Lecture 5:
Backpropagation
Announcements

• Assignment 1 due Thursday, 11:59
  • You can use up to 3 late days (making it due Sunday at midnight)

• Default final project will be released February 1st
  • To help you choose which project option you want to do

• Final project proposal due February 8th
  • See website for details and inspiration
Overview Today:

- From one-layer to multi layer neural networks!
- Fully vectorized gradient computation
- The backpropagation algorithm
- (Time permitting) Class project tips
Remember: One-layer Neural Net

\[ s = u^T h \]

\[ h = f(Wx + b) \]

\[ x \quad (\text{input}) \]

\[ x = [ \text{museums} \quad x_{\text{in}} \quad x_{\text{Paris}} \quad x_{\text{are}} \quad x_{\text{amazing}} ] \]
Two-layer Neural Net

\[ s = u^T h_2 \]

\[ h_2 = f(W_2 h_1 + b_2) \]

\[ h_1 = f(W_1 x + b_1) \]

\[ x \text{ (input)} \]

\[ x = [x_{\text{museums}} \ x_{\text{in}} \ x_{\text{Paris}} \ x_{\text{are}} \ x_{\text{amazing}}] \]
Repeat as Needed!

\[ s = u^T h_3 \]

\[ h_3 = f(W_3 h_2 + b_3) \]

\[ h_2 = f(W_2 h_1 + b_2) \]

\[ h_1 = f(W_1 x + b_1) \]

\[ x \quad \text{(input)} \]
Why Have Multiple Layers?

- Hierarchical representations -> neural net can represent complicated features
- Better results!

<table>
<thead>
<tr>
<th># Layers</th>
<th>Machine Translation Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>23.7</td>
</tr>
<tr>
<td>4</td>
<td>25.3</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>

From Transformer Network (will cover in a later lecture)
Remember: Stochastic Gradient Descent

- Update equation:

\[
\theta_{new} = \theta_{old} - \alpha \nabla_{\theta} J(\theta)
\]

\(\alpha = \text{step size or learning rate}\)
Remember: Stochastic Gradient Descent

• Update equation:

\[ \theta_{new} = \theta_{old} - \alpha \nabla \theta J(\theta) \]

\( \alpha = \text{step size or learning rate} \)

• This Lecture: How do we compute \( \nabla \theta J(\theta) \)?
  • By hand
  • Algorithmically (the backpropagation algorithm)
Why learn all these details about gradients?

• Modern deep learning frameworks compute gradients for you

• But why take a class on compilers or systems when they are implemented for you?
  • Understanding what is going on under the hood is useful!

• Backpropagation doesn’t always work perfectly.
  • Understanding why is crucial for debugging and improving models
  • Example in future lecture: exploding and vanishing gradients
Quickly Computing Gradients by Hand

• Review of multivariable derivatives

• Fully vectorized gradients
  • Much faster and more useful than non-vectorized gradients
  • But doing a non-vectorized gradient can be good practice, see slides in last week’s lecture for an example
  • Lecture notes cover this material in more detail
Gradients

• Given a function with 1 output and $n$ inputs

$$f(x) = f(x_1, x_2, \ldots, x_n)$$

• Its gradient is a vector of partial derivatives

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}$$
Jacobian Matrix: Generalization of the Gradient

- Given a function with \( m \) outputs and \( n \) inputs
  \[
  f(x) = [f_1(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n)]
  \]

- Its Jacobian is an \( m \times n \) matrix of partial derivatives

\[
\frac{\partial f}{\partial x} = \begin{bmatrix}
  \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
  \vdots & \ddots & \vdots \\
  \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]

\[
(\frac{\partial f}{\partial x})_{ij} = \frac{\partial f_i}{\partial x_j}
\]
Chain Rule For Jacobians

- For one-variable functions: multiply derivatives
  \[ z = 3y \]
  \[ y = x^2 \]
  \[ \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = (3)(2x) = 6x \]

- For multiple variables: multiply Jacobians
  \[ h = f(z) \]
  \[ z = Wx + b \]
  \[ \frac{\partial h}{\partial x} = \frac{\partial h}{\partial z} \frac{\partial z}{\partial x} = \ldots \]
Example Jacobian: Activation Function

\[ h = f(z), \text{ what is } \frac{\partial h}{\partial z} \？ \quad h, z \in \mathbb{R}^n \]

\[ h_i = f(z_i) \]
Example Jacobian: Activation Function

$h = f(z)$, what is $\frac{\partial h}{\partial z}$? 

$h_i = f(z_i)$

$h, z \in \mathbb{R}^n$

Function has $n$ outputs and $n$ inputs -> $n$ by $n$ Jacobian
Example Jacobian: Activation Function

\[ h = f(z), \text{ what is } \frac{\partial h}{\partial z}? \]
\[ h_i = f(z_i) \]

\[
\left( \frac{\partial h}{\partial z} \right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \]

\( h, z \in \mathbb{R}^n \)

definition of Jacobian
Example Jacobian: Activation Function

\[ h = f(z), \text{ what is } \frac{\partial h}{\partial z} ? \quad h, z \in \mathbb{R}^n \]

\[ h_i = f(z_i) \]

\[
\left( \frac{\partial h}{\partial z} \right)_{i,j} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \\
= \begin{cases} 
  f'(z_i) & \text{if } i = j \\
  0 & \text{if otherwise} 
\end{cases}
\]

definition of Jacobian

regular 1-variable derivative
Example Jacobian: Activation Function

\[ h = f(z), \text{ what is } \frac{\partial h}{\partial z}? \quad h, z \in \mathbb{R}^n \]

\[ h_i = f(z_i) \]

\[
\left( \frac{\partial h}{\partial z} \right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i)
\]

definition of Jacobian

\[
= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}
\]

regular 1-variable derivative

\[
\frac{\partial h}{\partial z} = \begin{pmatrix} f'(z_1) & 0 \\ \vdots & \ddots \\ 0 & \cdots & f'(z_n) \end{pmatrix} = \text{diag}(f'(z))
\]
Other Jacobians

\[ \frac{\partial}{\partial x} (Wx + b) = W \]
Other Jacobians

\[ \frac{\partial}{\partial x} (Wx + b) = W \]

\[ \frac{\partial}{\partial b} (Wx + b) = I \quad \text{(Identity matrix)} \]
Other Jacobians

\[ \frac{\partial}{\partial x} (Wx + b) = W \]

\[ \frac{\partial}{\partial b} (Wx + b) = I \quad \text{(Identity matrix)} \]

\[ \frac{\partial}{\partial u} (u^T h) = h^T \]
Other Jacobians

\[
\frac{\partial}{\partial x} (Wx + b) = W
\]

\[
\frac{\partial}{\partial b} (Wx + b) = I \quad \text{(Identity matrix)}
\]

\[
\frac{\partial}{\partial u} (u^T h) = h^T
\]

• Compute these at home for practice!
  • Check your answers with the lecture notes
Back to Neural Nets!

\[ s = u^T h \]

\[ h = f(Wx + b) \]

\[ x \quad (\text{input}) \]

\[ x = [x_{\text{museums}} \ x_{\text{in}} \ x_{\text{Paris}} \ x_{\text{are}} \ x_{\text{amazing}}] \]
Back to Neural Nets!

• Let’s find $\frac{\partial s}{\partial b}$

• In practice we care about the gradient of the loss, but we will compute the gradient of the score for simplicity

\[ s = u^T h \]

\[ h = f(Wx + b) \]

\[ x \quad (\text{input}) \]

\[ x = [x_{\text{museums}}, x_{\text{in}}, x_{\text{Paris}}, x_{\text{are}}, x_{\text{amazing}}] \]
1. Break up equations into simple pieces

\[ s = u^T h \]

\[ h = f(Wx + b) \]

\[ x \text{ (input)} \]

\[ s = u^T h \]

\[ h = f(z) \]

\[ z = Wx + b \]

\[ x \text{ (input)} \]
2. Apply the chain rule

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \quad (\text{input}) \]

\[ \frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b} \]
2. Apply the chain rule

\[
\begin{align*}
    s &= u^T h \\
    h &= f(z) \\
    z &= Wx + b \\
    x &= \text{(input)}
\end{align*}
\]

\[
\frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b}
\]
2. Apply the chain rule

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = W x + b \]
\[ x \text{ (input)} \]

\[ \frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b} \]
2. Apply the chain rule

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \ (input) \]

\[
\frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b}
\]
3. Write out the Jacobians

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \quad (\text{input}) \]

Useful Jacobians from previous slide

\[ \frac{\partial}{\partial h} (u^T h) = u^T \]
\[ \frac{\partial}{\partial z} (f(z)) = \text{diag}(f'(z)) \]
\[ \frac{\partial}{\partial b} (Wx + b) = I \]
3. Write out the Jacobians

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]

Useful Jacobians from previous slide

\[ \frac{\partial}{\partial h} (u^T h) = u^T \]
\[ \frac{\partial}{\partial z} (f(z)) = \text{diag}(f'(z)) \]
\[ \frac{\partial}{\partial b} (Wx + b) = I \]
3. Write out the Jacobians

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = W x + b \]
\[ x \text{ (input)} \]

Useful Jacobians from previous slide

\[ \frac{\partial}{\partial h}(u^T h) = u^T \]
\[ \frac{\partial}{\partial z}(f(z)) = \text{diag}(f'(z)) \]
\[ \frac{\partial}{\partial b}(W x + b) = I \]
3. Write out the Jacobians

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]

Useful Jacobians from previous slide
\[ \frac{\partial}{\partial h}(u^T h) = u^T \]
\[ \frac{\partial}{\partial z}(f(z)) = \text{diag}(f'(z)) \]
\[ \frac{\partial}{\partial b}(Wx + b) = I \]
3. Write out the Jacobians

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]

Useful Jacobians from previous slide

\[ \frac{\partial}{\partial h} (u^T h) = u^T \]
\[ \frac{\partial}{\partial z} (f(z)) = \text{diag}(f'(z)) \]
\[ \frac{\partial}{\partial b} (Wx + b) = I \]

\[ \frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b} \]

\[ = u^T \text{diag}(f'(z))I \]

\[ = u^T \circ f'(z) \]
Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial W}$
- Using the chain rule again:

$$\frac{\partial s}{\partial W} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial W}$$
Re-using Computation

• Suppose we now want to compute \( \frac{\partial s}{\partial W} \)
  • Using the chain rule again:

\[
\begin{align*}
\frac{\partial s}{\partial W} &= \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial W} \\
\frac{\partial s}{\partial b} &= \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b}
\end{align*}
\]

The same! Let’s avoid duplicated computation...
Re-using Computation

- Suppose we now want to compute \( \frac{\partial s}{\partial W} \)

  - Using the chain rule again:

\[
\begin{align*}
\frac{\partial s}{\partial W} &= \delta \frac{\partial z}{\partial W} \\
\frac{\partial s}{\partial b} &= \delta \frac{\partial z}{\partial b} \\
\delta &= \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} = u^T \circ f'(z)
\end{align*}
\]
Derivative with respect to Matrix

- What does $\frac{\partial s}{\partial W}$ look like? $W \in \mathbb{R}^{n \times m}$
- 1 output, $nm$ inputs: 1 by $nm$ Jacobian?
  - Inconvenient to do $\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$
Derivative with respect to Matrix

• What does $\frac{\partial s}{\partial W}$ look like? $W \in \mathbb{R}^{n \times m}$

• 1 output, $nm$ inputs: 1 by $nm$ Jacobian?
  • Inconvenient to do $\theta_{new} = \theta_{old} - \alpha \nabla_{\theta} J(\theta)$

• Instead follow convention: shape of the gradient is shape of parameters

  So $\frac{\partial s}{\partial W}$ is $n$ by $m$:

$$
\begin{bmatrix}
\frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\
\vdots & \ddots & \vdots \\
\frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}}
\end{bmatrix}
$$
Derivative with respect to Matrix

- Remember $\frac{\partial s}{\partial W} = \delta \frac{\partial z}{\partial W}$
- $\delta$ is going to be in our answer
- The other term should be $x$ because $z = Wx + b$

- It turns out $\frac{\partial s}{\partial W} = \delta^T x^T$
Why the Transposes?

\[
\frac{\partial s}{\partial W} = \delta^T x^T
\]

\[
[n \times m] [n \times 1][1 \times m]
\]

• Hacky answer: this makes the dimensions work out
  • Useful trick for checking your work!
• Full explanation in the lecture notes
Why the Transposes?

\[ \frac{\partial s}{\partial \mathbf{W}} = \delta^T \mathbf{x}^T = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} \begin{bmatrix} x_1 \\ \cdots \\ x_m \end{bmatrix} = \begin{bmatrix} \delta_1 x_1 & \cdots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \cdots & \delta_n x_m \end{bmatrix} \]

- Hacky answer: this makes the dimensions work out
  - Useful trick for checking your work!
- Full explanation in the lecture notes
What shape should derivatives be?

\[ \frac{\partial s}{\partial b} = u^T \circ f'(z) \] is a row vector

- But convention says our gradient should be a column vector because \( b \) is a column vector...

- Disagreement between Jacobian form (which makes the chain rule easy) and the shape convention (which makes implementing SGD easy)

- We expect answers to follow the shape convention

- But Jacobian form is useful for computing the answers
What shape should derivatives be?

- Two options:
  - 1. Use Jacobian form as much as possible, reshape to follow the convention at the end:
    - What we just did. But at the end transpose $\frac{\partial s}{\partial b}$ to make the derivative a column vector, resulting in $\delta^T$.
  - 2. Always follow the convention
    - Look at dimensions to figure out when to transpose and/or reorder terms.
Notes on PA1

• Don’t worry if you used some other method for gradient computation (as long as your answer is right and you are consistent!)

• This lecture we computed the gradient of the score, but in PA1 its of the loss

• Don’t forget to replace $f'$ with the actual derivative

• PA1 uses $xW + b$ for the linear transformation: gradients are different!
Backpropagation

• Compute gradients algorithmically

• Converting what we just did by hand into an algorithm

• Used by deep learning frameworks (TensorFlow, PyTorch, etc.)
Computational Graphs

- Representing our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations

\[
\begin{align*}
s &= u^T h \\
h &= f(z) \\
z &= Wx + b \\
x & \text{ (input)}
\end{align*}
\]
Computational Graphs

- Representing our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

\[
\begin{align*}
  s &= u^T h \\
  h &= f(z) \\
  z &= Wx + b \\
  x &= \text{(input)}
\end{align*}
\]
Computational Graphs

- Representing our neural net equations as a graph

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = c + b \]

"Forward Propagation"
Backpropagation

• Go backwards along edges
  • Pass along gradients

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]
Backpropagation: Single Node

- Node receives an “upstream gradient”
- Goal is to pass on the correct “downstream gradient”

\[ h = f(z) \]
Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

\[ h = f(z) \]
Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

\[ h = f(z) \]
Backpropagation: Single Node

- Each node has a **local gradient**
  - The gradient of its output with respect to its input

- \([\text{downstream gradient}] = [\text{upstream gradient}] \times [\text{local gradient}]\)
Backpropagation: Single Node

- What about nodes with multiple inputs?

\[ z = Wx \]
Backpropagation: Single Node

- Multiple inputs -> multiple local gradients

\[ z = Wx \]
An Example

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]
An Example

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]

Forward prop steps

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]
An Example

\[
f(x, y, z) = (x + y) \max(y, z)
\]
\[
x = 1, y = 2, z = 0
\]

Forward prop steps

\[
a = x + y
\]
\[
b = \max(y, z)
\]
\[
f = ab
\]

\[
\begin{aligned}
\text{Forward prop steps} \\
& a = x + y \\
& b = \max(y, z) \\
& f = ab
\end{aligned}
\]

\[
\begin{aligned}
x & \rightarrow 1 \\
y \quad & \rightarrow 2 \\
z & \rightarrow 0 \\
\text{max} & \rightarrow 2 \\
+ & \rightarrow 3 \\
\times & \rightarrow 6
\end{aligned}
\]
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps

\[ a = x + y \]

\[ b = \text{max}(y, z) \]

\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]

\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]

\[ f(x, y, z) = (x + y) \text{max}(y, z) \]

\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

\[
\begin{align*}
f(x, y, z) &= (x + y) \max(y, z) \\
x &= 1, y = 2, z = 0
\end{align*}
\]
An Example

Forward prop steps

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

\[
\begin{align*}
f(x, y, z) &= (x + y) \max(y, z) \\
x &= 1, y = 2, z = 0
\end{align*}
\]
An Example

Forward prop steps

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps

\[ a = x + y \]
\[ b = \text{max}(y, z) \]
\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

upstream * local = downstream
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]
An Example

Forward prop steps

\[ a = x + y \]

\[ b = \max (y, z) \]

\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]

\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]

\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

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

\[ x = 1, y = 2, z = 0 \]
Gradients add at branches
Gradients add at branches

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial y} + \frac{\partial f}{\partial b} \frac{\partial b}{\partial y} \]
Node Intuitions

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]

- + “distributes” the upstream gradient
Node Intuitions

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]

- + “distributes” the upstream gradient
- max “routes” the upstream gradient
Node Intuitions

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]

- + “distributes” the upstream gradient
- max “routes” the upstream gradient
- * “switches” the upstream gradient
Efficiency: compute all gradients at once

• Incorrect way of doing backprop:
  • First compute $\frac{\partial s}{\partial b}$

\[
\begin{align*}
  s &= u^T h \\
  h &= f(z) \\
  z &= Wx + b
\end{align*}
\]

\[x \quad \text{(input)}\]
Efficiency: compute all gradients at once

- Incorrect way of doing backprop:
  - First compute \( \frac{\partial s}{\partial b} \)
  - Then independently compute \( \frac{\partial s}{\partial W} \)
  - Duplicated computation!

\[
\begin{align*}
  s &= u^T h \\
  h &= f(z) \\
  z &= Wx + b \\
  x &= \text{(input)}
\end{align*}
\]
Efficiency: compute all gradients at once

- Correct way:
  - Compute all the gradients at once
  - Analogous to using $\delta$ when we computed gradients by hand

\[
\begin{align*}
    s &= u^T h \\
    h &= f(z) \\
    z &= Wx + b \\
    x &= \text{(input)}
\end{align*}
\]
Backprop Implementations

class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
Implementation: forward/backward API

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

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

\( (x, y, z \text{ are scalars}) \)

```python
class MultiplyGate(object):
    def forward(x, y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```
Implementation: forward/backward API

\[ (x, y, z \text{ are scalars}) \]

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

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

- For small $h$, \( f'(x) \approx \frac{f(x + h) - f(x - h)}{2h} \)
- Easy to implement
- But approximate and very slow:
  - Have to recompute $f$ for every parameter of our model
- Useful for checking your implementation
Summary

- Backpropagation: recursively apply the chain rule along computational graph
  - \([\text{downstream gradient}] = [\text{upstream gradient}] \times [\text{local gradient}]\)
- Forward pass: compute results of operation and save intermediate values
- Backward: apply chain rule to compute gradient
Project Types

1. Apply existing neural network model to a new task

2. Implement a complex neural architecture(s)
   - This is what PA4 will have you do!

3. Come up with a new model/training algorithm/etc.
   - Get 1 or 2 working first

   - See project page for some inspiration
Must-haves (choose-your-own final project)

- 10,000+ labeled examples by milestone
- Feasible task
- Automatic evaluation metric
- NLP is central
Details matter!

• Split your data into train/dev/test: only look at test for final experiments
• Look at your data, collect summary statistics
• Look at your model’s outputs, do error analysis
• Tuning hyperparameters is important
• Writeup quality is important
  • Look at last-year’s prize winners for examples
Project Advice

• Implement simplest possible model first (e.g., average word vectors and apply logistic regression) and improve it
  • Having a baseline system is crucial

• First overfit your model to train set (get really good training set results)
  • Then regularize it so it does well on the dev set

• Start early!