Lecture 10: Backpropagation

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer

This lecture uses material from:
• A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey
• Stanford CS231n
• Deep Learning by I. Goodfellow
Important components of a CNN

### CNN Architecture
- CONV size, stride, pad, depth
- ReLU & other nonlinearities
- POOL methods
- # of layers, dimensions per layer
- Fully connected layers

### Loss function & optimization
- Type of loss function
- Regularization
- Gradient descent method
- SGD batch and step size

Other specifics: Pre-processing, initialization, dropout, batch normalization, augmentation
Important components of a CNN

**CNN Architecture**

- CONV size, stride, pad, depth
- ReLU & other nonlinearities
- POOL methods
- # of layers, dimensions per layer
- Fully connected layers

**Loss function & optimization**

- Type of loss function
- Regularization
- Gradient descent method
- SGD batch and step size

**Other specifics:** Pre-processing, initialization, dropout, batch normalization, augmentation

How does the optimizer actually work???
Our very basic convolutional neural network

\[ L_{\text{in}}(y, f(W,x)) = \text{cross_entropy}(y, f(W,x)) \]

Training error

\[ \frac{dL}{dW} = \frac{y^*}{f(W,x)} \]

\[ L(w) = \frac{1}{N} \sum_{i=1}^{N} \ln(1 + e^{y_i \max(0, W_3 \max(0, W_2 \max(0, W_1 x_i)))}) \]

\( W_1 \) and \( W_2 \) are banded Toeplitz matrices, \( W_3 \) is a full matrix

3-layer network
Our very basic convolutional neural network

Training error

\[ L_{\text{in}}(y, f(W,x)) = \text{cross_entropy}(y, f(W,x)) \]

Output

\[ y^* = f(W,x) \]

Model

Ex. \([x_1, y_1]\) Ex. \([x_K, y_K]\)

Training Data

Forward pass: from \(x_i\) and current \(W\)'s, find \(L_{\text{in}}\)

\[ L_{\text{in}} = \max w_3 \text{ max } w_2 \text{ max } w_1 \]

\[ y^* \]
Our very basic convolutional neural network

\[ L_{\text{in}}(y, f(W,x)) = \text{cross_entropy}(y, f(W,x)) \]

Training error

Output \( y^* \)

Model \( y^* = f(W,x) \)

Ex. \([x_1, y_1]\) Ex. \([x_K, y_K]\)

Training Data

\[ L_{\text{in}} = \max \]

Given a new \( L_{\text{in}} \), want to update \( W \)'s to make \( L_{\text{in}} \) smaller!

Backwards pass: from new \( L_{\text{in}} \), need to update \( W \)'s! How?
Review: how can we determine the optimal $W$?

- Here, let’s assume we’ll use the steepest descent algorithm to “go down the hill”: 
Review: how can we determine the optimal $W$?

- Here, let’s assume we’ll use the steepest descent algorithm to “go down the hill”:

  Input: labeled training examples $[x_i, y_i]$ for $i=1$ to $N$, initial guess of $W$'s

  while loss function is still decreasing:
    Compute loss function $L(W, x_i, y_i)$
    Update $W$ to make $L$ smaller:
    $\frac{dL}{dW} = \text{evaluate\_gradient}(W, x_i, y_i, L)$
    $W = W - \text{step\_size} * \frac{dL}{dW}$

```python
while previous_step_size > precision and iters < max_iters
    prev_W = cur_W
    cur_W -= gamma * differential_dL(CNN_model, prev_W)
    previous_step_size = abs(cur_W - prev_W)
    iters += 1
```
Review: how can we determine the optimal $W$?

- Here, let’s assume we’ll use the steepest descent algorithm to “go down the hill”:

Input: labeled training examples $[x_i, y_i]$ for $i=1$ to $N$, initial guess of $W$’s

while loss function is still decreasing:
  Compute loss function $L(W,x_i,y_i)$
  Update $W$ to make $L$ smaller:
  \[
  \frac{dL}{dW} = \text{evaluate\_gradient}(W, x_i, y_i, L) \\
  W = W - \text{step\_size} \times \frac{dL}{dW}
  \]

Options to evaluate $dL/dW$:
1. Numerical gradient
2. Analytic gradient
3. Automatic differentiation

```python
while previous_step_size > precision and iters < max_iters
  prev_W = cur_W
  cur_W = gamma * differential_dL(CNN_model, prev_W)
  previous_step_size = abs(cur_W - prev_W)
  iters += 1
```
Review: how can we determine the optimal $W$?

- Here, let’s assume we’ll use the steepest descent algorithm to “go down the hill”:

  Input: labeled training examples $[x_i, y_i]$ for $i=1$ to $N$, initial guess of $W$’s

  while loss function is still decreasing:
    Compute loss function $L(W, x_i, y_i)$
    Update $W$ to make $L$ smaller:
      $$dL/dW = \text{evaluate\_gradient}(W, x_i, y_i, L)$$
      $$W = W - \text{step\_size} \times dL/dW$$

Options to evaluate $dL/dW$:
1. Numerical gradient
2. Analytic gradient
3. Automatic differentiation

*Note: Other gradient descent methods require the same fundamental calculation. So how gradient is computed is a different problem than how it is used
1. Numerical gradient, a simple example

With a matrix, compute this for each entry:

\[
\frac{dL(W_i)}{dW_i} = \lim_{h \to 0} \frac{L(W_i + h) - L(W_i)}{h}
\]

Example:

\[W = [1,2;3,4] \quad W_1+h = [1.001,2;3,4]\]

\[L(W, x, y) = 12.79 \quad L(W_1+h, x, y) = 12.8\]

\[dL(W_1)/dW_1 = 12.8-12.79/.001 \quad dL(W_1)/dW_1 = 10\]

**Pros:** Simple! Easy to code up!

**Cons:** Slow...really slow. And approximate
2. Analytic gradient, a simple example

Linear classification boundary

\[ y = \begin{cases} -1 & \text{for } y = -1 \\ +1 & \text{for } y = +1 \end{cases} \]

\[ \mathbf{x}_1 \quad \mathbf{x}_2 \]

Pros: Fast and exact

Cons: Error prone, especially with deep networks

\[ L = \frac{1}{N} \sum_{i=1}^{N} (w^T \mathbf{x}_i - y_i)^2 \]

\[ \nabla L(w) = \frac{2}{N} X^T (Xw - y) \]

Evaluate and use to update \( W \)
3. Automatic differentiation – what we’ll use without knowing it

Resources:

- Stanford CS231n, Lecture 4 notes and resources
  - http://cs231n.stanford.edu/syllabus

- I. Goodfellow et al., Deep Learning Chapter 6 Section 5
  - https://www.deeplearningbook.org/contents/mlp.html

- A. Baydin et al., “Automatic differentiation in machine learning: a survey”
3. Automatic differentiation – what is it?

- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- Repeat process in lock-step with evaluation of final function

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey
3. Automatic differentiation – what is it?

- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- Repeat process in lock-step with evaluation of final function

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey
3. Automatic differentiation – what is it?

- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- **Repeat process in lock-step with evaluation of final function**
3. Automatic differentiation – what is it?

- Not solely numerical or analytic
- Use insights into formation of final function
- Split into elementary operations
- Perform analytic (symbolic) differentiation at elementary operation level
- Keep intermediate numeric results
- Repeat process in lock-step with evaluation of final function

A. Baydin et al., Automatic Differentiation in Machine Learning: a Survey
Automatic differentiation on computational graphs

\[ f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]
Automatic differentiation on computational graphs

To both determine $f$ and find $\frac{df}{dx_i}$:

- Create graph of local operations
- Compute analytic (symbolic) gradient at each node (unit) in graph
- Use inter-relationships to establish final desired gradient, $\frac{df}{dx_1}$
  - Forward differentiation
  - Backwards differentiation = Backpropagation
Automatic differentiation on computational graphs

\[ f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]

Forward Primal Trace

\[
\begin{align*}
  v_{-1} &= x_1 & = 2 \\
  v_0 &= x_2 & = 5 \\
  v_1 &= \ln v_{-1} & = \ln 2 \\
  v_2 &= v_{-1} \times v_0 & = 2 \times 5 \\
  v_3 &= \sin v_0 & = \sin 5 \\
  v_4 &= v_1 + v_2 & = 0.693 + 10 \\
  v_5 &= v_4 - v_3 & = 10.693 + 0.959 \\
  y &= v_5 & = 11.652
\end{align*}
\]
Forward automatic differentiation

\[
f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)
\]

**Forward Primal Trace**

| \(v_{-1} = x_1\) | 2 |
| \(v_0 = x_2\) | 5 |
| \(v_1 = \ln(v_{-1})\) | \(\ln 2\) |
| \(v_2 = v_{-1} \times v_0\) | \(2 \times 5\) |
| \(v_3 = \sin(v_0)\) | \(\sin 5\) |
| \(v_4 = v_1 + v_2\) | 0.693 + 10 |
| \(v_5 = v_4 - v_3\) | 10.693 + 0.959 |

\(y = v_5\) | 11.652 |

**Forward Tangent (Derivative) Trace**

| \(\dot{v}_{-1} = \dot{x}_1\) | 1 |
| \(\dot{v}_0 = \dot{x}_2\) | 0 |
| \(\dot{v}_1 = \dot{v}_{-1}/v_{-1}\) | \(1/2\) |
| \(\dot{v}_2 = \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}\) | \(1 \times 5 + 0 \times 2\) |
| \(\dot{v}_3 = \dot{v}_0 \times \cos(v_0)\) | \(0 \times \cos 5\) |
| \(\dot{v}_4 = \dot{v}_1 + \dot{v}_2\) | 0.5 + 5 |
| \(\dot{v}_5 = \dot{v}_4 - \dot{v}_3\) | 5.5 - 0 |

\(\dot{y} = \dot{v}_5\) | 5.5 |
Forward automatic differentiation

\begin{align*}
    f(x_1, x_2) &= \ln(x_1) + x_1 x_2 - \sin(x_2) \\
    \text{Compute local derivative for all inputs and accumulate with chain rule}
\end{align*}

**Forward Primal Trace**

| \(v_{-1}\) | 2 |
| \(v_0\) | 5 |
| \(v_1\) | \(\ln v_{-1} = \ln 2\) |
| \(v_2\) | \(v_{-1} \times v_0 = 2 \times 5\) |
| \(v_3\) | \(\sin v_0 = \sin 5\) |
| \(v_4\) | \(v_1 + v_2 = 0.693 + 10\) |
| \(v_5\) | \(v_4 - v_3 = 10.693 + 0.959\) |
| \(y\) | 11.652 |

**Forward Tangent (Derivative) Trace**

| \(\dot{v}_{-1}\) | \(\dot{x}_1\) |
| \(\dot{v}_0\) | \(\dot{x}_2\) |
| \(\dot{v}_1\) | \(\dot{v}_{-1}/v_{-1} = 1/2\) |
| \(\dot{v}_2\) | \(\dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1} = 1 \times 5 + 0 \times 2\) |
| \(\dot{v}_3\) | \(\dot{v}_0 \times \cos v_0 = 0 \times \cos 5\) |
| \(\dot{v}_4\) | \(\dot{v}_1 + \dot{v}_2 = 0.5 + 5\) |
| \(\dot{v}_5\) | \(\dot{v}_4 - \dot{v}_3 = 5.5 - 0\) |
| \(\dot{y}\) | 5.5 |
Forward automatic differentiation

\[ f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]

Forward Primal Trace

\[
\begin{align*}
v_{-1} &= x_1 & \text{value} &= 2 \\
v_0 &= x_2 & \text{value} &= 5 \\
v_1 &= \ln v_{-1} & \text{value} &= \ln 2 \\
v_2 &= v_{-1} \times v_0 & \text{value} &= 2 \times 5 \\
v_3 &= \sin v_0 & \text{value} &= \sin 5 \\
v_4 &= v_1 + v_2 & \text{value} &= 0.693 + 10 \\
v_5 &= v_4 - v_3 & \text{value} &= 10.693 + 0.959 \\
\end{align*}
\]

\[ y = v_5 = 11.652 \]

Forward Tangent (Derivative) Trace

\[
\begin{align*}
\dot{v}_{-1} &= \dot{x}_1 & \text{value} &= 1 \\
\dot{v}_0 &= \dot{x}_2 & \text{value} &= 0 \\
\dot{v}_1 &= \dot{v}_{-1} / v_{-1} & \text{value} &= 1/2 \\
\dot{v}_2 &= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1} & \text{value} &= 1 \times 5 + 0 \times 2 \\
\dot{v}_3 &= \dot{v}_0 \times \cos v_0 & \text{value} &= 0 \times \cos 5 \\
\dot{v}_4 &= \dot{v}_1 + \dot{v}_2 & \text{value} &= 0.5 + 5 \\
\dot{v}_5 &= \dot{v}_4 - \dot{v}_3 & \text{value} &= 5.5 - 0 \\
\end{align*}
\]

\[ \dot{y} = \dot{v}_5 = 5.5 \]

Leads to final desired \( df/dx_1 \)
Problem:
- For N inputs, need to compute this N times, setting x_i to 1 each time...
Machine Learning and Imaging – Roarke Horstmeyer (2020)

Forward automatic differentiation

\[ f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]

**Problem:**
- For N inputs, need to compute this N times, setting \( x_i \) to 1 each time…

**Solution:**
Work backwards from end to start with backpropagation

<table>
<thead>
<tr>
<th>Forward Primal Trace</th>
<th>Forward Tangent (Derivative) Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_{-1} = x_1 ) = 2</td>
<td>( \dot{v}_{-1} = \dot{x}_1 ) = 1</td>
</tr>
<tr>
<td>( v_0 = x_2 ) = 5</td>
<td>( \dot{v}_0 = \dot{x}_2 ) = 0</td>
</tr>
<tr>
<td>( v_1 = \ln v_{-1} ) = \ln 2</td>
<td>( \dot{v}<em>1 = \dot{v}</em>{-1}/v_{-1} ) = 1/2</td>
</tr>
<tr>
<td>( v_2 = v_{-1} \times v_0 ) = 2 \times 5</td>
<td>( \dot{v}<em>2 = \dot{v}</em>{-1} \times v_0 + \dot{v}<em>0 \times v</em>{-1} ) = 1 \times 5 + 0 \times 2</td>
</tr>
<tr>
<td>( v_3 = \sin v_0 ) = \sin 5</td>
<td>( \dot{v}_3 = \dot{v}_0 \times \cos v_0 ) = 0 \times \cos 5</td>
</tr>
<tr>
<td>( v_4 = v_1 + v_2 ) = 0.693 + 10</td>
<td>( \dot{v}_4 = \dot{v}_1 + \dot{v}_2 ) = 0.5 + 5</td>
</tr>
<tr>
<td>( v_5 = v_4 - v_3 ) = 10.693 + 0.959</td>
<td>( \dot{v}_5 = \dot{v}_4 - \dot{v}_3 ) = 5.5 - 0</td>
</tr>
<tr>
<td>( y = v_5 ) = 11.652</td>
<td>( \dot{y} = \dot{v}_5 ) = 5.5</td>
</tr>
</tbody>
</table>
Backpropagation explanation from Stanford CS231N Slides


Other useful info here: http://cs231n.github.io/optimization-2/
Backpropagation explanation from Stanford CS231N Slides

Treat intermediate nodes like a dummy variable $z$, for $L(w_1)$

Key Idea: $\frac{dL}{dw_1} = (\frac{dL}{dz})(\frac{dz}{dw_1})$
Modularized implementation: forward / backward API

```python
class MultiplyGate(object):
    def forward(x, y):
        z = x * y
        self.x = x  # must keep these around!
        self.y = y
        return z

    def backward(dz):
        dx = self.y * dz  # [dz/dx * dL/dz]
        dy = self.x * dz  # [dz/dy * dL/dz]
        return [dx, dy]
```

(x, y, z are scalars)

Local gradient  Upstream gradient variable
How Tensorflow actually works: create a whole extra new graph

Final desired derivative
Last thing – matrix and vector derivatives

Here’s a review:

\[ u = Wv \]

\[ \frac{du}{dv} = \]
Last thing – matrix and vector derivatives

Here’s a review:

\[ u = Wv \]

\[ \frac{du}{dv} = \]

\[ u_3 = W_{3,1}v_1 + W_{3,2}v_2 + \ldots + W_{3,M}v_M \]

\[ \frac{\partial u_3}{\partial v_2} = \frac{\partial}{\partial v_2}(W_{3,1}v_1 + W_{3,2}v_2 + \ldots + W_{3,M}v_M) = \frac{\partial}{\partial v_2}W_{3,2}v_2 = W_{3,2} \]

\[ \frac{\partial u_i}{\partial v_j} = W_{i,j} \]

\[ \frac{du}{dv} = W \]

• When confused, write out one entry, solve derivative and generalize

• Use dimensionality to help (if x has N elements, and y has M, then dy/dx must be NxM

• Take advantage of *The Matrix Cookbook*:
  • https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf
Let’s go through an example:

\[ L = \| W_2 \, \text{ReLU}(W_1 \, X) \|_2^2 \]

\[ \frac{dL}{dW_1} = ? \quad \frac{dL}{dW_2} = ? \]

(2-layer network with MSE where we neglect labels \( y \) for now)
Let's go through an example:

\[ L = \| W_2 \text{ReLU}(W_1 X) \|_2^2 \]

(2-layer network with MSE where we neglect labels \( y \) for now)

\[ \frac{dL}{dW_1} = ? \quad \frac{dL}{dW_2} = ? \]

\[
\begin{align*}
  z_1 &= XW_1 \\
  h_1 &= \text{ReLU}(z_1) \\
  \hat{y} &= h_1 W_2 \\
  L &= \|\hat{y}\|_2^2
\end{align*}
\]

Forward pass: solve for \( z_1, h_1, y \) and \( L \)
Let's go through an example:

\[ L = \| W_2 \text{ReLU}(W_1 X) \|_2^2 \]

\[ \frac{dL}{dW_1} = ? \quad \frac{dL}{dW_2} = ? \]

Gradients for scalar \( L \) will have same shape as denominator

Backpropagation from \( L \) to \( W_2 \)
Let's go through an example:

\[ L = \| W_2 \text{ReLU}(W_1 X) \|_2^2 \]

\[ \frac{dL}{dW_1} = ? \quad \frac{dL}{dW_2} = ? \]

(2-layer network with MSE where we neglect labels \( y \) for now)

Gradients for scalar \( L \) will have same shape as denominator

\[
\frac{\partial L}{\partial W_2} = \frac{\partial \hat{y}}{\partial W_2} \frac{\partial L}{\partial \hat{y}}
\]
Let's go through an example:

$$L = \| W_2 \text{ReLU}(W_1 X) \|_2^2$$

(2-layer network with MSE where we neglect labels $y$ for now)

$$\frac{dL}{dW_1} = ? \quad \frac{dL}{dW_2} = ?$$

$$z_1 = X W_1$$

$$h_1 = \text{ReLU}(z_1)$$

$$\hat{y} = h_1 W_2$$

$$L = \|\hat{y}\|_2^2$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial \hat{y}}{\partial W_2} \frac{\partial L}{\partial \hat{y}} = 2h_1^T \hat{y}$$

$$\frac{\partial \hat{y}}{\partial W_2} = h_1^T$$

$$\frac{\partial L}{\partial \hat{y}} = 2\hat{y}$$
Let's go through an example:

\[ L = \| W_2 \text{ReLU}(W_1 X) \|_2^2 \]

(2-layer network with MSE where we neglect labels \( y \) for now)

\[
\begin{align*}
    z_1 &= X W_1 \\
    h_1 &= \text{ReLU}(z_1) \\
    \hat{y} &= h_1 W_2 \\
    L &= \| \hat{y} \|_2^2 \\

    \frac{\partial L}{\partial h_1} &= \frac{\partial L}{\partial \hat{y}} W_2^T \\
    \frac{\partial h_1}{\partial z_1} &= \frac{\partial L}{\partial h_1} \circ I[h_1 > 0] \\

    \frac{dL}{dW_1} &= \frac{dh_1}{\partial W_1} = x^\top \frac{\partial h_1}{\partial z_1}
\end{align*}
\]

Rely on dimensional analysis to select order of operations
Let’s go through an example:

\[ L = \| W_2 \text{ReLU}(W_1 X) \|_2^2 \]

\[ z_1 = XW_1 \]
\[ h_1 = \text{ReLU}(z_1) \]
\[ \hat{y} = h_1W_2 \]
\[ L = \| \hat{y} \|_2^2 \]

\[ \frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} W_2^\top \]
\[ \frac{\partial h_1}{\partial z_1} = \frac{\partial L}{\partial h_1} \circ I[h_1 > 0] \]

\[ \frac{dL}{dW_1} = \begin{bmatrix} dh_1 \\ x^\top \frac{\partial h_1}{\partial z_1} \end{bmatrix} \]

```python
import numpy as np

# forward prop
z_1 = np.dot(X, W_1)
h_1 = np.maximum(z_1, 0)
y_hat = np.dot(h_1, W_2)
L = np.sum(y_hat**2)

# backward prop
dy_hat = 2.0*y_hat
dW2 = h_1.T.dot(dy_hat)
dh1 = dy_hat.dot(W_2.T)
dz1 = dh1.copy()
dz1[z1 < 0] = 0
dW1 = X.T.dot(dz1)
```
Summary

- Tensorflow: define variables, series of operations & a cost function
- When you hit enter, Tensorflow effectively forms two graphs
  - Forward graph to evaluate function at each node
  - **Backprop:** Backwards graph that includes *local* derivatives of each operation as symbolic functions, as well as connections
- Tensorflow will go through the forward graph & save numerical results, then the backwards graph, to update weights via local operations, to minimize cost function
- Uses more impressive operations to do this with vectors and matrices efficiently