Neural Network: Training & Backpropagation
Announcements
Recap

A two layer fully connected feedforward NN:
$\text{ReLU}(x) = \max \{x, 0\}$

Recap

A two layer fully connected feedforward NN:

$$\hat{y} = \begin{pmatrix} h(x) \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha^T \text{ReLU}(W^2 \text{ReLU}(W^1 x)) + b \end{pmatrix}$$
Outline of Today

1. Training NNs via SGD

2. A Naive approach of computing gradients

3. Backpropagation: efficient way of computing gradients
Training neural network via SGD

Square loss on training example \((x, y)\)

\[
h(x) := \alpha^\top \text{ReLU} \left( W[2] \text{ReLU} \left( W[1] x \right) \right) + b
\]
Training neural network via SGD

Square loss on training example \((x, y)\)

\[
h(x) := \alpha^\top \text{ReLU} \left( W[2] \text{ReLU} \left( W[1] x \right) \right) + b
\]

\[
\ell(h(x), y) = (\hat{y} - y)^2, \text{ where } \hat{y} = h(x)
\]
Training neural network via SGD

Square loss on training example \((x, y)\)

\[
h(x) := \alpha^\top \text{ReLU} \left( W^2 \text{ReLU} \left( W^1 x \right) \right) + b
\]

Trainable parameters \(W^1, W^2, \alpha, b\)

\[
\ell (h(x), y) = (\hat{y} - y)^2, \text{ where } \hat{y} = h(x)
\]
Training neural network via SGD

Square loss on training example \((x, y)\)

\[
h(x) := \alpha^\top \text{ReLU} \left( W[2] \text{ReLU} \left( W[1] x \right) \right) + b
\]

\[
\ell(h(x), y) = (\hat{y} - y)^2, \text{ where } \hat{y} = h(x)
\]

Trainable parameters \(W[1], W[2], \alpha, b\)

Compute gradients:

\[
\frac{\partial \ell(h(x), y)}{\partial W[1]} \quad \frac{\partial \ell(h(x), y)}{\partial W[2]} \quad \frac{\partial \ell(h(x), y)}{\partial \alpha} \quad \frac{\partial \ell(h(x), y)}{\partial b}
\]
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]

For epoch \( t = 1 \) to \( T \):
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]

// go through dataset multiple times

For epoch \( t = 1 \) to \( T \):
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]

For epoch \( t = 1 \) to \( T \):  
Randomly shuffle the data
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]

// go through dataset multiple times

For epoch \( t = 1 \) to \( T \):

Randomly shuffle the data // important (unbiased estimate of the true gradient)
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]  // go through dataset multiple times

For epoch \( t = 1 \) to \( T \):

- Randomly shuffle the data
- Split the data into \( n/B \) many batches \( \mathcal{D}_i \), each with size \( B \)  // important (unbiased estimate of the true gradient)
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]  

// go through dataset multiple times

For epoch \( t = 1 \) to \( T \):

Randomly shuffle the data

Split the data into \( n/B \) many batches \( D_i \), each with size \( B \)

For \( i = 1 \) to \( n/B \)

// important (unbiased estimate of the true gradient)
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]  // go through dataset multiple times

For epoch \( t = 1 \) to \( T \):  // important (unbiased estimate of the true gradient)

Randomly shuffle the data

Split the data into \( n/B \) many batches \( \mathcal{D}_i \), each with size \( B \)

For \( i = 1 \) to \( n/B \)

\[
\text{Mini-batch gradient } g = \sum_{x,y \in \mathcal{D}_i} \nabla_{\theta} \ell(h_\theta(x), y)/B
\]
Training neural network via SGD

Mini-batch Stochastic gradient descent

\[ \theta = [W^{[1]}, W^{[2]}, \alpha, b] \]

For epoch \( t = 1 \) to \( T \):

- Randomly shuffle the data
- Split the data into \( n/B \) many batches \( \mathcal{D}_i \), each with size \( B \)

For \( i = 1 \) to \( n/B \)

- Mini-batch gradient \( g = \sum_{x,y \in \mathcal{D}_i} \nabla_\theta \mathcal{L}(h_\theta(x), y)/B \)

\[ \theta = \theta - \eta g \]
Training neural network via SGD

SGD helps avoiding local minima and saddle point

A local minima

A saddle point
Training neural network via SGD

SGD tends to converge to a flat region

Training loss
Training neural network via SGD

SGD tends to converge to a flat region
Training neural network via SGD

SGD tends to converge to a flat region

A flat local minima solution can help generalizes better to test data
Training neural network via SGD

SGD tends to converge to a flat region

A flat local minima solution can help generalizes better to test data
Outline of Today

1. Training NNs via SGD

2. A Naive approach of computing gradients

3. Backpropagation: efficient way of computing gradients
A naive algorithm

Consider the following one-dim case with identity transformation
A naive algorithm

Consider the following one-dim case with identity transformation

\[ \hat{y} = aw_T \ldots w_2w_1x \]
A naive algorithm

Consider the following one-dim case with identity transformation

\[ \hat{y} = aw_T \cdots w_2 w_1 x \]

Let's compute derivatives \( \frac{\partial \hat{y}}{\partial a}, \forall a = 1, \ldots T \)

\[ \frac{\partial \hat{y}}{\partial w_i} = \frac{2}{\partial w_i} \cdot \frac{2 \hat{y}}{\partial \hat{y}} \]
A naive algorithm

Consider the following one-dim case with identity transformation

\[ y = a w_T \cdots w_2 w_1 x \]

Let's compute derivatives \( \frac{\partial y}{\partial w_i}, \forall i = 1, \ldots T \)

Via chain rule:

\[ \frac{\partial \ell}{\partial w_i} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_i} \]
A naive algorithm

\[ z_0 = x \]

\[ z_1 = w_1 x = w_1 z_0 \]

\[ z_2 = w_2 z_1 \]

\[ z_T \]

\[ \hat{y} \]
A naive algorithm

\[
x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y}
\]

\[z_0 = x\]
\[z_2 = w_2 z_1\]

Via chain rule:

\[
\frac{2\hat{y}}{2w_1} = \frac{2\hat{y}}{2z_2} \cdot \frac{2z_2}{2z_{T-2}} \cdot \frac{2z_{T-1}}{2z_{T-1}} \cdots \frac{2z_1}{2w_1}
\]
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \quad z_2 = w_2 z_1 \]

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1}
\]
A naive algorithm

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \cdots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \quad \text{// computation: T}
\]
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \quad z_2 = w_2 z_1 \]

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \quad \text{// computation: T}
\]

\[
\frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_2}
\]
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \quad z_2 = w_2 z_1 \]

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \quad \text{// computation: T}
\]

\[
\frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_2} \quad \text{// computation: T-1}
\]
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \]

\[ z_2 = w_2 z_1 \]

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1}
\]

// computation: T

\[
\frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_2}
\]

// computation: T-1

\[
\frac{\partial \hat{y}}{\partial w_T} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial w_T}
\]
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \]

\[ z_2 = w_2z_1 \]

Via chain rule:

\[ \frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \]

// computation: T

\[ \frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_2} \]

// computation: T-1

\[ \frac{\partial \hat{y}}{\partial w_T} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial w_T} \]

// computation: 1
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \quad z_2 = w_2 z_1 \]

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \quad \text{// computation: T}
\]

\[
\frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_2} \quad \text{// computation: T-1}
\]

\[
\frac{\partial \hat{y}}{\partial w_T} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial w_T} \quad \text{// computation: 1}
\]

Total complexity:

\[ 1 + 2 + \ldots + T = O(T^2) \]
A naive algorithm

\[ x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \ldots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y} \]

\[ z_0 = x \quad z_2 = w_2z_1 \]

Via chain rule:

\[
\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \quad / \text{ computation: } T
\]

\[
\frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial z_{T-1}} \ldots \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_2} \quad / \text{ computation: } T-1
\]

\[
\frac{\partial \hat{y}}{\partial w_T} = \frac{\partial \hat{y}}{\partial z_T} \frac{\partial z_T}{\partial w_T} \quad / \text{ computation: } 1
\]

Total complexity:

\[ 1 + 2 + \ldots + T = O(T^2) \]

Quadratic in size of the graph!
Summary so far

What we did:
for each edge weight $w_i$, apply chain rule to calculate $\frac{\partial y}{\partial w_i}$
Summary so far

What we did:
for each edge weight $w_i$, apply chain rule to calculate $\frac{\partial y}{\partial w_i}$

What we got:
Able to compute gradient in running time $O \left( (\text{size of graph})^2 \right)$
Summary so far

What we did:
for each edge weight $w_i$, apply chain rule to calculate $\frac{\partial y}{\partial w_i}$

What we got:
Able to compute gradient in running time $O \left( (\text{size of graph})^2 \right)$

Can we do better in running time?
Outline of Today

1. Training NNs via SGD

2. A Naive approach of computing gradients

3. Backpropagation: efficient way of computing gradients
...Hinton popularized what they termed a “backpropagation” algorithm ... in 1986.

…the algorithm propagated measures of the errors produced by the network’s guesses backwards through its neurons, starting with those directly connected to the outputs.
...Hinton popularized what they termed a “backpropagation” algorithm … in 1986.

...the algorithm propagated measures of the errors produced by the network’s guesses backwards through its neurons, starting with those directly connected to the outputs.

This allowed networks with intermediate “hidden” neurons between input and output layers to learn efficiently, overcoming the limitations noted by Minsky and Papert.
Overview of backpropagation

Forward pass followed by a backward pass

Forward pass:

\[
x[1] x[2] \cdots x[d+1] = \frac{1}{\sum_{i=1}^{d+1} y_i W[i]}
\]
Overview of backpropagation

Forward pass followed by a backward pass

Forward pass:

Store input & output of all neurons
Overview of backpropagation

Forward pass followed by a backward pass

Forward pass:

\[
x^{[1]} = x^{[2]} = \ldots = x^{[d+1]} = 1
\]

\[
x^{[1]} \cdot W^{[1]} \cdot W^{[2]} = y
\]

Store input & output of all neurons

backward pass:

\[
x^{[1]} = x^{[2]} = \ldots = x^{[d+1]} = 1
\]

\[
x^{[1]} \cdot W^{[1]} \cdot W^{[2]} = y
\]
Overview of backpropagation

Forward pass followed by a backward pass

**Forward pass:**
- $x[1]$
- $x[2]$
- $x[d+1] = 1$
- $W[1]$ and $W[2]$
- Store input & output of all neurons

**backward pass:**
- $x[1]$
- $x[2]$
- $x[d+1] = 1$
- $W[1]$ and $W[2]$
- Compute derivatives
A Forward Pass: from $t = 0$ to $T$

$U = W^1 x^n$  

$Z = \text{ReLU}(U)$
A Forward Pass: from $t = 0$ to $T$

$z^0 = x$

1st layer of ReLU

2nd layer of ReLU
A Forward Pass: from $t = 0$ to $T$

$z^0 = x \quad u^1 = W^{[1]}z^0$

1st layer of ReLU

2nd layer of ReLU

$W^{[1]}$

$W^{[2]}$
A Forward Pass: from $t = 0$ to $T$

$z^0 = x$
$u^1 = W^{[1]} z^0$
$z^1 = \text{ReLU}(u^1)$

$u = W^{[2]} z^1$
$z^2 = \text{ReLU}(u)$
A Forward Pass: from $t = 0$ to $T$

- $z^0 = x$
- $u^1 = W[1]z^0$
- $z^1 = \text{ReLU}(u^1)$

$z^T = \text{ReLU}(u^T)$
A Forward Pass: from $t = 0$ to $T$

\[ z^0 = x \quad u^1 = W[1]z^0 \quad z^1 = \text{ReLU}(u^1) \]

1st layer of ReLU

2nd layer of ReLU

\[ \hat{y} = \langle \alpha, z^T \rangle \]

\[ z^T = \text{ReLU}(u^T) \]
Summary of the forward pass

All nodes’ values (i.e., z, u, ̂y) are computed and stored
Summary of the forward pass

All nodes’ values (i.e., z, u, \( \hat{y} \)) are computed and stored

Q: what is the computation complexity of the forward pass?
Summary of the forward pass

All nodes’ values (i.e., z, u, \( \hat{y} \)) are computed and stored

Q: what is the computation complexity of the forward pass?

A: linear in # of Edges + # of nodes
The backward Pass

Claim: to compute $\partial \hat{y} / \partial w$, $\forall$ edge $w$, it suffices to compute $\partial \hat{y} / \partial z$, $\forall$ node $z$. 
Claim: to compute $\frac{\partial \hat{y}}{\partial w}, \forall$ edge $w$, it suffices to compute $\frac{\partial \hat{y}}{\partial z}, \forall$ node $z$. 
The backward Pass

Claim: to compute $\partial \hat{y} / \partial w$, $\forall$ edge $w$, it suffices to compute $\partial \hat{y} / \partial z$, $\forall$ node $z$.

Proof:
WLOG consider $\partial \hat{y} / \partial w_{2,1}^t$
The backward Pass

Claim: to compute $\frac{\partial \hat{y}}{\partial w}$, $\forall$ edge $w$, it suffices to compute $\frac{\partial \hat{y}}{\partial z}$, $\forall$ node $z$.

Proof:
WLOG consider $\frac{\partial \hat{y}}{\partial w_{2,1}^t}$

$\frac{\partial \hat{y}}{\partial w_{2,1}^t}$
The backward Pass

Claim: to compute $\partial \hat{y} / \partial w$, ∀ edge $w$, it suffices to compute $\partial \hat{y} / \partial z$, ∀ node $z$.

Proof:
WLOG consider $\partial \hat{y} / \partial w_{2,1}^t$

$$\frac{\partial \hat{y}}{\partial w_{2,1}^t} = \frac{\partial \hat{y}}{\partial z_1^t} \cdot \frac{\partial z_1^t}{\partial u_1^t} \cdot \frac{\partial u_1^t}{\partial w_{2,1}^t}$$
The backward Pass

Claim: to compute $\frac{\partial \hat{y}}{\partial w}$, for all edge $w$, it suffices to compute $\frac{\partial \hat{y}}{\partial z}$, for all node $z$.

Proof:

WLOG consider $\frac{\partial \hat{y}}{\partial w_{2,1}^t}$

$$
\frac{\partial \hat{y}}{\partial w_{2,1}^t} = \frac{\partial \hat{y}}{\partial z_1^t} \cdot \frac{\partial z_1^t}{\partial u_1^t} \cdot \frac{\partial u_1^t}{\partial w_{2,1}^t}
$$

$$
= \frac{\partial \hat{y}}{\partial z_1^t} \cdot \sigma'(u_1^t) \cdot z_2^{t-1}
$$
The backward Pass

Claim: to compute $\partial \hat{y} / \partial w$, $\forall$ edge $w$, it suffices to compute $\partial \hat{y} / \partial z$, $\forall$ node $z$.

Proof:

WLOG consider $\partial \hat{y} / \partial w_{2,1}^t$

$$
\frac{\partial \hat{y}}{\partial w_{2,1}^t} = \frac{\partial \hat{y}}{\partial z_1^t} \cdot \frac{\partial z_1^t}{\partial u_1^t} \cdot \frac{\partial u_1^t}{\partial w_{2,1}^t} = \left(\frac{\partial \hat{y}}{\partial z_1^t}\right) \cdot \sigma'(u_1^t) \cdot z_2^{t-1}
$$

Given by assumption
The backward Pass

Claim: to compute $\frac{\partial \hat{y}}{\partial w}$, $\forall$ edge $w$, it suffices to compute $\frac{\partial \hat{y}}{\partial z}$, $\forall$ node $z$.

Proof:

WLOG consider $\frac{\partial \hat{y}}{\partial w_{2,1}^t}$

$$\frac{\partial \hat{y}}{\partial w_{2,1}^t} = \frac{\partial \hat{y}}{\partial z_1^t} \cdot \frac{\partial z_1^t}{\partial u_1^t} \cdot \frac{\partial u_1^t}{\partial w_{2,1}^t}$$

Given by assumption

Derivative of ReLU
The backward Pass

Claim: to compute $\partial \hat{y} / \partial w$, ∀ edge $w$, it suffices to compute $\partial \hat{y} / \partial z$, ∀ node $z$.

Proof:

WLOG consider $\partial \hat{y} / \partial w^t_{2,1}$

$$
\frac{\partial \hat{y}}{\partial w^t_{2,1}} = \frac{\partial \hat{y}}{\partial z^t_1} \cdot \frac{\partial z^t_1}{\partial u^t_1} \cdot \frac{\partial u^t_1}{\partial w^t_{2,1}}
$$

Given by assumption

Derivative of ReLU

Known from forward pass
The backward Pass

We compute $\frac{\partial \hat{y}}{\partial z^t}$ backwards in time from $t = T$ to $t = 1$:
The backward Pass: base case

Base case: compute $\frac{\partial \hat{y}}{\partial z^T}$, for all node $z$ at $T$-th Layer
The backward Pass: base case

Base case: compute $\partial \hat{y} / \partial z_i^T$, for all node $z$ at $T$-th Layer

$$\partial \hat{y} / \partial z_i^T = a_i$$
The backward Pass: induction step

Assume that we have computed $\partial \hat{y}/\partial z_i^t$, $\forall i$
The backward Pass: induction step

Assume that we have computed $\frac{\partial \hat{y}}{\partial z_i^t}, \forall i$

WLOG, consider $\frac{\partial \hat{y}}{\partial z_2^{t-1}}$
The backward Pass: induction step

Assume that we have computed $\frac{\partial \hat{y}}{\partial z_i^t}$, $\forall i$

WLOG, consider $\frac{\partial \hat{y}}{\partial z_2^{t-1}}$

Step 1: for all $i$, $\frac{\partial \hat{y}}{\partial u_i^t} = \frac{\partial \hat{y}}{\partial z_i^t} \frac{\partial z_i^t}{\partial u_i^t}$
The backward Pass: induction step

Assume that we have computed $\frac{\partial \hat{y}}{\partial z_i^t}, \forall i$

WLOG, consider $\frac{\partial \hat{y}}{\partial z_2^t}$

Step 1: for all $i$, 
$$\frac{\partial \hat{y}}{\partial u_i^t} = \frac{\partial \hat{y}}{\partial z_i^t} \cdot \frac{\partial z_i^t}{\partial u_i^t}$$

$$= \frac{\partial \hat{y}}{\partial z_i^t} \cdot \sigma'(u_i^t)$$

Given by assumption
The backward Pass: induction step

Assume that we have computed $\frac{\partial \hat{y}}{\partial z_i^t}, \forall i$

WLOG, consider $\frac{\partial \hat{y}}{\partial z_2^{t-1}}$

Step 1: for all $i$, \[
\frac{\partial \hat{y}^t}{\partial u_i^t} = \frac{\partial \hat{y}}{\partial z_i^t} \frac{\partial z_i^t}{\partial u_i^t} = \frac{\partial \hat{y}}{\partial z_i^t} \cdot \sigma'(u_i^t)
\]
The backward Pass: induction step

Assume that we have computed $\frac{\partial \hat{y}}{\partial z^t_i}, \forall i$

WLOG, consider $\frac{\partial \hat{y}}{\partial z^{t-1}_2}$

Step 1: for all $i$, $\frac{\partial \hat{y}}{\partial u^t_i} = \frac{\partial \hat{y}}{\partial z^t_i} \frac{\partial z^t_i}{\partial u^t_i} = \frac{\partial \hat{y}}{\partial z^t_i} \cdot \sigma'(u^t_i)$
The backward Pass: induction step

Assume that we have computed $\frac{\partial \ell}{\partial z_i^t}, \forall i$

After step 1, we have $\frac{\partial \hat{y}}{\partial u_i^t}, \forall i$
The backward Pass: induction step

Assume that we have computed $\frac{\partial \ell}{\partial z_i^t}$, $\forall i$

After step 1, we have $\frac{\partial \hat{y}}{\partial u_i^t}$, $\forall i$

Via multivariate chain rule:

$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial \hat{y}}{\partial u_1} \frac{\partial u_1}{\partial x} + \frac{\partial \hat{y}}{\partial u_2} \frac{\partial u_2}{\partial x} + \frac{\partial \hat{y}}{\partial u_3} \frac{\partial u_3}{\partial x}$$
The backward Pass: induction step

Assume that we have computed $\frac{\partial \ell}{\partial z_i^t}, \forall i$

After step 1, we have $\frac{\partial \hat{y}}{\partial u_i^t}, \forall i$

Via multivariate chain rule:

Step 2: $\frac{\partial \hat{y}}{\partial z_{2}^{t-1}} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u_i^t} \frac{\partial u_i^t}{\partial z_{2}^{t-1}}$

Computed after step 1
The backward Pass: induction step

Assume that we have computed $\frac{\partial \ell}{\partial z^t_i}, \forall i$

After step 1, we have $\frac{\partial \hat{y}}{\partial u^t_i}, \forall i$

Via multivariate chain rule:

Step 2:

$$
\frac{\partial \hat{y}}{\partial z^{t-1}_2} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u^t_i} \frac{\partial u^t_i}{\partial z^{t-1}_2} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u^t_i} \cdot w^t_{2,i}
$$
The backward Pass: induction step

Assume that we have computed $\partial \ell / \partial z^t_i$, $\forall i$

After step 1, we have $\partial \hat{y} / \partial u^t_i$, $\forall i$

Via multivariate chain rule:

Step 2:

$$\frac{\partial \hat{y}}{\partial z^{t-1}_2} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u^t_i} \cdot \frac{\partial u^t_i}{\partial z^{t-1}_2}$$

$$= \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u^t_i} \cdot w_{2,i}^t$$

We are done at node $z^{t-1}_2$!
The backward Pass: induction step

Assume that we have computed $\frac{\partial \ell}{\partial z^t_i}, \forall i$

After step 1, we have $\frac{\partial \hat{y}}{\partial u^t_i}, \forall i$

Via multivariate chain rule:

Step 2: $\frac{\partial \hat{y}}{\partial z^{t-1}_2} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u^t_i} \frac{\partial u^t_i}{\partial z^{t-1}_2}$

$= \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u^t_i} \cdot w^t_{2,i}$

We are done at node $z^{t-1}_2$!
The backward Pass: induction step

Assume that we have computed $\frac{\partial \ell}{\partial z_i^t}, \forall i$

After step 1, we have $\frac{\partial \hat{y}}{\partial u_i^t}, \forall i$

Via multivariate chain rule:

Step 2:  \[
\frac{\partial \hat{y}}{\partial z_{2}^{t-1}} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u_i^t} \frac{\partial u_i^t}{\partial z_{2}^{t-1}} \\
= \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u_i^t} \cdot w_{2,i}^t
\]

We are done at node $z_{2}^{t-1}$!

Repeat this for all $z_i^{t-1}, \forall i$
Summary of backward pass

\[ z_t = \frac{z_{t-1}}{2} \]

\[ u_t = \begin{cases} 1 & \text{if } z_{t-1} > 0 \\ \frac{z_{t-1}}{2} & \text{if } z_{t-1} \leq 0 \end{cases} \]

\[ w_{2,1}^t \]

\[ w_{2,2}^t \]

\[ w_{2,3}^t \]
The computation from $\frac{\partial \hat{y}}{\partial z^t}$ to $\frac{\partial \hat{y}}{\partial z^{t-1}}$ is the # of all edges in the sub-graph
The computation from $\partial \hat{y} / z^t$ to $\partial \hat{y} / z^{t-1}$ is the # of all edges in the sub-graph

**Total computation:** # of edges + # of nodes!
Summary of backward pass

The computation from $\partial \hat{y}/z^t$ to $\partial \hat{y}/z^{t-1}$ is the # of all edges in the sub-graph

Total computation: # of edges + # of nodes!

Exercise: can you express backward pass in matrix-vector format?
Summary for today

1. Naively compute all derivatives wrt edges using chain rule takes \((E + V)^2\) time
Summary for today

1. Naively compute all derivatives wrt edges using chain rule takes $(E + V)^2$ time

2. Backpropagation: forward pass & backward pass takes $O(E + V)$ time
Summary for today

1. Naively compute all derivatives wrt edges using chain rule takes $(E + V)^2$ time

2. Backpropagation: forward pass & backward pass takes $O(E + V)$ time

Forward pass: $x = z^0 \rightarrow u^1 \rightarrow z^1 \rightarrow \ldots \rightarrow z^t \rightarrow u^{t+1} \rightarrow z^{t+1} \ldots \rightarrow z^T \rightarrow \hat{y}$
Summary for today

1. Naively compute all derivatives wrt edges using chain rule takes $(E + V)^2$ time

2. Backpropagation: forward pass & backward pass takes $O(E + V)$ time

Forward pass: $x = z^0 \rightarrow u^1 \rightarrow z^1 \rightarrow \ldots \rightarrow z^t \rightarrow u^{t+1} \rightarrow z^{t+1} \ldots \rightarrow z^T \rightarrow \hat{y}$

Backward pass: $\frac{\partial \hat{y}}{\partial z^T} \rightarrow \frac{\partial \hat{y}}{\partial z^{T-1}} \rightarrow \ldots \rightarrow \frac{\partial \hat{y}}{\partial z^1} \overset{\text{claim}}{=} \frac{\partial y}{\partial x}$