch) vs batch gradient descent.
- Input normalization / rotation.
- Adaptive learning rates, RMSprop.

Knowing the details of these methods is less important than being able to diagnose what's going wrong in the learning.

Thinking about gradient descent

Three ways to simplify our picture of the dynamics of SGD:

1. Smooth objective functions can be locally approximated by a quadratic function, i.e. the second-order Taylor approximation. The level sets of a quadratic function are ellipses.

Thinking about gradient descent

Three ways to simplify our picture of the dynamics of SGD:

1. Smooth objective functions can be locally approximated by a quadratic function, i.e. the second-order Taylor approximation. The level sets of a quadratic function are ellipses.
2. We can understand the dynamics for general quadratics by looking at axis-aligned quadratics, since every quadratic function is a rotation of an axis-aligned one.

Thinking about gradient descent

Three ways to simplify our picture of the dynamics of SGD:

1. Smooth objective functions can be locally approximated by a quadratic function, i.e. the second-order Taylor approximation. The level sets of a quadratic function are ellipses.
2. We can understand the dynamics for general quadratics by looking at axis-aligned quadratics, since every quadratic function is a rotation of an axis-aligned one.
3. For axis-aligned quadratics, every weight evolves independently. Therefore, we can understand the dynamics by looking at one weight at a time.

Thinking about gradient descent

Three ways to simplify our picture of the dynamics of SGD:

1. Smooth objective functions can be locally approximated by a quadratic function, i.e. the second-order Taylor approximation. The level sets of a quadratic function are ellipses.
2. We can understand the dynamics for general quadratics by looking at axis-aligned quadratics, since every quadratic function is a rotation of an axis-aligned one.
3. For axis-aligned quadratics, every weight evolves independently. Therefore, we can understand the dynamics by looking at one weight at a time.

Therefore, we really only need to look at quadratics in one variable!

Question 1 : Setting learning rates

Suppose we have the following loss function over a real number w.

$$C(w) = \frac{1}{2}w^2$$

we would like to minimize this function using gradient descent. so we start at an initial value $w^{(0)} = 1$ and update the weight at time step t as

$$w^{(t+1)} = w^{(t)} - \epsilon \frac{\partial C}{\partial w}\bigg|_t$$

Compute the weights after the first three steps : $w^{(1)}$, $w^{(2)}$ and $w^{(3)}$ in each of the following situations

- $\epsilon = 0.5$.
- $\epsilon = 1$.
- $\epsilon = 2$.
- $\epsilon = 3$.

in which case(s) do you think, the model will converge if we keep updating w for a long time ?

Question 1 : Setting learning rates

Suppose we have the following loss function over a real number w.

$$C(w) = \frac{1}{2}w^2$$

We would like to minimize this function using gradient descent. So we start we an initial value $w^{(0)}$ and update the weight at time step t as

$$w^{(t+1)} = w^{(t)} - \epsilon \frac{\partial C}{\partial w}\bigg|_t$$

Compute the weights after the first three steps : $w^{(1)}$, $w^{(2)}$ and $w^{(3)}$ in each of the following situations

- $\epsilon = 0.5$. Converges gradually.
- $\epsilon = 1$. Converges in one step.
- $\epsilon = 2$. Oscillates.
- $\epsilon = 3$. Explodes!

In which case(s) do you think, the model will converge if we keep updating w for a long time ?

Question 2 : Momentum

Suppose we have the following loss function over a real number w

$$C(w) = \frac{1}{2}\frac{w^2}{100}.$$

We would like to minimize this function using gradient descent with momentum. so we start at an initial value $w^{(0)} = 1, v^{(0)} = 0$ and update the weight at time step t as

$$\begin{aligned} v^{(t+1)} &= \mu v^{(t)} - \epsilon \left.\frac{\partial C}{\partial w}\right|_t \\ w^{(t+1)} &= w^{(t)} + v^{(t+1)} \end{aligned}$$

Compute the weights after the first three steps : $w^{(1)}$, $w^{(2)}$ and $w^{(3)}$ in each of the following situations

- $\epsilon = 1$, $\mu = 0$.
- $\epsilon = 1$, $\mu = 1$.

Which case converges faster?

Question 2 : Momentum

Solution

$$
\begin{aligned}
\left.\frac{\partial C}{\partial w}\right|_t &= \frac{w^{(t)}}{100} \\
v^{(t+1)} &= \mu v^{(t)} - \epsilon \left.\frac{\partial C}{\partial w}\right|_t \\
w^{(t+1)} &= w^{(t)} + v^{(t+1)}
\end{aligned}
$$

- $w^{(0)} = 1,\ \epsilon = 1,\ \mu = 0.$

  No momentum. Therefore $v^{(t+1)} = -\epsilon \left.\frac{\partial C}{\partial w}\right|_t$.

  $w^{(1)} = w^{(0)} - \epsilon \frac{w^{(0)}}{100} = 1 - 1.\frac{1}{100} = 0.99$

  $w^{(2)} = w^{(1)} - \epsilon \frac{w^{(1)}}{100} \approx 0.99 - 1.\frac{0.99}{100} \approx 0.98$

  $w^{(3)} = w^{(2)} - \epsilon \frac{w^{(2)}}{100} \approx 0.98 - 1.\frac{0.98}{100} \approx 0.97$

- $w^{(0)} = 1,\ \epsilon = 1,\ \mu = 1.$

  $v^{(1)} = \mu v^{(0)} - \epsilon \frac{w^{(0)}}{100} = 0 - 1.\frac{1}{100} = -0.01,\ w^{(1)} = w^{(0)} + v^{(0)} = 1 - 0.01 = 0.99$

  $v^{(2)} = \mu v^{(1)} - \epsilon \frac{w^{(1)}}{100} = -0.01 - 1.\frac{0.99}{100} \approx -0.02,\ w^{(2)} = w^{(1)} + v^{(1)} \approx 0.99 - 0.02 = 0.97$

  $v^{(3)} = \mu v^{(2)} - \epsilon \frac{w^{(2)}}{100} = -0.02 - 1.\frac{0.97}{100} \approx -0.03,\ w^{(3)} = w^{(2)} + v^{(2)} \approx 0.97 - 0.03 = 0.94$

w goes faster towards 0 with momentum.

We saw how momentum works in simple 1-D error functions.

In reality, we'd never use a momentum of 1.

- The weights will oscillate, rather than converge.
- The updates may even blow up.
- However, typical values are very close to 1, e.g. 0.9 to 0.999.

Stochastic Gradient Descent

We want a good estimate of the gradient $\frac{\partial C}{\partial \mathbf{w}}$.

Stochastic Gradient Descent

We want a good estimate of the gradient $\frac{\partial C}{\partial \mathbf{w}}$.
Each data point gives us an estimate of this gradient.
We take the sum (or average) of these gradient and use that to make an update.

Stochastic Gradient Descent

We want a good estimate of the gradient $\frac{\partial C}{\partial \mathbf{w}}$.
Each data point gives us an estimate of this gradient.
We take the sum (or average) of these gradient and use that to make an update.
Full-batch Gradient Descent - Take the sum of all data points.
Mini-batch (Stochastic) Gradient Descent - Take the sum of a few randomly chosen data points.

Stochastic Gradient Descent

We want a good estimate of the gradient $\frac{\partial C}{\partial \mathbf{w}}$.
Each data point gives us an estimate of this gradient.
We take the sum (or average) of these gradient and use that to make an update.
Full-batch Gradient Descent - Take the sum of all data points.
Mini-batch (Stochastic) Gradient Descent - Take the sum of a few randomly chosen data points.
When would one be preferable, if we have -

- A dataset of 10 million examples for a binary classification problem ?
- A dataset of 100 examples for a binary classification problem ?

Measuring generalization

Suppose we use all these tricks and train a huge model with millions of parameters to do very well on the training data.
Are we done?

- We need to be careful!
- We want the model to generalize : work well on data that has not been trained on.

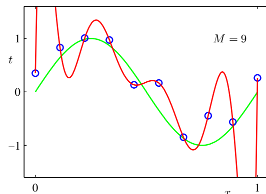
Measuring generalization

Suppose we use all these tricks and train a huge model with millions of parameters to do very well on the training data.
Are we done?

- We need to be careful!
- We want the model to generalize : work well on data that has not been trained on.
- We can always do well on training data, just by memorizing it (putting it in a table).
- We need to make sure we are not effectively training a fancy lookup table!

Measuring generalization

Overfitting : The model has memorized the training data.



Measuring generalization

It's hard to eliminate overfitting, but we at least want to know if it's occurring.

- Hold out part of your data to use as the test set. The algorithm doesn't get to see this during training. It's used to measure performance at the very end.
- Suppose we need to tune hyperparameters (e.g. learning rate, or number of hidden units; also called metaparameters). Should we choose the ones which perform best on the training data? The test data?

Measuring generalization

It's hard to eliminate overfitting, but we at least want to know if it's occurring.

- Hold out part of your data to use as the test set. The algorithm doesn't get to see this during training. It's used to measure performance at the very end.
- Suppose we need to tune hyperparameters (e.g. learning rate, or number of hidden units; also called metaparameters). Should we choose the ones which perform best on the training data? The test data?
  - you should hold out yet more of the data as the validation set, sometimes called the development set.

Measuring generalization

True or false:

- The performance on the test set is a good measure of how your learned model will perform in practice.
- Ideally, the model should be simple enough that the test set performance is not much worse than the training set performance.

Hint: these are partly true and partly false.