CSE 152: Computer Vision
Hao Su

Lecture 8: Network Optimization

Credit: Manmohan Chandraker
Gradient descent

\[ C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \]

- parameters to compute
- \# of input samples

\[ \Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \]

Small changes in parameters to leads to small changes in output

\[ \nabla C \equiv \begin{pmatrix} \frac{\partial C}{\partial v_1} \\ \frac{\partial C}{\partial v_2} \end{pmatrix}^T \]

Gradient vector!

\[ \Delta v = -\eta \nabla C \]

Change the parameter using learning rate (positive) and gradient vector!

\[ v \rightarrow v' = v - \eta \nabla C \]

Update rule!
Stochastic gradient descent

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

Update rules for each parameter:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

Cost function is a sum over all the training samples:

$$C(w, b) = \sum_x C_x(w, b), \text{ where } C_x(w, b) = \frac{1}{2} \|y(x) - a\|^2$$

Gradient from entire training set:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

Usually, $n$ is very large.
Stochastic gradient descent

Gradient from entire training set:

\[ \nabla C = \frac{1}{n} \sum_x \nabla C_x \]

- For large training data, gradient computation takes a long time
  - Leads to “slow learning”
- Instead, consider a mini-batch with \( m \) samples
- If sample size is large enough, properties approximate the dataset

\[ \sum_{j=1}^{m} \frac{\nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C. \]
Stochastic gradient descent

What if the loss function has a local minima or saddle point?

Zero gradient, gradient descent gets stuck
Stochastic gradient descent

What if loss changes quickly in one direction and slowly in another? What does gradient descent do? Very slow progress along shallow dimension, jitter along steep direction

Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large
Stochastic gradient descent

Our gradients come from minibatches so they can be noisy!

\[
L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)
\]

\[
\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)
\]
Stochastic gradient descent

Momentum update:

Velocity

Gradient

actual step

SGD

\[ x_{t+1} = x_t - \alpha \nabla f(x_t) \]

SGD+Momentum

\[ v_{t+1} = \rho v_t + \nabla f(x_t) \]

\[ x_{t+1} = x_t - \alpha v_{t+1} \]

Build up velocity as a running mean of gradients.
Layer to layer relationship

- $b^l_j$ is the bias term in the jth neuron in the lth layer.
- $a^l_j$ is the activation in the jth neuron in the lth layer.
- $z^l_j$ is the weighted input to the jth neuron in the lth layer.

\[
a^l_j = \sigma(z^l_j) \\
z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j \\
a^l_j = \sigma \left( \sum_k w^l_{jk} a^{l-1}_k + b^l_j \right)
\]
Cost and gradient computation

The goal of the backpropagation algorithm is to compute the gradients $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ of the cost function $C$ with respect to each and every weight and bias parameter. Note that backpropagation is only used to compute the gradients.

$$C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2$$

Stochastic gradient descent is the training algorithm.
Chain rule of differentiation

In order to differentiate a function \( z = f(g(x)) \) w.r.t \( x \), we can do the following:

Let \( y = g(x) \), \( z = f(y) \), \( \frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx} \)

Let \( x \in \mathbb{R}^m \), \( y \in \mathbb{R}^n \), \( g \) maps from \( \mathbb{R}^m \) to \( \mathbb{R}^n \), and \( f \) maps from \( \mathbb{R}^n \) to \( \mathbb{R} \). If \( y = g(x) \) and \( z = f(y) \), then

\[
\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}
\]

This is all you need to know to get the gradients in a neural network!
Backpropagation

In order to differentiate a function $z = f(g(x))$ w.r.t $x$, we can do the following:

Let $y = g(x)$, $z = f(y)$, $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$

Let $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and $f$ maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

This is all you need to know to get the gradients in a neural network!
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

**Backpropagation: a simple example**

\[ f(x, y, z) = (x + y)z \]

**e.g.** \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

**Want:** \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}
\]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}
\]
Backpropagation example
Backpropagation example
Backpropagation example

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

"local gradient"

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} \]

\[ \frac{\partial L}{\partial z} \]

gradients
Backpropagation example

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

“local gradient”

Gradients
Over-fitting

- Instead of 60000 training images, we use only 1000 training images and check the performance on the test data.
More data prevents over-fitting

But not always feasible to have more data that is relevant.
Regularization reduces over-fitting

- If not enough data, can instead limit model complexity.
- Regularization places constraints on the model, so reduces its complexity.
L2 regularization

Let $C_0$ be original cost. Define

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- The first term is just the usual $C_0 \equiv \frac{1}{2n} \sum \|y(x) - a\|^2$
- Here $\lambda$ is the regularization parameter and $n$ is the size of our training set.

Partial derivatives:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} W$$

Update rule:

$$W \rightarrow W - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} W$$
Regularization reduces over-fitting
Dropout as a regularization

• Modify the network itself
  – Randomly delete half the hidden neurons in the network
  – Repeat several times to learn weights and biases
  – At runtime, twice as many neurons, so halve the weights outgoing from a neuron
Dropout as a regularization

- Averaging or voting scheme to decide output
  - Forces neurons to learn independent of others
  - Same training data, but random initializations
  - Each network over-fits in a different way
  - Robustness: average output not sensitive to particular mode
Data augmentation as regularization
Data augmentation as regularization

Horizontal flips
Data augmentation as regularization

Random crops and scales

1. Pick random $L$ in range $[256, 480]$
2. Resize training image, short side = $L$
3. Sample random 224 x 224 patch
Data augmentation as regularization

Color jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all \([R, G, B]\) pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image
Data augmentation as regularization

Color jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

Can do a lot more: rotation, shear, non-rigid, motion blur, lens distortions, ....