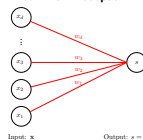


Lecture 3 - Back Propagation

DD2424

April 3, 2017

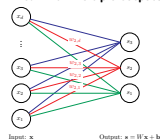
Linear with 1 output



Final decision:

$$g(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

Linear with multiple outputs



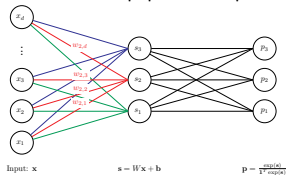
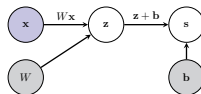
Final decision:

$$g(\mathbf{x}) = \arg \max_j s_j$$

Classification functions we have encountered so far

Computational graph of the multiple linear function

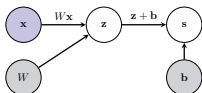
Linear with multiple probabilistic outputs

Final decision: $g(\mathbf{x}) = \arg \max_j p_j$ 

The computational graph:

- Represents order of computations.
- Displays the dependencies between the computed quantities.
- User input, parameters that have to be learnt.

Computational Graph helps automate gradient computations.



- Assume have labelled training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
- Set W, b so they correctly & robustly predict labels of the $\mathbf{x}_i$'s
- Need then to
 - Measure the quality of the prediction's based on W, b .
 - Find the optimal W, b relative to the quality measure on the training data.

Multi-class SVM loss



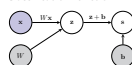
$$l_{\text{SVM}}(\mathbf{s}, y) = \sum_{\substack{j=1 \\ j \neq y}}^C \max(0, s_j - s_y + 1)$$

Cross-entropy loss

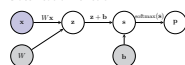


$$l_{\text{CE}}(\mathbf{p}, y) = -\log(p_y)$$

Classification function



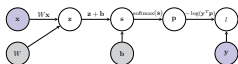
Classification function



Computational graph of the complete loss function

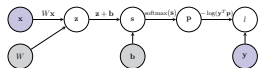
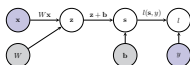
How do we learn W, b ?

- Linear scoring function + SOFTMAX + cross-entropy loss

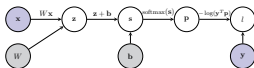


where $\mathbf{y}$ is the 1-hot response vector induced by the label y .

- Linear scoring function + multi-class SVM loss



- Assume have labelled training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
- Set W, b so they correctly & robustly predict labels of the $\mathbf{x}_i$'s
- Need then to
 - measure the quality of the prediction's based on W, b .
 - find an optimal W, b relative to the quality measure on the training data.

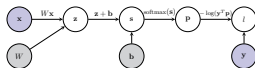


- Let l be the loss function defined by the computational graph.
- Find W, b by optimizing

$$\arg \max_{W, b} \frac{1}{|\mathcal{D}|} \sum_{(x, y) \in \mathcal{D}} l(x, y, W, b)$$

- Solve using a variant of **mini-batch gradient descent**
 $\implies$ need to efficiently compute the gradient vectors

$$\nabla_W l(x, y, W, b)|_{(x, y) \in \mathcal{D}} \quad \text{and} \quad \nabla_b l(x, y, W, b)|_{(x, y) \in \mathcal{D}}$$



- Let l be the complete loss function defined by the computational graph.

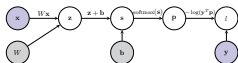
- How do we efficiently compute the gradient vectors

$$\nabla_W l(x, y, W, b)|_{(x, y) \in \mathcal{D}} \quad \text{and} \quad \nabla_b l(x, y, W, b)|_{(x, y) \in \mathcal{D}}?$$

- Answer: **Back Propagation**

Today's lecture: Gradient computations in neural networks

- For our learning approach need to be able to compute gradients efficiently.
- BackProp is algorithm for achieving given the form of many of our classifiers and loss functions.



- BackProp relies on the **chain rule** applied to the **composition of functions**.
- Example: the composition of functions

$$l(x, y, W, b) = -\log(y^T \text{softmax}(Wx + b))$$

linear classifier then **SOFTMAX** then cross-entropy loss

Chain Rule for functions with a scalar input and a scalar output

- Have two functions $g : \mathbb{R} \rightarrow \mathbb{R}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$.
- Define $h : \mathbb{R} \rightarrow \mathbb{R}$ as the composition of f and g :

$$h(x) = (f \circ g)(x) = f(g(x))$$

- How do we compute

$$\frac{dh(x)}{dx} ?$$

- Use the chain rule.

- Have functions $f, g : \mathbb{R} \rightarrow \mathbb{R}$ and define $h : \mathbb{R} \rightarrow \mathbb{R}$ as

$$h(x) = (f \circ g)(x) = f(g(x))$$

- Derivative of h w.r.t. x is given by the Chain Rule.

- **Chain Rule**

$$\frac{dh(x)}{dx} = \frac{df(y)}{dy} \frac{dg(x)}{dx} \quad \text{where } y = g(x)$$

Example of the Chain Rule in action

The composition of n functions

- Have

$$g(x) = x^2, \quad f(x) = \sin(x)$$

- One composition of these two functions is

$$h(x) = f(g(x)) = \sin(x^2)$$

- According to the **chain rule**

$$\begin{aligned} \frac{dh(x)}{dx} &= \frac{df(y)}{dy} \frac{dg(x)}{dx} \quad \leftarrow \text{where } y = x^2 \\ &= \frac{d \sin(y)}{dy} \frac{dx^2}{dx} \\ &= \cos(y) 2x \\ &= 2x \cos(x^2) \quad \leftarrow \text{plug in } y = x^2 \end{aligned}$$

- Have functions $f_1, \dots, f_n : \mathbb{R} \rightarrow \mathbb{R}$

- Define function $h : \mathbb{R} \rightarrow \mathbb{R}$ as the composition of f_j 's

$$h(x) = (f_n \circ f_{n-1} \circ \dots \circ f_1)(x) = f_n(f_{n-1}(\dots(f_1(x))\dots))$$

- Can we compute the derivative

$$\frac{dh(x)}{dx} ?$$

- Yes recursively apply the CHAIN RULE

- Have functions $f_1, \dots, f_n : \mathbb{R} \rightarrow \mathbb{R}$

- Define function $h : \mathbb{R} \rightarrow \mathbb{R}$ as the composition of f_j 's

$$h(x) = (f_n \circ f_{n-1} \circ \dots \circ f_1)(x) = f_n(f_{n-1}(\dots(f_1(x))\dots))$$

- Can we compute the derivative

$$\frac{dh(x)}{dx} \quad ?$$

- Yes recursively apply the CHAIN RULE

$$h(x) = (f_n \circ f_{n-1} \circ \dots \circ f_1)(x)$$

- Define

$$g_j = f_n \circ f_{n-1} \circ \dots \circ f_j$$

- Therefore $g_1 = h$, $g_n = f_n$ and

$$g_j = g_{j+1} \circ f_j \quad \text{for } j = 1, \dots, n-1$$

- Let $y_j = f_j(y_{j-1})$ and $y_0 = x$ then

$$y_n = g_j(y_{j-1}) \quad \text{for } j = 1, \dots, n$$

- Apply the **Chain Rule**:

- For $j = 1, 2, 3, \dots, n-1$

$$\begin{aligned} \frac{dy_n}{dy_{j-1}} &= \frac{dg_j(y_{j-1})}{dy_{j-1}} = \frac{d(g_{j+1} \circ f_j)(y_{j-1})}{dy_{j-1}} = \frac{dg_{j+1}(y_j)}{dy_j} \frac{df_j(y_{j-1})}{dy_{j-1}} \\ &= \frac{dy_n}{dy_j} \frac{dy_j}{dy_{j-1}} \end{aligned}$$

The Chain Rule for the composition of n functions

Recursively applying this fact gives:

$$\begin{aligned} \frac{dh(x)}{dx} &= \frac{dg_1(x)}{dx} && \leftarrow \text{Apply } h = g_1 \\ &= \frac{d(g_2 \circ f_1)(x)}{dx} && \leftarrow \text{Apply } g_1 = g_2 \circ f_1 \\ &= \frac{dg_2(y_1)}{dy_1} \frac{df_1(x)}{dx} && \leftarrow \text{Apply chain rule \& } y_1 = f_1(x) \\ &= \frac{d(g_3 \circ f_2)(y_1)}{dy_1} \frac{df_1(x)}{dx} && \leftarrow \text{Apply } g_2 = g_3 \circ f_2 \\ &= \frac{dg_3(y_2)}{dy_2} \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx} && \leftarrow \text{Apply chain rule \& } y_2 = f_2(y_1) \\ &\vdots \\ &= \frac{dg_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \dots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx} \\ &= \frac{df_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \dots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx} && \leftarrow \text{Apply } g_n = f_n \end{aligned}$$

where $y_j = (f_j \circ f_{j-1} \circ \dots \circ f_1)(x) = f_j(y_{j-1})$.

Summary: Chain Rule for a composition of n functions

- Have $f_1, \dots, f_n : \mathbb{R} \rightarrow \mathbb{R}$ and define h as their composition

$$h(x) = (f_n \circ f_{n-1} \circ \dots \circ f_1)(x)$$

- Then

$$\begin{aligned} \frac{dh(x)}{dx} &= \frac{df_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \dots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx} \\ &= \frac{dy_n}{dy_{n-1}} \frac{dy_{n-1}}{dy_{n-2}} \dots \frac{dy_2}{dy_1} \frac{dy_1}{dx} \end{aligned}$$

where $y_j = (f_j \circ f_{j-1} \circ \dots \circ f_1)(x) = f_j(y_{j-1})$.

- Remember: As $y_0 = x$ then for $j = n-1, n-2, \dots, 0$

$$\frac{dy_n}{dy_j} = \frac{dy_n}{dy_{j+1}} \frac{dy_{j+1}}{dy_j}$$

- Have $f_1, \dots, f_n : \mathbb{R} \rightarrow \mathbb{R}$ and define h as their composition

$$h(x) = (f_n \circ f_{n-1} \circ \dots \circ f_1)(x)$$

- Then

$$\begin{aligned} \frac{dh(x)}{dx} &= \frac{df_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \dots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx} \\ &= \frac{dy_n}{dy_{n-1}} \frac{dy_{n-1}}{dy_{n-2}} \dots \frac{dy_2}{dy_1} \frac{dy_1}{dx} \end{aligned}$$

$$\text{where } y_j = (f_j \circ f_{j-1} \circ \dots \circ f_1)(x) = f_j(y_{j-1}).$$

- Remember:** As $y_0 = x$ then for $j = n-1, n-2, \dots, 0$

$$\frac{dy_n}{dy_j} = \frac{dy_n}{dy_{j+1}} \frac{dy_{j+1}}{dy_j}$$

$$\frac{dh(x)}{dx} = \frac{dy_n}{dx} = \frac{dy_n}{dy_{n-1}} \frac{dy_{n-1}}{dy_{n-2}} \dots \frac{dy_2}{dy_1} \frac{dy_1}{dx}$$

Computation of $\frac{dy_n}{dx}$ relies on:

- Record keeping:** Compute and record values of the y_j 's.
- Iteratively aggregate** local gradients.

For $j = n-1, n, \dots, 1$

$$\text{- Compute local derivative: } \frac{df_{j+1}(y_j)}{dy_j} = \frac{dy_{j+1}}{dy_j}$$

- Aggregate:

$$\frac{dy_n}{dy_j} = \frac{dy_n}{dy_{j+1}} \frac{dy_{j+1}}{dy_j}$$

$$\text{Remember } \frac{dy_n}{dy_{j+1}} = \frac{dy_n}{dy_{n-1}} \frac{dy_{n-1}}{dy_{n-2}} \dots \frac{dy_{j+2}}{dy_{j+1}}$$

This is Backprop algorithm given a chain dependency between the y_j 's.

Exploit structure to compute gradient

Compute gradient of h at a point x^*

$$\frac{dh(x)}{dx} = \frac{dy_n}{dx} = \frac{dy_n}{dy_{n-1}} \frac{dy_{n-1}}{dy_{n-2}} \dots \frac{dy_2}{dy_1} \frac{dy_1}{dx}$$

Computation of $\frac{dy_n}{dx}$ relies on:

- Record keeping:** Compute and record values of the y_j 's.
- Iteratively aggregate** local gradients.

For $j = n-1, n, \dots, 1$

$$\text{- Compute local derivative: } \frac{df_{j+1}(y_j)}{dy_j} = \frac{dy_{j+1}}{dy_j}$$

- Aggregate:

$$\frac{dy_n}{dy_j} = \frac{dy_n}{dy_{j+1}} \frac{dy_{j+1}}{dy_j}$$

$$\text{Remember } \frac{dy_n}{dy_{j+1}} = \frac{dy_n}{dy_{n-1}} \frac{dy_{n-1}}{dy_{n-2}} \dots \frac{dy_{j+2}}{dy_{j+1}}$$

This is Backprop algorithm given a chain dependency between the y_j 's.

$$h(x) = (f_n \circ f_{n-1} \circ \dots \circ f_1)(x)$$

- Have a value for $x = x^*$
- Want to (efficiently) compute

$$\left. \frac{dh(x)}{dx} \right|_{x=x^*}$$

- Use the **Back-Propagation** algorithm.
- It consists of a **Forward** and **Backward** pass.



Evaluate $h(x^*)$ and keep track of the intermediary results

- Compute $y_1^* = f_1(x^*)$.
- for $j = 2, 3, \dots, n$
- Keep a record of $y_1^*, \dots, y_n^*$.

$$y_j^* = f_j(y_{j-1}^*)$$



Compute local f_j gradients and aggregate:

- Set $g = 1$.
- for $j = n, n-1, \dots, 2$

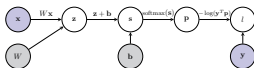
$$g = g \times \left. \frac{df_j(y_{j-1})}{dy_{j-1}} \right|_{y_{j-1}=y_{j-1}^*}$$



Note: $g = \left. \frac{dy}{dy_{j-1}} \right|_{y_{j-1}=y_{j-1}^*}$

- Then $\left. \frac{dh(x)}{dx} \right|_{x=x^*} = g \times \left. \frac{df_1(x)}{dx} \right|_{x=x^*}$

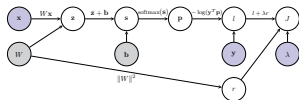
Problem 1: But what if I don't have a chain?



- This computational graph is **not a chain**.
- Some nodes have multiple parents.
- The function represented by graph is

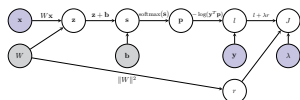
$$l(\mathbf{x}, \mathbf{y}, W, \mathbf{b}) = -\log(\mathbf{y}^T \text{Softmax}(W\mathbf{x} + \mathbf{b}))$$

Problem 1a: And when a regularization term is added..

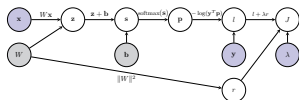


- This computational graph is **not a chain**.
 - Some nodes have **multiple parents** and others **multiple children**.
 - The function represented by graph is
- $$J(\mathbf{x}, \mathbf{y}, W, \mathbf{b}, \lambda) = -\log(\mathbf{y}^T \text{Softmax}(W\mathbf{x} + \mathbf{b})) + \lambda \sum_{i,j} W_{i,j}^2$$

• How is the back-propagation algorithm defined in these cases?

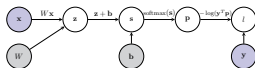


- This computational graph is **not a chain**.
- Some nodes have **multiple parents** and others **multiple children**.
- The function represented by graph is
$$J(\mathbf{x}, \mathbf{y}, W, \mathbf{b}, \lambda) = -\log(\mathbf{y}^T \text{Softmax}(W\mathbf{x} + \mathbf{b})) + \lambda \sum_{i,j} W_{i,j}^2$$
- How is the back-propagation algorithm defined in these cases?



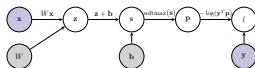
- The function represented by graph:
$$J(\mathbf{x}, \mathbf{y}, W, \mathbf{b}, \lambda) = -\log(\mathbf{y}^T \text{Softmax}(W\mathbf{x} + \mathbf{b})) + \lambda \sum_{i,j} W_{i,j}^2$$
- Nearly all of the inputs and intermediary outputs are **vectors** or **matrices**.
- How are the derivatives defined in this case?

Issues we need to sort out



- Back-propagation when the computational graph is **not a chain**.
- Derivative computations when the inputs and outputs are not scalars.
- Will address these issues now. First the derivatives of vectors.

Issues we need to sort out



- Back-propagation when the computational graph is **not a chain**.
- Derivative computations when the inputs and outputs are not scalars.
- Will address these issues now. First the derivatives of vectors.

Chain Rule for functions with vector inputs and vector outputs

- Have two functions $g: \mathbb{R}^d \rightarrow \mathbb{R}^m$ and $f: \mathbb{R}^m \rightarrow \mathbb{R}^c$.
- Define $h: \mathbb{R}^d \rightarrow \mathbb{R}^c$ as the composition of f and g :

$$h(\mathbf{x}) = (f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$$

- Consider

$$\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$$

- How is it defined and computed?
- What's the chain rule for vector valued functions?

Chain Rule for vector input and output

- Let $\mathbf{y} = h(\mathbf{x})$ where each $h: \mathbb{R}^d \rightarrow \mathbb{R}^c$ then

$$\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_d} \\ \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_2}{\partial x_d} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_c}{\partial x_1} & \cdots & \frac{\partial y_c}{\partial x_d} \end{pmatrix} \quad \leftarrow \text{this is a Jacobian matrix}$$

and is a matrix of size $c \times d$.

- **Chain Rule** says if $h = f \circ g$ ($g: \mathbb{R}^d \rightarrow \mathbb{R}^m$ and $f: \mathbb{R}^m \rightarrow \mathbb{R}^c$) then

$$\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$$

where $\mathbf{z} = g(\mathbf{x})$ and $\mathbf{y} = f(\mathbf{z})$.

- Both $\frac{\partial \mathbf{y}}{\partial \mathbf{z}}$ ($c \times m$) and $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ ($m \times d$) defined silly to $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$.

Chain Rule for vector input and scalar output

The cost functions we will examine usually have a scalar output

- Let $\mathbf{x} \in \mathbb{R}^d$, $f: \mathbb{R}^d \rightarrow \mathbb{R}^m$ and $g: \mathbb{R}^m \rightarrow \mathbb{R}$

$$\mathbf{z} = f(\mathbf{x})$$

$$s = g(\mathbf{z})$$

- The **Chain Rule** says gradient of output w.r.t. input

$$\frac{\partial s}{\partial \mathbf{x}} = \left(\frac{\partial s}{\partial z_1} \quad \cdots \quad \frac{\partial s}{\partial z_d} \right)$$

is given by a gradient times a Jacobian:

$$\frac{\partial s}{\partial \mathbf{x}} = \underbrace{\frac{\partial s}{\partial \mathbf{z}}}_{1 \times m} \underbrace{\frac{\partial \mathbf{z}}{\partial \mathbf{x}}}_{m \times d}$$

where

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_d} \\ \frac{\partial z_2}{\partial x_1} & \cdots & \frac{\partial z_2}{\partial x_d} \\ \vdots & \vdots & \vdots \\ \frac{\partial z_m}{\partial x_1} & \cdots & \frac{\partial z_m}{\partial x_d} \end{pmatrix}$$

- $f_1: \mathbb{R}^d \rightarrow \mathbb{R}^{m_1}, f_2: \mathbb{R}^d \rightarrow \mathbb{R}^{m_2}$ and $g: \mathbb{R}^n \rightarrow \mathbb{R}$ ($n = m_1 + m_2$)

$$\mathbf{z}_1 = f_1(\mathbf{x}),$$

$$\mathbf{z}_2 = f_2(\mathbf{x})$$

$$s = g(\mathbf{z}_1, \mathbf{z}_2) = g(\mathbf{v}) \quad \text{where } \mathbf{v} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{pmatrix}.$$

- Chain Rule** says gradient of the output w.r.t. the input

$$\frac{\partial s}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial s}{\partial \mathbf{z}_1} & \dots & \frac{\partial s}{\partial \mathbf{z}_2} \end{pmatrix}$$

is given by:

$$\frac{\partial s}{\partial \mathbf{x}} = \underbrace{\frac{\partial s}{\partial \mathbf{v}}}_{1 \times n} \underbrace{\frac{\partial \mathbf{v}}{\partial \mathbf{x}}}_{n \times d}$$

But

$$\frac{\partial s}{\partial \mathbf{v}} = \begin{pmatrix} \frac{\partial s}{\partial \mathbf{z}_1} & \frac{\partial s}{\partial \mathbf{z}_2} \end{pmatrix} \quad \text{and} \quad \frac{\partial \mathbf{v}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{z}_2}{\partial \mathbf{x}} \end{pmatrix}$$

$\Rightarrow$

$$\frac{\partial s}{\partial \mathbf{x}} = \frac{\partial s}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{x}} = \underbrace{\frac{\partial s}{\partial \mathbf{z}_1}}_{1 \times m_1} \underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m_1 \times d} + \underbrace{\frac{\partial s}{\partial \mathbf{z}_2}}_{1 \times m_2} \underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{x}}}_{m_2 \times d}$$

- $f_i: \mathbb{R}^d \rightarrow \mathbb{R}^{m_i}$ for $i = 1, \dots, t$ and $g: \mathbb{R}^n \rightarrow \mathbb{R}$ ($n = m_1 + \dots + m_t$)

$$\mathbf{z}_i = f_i(\mathbf{x}), \quad \text{for } i = 1, \dots, t$$

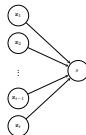
$$s = g(\mathbf{z}_1, \dots, \mathbf{z}_n)$$

- Consequence of the **Chain Rule**

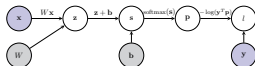
$$\frac{\partial s}{\partial \mathbf{x}} = \sum_{i=1}^t \frac{\partial s}{\partial \mathbf{z}_i} \frac{\partial \mathbf{z}_i}{\partial \mathbf{x}}$$

- Computational graph interpretation. Let $C_{\mathbf{x}}$ be the children nodes of $\mathbf{x}$ then

$$\frac{\partial s}{\partial \mathbf{x}} = \sum_{\mathbf{z} \in C_{\mathbf{x}}} \frac{\partial s}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$$



Issues we need to sort out

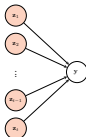


- Back-propagation when the computational graph is **not a chain**.
- Derivative computations when the inputs and outputs are not scalars. ✓
- Will now describe Back-prop for non-chains.

Back-propagation for non-chain computational graphs

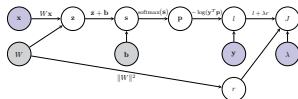
- Have node y .
- Denote the set of y 's parent nodes by $\mathcal{P}_y$ and their values by

$$V_{\mathcal{P}_y} = \{z.\text{value} \mid z \in \mathcal{P}_y\}$$



- Given $V_{\mathcal{P}_y}$ can now apply the function f_z

$$y.\text{value} = f_y(V_{\mathcal{P}_y})$$



- Consider node W in the above graph. Its children are $\{z, r\}$. Applying the chain rule

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial r} \frac{\partial r}{\partial W} + \frac{\partial J}{\partial z} \frac{\partial z}{\partial W}$$

- In general for node c with children specified by $\mathcal{C}_c$:

$$\frac{\partial J}{\partial c} = \sum_{u \in \mathcal{C}_c} \frac{\partial J}{\partial u} \frac{\partial u}{\partial c}$$

Pseudo-Code for the Generic Forward Pass

```

procedure EVALUATEGRAPHFN( $G$ )
     $S = \text{GetStartNodes}(G)$ 
    for  $s \in S$  do
         $\text{ComputeBranch}(s, G)$ 
    end for
end procedure

```

▷ G is the computational graph
▷ a start node has no parent and its value is already set

```

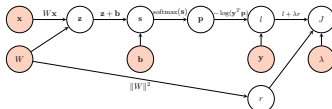
procedure COMPUTEBRANCH( $s, G$ )
     $\mathcal{C}_s = \text{GetChildren}(s, G)$ 
    for each  $n \in \mathcal{C}_s$  do
        if  $n$  computed then
             $\mathcal{P}_n = \text{GetParents}(n, G)$ 
            if  $\text{CheckAllNodesComputed}(\mathcal{P}_n)$  then ▷ Or not all parents of children are computed
                 $f_n = \text{GetNodeFn}(n)$ 
                 $n.\text{value} = f_n(\mathcal{P}_n)$ 
                 $n.\text{computed} = \text{true}$ 
                 $\text{ComputeBranch}(n, G)$ 
            end if
        end if
    end for
end procedure

```

▷ recursive fn evaluating nodes
▷ Try to evaluate each children node
▷ Unless child is already computed

Generic Forward Pass

Identify Start Nodes



```

procedure EVALUATEGRAPHFN( $G$ )
     $S = \text{GetStartNodes}(G)$ 
    for  $s \in S$  do
         $\text{ComputeBranch}(s, G)$ 
    end for
end procedure

```

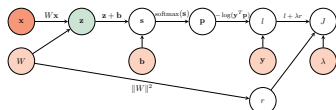
▷ G is the computational graph

```

procedure COMPUTEBRANCH( $s, G$ )
     $\mathcal{C}_s = \text{GetChildren}(s, G)$ 
    for each  $n \in \mathcal{C}_s$  do
        if  $n$  is computed then
             $\mathcal{P}_n = \text{GetParents}(n, G)$ 
            if  $\text{CheckAllNodesComputed}(\mathcal{P}_n)$  then
                 $f_n = \text{GetNodeFn}(n)$ 
                 $n.\text{value} = f_n(\mathcal{P}_n)$ 
                 $n.\text{computed} = \text{true}$ 
                 $\text{ComputeBranch}(n, G)$ 
            end if
        end if
    end for
end procedure

```

Order in which nodes are evaluated



```

procedure EVALUATEGRAPHFN(G) ▷ G is the computational graph
  S = GetStartNodes(G)
  for s ∈ S do
    ComputeBranch(s, G)
  end for
end procedure

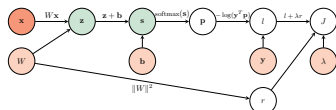
```

```

procedure COMPUTEBRANCH(s, G)
  Ps = GetParents(s, G)
  for each n ∈ Ps do
    if !n.computed then
      Cn = GetChildren(n, G)
      if CheckAllNodesComputed(Cn) then
        fn = GetNodeFn(n)
        n.value = fn(Cn)
        n.computed = true
        ComputeBranch(n, G)
      end if
    end if
  end for
end procedure

```

Order in which nodes are evaluated



```

procedure EVALUATEGRAPHFN(G) ▷ G is the computational graph
  S = GetStartNodes(G)
  for s ∈ S do
    ComputeBranch(s, G)
  end for
end procedure

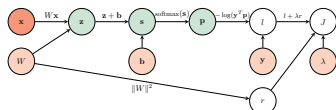
```

```

procedure COMPUTEBRANCH(s, G)
  Ps = GetParents(s, G)
  for each n ∈ Ps do
    if !n.computed then
      Cn = GetChildren(n, G)
      if CheckAllNodesComputed(Cn) then
        fn = GetNodeFn(n)
        n.value = fn(Cn)
        n.computed = true
        ComputeBranch(n, G)
      end if
    end if
  end for
end procedure

```

Order in which nodes are evaluated



```

procedure EVALUATEGRAPHFN(G) ▷ G is the computational graph
  S = GetStartNodes(G)
  for s ∈ S do
    ComputeBranch(s, G)
  end for
end procedure

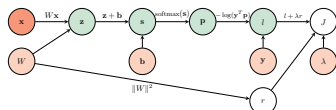
```

```

procedure COMPUTEBRANCH(s, G)
  Ps = GetParents(s, G)
  for each n ∈ Ps do
    if !n.computed then
      Cn = GetChildren(n, G)
      if CheckAllNodesComputed(Cn) then
        fn = GetNodeFn(n)
        n.value = fn(Cn)
        n.computed = true
        ComputeBranch(n, G)
      end if
    end if
  end for
end procedure

```

Order in which nodes are evaluated



```

procedure EVALUATEGRAPHFN(G) ▷ G is the computational graph
  S = GetStartNodes(G)
  for s ∈ S do
    ComputeBranch(s, G)
  end for
end procedure

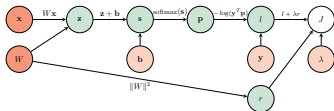
```

```

procedure COMPUTEBRANCH(s, G)
  Ps = GetParents(s, G)
  for each n ∈ Ps do
    if !n.computed then
      Cn = GetChildren(n, G)
      if CheckAllNodesComputed(Cn) then
        fn = GetNodeFn(n)
        n.value = fn(Cn)
        n.computed = true
        ComputeBranch(n, G)
      end if
    end if
  end for
end procedure

```

Order in which nodes are evaluated



```

procedure EVALUATEGRAPHFN(G)
  ▷ G is the computational graph
  S = GetStartNodes(G)
  for s ∈ S do
    ComputeBranch(s, G)
  end for
end procedure

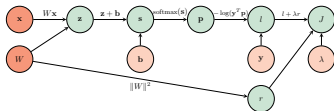
```

```

procedure COMPUTEBRANCH(s, G)
  Ps = GetParents(s, G)
  for each n ∈ Ps do
    if !n.computed then
      Cn = GetChildren(n, G)
      if CheckAllNodesComputed(Cn) then
        fn = GetNodeFn(n)
        n.value = fn(Cn)
        n.computed = true
        ComputeBranch(n, G)
      end if
    end if
  end for
end procedure

```

Order in which nodes are evaluated



```

procedure EVALUATEGRAPHFN(G)
  ▷ G is the computational graph
  S = GetStartNodes(G)
  for s ∈ S do
    ComputeBranch(s, G)
  end for
end procedure

```

```

procedure COMPUTEBRANCH(s, G)
  Ps = GetParents(s, G)
  for each n ∈ Ps do
    if !n.computed then
      Cn = GetChildren(n, G)
      if CheckAllNodesComputed(Cn) then
        fn = GetNodeFn(n)
        n.value = fn(Cn)
        n.computed = true
        ComputeBranch(n, G)
      end if
    end if
  end for
end procedure

```

Pseudo-Code for the Generic Backward Pass

```

procedure PERFORMBACKPASS(G)
  J = GetResultNode(G)
  BackOp(J, G)
end procedure

```

▷ node with the value of cost function
▷ Start the Backward-pass

```

procedure BACKOP(s, G)
  Cs = GetChildren(s, G)
  if Cs = {} then
    s.Grad = 1
  end if
  if AllGradientsComputed(Cs) then
    s.Grad = 0
    for each c ∈ Cs do
      s.Grad += c.Grad * c.s.Jacobian
    end for
    s.GradComputed = true
  end if
  for each p ∈ Ps do
    s.p.Jacobian =  $\frac{\partial f_{\mathbf{s}}(\mathbf{P}_{\mathbf{s}})}{\partial \mathbf{p}}$ 
    BackOp(p, G)
  end for
end procedure

```

▷ At the result node

▷ Have computed all $\frac{\partial J}{\partial c}$ where $c \in C_s$

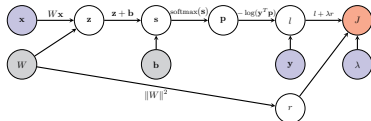
▷ $\frac{\partial J}{\partial s} = \frac{\partial J}{\partial c} \frac{\partial c}{\partial s}$

▷ Compute the Jacobian of $f_{\mathbf{s}}$ w.r.t. each parent node

▷ $\frac{\partial f_{\mathbf{s}}(\mathbf{P}_{\mathbf{s}})}{\partial \mathbf{p}} = \frac{\partial s}{\partial \mathbf{p}}$

Generic Backward Pass: Order of computations

Identify Result Node



```

procedure PERFORMBACKPASS(G)
  J = GetResultNode(G)
  BackOp(J, G)
end procedure

```

```

procedure BACKOP(s, G)
  Cs = GetChildren(s, G)
  if Cs = {} then
    s.Grad = 1
  else
    if AllGradientsComputed(Cs) then
      s.Grad = 0
      for each c ∈ Cs do
        s.Grad += c.Grad * c.s.Jacobian
      end for
      s.GradComputed = true
    end if
    for each p ∈ Ps do
      s.p.Jacobian =  $\frac{\partial f_{\mathbf{s}}(\mathbf{P}_{\mathbf{s}})}{\partial \mathbf{p}}$ 
      BackOp(p, G)
    end for
  end procedure

```

▷ At the result node

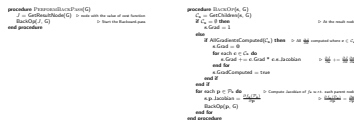
▷ All $\frac{\partial J}{\partial c}$ computed where $c \in C_s$

▷ $\frac{\partial J}{\partial s} = \frac{\partial J}{\partial c} \frac{\partial c}{\partial s}$

▷ Compute Jacobian of $f_{\mathbf{s}}$ w.r.t. each parent node

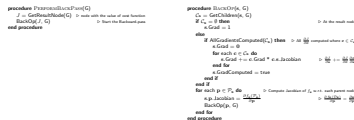
▷ $\frac{\partial f_{\mathbf{s}}(\mathbf{P}_{\mathbf{s}})}{\partial \mathbf{p}} = \frac{\partial s}{\partial \mathbf{p}}$

Compute Jacobian of current node w.r.t. its child

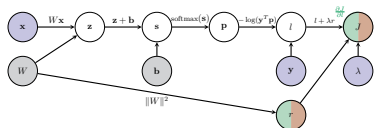


Generic Backward Pass: Order of computations

Compute Jacobian of current node w.r.t. its child



Compute Jacobian of current node w.r.t. its child



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G) ▷ with the value of use function
  BackOp(J, G) ▷ Start the Backward pass
end procedure

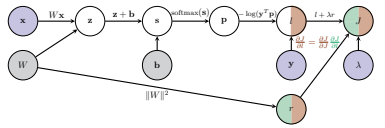
```

```

procedure BackOp(e, G)
  J ← GetChildNode(e, G)
  if  $C_e = 0$  then
    a.Grad ← 1
  else
    if AllGradientComputed( $C_e$ ) then ▷ all computed when  $e \in C_e$ 
      a.Grad ← 0
      for each  $c \in C_e$  do
        a.Grad ← a.Grad * e.a.Jacobian
      end for
      a.GradComputed ← true
    end if
    and if
      for each  $p \in P_e$  do ▷ Compute Jacobian of  $J_e$  w.r.t. each parent node
        e.p.Jacobian ←  $\frac{\partial J_e}{\partial p}$ 
        BackOp(p, G) ▷ Start the Backward pass
      end for
    end procedure
  end procedure

```

Compute Gradient of current node



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G) ▷ with the value of use function
  BackOp(J, G) ▷ Start the Backward pass
end procedure

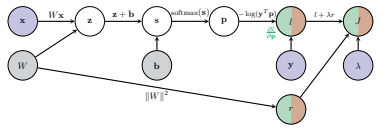
```

```

procedure BackOp(e, G)
  J ← GetChildNode(e, G)
  if  $C_e = 0$  then
    a.Grad ← 1
  else
    if AllGradientComputed( $C_e$ ) then ▷ all computed when  $e \in C_e$ 
      a.Grad ← 0
      for each  $c \in C_e$  do
        a.Grad ← a.Grad * e.a.Jacobian
      end for
      a.GradComputed ← true
    end if
    and if
      for each  $p \in P_e$  do ▷ Compute Jacobian of  $J_e$  w.r.t. each parent node
        e.p.Jacobian ←  $\frac{\partial J_e}{\partial p}$ 
        BackOp(p, G) ▷ Start the Backward pass
      end for
    end procedure
  end procedure

```

Compute Jacobian of current node w.r.t. its child



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G) ▷ with the value of use function
  BackOp(J, G) ▷ Start the Backward pass
end procedure

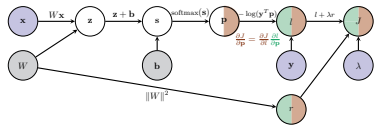
```

```

procedure BackOp(e, G)
  J ← GetChildNode(e, G)
  if  $C_e = 0$  then
    a.Grad ← 1
  else
    if AllGradientComputed( $C_e$ ) then ▷ all computed when  $e \in C_e$ 
      a.Grad ← 0
      for each  $c \in C_e$  do
        a.Grad ← a.Grad * e.a.Jacobian
      end for
      a.GradComputed ← true
    end if
    and if
      for each  $p \in P_e$  do ▷ Compute Jacobian of  $J_e$  w.r.t. each parent node
        e.p.Jacobian ←  $\frac{\partial J_e}{\partial p}$ 
        BackOp(p, G) ▷ Start the Backward pass
      end for
    end procedure
  end procedure

```

Compute Gradient of current node



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G) ▷ with the value of use function
  BackOp(J, G) ▷ Start the Backward pass
end procedure

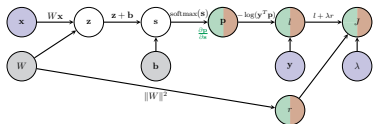
```

```

procedure BackOp(e, G)
  J ← GetChildNode(e, G)
  if  $C_e = 0$  then
    a.Grad ← 1
  else
    if AllGradientComputed( $C_e$ ) then ▷ all computed when  $e \in C_e$ 
      a.Grad ← 0
      for each  $c \in C_e$  do
        a.Grad ← a.Grad * e.a.Jacobian
      end for
      a.GradComputed ← true
    end if
    and if
      for each  $p \in P_e$  do ▷ Compute Jacobian of  $J_e$  w.r.t. each parent node
        e.p.Jacobian ←  $\frac{\partial J_e}{\partial p}$ 
        BackOp(p, G) ▷ Start the Backward pass
      end for
    end procedure
  end procedure

```

Compute Jacobian of current node w.r.t. its child



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G)
  BackChp(J, G)
end procedure

```

▷ ready with the value of user function

```

procedure BackChp(G)
  G ← GetChildNode(G)
  if CG = 0 then
    a Grad = 1
  else

```

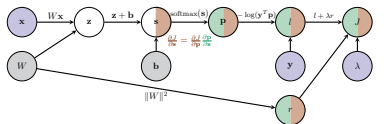
▷ At the result node

```

    if AllGradientComputed(CG) then ▷ all  $\frac{\partial G}{\partial C_i}$  computed where  $i \in C_G$ 
      a Grad = 0
      for each  $c \in C_G$  do
        a Grad  $\leftarrow$  a Grad * a.c.Jacobian ▷  $\frac{\partial G}{\partial C_i} = \frac{\partial G}{\partial C_i} \cdot \frac{\partial C_i}{\partial x}$ 
        and for
          a GradComputed = true
      end if
    end if
    for each  $p \in P_G$  do
      a.p.Jacobian =  $\frac{\partial G}{\partial p}$  ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
      BackChp(p, G) ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
    end for
  end procedure

```

Compute Gradient of current node



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G)
  BackChp(J, G)
end procedure

```

▷ ready with the value of user function

```

procedure BackChp(G)
  G ← GetChildNode(G)
  if CG = 0 then
    a Grad = 1
  else

```

▷ At the result node

```

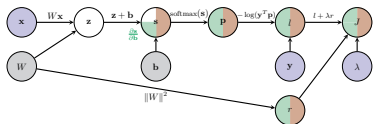
    if AllGradientComputed(CG) then ▷ all  $\frac{\partial G}{\partial C_i}$  computed where  $i \in C_G$ 
      a Grad = 0
      for each  $c \in C_G$  do
        a Grad  $\leftarrow$  a Grad * a.c.Jacobian ▷  $\frac{\partial G}{\partial C_i} = \frac{\partial G}{\partial C_i} \cdot \frac{\partial C_i}{\partial x}$ 
        and for
          a GradComputed = true
      end if
    end if
    for each  $p \in P_G$  do
      a.p.Jacobian =  $\frac{\partial G}{\partial p}$  ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
      BackChp(p, G) ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
    end for
  end procedure

```

Generic Backward Pass: Order of computations

Generic Backward Pass: Order of computations

Compute Jacobian of current node w.r.t. its child



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G)
  BackChp(J, G)
end procedure

```

▷ ready with the value of user function

```

procedure BackChp(G)
  G ← GetChildNode(G)
  if CG = 0 then
    a Grad = 1
  else

```

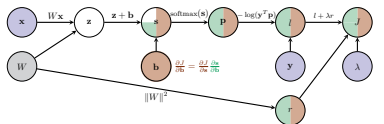
▷ At the result node

```

    if AllGradientComputed(CG) then ▷ all  $\frac{\partial G}{\partial C_i}$  computed where  $i \in C_G$ 
      a Grad = 0
      for each  $c \in C_G$  do
        a Grad  $\leftarrow$  a Grad * a.c.Jacobian ▷  $\frac{\partial G}{\partial C_i} = \frac{\partial G}{\partial C_i} \cdot \frac{\partial C_i}{\partial x}$ 
        and for
          a GradComputed = true
      end if
    end if
    for each  $p \in P_G$  do
      a.p.Jacobian =  $\frac{\partial G}{\partial p}$  ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
      BackChp(p, G) ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
    end for
  end procedure

```

Compute Gradient of current node



```

procedure PreprocessFunc(G)
  J ← GetFuncNode(G)
  BackChp(J, G)
end procedure

```

▷ ready with the value of user function

```

procedure BackChp(G)
  G ← GetChildNode(G)
  if CG = 0 then
    a Grad = 1
  else

```

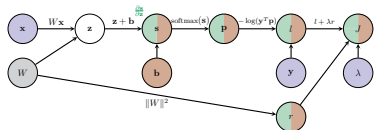
▷ At the result node

```

    if AllGradientComputed(CG) then ▷ all  $\frac{\partial G}{\partial C_i}$  computed where  $i \in C_G$ 
      a Grad = 0
      for each  $c \in C_G$  do
        a Grad  $\leftarrow$  a Grad * a.c.Jacobian ▷  $\frac{\partial G}{\partial C_i} = \frac{\partial G}{\partial C_i} \cdot \frac{\partial C_i}{\partial x}$ 
        and for
          a GradComputed = true
      end if
    end if
    for each  $p \in P_G$  do
      a.p.Jacobian =  $\frac{\partial G}{\partial p}$  ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
      BackChp(p, G) ▷  $\frac{\partial G}{\partial p} = \frac{\partial G}{\partial p} \cdot \frac{\partial p}{\partial x}$ 
    end for
  end procedure

```

Compute Jacobian of current node w.r.t. its child



```

procedure PreorderTraverse(G)
  J ← GetInitialNode(G)
  BackOp(J, G)
end procedure

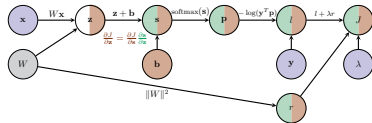
```

```

procedure BackOp(p, G)
  J ← GetChildNode(p, G)
  if  $C_p = \emptyset$  then
    a Grad ← 1
  else
    if AllGradsComputed( $C_p$ ) then
      a Grad ← 0
    for each  $c \in C_p$  do
      a Grad ← a Grad * c.a.Jacobian
    end for
    a GradComputed ← true
  end if
  for each  $p' \in P_p$  do
    p'.Jacobian ←  $\frac{\partial J}{\partial p'}$ 
    BackOp(p', G)
  end for
end procedure

```

Compute Gradient of current node



```

procedure PreorderTraverse(G)
  J ← GetInitialNode(G)
  BackOp(J, G)
end procedure

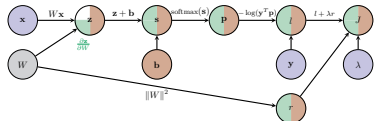
```

```

procedure BackOp(p, G)
  J ← GetChildNode(p, G)
  if  $C_p = \emptyset$  then
    a Grad ← 1
  else
    if AllGradsComputed( $C_p$ ) then
      a Grad ← 0
    for each  $c \in C_p$  do
      a Grad ← a Grad * c.a.Jacobian
    end for
    a GradComputed ← true
  end if
  for each  $p' \in P_p$  do
    p'.Jacobian ←  $\frac{\partial J}{\partial p'}$ 
    BackOp(p', G)
  end for
end procedure

```

Compute Jacobian of current node w.r.t. its child



```

procedure PreorderTraverse(G)
  J ← GetInitialNode(G)
  BackOp(J, G)
end procedure

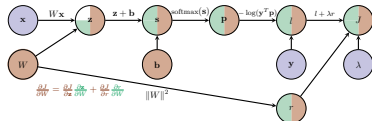
```

```

procedure BackOp(p, G)
  J ← GetChildNode(p, G)
  if  $C_p = \emptyset$  then
    a Grad ← 1
  else
    if AllGradsComputed( $C_p$ ) then
      a Grad ← 0
    for each  $c \in C_p$  do
      a Grad ← a Grad * c.a.Jacobian
    end for
    a GradComputed ← true
  end if
  for each  $p' \in P_p$  do
    p'.Jacobian ←  $\frac{\partial J}{\partial p'}$ 
    BackOp(p', G)
  end for
end procedure

```

Compute Gradient of current node



```

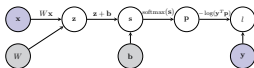
procedure PreorderTraverse(G)
  J ← GetInitialNode(G)
  BackOp(J, G)
end procedure

```

```

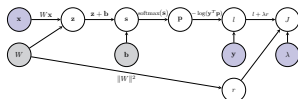
procedure BackOp(p, G)
  J ← GetChildNode(p, G)
  if  $C_p = \emptyset$  then
    a Grad ← 1
  else
    if AllGradsComputed( $C_p$ ) then
      a Grad ← 0
    for each  $c \in C_p$  do
      a Grad ← a Grad * c.a.Jacobian
    end for
    a GradComputed ← true
  end if
  for each  $p' \in P_p$  do
    p'.Jacobian ←  $\frac{\partial J}{\partial p'}$ 
    BackOp(p', G)
  end for
end procedure

```



- Back-propagation when the computational graph is **not** a chain. ✓
- Derivative computations when the inputs and outputs are not scalars. ✓
- Let's now compute some gradients!

Compute gradients for

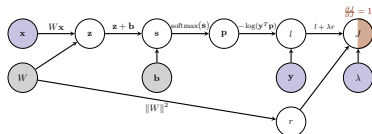


linear scoring function + SOFTMAX + cross-entropy loss + Regularization

- Assume the forward pass has been completed.
- $\implies$ value for every node is known.

Generic Backward Pass: Gradient of current node

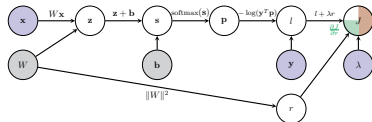
Compute Gradient of node J



$$\frac{\partial J}{\partial J} = 1$$

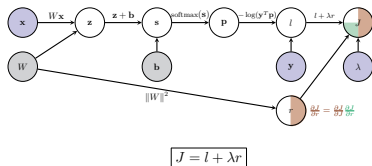
Generic Backward Pass: Order of computations

Compute Jacobian of node J w.r.t. its child r

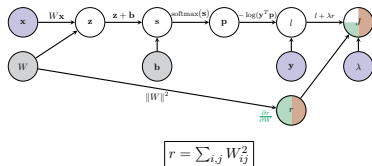


$$J = l + \lambda r$$

$$\frac{\partial J}{\partial r} = \lambda$$

Compute Gradient of node r 

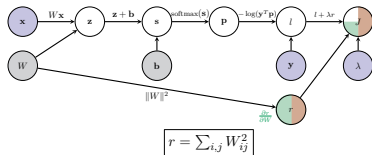
$$\frac{\partial J}{\partial r} = \frac{\partial J}{\partial J} \frac{\partial J}{\partial r} = \lambda$$

Compute Jacobian of node r w.r.t. its child W 

$$\frac{\partial r}{\partial W} = ?$$

Derivative of a scalar w.r.t. a matrix

Generic Backward Pass: Compute Jacobian



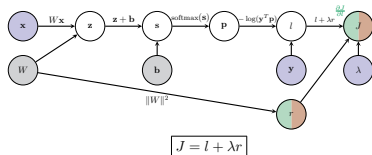
- Jacobian to compute: $\frac{\partial r}{\partial W} = \begin{pmatrix} \frac{\partial r}{\partial W_{11}} & \frac{\partial r}{\partial W_{12}} & \cdots & \frac{\partial r}{\partial W_{1d}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r}{\partial W_{C1}} & \frac{\partial r}{\partial W_{C2}} & \cdots & \frac{\partial r}{\partial W_{Cd}} \end{pmatrix}$

- The individual derivatives: $\frac{\partial r}{\partial W_{ij}} = 2W_{ij}$

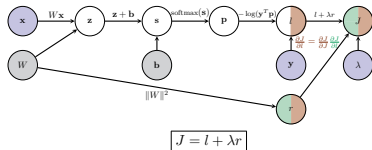
- Putting it together in matrix notation

$$\frac{\partial r}{\partial W} = 2W$$

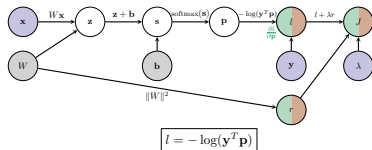
Generic Backward Pass: Order of computations

Compute Jacobian of node J w.r.t. its child l 

$$\frac{\partial J}{\partial l} = 1$$

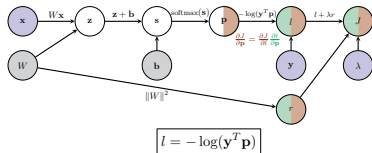
Compute Gradient of node l 

$$\frac{\partial J}{\partial l} = \frac{\partial J}{\partial J} \frac{\partial J}{\partial l} = 1$$

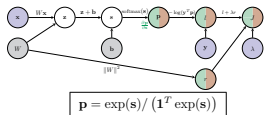
Compute Jacobian of node l w.r.t. its child p 

- The Jacobian we want to compute: $\frac{\partial l}{\partial \mathbf{p}} = \left(\frac{\partial l}{\partial p_1}, \frac{\partial l}{\partial p_2}, \dots, \frac{\partial l}{\partial p_C} \right)$
- The individual derivatives: $\frac{\partial l}{\partial p_i} = -\frac{y_i}{\mathbf{y}^T \mathbf{p}}$ for $i = 1, \dots, C$
- Putting it together:

$$\frac{\partial l}{\partial \mathbf{p}} = -\frac{\mathbf{y}^T}{\mathbf{y}^T \mathbf{p}}$$

Compute Gradient of node p 

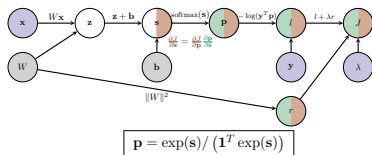
$$\frac{\partial J}{\partial \mathbf{p}} = \frac{\partial J}{\partial l} \frac{\partial l}{\partial \mathbf{p}}$$

Compute Jacobian of node p w.r.t. its child s 

- The Jacobian we need to compute: $\frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \begin{pmatrix} \frac{\partial p_1}{\partial s_1} & \dots & \frac{\partial p_1}{\partial s_C} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_C}{\partial s_1} & \dots & \frac{\partial p_C}{\partial s_C} \end{pmatrix}$
- The individual derivatives:

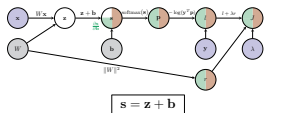
$$\frac{\partial p_i}{\partial s_j} = \begin{cases} p_i(1 - p_i) & \text{if } i = j \\ -p_i p_j & \text{otherwise} \end{cases}$$
- Putting it together in vector notation: $\frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \text{diag}(\mathbf{p}) - \mathbf{p} \mathbf{p}^T$

Compute Gradient of node s



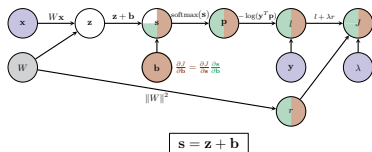
$$\frac{\partial J}{\partial s} = \frac{\partial J}{\partial p} \frac{\partial p}{\partial s}$$

Compute Jacobian of node s w.r.t. its child b



- The Jacobian we need to compute: $\frac{\partial s}{\partial b} = \begin{pmatrix} \frac{\partial s_1}{\partial b_1} & \dots & \frac{\partial s_1}{\partial b_C} \\ \vdots & & \vdots \\ \frac{\partial s_C}{\partial b_1} & \dots & \frac{\partial s_C}{\partial b_C} \end{pmatrix}$
- The individual derivatives: $\frac{\partial s_i}{\partial b_j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$
- In vector notation: $\frac{\partial s}{\partial b} = I_C \leftarrow \text{the identity matrix of size } C \times C$

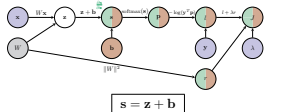
Compute Gradient of node b



gradient needed for mini-batch g.d. training as b parameter of the model $\Rightarrow$

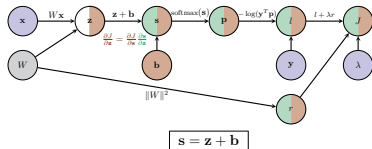
$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial s} \frac{\partial s}{\partial b}$$

Compute Jacobian of node s w.r.t. its child z



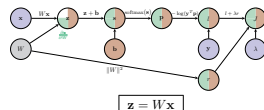
- The Jacobian we need to compute: $\frac{\partial s}{\partial z} = \begin{pmatrix} \frac{\partial s_1}{\partial z_1} & \dots & \frac{\partial s_1}{\partial z_C} \\ \vdots & & \vdots \\ \frac{\partial s_C}{\partial z_1} & \dots & \frac{\partial s_C}{\partial z_C} \end{pmatrix}$
- The individual derivatives: $\frac{\partial s_i}{\partial z_j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$
- In vector notation: $\frac{\partial s}{\partial z} = I_C \leftarrow \text{the identity matrix of size } C \times C$

Compute Gradient of node z



$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial s} \frac{\partial s}{\partial z}$$

Compute Jacobian of node z w.r.t. its child W



- No consistent definition for "Jacobian" of vector w.r.t. matrix.
- Instead re-arrange W ($C \times d$) into a vector $\text{vec}(W)$ ($Cd \times 1$)

$$W = \begin{pmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_C^T \end{pmatrix} \text{ then } \text{vec}(W) = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_C \end{pmatrix}$$

- Then

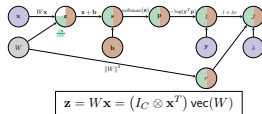
$$z = (I_C \otimes x^T) \text{vec}(W)$$

where $\otimes$ denotes the **Kronecker product** between two matrices.

Generic Backward Pass: Order of computations

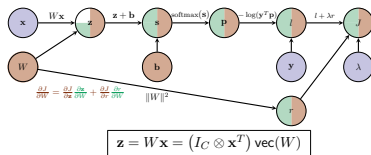
Generic Backward Pass: Order of computations

Compute Jacobian of node z w.r.t. its child W



- Let $v = \text{vec}(W)$. Jacobian to compute: $\frac{\partial z}{\partial v} = \begin{pmatrix} \frac{\partial z_1}{\partial v_1} & \cdots & \frac{\partial z_1}{\partial v_{dC}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_C}{\partial v_1} & \cdots & \frac{\partial z_C}{\partial v_{dC}} \end{pmatrix}$
- The individual derivatives: $\frac{\partial z_j}{\partial v_j} = \begin{cases} x_{j-(i-1)d} & \text{if } (i-1)d + 1 \leq j \leq id \\ 0 & \text{otherwise} \end{cases}$
- In vector notation: $\frac{\partial z}{\partial v} = I_C \otimes x^T$

Compute Gradient of node W

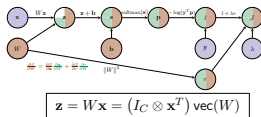


gradient needed for learning $\rightarrow$

$$\frac{\partial J}{\partial \text{vec}(W)} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial \text{vec}(W)} + \frac{\partial J}{\partial r} \frac{\partial r}{\partial \text{vec}(W)}$$

$$= (g_1 x^T \quad g_2 x^T \quad \cdots \quad g_C x^T) + 2\lambda \text{vec}(W)^T$$

if we set $g = \frac{\partial J}{\partial z}$.

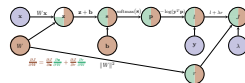
Compute Gradient of node W 

Can convert

$$\frac{\partial J}{\partial \text{vec}(W)} = (g_1 \mathbf{x}^T \quad g_2 \mathbf{x}^T \quad \dots \quad g_C \mathbf{x}^T) + 2\lambda \text{vec}(W)^T$$

(where $\mathbf{g} = \frac{\partial J}{\partial \mathbf{z}}$) from a vector ($1 \times C d$) back to a 2D matrix ($C \times d$):

$$\frac{\partial J}{\partial W} = \begin{pmatrix} g_1 \mathbf{x}^T \\ g_2 \mathbf{x}^T \\ \vdots \\ g_C \mathbf{x}^T \end{pmatrix} + 2\lambda W = \mathbf{g}^T \mathbf{x}^T + 2\lambda W$$



linear scoring function + SOFTMAX + cross-entropy loss + Regularization

$$\mathbf{g} = \frac{\partial J}{\partial \mathbf{l}} = 1$$

$$\mathbf{g} \leftarrow \mathbf{g} \frac{\partial \mathbf{l}}{\partial \mathbf{p}} = \left(-\frac{\mathbf{y}^T}{\mathbf{y}^T \mathbf{p}} \right) \leftarrow \frac{\partial J}{\partial \mathbf{p}}$$

$$\mathbf{g} \leftarrow \mathbf{g} \frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \mathbf{g} \left(\text{diag}(\mathbf{p}) - \mathbf{p} \mathbf{p}^T \right) \leftarrow \frac{\partial J}{\partial \mathbf{s}}$$

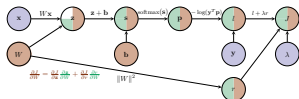
$$\mathbf{g} \leftarrow \mathbf{g} \frac{\partial \mathbf{s}}{\partial \mathbf{z}} = \mathbf{g} \mathbf{I}_C \leftarrow \frac{\partial J}{\partial \mathbf{z}}$$

Then

$$\frac{\partial J}{\partial \mathbf{b}} = \mathbf{g}$$

$$\frac{\partial J}{\partial W} = \mathbf{g}^T \mathbf{x}^T + 2\lambda W$$

Aggregating the Gradient computations



linear scoring function + SOFTMAX + cross-entropy loss + Regularization

1. Let

$$\mathbf{g} = -\frac{\mathbf{y}^T}{\mathbf{y}^T \mathbf{p}} \left(\text{diag}(\mathbf{p}) - \mathbf{p} \mathbf{p}^T \right)$$

2. The gradient of J w.r.t. the bias vector is the $1 \times d$ vector

$$\frac{\partial J}{\partial \mathbf{b}} = \mathbf{g}$$

3. The gradient of J w.r.t. the weight matrix W is the $c \times d$ matrix

$$\frac{\partial J}{\partial W} = \mathbf{g}^T \mathbf{x}^T + 2\lambda W$$

Gradient Computations for a mini-batch

- Have explicitly described the gradient computations for one training example $(\mathbf{x}, y)$.
- In general, want to compute the gradients of the cost function for a mini-batch $\mathcal{D}$.

$$J(\mathcal{D}, W, \mathbf{b}) = L(\mathcal{D}, W, \mathbf{b}) + \lambda \|W\|^2$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l(\mathbf{x}, y, W, \mathbf{b}) + \lambda \|W\|^2$$

- The gradients we need to compute are

$$\frac{\partial J(\mathcal{D}, W, \mathbf{b})}{\partial W} = \frac{\partial L(\mathcal{D}, W, \mathbf{b})}{\partial W} + 2\lambda W = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \frac{\partial l(\mathbf{x}, y, W, \mathbf{b})}{\partial W} + 2\lambda W$$

$$\frac{\partial J(\mathcal{D}, W, \mathbf{b})}{\partial \mathbf{b}} = \frac{\partial L(\mathcal{D}, W, \mathbf{b})}{\partial \mathbf{b}} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \frac{\partial l(\mathbf{x}, y, W, \mathbf{b})}{\partial \mathbf{b}}$$

linear scoring function + **SOFTMAX** + cross-entropy loss + Regularization

- Compute gradients of l w.r.t. W , $\mathbf{b}$ for each $(\mathbf{x}, y) \in \mathcal{D}^{(t)}$:

- Set all entries in $\frac{\partial L}{\partial \mathbf{b}}$ and $\frac{\partial L}{\partial W}$ to zero.
- for $(\mathbf{x}, y) \in \mathcal{D}^{(t)}$

1. Let

$$\mathbf{g} = -\frac{\mathbf{y}^T}{\mathbf{y}^T \mathbf{p}} \left(\text{diag}(\mathbf{p}) - \mathbf{p} \mathbf{p}^T \right)$$

2. Add gradient of l w.r.t. $\mathbf{b}$ computed at $(\mathbf{x}, y)$

$$\frac{\partial L}{\partial \mathbf{b}} += \mathbf{g}$$

3. Add gradient of l w.r.t. W computed at $(\mathbf{x}, y)$

$$\frac{\partial L}{\partial W} += \mathbf{g}^T \mathbf{x}^T$$

- Divide by the number of entries in $\mathcal{D}^{(t)}$:

$$\frac{\partial L}{\partial W} /= |\mathcal{D}^{(t)}|, \quad \frac{\partial L}{\partial \mathbf{b}} /= |\mathcal{D}^{(t)}|$$

- Add the gradient for the regularization term

$$\frac{\partial J}{\partial W} = \frac{\partial L}{\partial W} + 2\lambda W, \quad \frac{\partial J}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{b}}$$