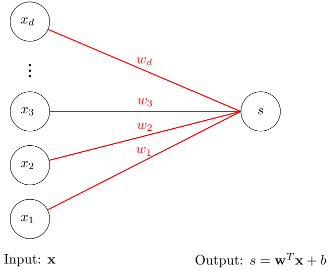
Lecture 3 - Back Propagation

DD2424, Josephine Sullivan

March 21, 2025

Classification functions we have encountered so far

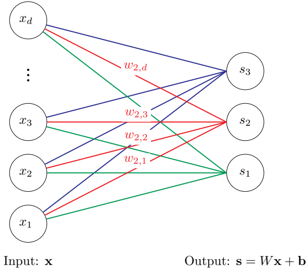
Linear with 1 output



Final decision:

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

Linear with multiple outputs

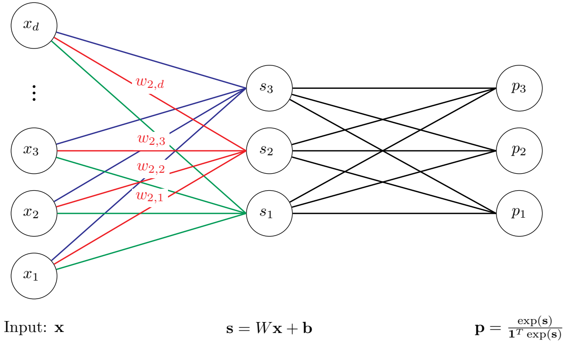


Final decision:

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

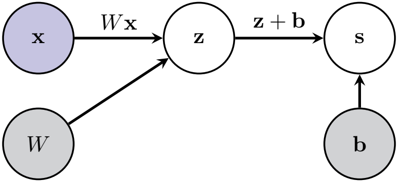
Classification functions we have encountered so far

Linear with multiple probabilistic outputs



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

Computational graph of the multiple linear function

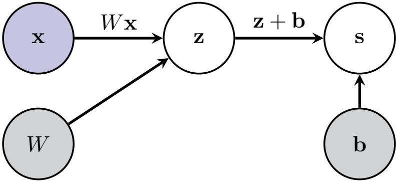


The computational graph:

- 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.

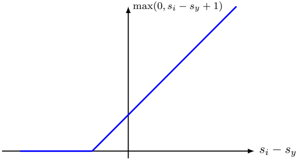
How do we learn $W, \mathbf{b}$?



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

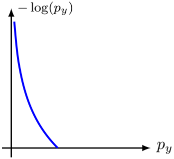
Quality measures a.k.a. loss functions we've encountered

Multi-class SVM loss



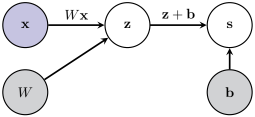
$$l_{\mathsf{SVM}}(\mathbf{s}, y) = \sum_{\substack{j=1 \\ j \neq y}}^{C} \max(0, s_j - s_y + 1)$$

Cross-entropy loss

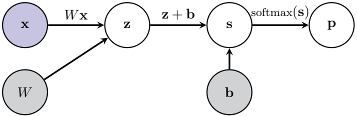


$$l_{\mathsf{cross}}(\mathbf{p}, y) = -\log(p_y)$$

Classification function



Classification function



- Let $\mathbf{y}$ and $\mathbf{p}$ both be vectors of size $C \times 1$.

- Both $\mathbf{y}$ and $\mathbf{p}$ represent a discrete pdf.

- **Cross-entropy** between these two pdf vectors is defined as

$$-\mathbf{y}^T \log(\mathbf{p})$$

- In ML commonly $\mathbf{y}$ is a one-hot encoding vector that is

$$y_i = \begin{cases} 0 & \text{if } i \neq \text{ground truth class} \\ 1 & \text{if } i \text{ is the ground truth class} \end{cases}$$

In this case and if y is the ground truth class then

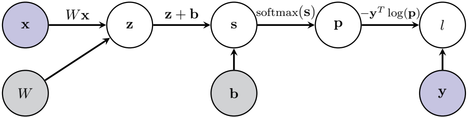
$$-\mathbf{y}^T \log(\mathbf{p}) = -\log(p_y)$$

Note on cross-entropy

- Let $\mathbf{y}$ and $\mathbf{p}$ both be vectors of size $C \times 1$.
- Both $\mathbf{y}$ and $\mathbf{p}$ represent a discrete pdf.
- **Cross-entropy** between these two pdf vectors is defined as
$$-\mathbf{y}^T \log(\mathbf{p})$$
- In ML commonly $\mathbf{y}$ is a one-hot encoding vector that is
$$y_i = \begin{cases} 0 & \text{if } i \neq \text{ground truth class} \\ 1 & \text{if } i \text{ is the ground truth class} \end{cases}$$
In this case and if y is the ground truth class then
$$-\mathbf{y}^T \log(\mathbf{p}) = -\log(p_y)$$

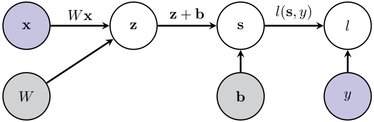
Computational graph of the complete loss function

- Linear scoring function + SoftMax + cross-entropy loss

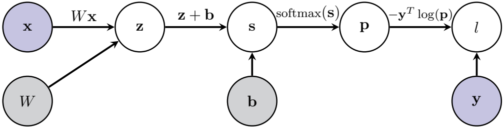


where $\mathbf{y}$ is the 1-hot response vector induced by the label y.

- Linear scoring function + multi-class SVM loss

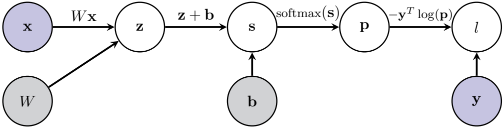


How do we learn $W, \mathbf{b}$?



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

How do we learn $W, \mathbf{b}$?



- Let l be the loss function defined by the computational graph.
- Find $W, \mathbf{b}$ by optimizing

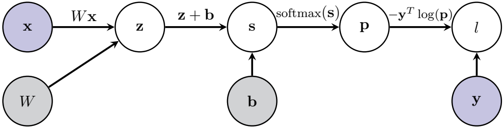
$$\arg\min_{W,\mathbf{b}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} l(\mathbf{x}, y, W, \mathbf{b})$$

- Solve using a variant of **mini-batch gradient descent**

$\Longrightarrow$ need to efficiently compute the gradient vectors

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

How do we compute these gradients?



- Let l be the complete loss function defined by the computational graph.
- How do we efficiently compute the gradient vectors

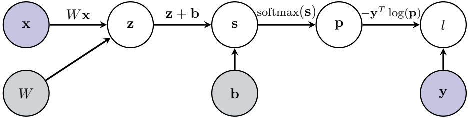
$$\nabla_W l(\mathbf{x}, y, W, \mathbf{b})|_{(\mathbf{x},y)\in\mathcal{D}} \quad \text{and} \quad \nabla_{\mathbf{b}} l(\mathbf{x}, y, W, \mathbf{b})|_{(\mathbf{x},y)\in\mathcal{D}}?$$

- Answer: Back Propagation

Today's lecture: Gradient computations in neural networks

- For our learning approach need to be able to compute gradients efficiently.

- **BackProp algorithm**: efficient gradient calculations for many of our classifiers and loss functions.



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

- Example: the composition of functions

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

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

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

Differentiation of the composition of functions

- Have two functions $g : \mathbb{R} \to \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$.
- Define $h : \mathbb{R} \to \mathbb{R}$ 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**

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

Example of the Chain Rule in action

- Have

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

- One composition of these two functions is

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

- According to the **chain rule**

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

The composition of n functions

- Have functions $f_1, \ldots, f_n : \mathbb{R} \to \mathbb{R}$

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

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

- Can we compute the derivative

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

- Yes recursively apply the CHAIN RULE

The composition of n functions

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

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

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

- Can we compute the derivative

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

- Yes recursively apply the CHAIN RULE

Chain Rule for the composition of n functions

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

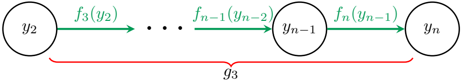
Computational graph for h

$$\underbrace{x \xrightarrow{f_1(x)} y_1 \xrightarrow{f_2(y_1)} y_2 \xrightarrow{f_3(y_2)} \cdots \xrightarrow{f_{n-1}(y_{n-2})} y_{n-1} \xrightarrow{f_n(y_{n-1})} y_n}_{h}$$

Chain Rule for the composition of n functions

Define

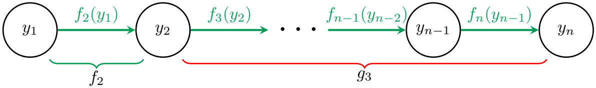
$$\textcolor{red}{g_j}(y_{j-1}) = \left(\textcolor{green}{f_n} \circ \textcolor{green}{f_{n-1}} \circ \cdots \circ \textcolor{green}{f_j}\right)(y_{j-1})$$



Chain Rule for the composition of n functions

Then for $j = 1, \ldots, n-1$ (and $g_1 = h$, $g_n = f_n$)

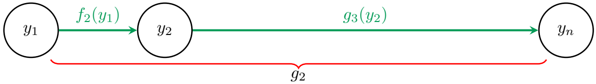
$$g_{j-1} = g_j \circ f_{j-1}$$



Chain Rule for the composition of n functions

Then for $j = 1, \ldots, n-1$ (and $g_1 = h$, $g_n = f_n$)

$$g_{j-1} = g_j \circ f_{j-1}$$



The Chain Rule for the composition of n functions

$$\underbrace{x \xrightarrow{f_1(x)} y_1 \xrightarrow{f_2(y_1)} y_2 \xrightarrow{f_3(y_2)} \cdots \xrightarrow{f_{n-1}(y_{n-2})} y_{n-1} \xrightarrow{f_n(y_{n-1})} y_n}_{h}$$

Summary so far:

- Have function h defined as the composition: (assuming $y_0 = x$)

$$y_n = h(y_0) = (f_n \circ f_{n-1} \circ \cdots \circ f_1)(y_0)$$

- Define **intermediary outputs** as

$$y_j = (f_j \circ f_{j-1} \circ \cdots \circ f_1)(y_0) = f_j(y_{j-1})$$

- Define h in terms of g_j's applied to **intermediary outputs**:

$$\begin{aligned} h(y_0) = y_n &= g_j(y_{j-1}) \\ &= (f_n \circ f_{n-1} \circ \cdots \circ f_{j+1} \circ f_j)(y_{j-1}) \\ &= (g_{j+1} \circ f_j)(y_{j-1}) \end{aligned}$$

The Chain Rule for the composition of n functions

Can recursively apply the **Chain Rule** to compute derivative of h w.r.t. x:

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

Summary: Chain Rule for a composition of n functions

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

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

- Then

$$\frac{dh(x)}{dx} = \frac{df_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \cdots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx}$$

$$= \frac{dy_n}{dy_{n-1}} \; \frac{dy_{n-1}}{dy_{n-2}} \; \cdots \frac{dy_2}{dy_1} \; \frac{dy_1}{dx}$$

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

- **Prev slide repeatedly used**: for $j = n-1, n-2, \ldots, 0$

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

Summary: Chain Rule for a composition of n functions

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

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

- Then

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

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

- **Prev slide repeatedly used**: for $j = n-1, n-2, \ldots, 0$

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

Compute gradient of h at a point x^*

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

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

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

- Can either compute it with
  - Forward mode auto-diff **or**
  - Reverse mode auto-diff a.k.a. **Back-Propagation** (algorithm consisting of a Forward and Backward pass).

Forward mode auto-diff to compute

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

where

$$\frac{dh(x)}{dx} = \frac{df_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \cdots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx}$$

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

Path graph: Forward mode auto-diff

$$x \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_{n-1}} y_{n-1} \xrightarrow{f_n} y_n$$

- Initialize

$$g = \left.\frac{df_1(x)}{dx}\right|_{x=x^*} \quad \textbf{and} \quad y^* = f_1(x^*)$$

- Then
  for $j = 2, 3, \ldots, n$ (i.e. compute local gradient, accumulate & compute local fn)

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

Note: $g = \left.\frac{dy_j}{dx}\right|_{x=x^*}$ at end of each iteration

At end of for-loop g corresponds to $\left.\frac{dh(x)}{dx}\right|_{x=x^*}$

Reverse mode auto-diff (aka back-propagation) to compute

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

where

$$\frac{dh(x)}{dx} = \frac{df_n(y_{n-1})}{dy_{n-1}} \frac{df_{n-1}(y_{n-2})}{dy_{n-2}} \cdots \frac{df_2(y_1)}{dy_1} \frac{df_1(x)}{dx}$$

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

Path graph: Back-Propagation - Forward Pass

$$x \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_{n-1}} y_{n-1} \xrightarrow{f_n} y_n$$

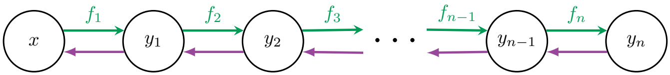
Evaluate $h(x^*)$ and keep track of the intermediary results

- Compute $y_1^* = f_1(x^*)$.

- Then
  for $j = 2, 3, \ldots, n$
  $$y_j^* = f_j(y_{j-1}^*)$$

- Keep a record of $y_1^*, \ldots, y_n^*$.

Path graph: Back-Propagation - Backward Pass

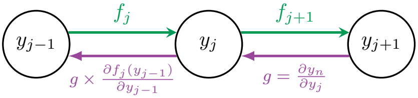


Compute local derivatves of f_j and aggregate:

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

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

Note: $g = \left. \frac{dy_n}{dy_{j-1}} \right|_{y_{j-1} = y^*_{j-1}}$ at end of each iteration



- Then $\left. \frac{dh(x)}{dx} \right|_{x = x^*} = g \times \left. \frac{d f_1(x)}{dx} \right|_{x = x^*}$

Forward or Reverse mode for deep networks?

- For this simple 1d example **forward mode** better than **reverse mode** as
  - needs just one pass and
  - do need to store the intermediary $y_1^*, y_2^*, \ldots, y_n^*$.

- But for deep learning **reverse mode** is the method predominantly used.

  More efficient as typically $\dim(\text{input}) \gg \dim(\text{output})$. (this comment will make more sense as the lecture progresses)

Forward or Reverse mode for deep networks?

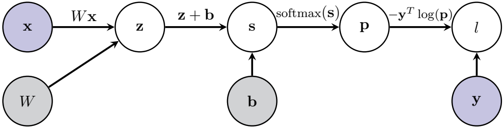
- For this simple 1d example **forward mode** better than **reverse mode** as
  - needs just one pass and
  - do need to store the intermediary $y_1^*, y_2^*, \ldots, y_n^*$.

- But for deep learning **reverse mode** is the method predominantly used.

  More efficient as typically $\dim(\text{input}) \gg \dim(\text{output})$. (this comment will make more sense as the lecture progresses)

But are we ready to apply the chain rule to our loss?

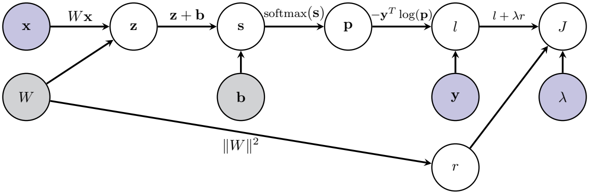
Problem 1: But what if I don't have a path graph?



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

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

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

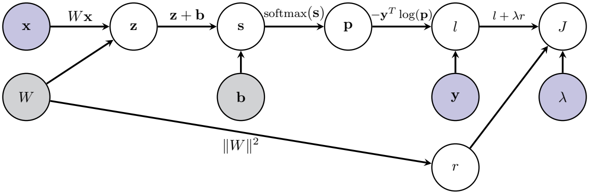


- This computational graph is **not a path graph**.
- Some nodes have **multiple parents** and others **multiple children**.
- The function represented by graph is

$$J(\mathbf{x}, \mathbf{y}, W, \mathbf{b}, \lambda) = -\mathbf{y}^T \log(\text{SoftMax}(W\mathbf{x} + \mathbf{b})) + \lambda \sum_{i,j} W_{i,j}^2$$

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

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

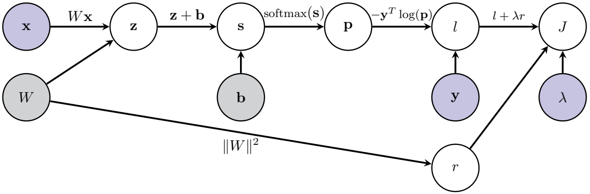


- This computational graph is **not a path graph**.
- Some nodes have **multiple parents** and others **multiple children**.
- The function represented by graph is

$$J(\mathbf{x}, \mathbf{y}, W, \mathbf{b}, \lambda) = -\mathbf{y}^T \log(\text{SoftMax}(W\mathbf{x} + \mathbf{b})) + \lambda \sum_{i,j} W_{i,j}^2$$

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

Problem 2: Don't have scalar inputs and outputs

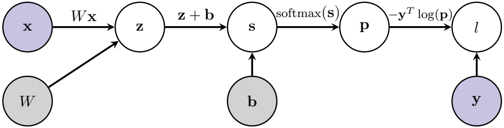


- The function represented by graph:

$$J(\mathbf{x}, \mathbf{y}, W, \mathbf{b}, \lambda) = -\mathbf{y}^T \log(\mathsf{SoftMax}(W\mathbf{x} + \mathbf{b})) + \lambda \sum_{i,j} W_{i,j}^2$$

- Nearly all of the inputs and intermediary outputs are **vectors** or **matrices**.

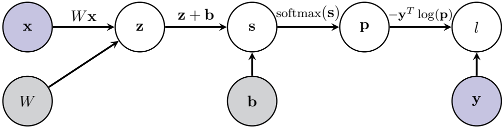
- How are the derivatives defined in this case?



- Back-propagation when the computational graph is **not a path graph**.

- Derivative computations when the inputs and outputs are not scalars.

- Will address these issues now. First the derivatives of vectors.



- Back-propagation when the computational graph is **not a path graph**.

- 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**

Chain Rule for vector input and output

- 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?

Brief technical interlude: Layout Convention for Matrix Calculus

Matrix Calculus 1

- Let $\mathbf{y}$ be a vector of length m, $\mathbf{x}$ be a vector of length n

- There are two conventions for writing $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$
  - **Numerator layout** (aka Jacobian formulation) $(m \times n)$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

  - **Denominator layout** (aka Hessian formulation) $(n \times m)$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

What about the gradients?

- If you chose **numerator layout** for $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ then the gradient $\frac{\partial y}{\partial \mathbf{x}}$ should be a row vector.
- If you chose **denominator layout** for $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ then the gradient $\frac{\partial y}{\partial \mathbf{x}}$ should be a column vector.

We mainly use "numerator layout" but may not be entirely consistent across all lectures and assignment instructions.

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 $h = f \circ g$ $\left(g : \mathbb{R}^d \rightarrow \mathbb{R}^m \text{ and } f : \mathbb{R}^m \rightarrow \mathbb{R}^c \right)$ 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 slly 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 $\mathbf{x} \in \mathbb{R}^d$, $f : \mathbb{R}^d \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}$

$$\begin{aligned} \mathbf{z} &= f(\mathbf{x}) \\ s &= g(\mathbf{z}) \end{aligned}$$

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

$$\frac{\partial s}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial s}{\partial x_1} & \cdots & \frac{\partial s}{\partial x_d} \end{pmatrix} \leftarrow \text{for consistency gradient def corresponds to Jacobian def.}$$

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

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

Technical Interlude

Forward & Reverse Mode derivative calculations

Chain Rule for vector input and scalar output

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

$$s = g(f(f(f(\mathbf{x}))))$$

Apply the chain rule recursively to get:

$$\frac{\partial s}{\partial \mathbf{x}} = \underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m} \underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m} \underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m} \underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d}$$

where $\mathbf{z}_1 = f(\mathbf{x}), \mathbf{z}_2 = f(\mathbf{z}_1)$ and $\mathbf{z}_3 = f(\mathbf{z}_2)$.

In **forward mode** calculate the gradient at $\mathbf{x}^*$ ordering the matrix multiplications as

$$\frac{\partial s}{\partial \mathbf{x}} = \left(\underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m} \left(\underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m} \left(\underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m} \underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d} \right) \right) \right)$$

Reverse Mode calculation of the gradient

In **reverse mode** calculate the gradient at $\mathbf{x}^*$ ordering the matrix multiplications as

$$\frac{\partial s}{\partial \mathbf{x}} = \left(\left(\left(\underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m} \underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m} \right) \underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m} \right) \underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d} \right)$$

Flops for Forward Vs Reverse Mode

What is computational complexity of computing $\frac{\partial s}{\partial \mathbf{x}}$??

$$\left(\underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m}\left(\underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m}\left(\underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m}\underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d}\right)\right)\right) \quad \text{Vs} \quad \left(\left(\left(\underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m}\underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m}\right)\underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m}\right)\underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d}\right)$$

Forward mode **Reverse Mode**

Flops for Forward Vs Reverse Mode

What is computational complexity of computing $\frac{\partial s}{\partial \mathbf{x}}$??

$$\left(\underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m}\left(\underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m}\left(\underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m}\underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d}\right)\right)\right) \quad \text{Vs} \quad \left(\left(\left(\underbrace{\frac{\partial s}{\partial \mathbf{z}_3}}_{1\times m}\underbrace{\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2}}_{m\times m}\right)\underbrace{\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}}_{m\times m}\right)\underbrace{\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}}_{m\times d}\right)$$

Forward mode $\propto 2m^2d + md$

Reverse Mode $\propto 2m^2 + md$

If d large $\implies$ **Forward mode** much more expensive than **Reverse mode**

End of this interlude

Two intermediary vector inputs and scalar output

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

$$\mathbf{z}_1 = f_1(\mathbf{x}) \qquad \mathbf{z}_2 = f_2(\mathbf{x})$$

$$s = g(\mathbf{z}_1, \mathbf{z}_2)$$

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

$$\frac{\partial s}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial s}{\partial x_1} & \cdots & \frac{\partial s}{\partial x_d} \end{pmatrix}$$

is given by

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

More generally

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

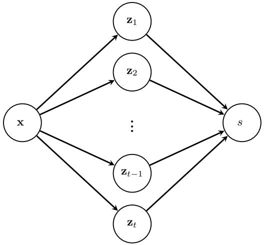
$$\begin{aligned} \mathbf{z}_i &= f_i(\mathbf{x}), \qquad \text{for } i = 1, \dots, t \\ s &= g(\mathbf{z}_1, \dots, \mathbf{z}_t) \end{aligned}$$

- Consequence of the **Chain Rule**

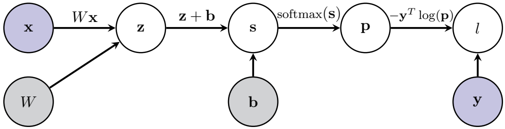
$$\frac{\partial s}{\partial \mathbf{x}} = \sum_{i=1}^{t} \frac{\partial s}{\partial \mathbf{z}_i} \frac{\partial \mathbf{z}_i}{\partial \mathbf{x}}$$

- Computational graph interpretation: Let $\mathcal{C}_{\mathbf{x}}$ be children nodes of $\mathbf{x}$ then

$$\frac{\partial s}{\partial \mathbf{x}} = \sum_{\mathbf{z} \in \mathcal{C}_{\mathbf{x}}} \frac{\partial s}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$$



Issues we need to sort out



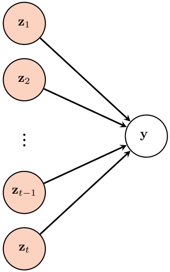
- Back-propagation when the computational graph is **not a path graph**.
- Derivative computations when the inputs and outputs are not scalars. ✓
- Will now describe Back-prop for non-path graphs.

Back-propagation for non-path computational graphs

Results that we need

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

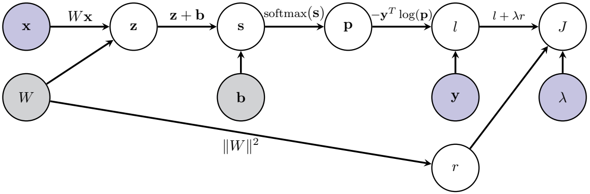
$$V_{\mathcal{P}_{\mathbf{y}}} = \{\mathbf{z}.\text{value} \mid \mathbf{z} \in \mathcal{P}_{\mathbf{y}}\}$$



- Given $V_{\mathcal{P}_{\mathbf{y}}}$ can apply the function $f_{\mathbf{z}}$

$$\mathbf{y}.\text{value} = f_{\mathbf{y}}(V_{\mathcal{P}_{\mathbf{y}}})$$

Results that we need but already know



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

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

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

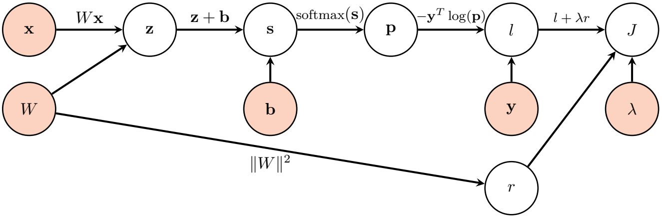
$$\frac{\partial J}{\partial \mathbf{c}} = \sum_{\mathbf{u} \in \mathcal{C}_{\mathbf{c}}} \frac{\partial J}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{c}}$$

Pseudo-Code for the Generic Forward Pass

```
procedure EvaulateGraphFn(G)                         ▷ G is the computational graph
    S = GetStartNodes(G)    ▷ a start node has no parent and its value is already set
    for s ∈ S do
        ComputeBranch(s, G)
    end for
end procedure

procedure ComputeBranch(s, G)                        ▷ recursive fn evaluating nodes
    C_s = GetChildren(s, G)
    for each n ∈ C_s do                        ▷ Try to evaluate each children node
        if !n.computed then                      ▷ Unless child is already computed
            P_n = GetParents(n, G)
            if CheckAllNodesComputed(P_n) then ▷ Or not all parents of children are computed
                f_n = GetNodeFn(n)
                n.value = f_n(P_n)
                n.computed = true
                ComputeBranch(n, G)
            end if
        end if
    end for
end procedure
```

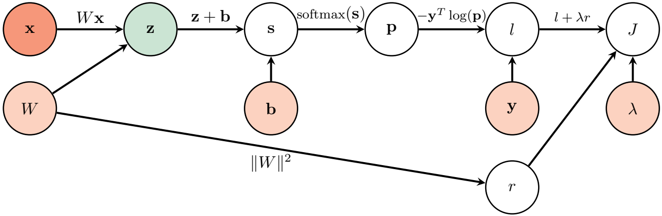
Identify Start Nodes



procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
  $\mathcal{S}$ = GetStartNodes(G)
  for $\mathbf{s} \in \mathcal{S}$ **do**
    ComputeBranch($\mathbf{s}$, G)
  end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
  $\mathcal{C}_{\mathbf{s}}$ = GetChildren($\mathbf{s}$, G)
  for each $\mathbf{n} \in \mathcal{C}_{\mathbf{s}}$ **do**
    if !$\mathbf{n}$.computed **then**
      $\mathcal{P}_{\mathbf{n}}$ = GetParents($\mathbf{n}$, G)
      if CheckAllNodesComputed($\mathcal{P}_{\mathbf{n}}$) **then**
        $f_{\mathbf{n}}$ = GetNodeFn($\mathbf{n}$)
        $\mathbf{n}$.value = $f_{\mathbf{n}}(\mathcal{P}_{\mathbf{n}})$
        $\mathbf{n}$.computed = true
        ComputeBranch($\mathbf{n}$, G)
      end if
    end if
  end for
end procedure

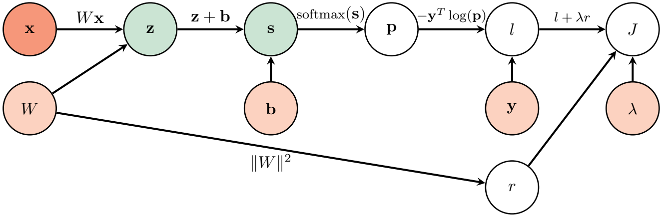
Order in which nodes are evaluated



procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
  $\mathcal{S}$ = GetStartNodes(G)
  for $\mathbf{s} \in \mathcal{S}$ **do**
    ComputeBranch($\mathbf{s}$, G)
  end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  for each $\mathbf{n} \in \mathcal{C}_\mathbf{s}$ **do**
    if !$\mathbf{n}$.computed **then**
      $\mathcal{P}_\mathbf{n}$ = GetParents($\mathbf{n}$, G)
      if CheckAllNodesComputed($\mathcal{P}_\mathbf{n}$) **then**
        $f_\mathbf{n}$ = GetNodeFn($\mathbf{n}$)
        $\mathbf{n}$.value = $f_\mathbf{n}(\mathcal{P}_\mathbf{n})$
        $\mathbf{n}$.computed = true
        ComputeBranch($\mathbf{n}$, G)
      end if
    end if
  end for
end procedure

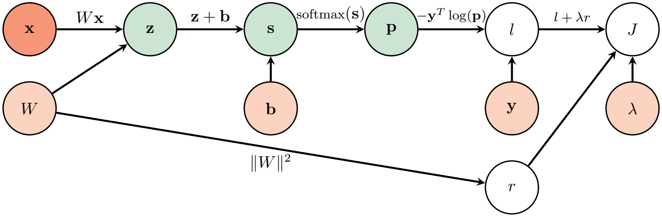
Order in which nodes are evaluated



procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
    $\mathcal{S}$ = GetStartNodes(G)
    for $\mathbf{s} \in \mathcal{S}$ **do**
        ComputeBranch($\mathbf{s}$, G)
    end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
    $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
    for each $\mathbf{n} \in \mathcal{C}_\mathbf{s}$ **do**
        if !$\mathbf{n}$.computed **then**
            $\mathcal{P}_\mathbf{n}$ = GetParents($\mathbf{n}$, G)
            if CheckAllNodesComputed($\mathcal{P}_\mathbf{n}$) **then**
                $f_\mathbf{n}$ = GetNodeFn($\mathbf{n}$)
                $\mathbf{n}$.value = $f_\mathbf{n}(\mathcal{P}_\mathbf{n})$
                $\mathbf{n}$.computed = true
                ComputeBranch($\mathbf{n}$, G)
            end if
        end if
    end for
end procedure

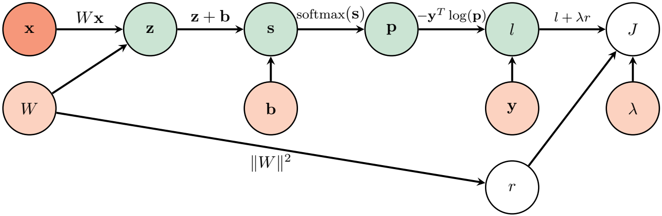
Order in which nodes are evaluated



procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
    $\mathcal{S}$ = GetStartNodes(G)
    for $\mathbf{s} \in \mathcal{S}$ **do**
        ComputeBranch($\mathbf{s}$, G)
    end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
    $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
    for each $\mathbf{n} \in \mathcal{C}_\mathbf{s}$ **do**
        if !$\mathbf{n}$.computed **then**
            $\mathcal{P}_\mathbf{n}$ = GetParents($\mathbf{n}$, G)
            if CheckAllNodesComputed($\mathcal{P}_\mathbf{n}$) **then**
                $f_\mathbf{n}$ = GetNodeFn($\mathbf{n}$)
                $\mathbf{n}$.value = $f_\mathbf{n}(\mathcal{P}_\mathbf{n})$
                $\mathbf{n}$.computed = true
                ComputeBranch($\mathbf{n}$, G)
            end if
        end if
    end for
end procedure

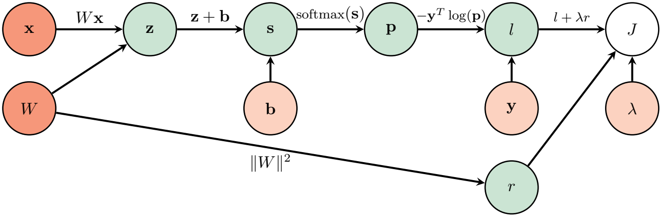
Order in which nodes are evaluated



procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
  $\mathcal{S}$ = GetStartNodes(G)
  for $\mathbf{s} \in \mathcal{S}$ **do**
    ComputeBranch($\mathbf{s}$, G)
  end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
  $\mathcal{C}_{\mathbf{s}}$ = GetChildren($\mathbf{s}$, G)
  for each $\mathbf{n} \in \mathcal{C}_{\mathbf{s}}$ **do**
    if !$\mathbf{n}$.computed **then**
      $\mathcal{P}_{\mathbf{n}}$ = GetParents($\mathbf{n}$, G)
      if CheckAllNodesComputed($\mathcal{P}_{\mathbf{n}}$) **then**
        $f_{\mathbf{n}}$ = GetNodeFn($\mathbf{n}$)
        $\mathbf{n}$.value = $f_{\mathbf{n}}(\mathcal{P}_{\mathbf{n}})$
        $\mathbf{n}$.computed = true
        ComputeBranch($\mathbf{n}$, G)
      end if
    end if
  end for
end procedure

Order in which nodes are evaluated

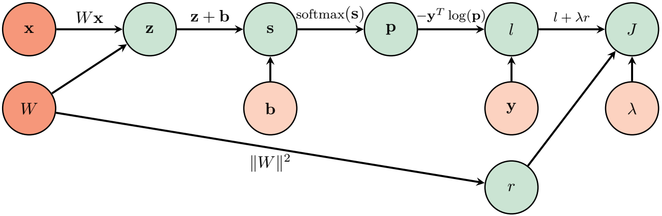


procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
  $\mathcal{S}$ = GetStartNodes(G)
  for $\mathbf{s} \in \mathcal{S}$ **do**
    ComputeBranch($\mathbf{s}$, G)
  end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  for each $\mathbf{n} \in \mathcal{C}_\mathbf{s}$ **do**
    if !$\mathbf{n}$.computed **then**
      $\mathcal{P}_\mathbf{n}$ = GetParents($\mathbf{n}$, G)
      if CheckAllNodesComputed($\mathcal{P}_\mathbf{n}$) **then**
        $f_\mathbf{n}$ = GetNodeFn($\mathbf{n}$)
        $\mathbf{n}$.value = $f_\mathbf{n}(\mathcal{P}_\mathbf{n})$
        $\mathbf{n}$.computed = true
        ComputeBranch($\mathbf{n}$, G)
      end if
    end if
  end for
end procedure

Generic Forward Pass

Order in which nodes are evaluated



procedure EVAULATEGRAPHFN(G) ▷ G is the computational graph
    $\mathcal{S}$ = GetStartNodes(G)
    for $\mathbf{s} \in \mathcal{S}$ **do**
        ComputeBranch($\mathbf{s}$, G)
    end for
end procedure

procedure COMPUTEBRANCH($\mathbf{s}$, G)
    $\mathcal{C}_{\mathbf{s}}$ = GetChildren($\mathbf{s}$, G)
    for each $\mathbf{n} \in \mathcal{C}_{\mathbf{s}}$ **do**
        if !$\mathbf{n}$.computed **then**
            $\mathcal{P}_{\mathbf{n}}$ = GetParents($\mathbf{n}$, G)
            if CheckAllNodesComputed($\mathcal{P}_{\mathbf{n}}$) **then**
                $f_{\mathbf{n}}$ = GetNodeFn($\mathbf{n}$)
                $\mathbf{n}$.value = $f_{\mathbf{n}}(\mathcal{P}_{\mathbf{n}})$
                $\mathbf{n}$.computed = true
                ComputeBranch($\mathbf{n}$, G)
            end if
        end if
    end for
end procedure

Pseudo-Code for the Generic Backward Pass

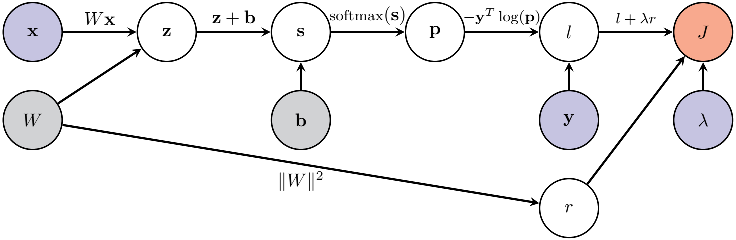
```
procedure PerformBackPass(G)
    J = GetResultNode(G)                                  ▷ node with the value of cost function
    BackOp(J, G)                                          ▷ Start the Backward-pass
end procedure

procedure BackOp(s, G)
    C_s = GetChildren(s, G)
    if C_s = ∅ then                                       ▷ At the result node
        s.Grad = 1
    end if
    if AllGradientsComputed(C_s) then                     ▷ Have computed all ∂J/∂c where c ∈ C_s
        s.Grad = 0
        for each c ∈ C_s do
            s.Grad += c.Grad * c.s.Jacobian               ▷ ∂J/∂s += (∂J/∂c)(∂c/∂s)
        end for
        s.GradComputed = true
    end if
    for each p ∈ P_s do                                   ▷ Compute the Jacobian of f_s w.r.t. each parent node
        s.p.Jacobian = ∂f_p(P_s)/∂p                       ▷ ∂f_s(P_s)/∂p = ∂s/∂p
        BackOp(p, G)
    end for
end procedure
```

Where: J is the result node; $\mathcal{C}_{\mathbf{s}}$ the children of $\mathbf{s}$; $\mathcal{P}_{\mathbf{s}}$ the parents of $\mathbf{s}$. Comments: $\frac{\partial J}{\partial \mathbf{s}} \mathrel{+}= \frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$; s.p.Jacobian $= \frac{\partial f_{\mathbf{p}}(\mathcal{P}_{\mathbf{s}})}{\partial \mathbf{p}}$; $\frac{\partial f_{\mathbf{s}}(\mathcal{P}_{\mathbf{s}})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$

Generic Backward Pass: Order of computations

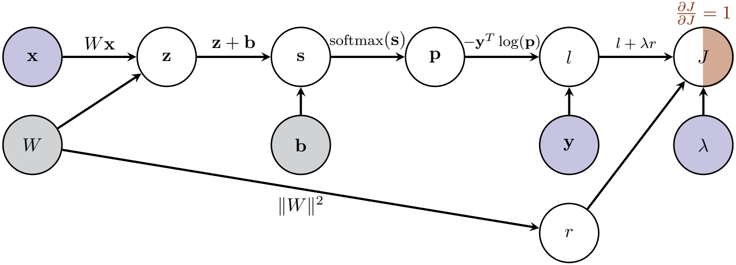
Identify Result Node



procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}} \mathrel{+}= \frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}.\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

Compute Gradient of current node

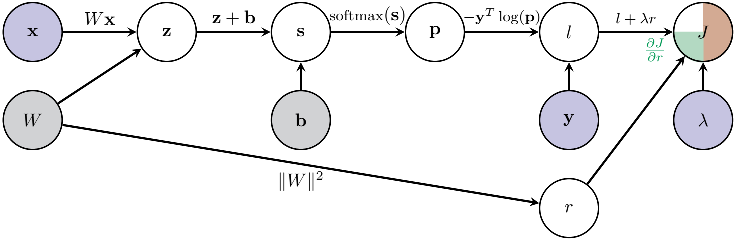


procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      s.Grad = **0**
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        s.Grad += **c**.Grad * **c**.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}} \mathrel{+}= \frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    s.**p**.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

Generic Backward Pass: Order of computations

Compute Jacobian of current node w.r.t. one parent

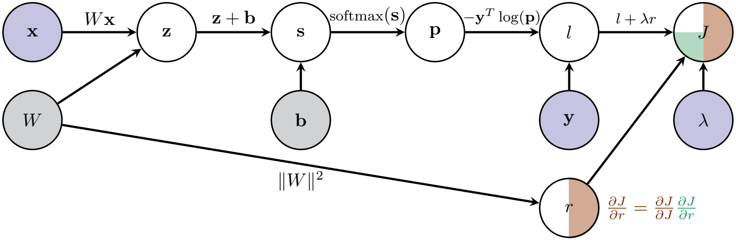


procedure PERFORMBACKPASS(G)
    J = GetResultNode(G) ▷ node with the value of cost function
    BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BACKOP($\mathbf{s}$, G)
    $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
    if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
        $\mathbf{s}$.Grad = 1
    else
        if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
            $\mathbf{s}$.Grad = $\mathbf{0}$
            for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
                $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}.\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
            end for
            $\mathbf{s}$.GradComputed = true
        end if
    end if
    for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
        $\mathbf{s}.\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
        BackOp($\mathbf{p}$, G)
    end for
end procedure

Generic Backward Pass: Order of computations

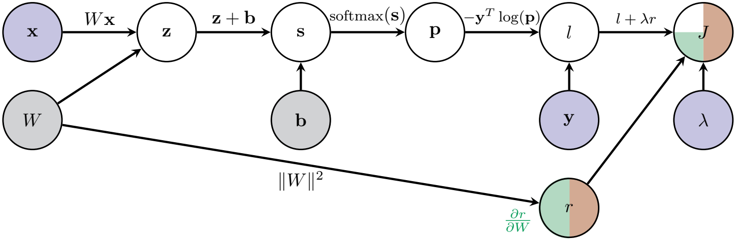
Compute Gradient of current node



procedure PERFORMBACKPASS(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BACKOP($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    $\mathbf{s}$.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      $\mathbf{s}$.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.$\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      $\mathbf{s}$.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}$.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

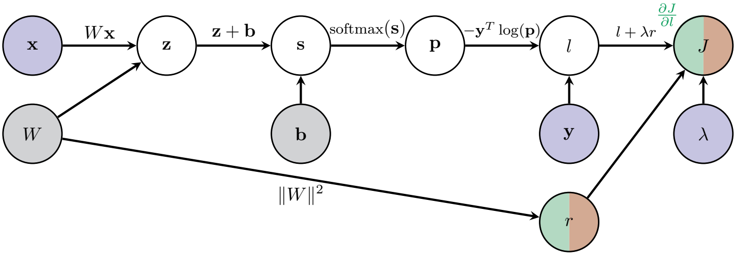
Compute Jacobian of current node w.r.t. one parent



procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    $\mathbf{s}$.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      $\mathbf{s}$.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.$\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      $\mathbf{s}$.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}$.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

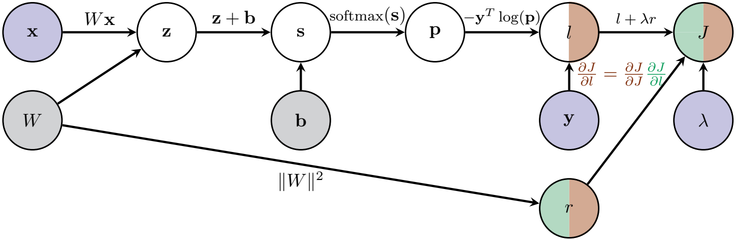
Compute Jacobian of current node w.r.t. one parent



procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    $\mathbf{s}$.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      $\mathbf{s}$.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.$\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      $\mathbf{s}$.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}$.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

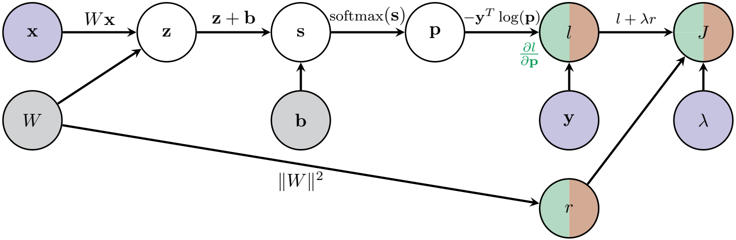
Compute Gradient of current node



procedure PERFORMBACKPASS(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BACKOP($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}.\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

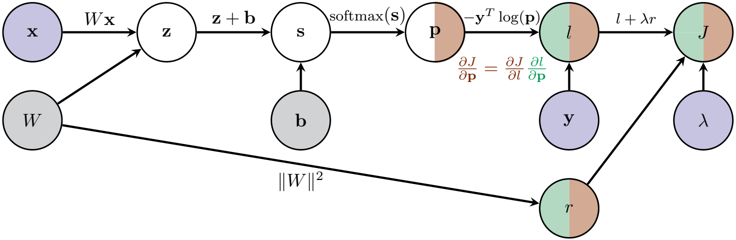
Compute Jacobian of current node w.r.t. one parent



procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}.\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

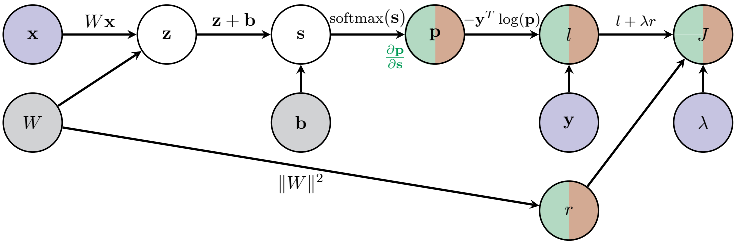
Compute Gradient of current node



procedure PERFORMBACKPASS(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BACKOP($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    $\mathbf{s}$.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      $\mathbf{s}$.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.$\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      $\mathbf{s}$.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}$.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

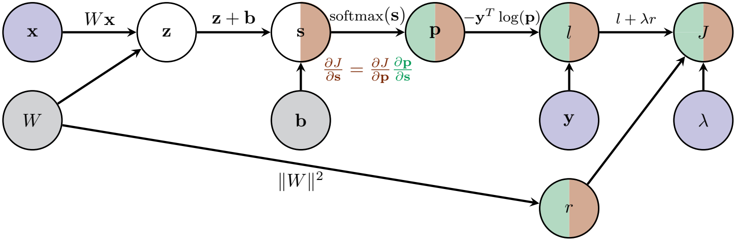
Compute Jacobian of current node w.r.t. one parent



procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}.\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

Compute Gradient of current node

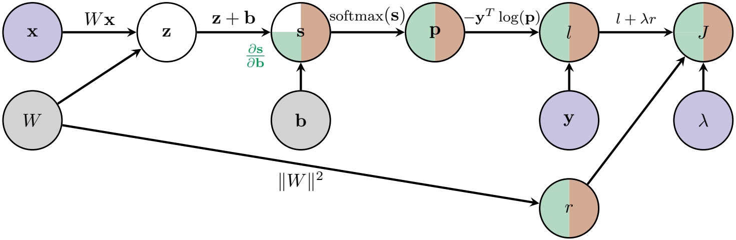


procedure PERFORMBACKPASS(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BACKOP($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    s.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

Generic Backward Pass: Order of computations

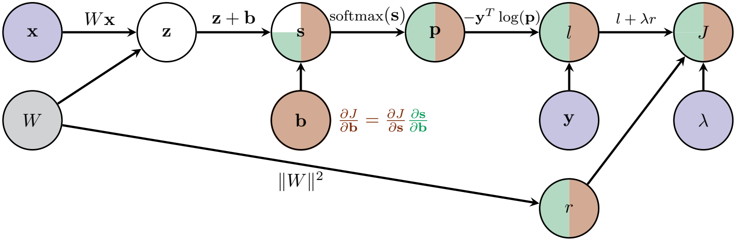
Compute Jacobian of current node w.r.t. one parent



procedure PERFORMBACKPASS(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BACKOP($\mathbf{s}$, G)
  $\mathcal{C}_{\mathbf{s}}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_{\mathbf{s}} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_{\mathbf{s}}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_{\mathbf{s}}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_{\mathbf{s}}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}} \mathrel{+}= \frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_{\mathbf{s}}$ **do** ▷ Compute Jacobian of $f_{\mathbf{s}}$ w.r.t. each parent node
    $\mathbf{s}.\mathbf{p}$.Jacobian = $\frac{\partial f_{\mathbf{s}}(\mathcal{P}_{\mathbf{s}})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_{\mathbf{s}}(\mathcal{P}_{\mathbf{s}})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

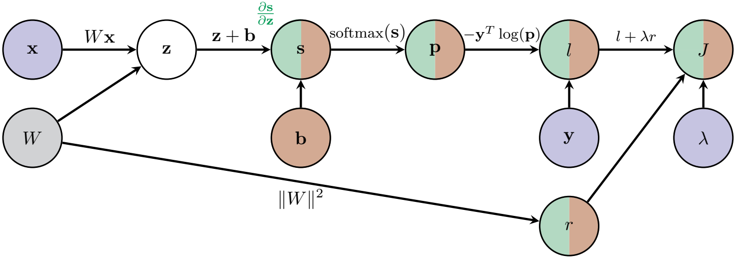
Compute Gradient of current node




```
procedure PerformBackPass(G)
    J = GetResultNode(G)   ▷ node with the value of cost function
    BackOp(J, G)           ▷ Start the Backward-pass
end procedure
```

```
procedure BackOp(s, G)
    C_s = GetChildren(s, G)
    if C_s = ∅ then                                   ▷ At the result node
        s.Grad = 1
    else
        if AllGradientsComputed(C_s) then    ▷ All ∂J/∂c computed where c ∈ C_s
            s.Grad = 0
            for each c ∈ C_s do
                s.Grad += c.Grad * c.s.Jacobian      ▷ ∂J/∂s += ∂J/∂c ∂c/∂s
            end for
            s.GradComputed = true
        end if
    end if
    for each p ∈ P_s do        ▷ Compute Jacobian of f_s w.r.t. each parent node
        s.p.Jacobian = ∂f_s(P_s)/∂p                  ▷ ∂f_s(P_s)/∂p = ∂s/∂p
        BackOp(p, G)
    end for
end procedure
```

Compute Jacobian of current node w.r.t. one parent

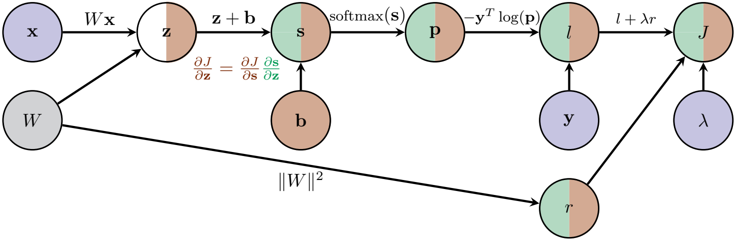


procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_{\mathbf{s}}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_{\mathbf{s}} = \emptyset$ **then** ▷ At the result node
    s.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_{\mathbf{s}}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_{\mathbf{s}}$
      s.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_{\mathbf{s}}$ **do**
        s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      s.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_{\mathbf{s}}$ **do** ▷ Compute Jacobian of $f_{\mathbf{s}}$ w.r.t. each parent node
    s.$\mathbf{p}$.Jacobian = $\frac{\partial f_{\mathbf{s}}(\mathcal{P}_{\mathbf{s}})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_{\mathbf{s}}(\mathcal{P}_{\mathbf{s}})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

Generic Backward Pass: Order of computations

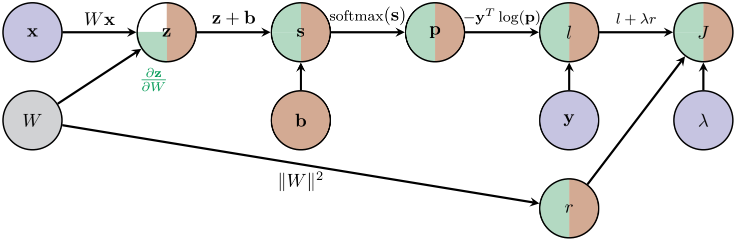
Compute Gradient of current node



procedure PerformBackPass(G)
  J = GetResultNode(G) ▷ node with the value of cost function
  BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
  $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
  if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
    $\mathbf{s}$.Grad = 1
  else
    if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
      $\mathbf{s}$.Grad = $\mathbf{0}$
      for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
        $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.$\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
      end for
      $\mathbf{s}$.GradComputed = true
    end if
  end if
  for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
    $\mathbf{s}$.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
    BackOp($\mathbf{p}$, G)
  end for
end procedure

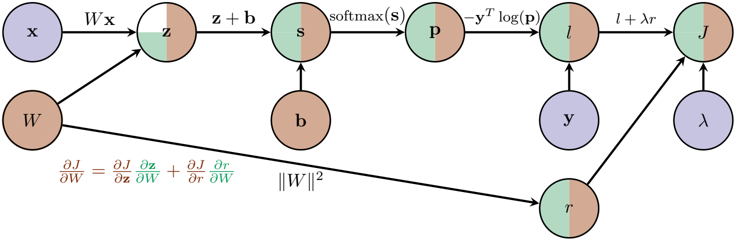
Compute Jacobian of current node w.r.t. one parent



procedure PerformBackPass(G)
    J = GetResultNode(G) ▷ node with the value of cost function
    BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
    $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
    if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
        $\mathbf{s}$.Grad = 1
    else
        if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
            $\mathbf{s}$.Grad = $\mathbf{0}$
            for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
                $\mathbf{s}$.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.$\mathbf{s}$.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}}$ += $\frac{\partial J}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
            end for
            $\mathbf{s}$.GradComputed = true
        end if
    end if
    for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
        $\mathbf{s}$.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
        BackOp($\mathbf{p}$, G)
    end for
end procedure

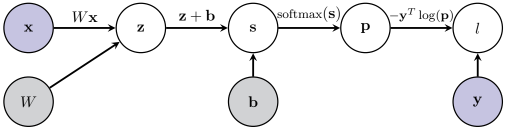
Compute Gradient of current node



procedure PerformBackPass(G)
    J = GetResultNode(G) ▷ node with the value of cost function
    BackOp(J, G) ▷ Start the Backward-pass
end procedure

procedure BackOp($\mathbf{s}$, G)
    $\mathcal{C}_\mathbf{s}$ = GetChildren($\mathbf{s}$, G)
    if $\mathcal{C}_\mathbf{s} = \emptyset$ **then** ▷ At the result node
        s.Grad = 1
    else
        if AllGradientsComputed($\mathcal{C}_\mathbf{s}$) **then** ▷ All $\frac{\partial J}{\partial \mathbf{c}}$ computed where $\mathbf{c} \in \mathcal{C}_\mathbf{s}$
            s.Grad = $\mathbf{0}$
            for each $\mathbf{c} \in \mathcal{C}_\mathbf{s}$ **do**
                s.Grad += $\mathbf{c}$.Grad * $\mathbf{c}$.s.Jacobian ▷ $\frac{\partial J}{\partial \mathbf{s}} \mathrel{+}= \frac{\partial J}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{s}}$
            end for
            s.GradComputed = true
        end if
    end if
    for each $\mathbf{p} \in \mathcal{P}_\mathbf{s}$ **do** ▷ Compute Jacobian of $f_\mathbf{s}$ w.r.t. each parent node
        s.$\mathbf{p}$.Jacobian = $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}}$ ▷ $\frac{\partial f_\mathbf{s}(\mathcal{P}_\mathbf{s})}{\partial \mathbf{p}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}}$
        BackOp($\mathbf{p}$, G)
    end for
end procedure

Issues we need to sort out



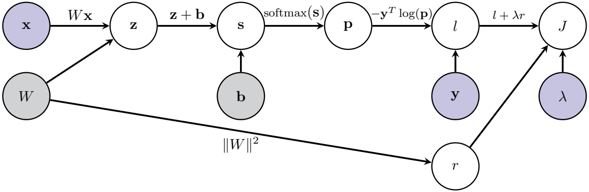
- Back-propagation when the computational graph is **not a path graph**. ✓

- Derivative computations when the inputs and outputs are not scalars. ✓

- Let's now compute some gradients!

Example of Back-Prop in action

Compute gradients for

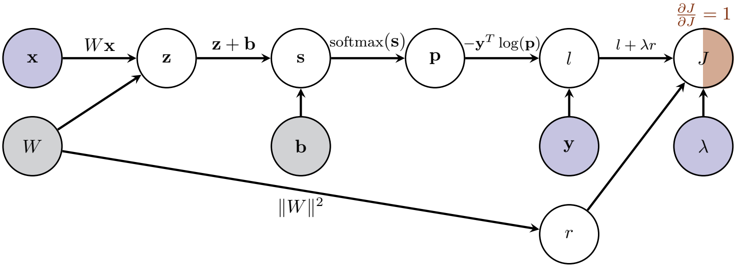


linear scoring function + SoftMax + cross-entropy loss + Regularization

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

Generic Backward Pass: Gradient of current node

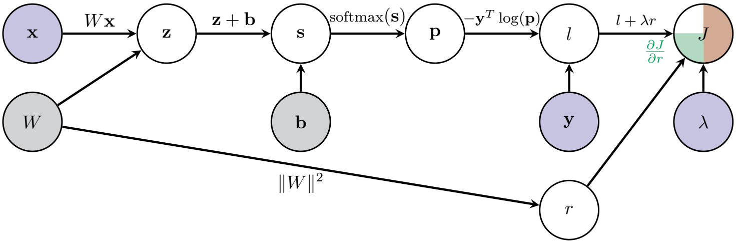
Compute Gradient of node J



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

Generic Backward Pass: Order of computations

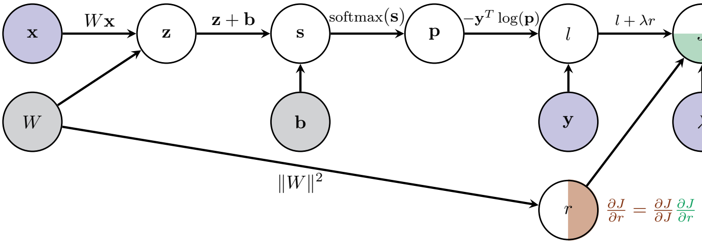
Compute Jacobian of node J w.r.t. its parent r



$$\boxed{J = l + \lambda r}$$

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

Compute Gradient of node r

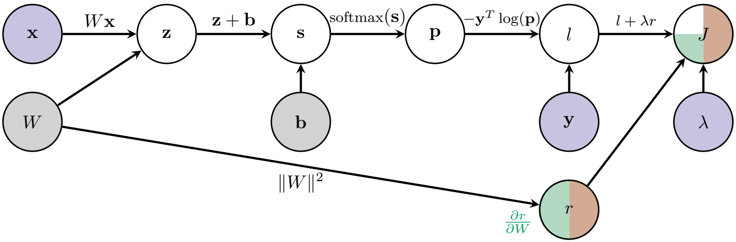


$$\boxed{J = l + \lambda r}$$

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

Generic Backward Pass: Order of computations

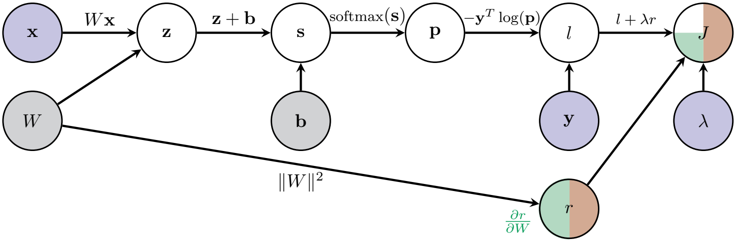
Compute Jacobian of node r w.r.t. its parent W



$$\boxed{r = \sum_{i,j} W_{ij}^2}$$

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

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

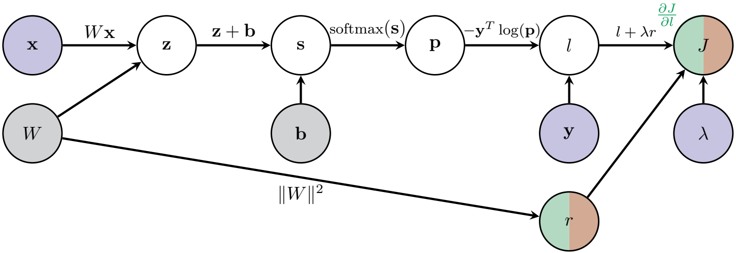


$$\boxed{r = \sum_{i,j} W_{ij}^2}$$

- Jacobian to compute: $\frac{\partial r}{\partial W} = \begin{pmatrix} \frac{\partial r}{\partial W_{11}} & \frac{\partial r}{\partial W_{12}} & \cdots & \cdots \frac{\partial r}{\partial W_{1d}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r}{\partial W_{C1}} & \frac{\partial r}{\partial W_{C2}} & \cdots & \cdots \frac{\partial r}{\partial W_{Cd}} \end{pmatrix}$ (W is $C \times d$)
- The individual derivatives: $\frac{\partial r}{\partial W_{ij}} = 2W_{ij}$
- Putting it together in matrix notation

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

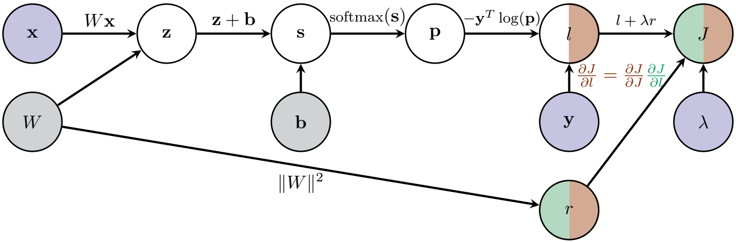
Compute Jacobian of node J w.r.t. its parent l



$$\boxed{J = l + \lambda r}$$

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

Compute Gradient of node l

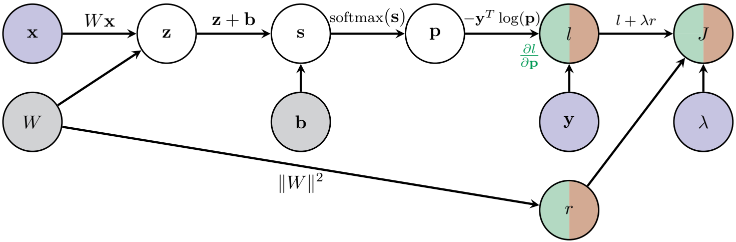


$$\boxed{J = l + \lambda r}$$

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

Generic Backward Pass: Order of computations

Compute Jacobian of node l w.r.t. its parent $\mathbf{p}$

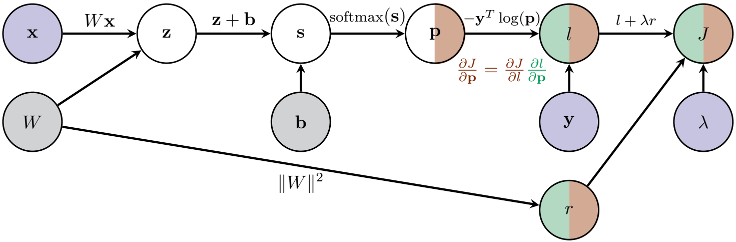


$$\boxed{l = -\mathbf{y}^T \log(\mathbf{p})}$$

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

$$\frac{\partial l}{\partial \mathbf{p}} = -\mathbf{y}^T \text{diag}\,(\mathbf{p})^{-1}$$

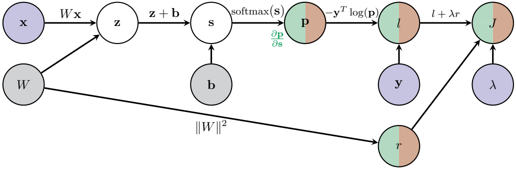
Compute Gradient of node $\mathbf{p}$



$$\boxed{l = -\mathbf{y}^T \log(\mathbf{p})}$$

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

Compute Jacobian of node $\mathbf{p}$ w.r.t. its parent $\mathbf{s}$



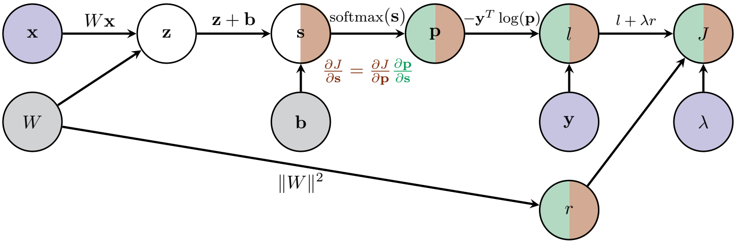
$$\boxed{\mathbf{p} = \exp(\mathbf{s}) / \left(\mathbf{1}^T \exp(\mathbf{s})\right)}$$

- The Jacobian we need to compute: $\frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \begin{pmatrix} \frac{\partial p_1}{\partial s_1} & \cdots & \frac{\partial p_1}{\partial s_C} \\ \vdots & \vdots & \vdots \\ \frac{\partial p_C}{\partial s_1} & \cdots & \frac{\partial p_C}{\partial s_C} \end{pmatrix}$
- 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: $\frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T$

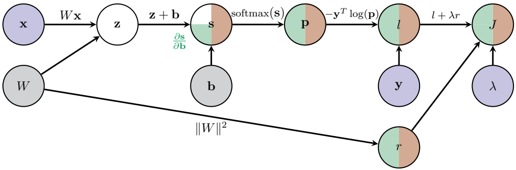
Compute Gradient of node $\mathbf{s}$



$$\boxed{\mathbf{p} = \exp(\mathbf{s})/\left(\mathbf{1}^T\exp(\mathbf{s})\right)}$$

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

Compute Jacobian of node $\mathbf{s}$ w.r.t. its parent $\mathbf{b}$

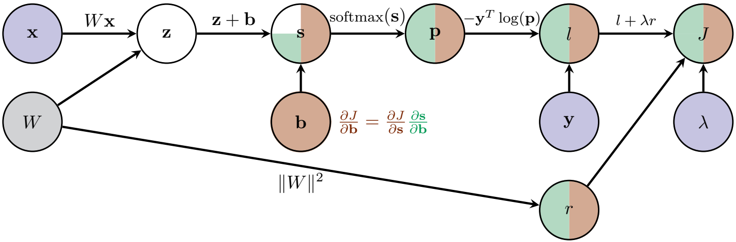


$$\boxed{\mathbf{s} = \mathbf{z} + \mathbf{b}}$$

- The Jacobian we need to compute: $\frac{\partial \mathbf{s}}{\partial \mathbf{b}} = \begin{pmatrix} \frac{\partial s_1}{\partial b_1} & \cdots & \frac{\partial s_1}{\partial b_C} \\ \vdots & \vdots & \vdots \\ \frac{\partial s_C}{\partial b_1} & \cdots & \frac{\partial s_C}{\partial b_C} \end{pmatrix}$
- 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: $\frac{\partial \mathbf{s}}{\partial \mathbf{b}} = I_C$ $\leftarrow$ the identity matrix of size $C \times C$

Generic Backward Pass: Order of computations

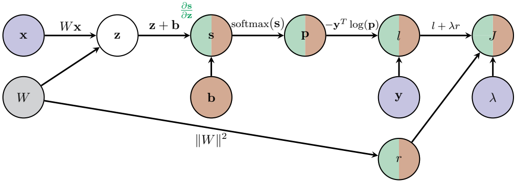
Compute Gradient of node $\mathbf{b}$



$$\boxed{\mathbf{s} = \mathbf{z} + \mathbf{b}}$$

gradient needed for mini-batch g.d.training as $\mathbf{b}$ parameter of the model $\rightarrow$ $$\frac{\partial J}{\partial \mathbf{b}} = \frac{\partial J}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{b}}$$

Compute Jacobian of node $\mathbf{s}$ w.r.t. its parent $\mathbf{z}$



$$\boxed{\mathbf{s} = \mathbf{z} + \mathbf{b}}$$

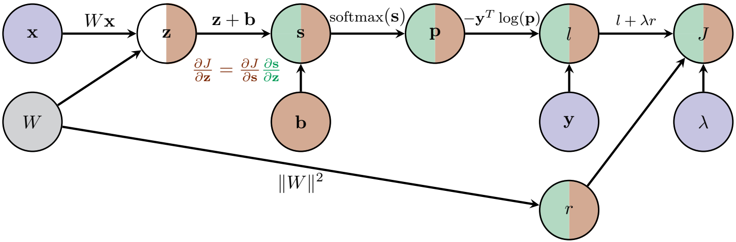
- The Jacobian we need to compute: $\frac{\partial \mathbf{s}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial s_1}{\partial z_1} & \cdots & \frac{\partial s_1}{\partial z_C} \\ \vdots & \vdots & \vdots \\ \frac{\partial s_C}{\partial z_1} & \cdots & \frac{\partial s_C}{\partial z_C} \end{pmatrix}$

- 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: $\frac{\partial \mathbf{s}}{\partial \mathbf{z}} = I_C$ $\leftarrow$ the identity matrix of size $C \times C$

Generic Backward Pass: Order of computations

Compute Gradient of node $\mathbf{z}$

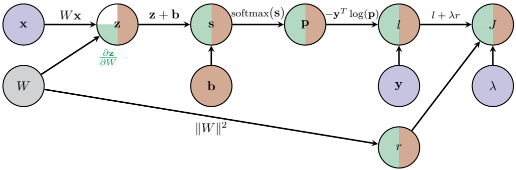


$$\boxed{\mathbf{s} = \mathbf{z} + \mathbf{b}}$$

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

Generic Backward Pass: Order of computations

Compute Jacobian of node $\mathbf{z}$ w.r.t. its parent W



$$\boxed{\mathbf{z} = W\mathbf{x}}$$

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

$$W = \begin{pmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_C^T \end{pmatrix} \quad \text{then} \quad \text{vec}(W) = \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_C \end{pmatrix}$$

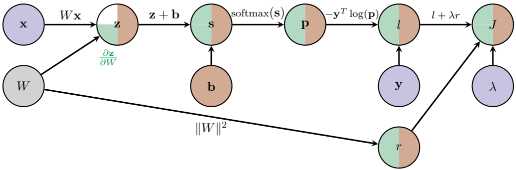
- Then

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

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

Generic Backward Pass: Order of computations

Compute Jacobian of node $\mathbf{z}$ w.r.t. one parent W

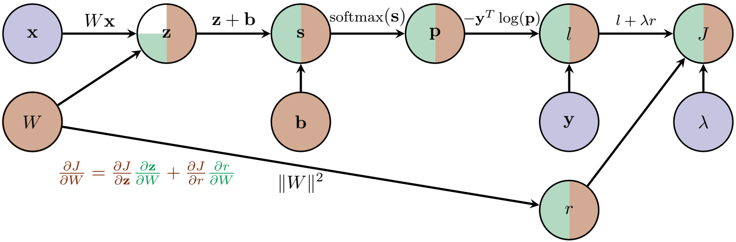


$$\boxed{\mathbf{z} = W\mathbf{x} = \left(I_C \otimes \mathbf{x}^T\right)\text{vec}(W)}$$

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

Generic Backward Pass: Order of computations

Compute Gradient of node W



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

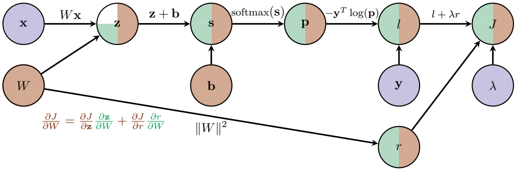
gradient needed for learning $\rightarrow$

$$\frac{\partial J}{\partial \text{vec}(W)} = \frac{\partial J}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \text{vec}(W)} + \frac{\partial J}{\partial r}\frac{\partial r}{\partial \text{vec}(W)}$$
$$= \begin{pmatrix} g_1\mathbf{x}^T & g_2\mathbf{x}^T & \cdots & g_C\mathbf{x}^T \end{pmatrix} + 2\lambda\,\text{vec}(W)^T$$

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

Generic Backward Pass: Order of computations

Compute Gradient of node W



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

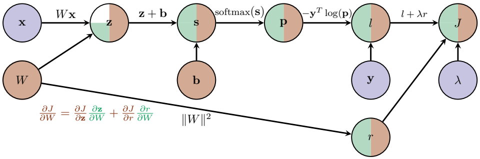
Can convert

$$\frac{\partial J}{\partial \text{vec}(W)} = \begin{pmatrix} g_1\mathbf{x}^T & g_2\mathbf{x}^T & \cdots & g_C\mathbf{x}^T \end{pmatrix} + 2\lambda\,\text{vec}(W)^T$$

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

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

Aggregating the Gradient computations



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

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

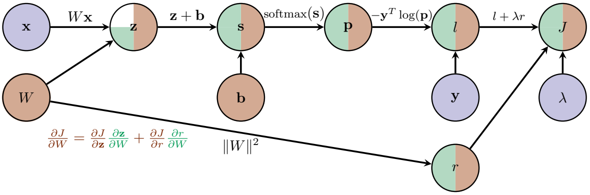
$$\mathbf{g} \leftarrow \mathbf{g}\frac{\partial l}{\partial \mathbf{p}} = -\mathbf{y}^T \text{diag}(\mathbf{p})^{-1} \quad \leftarrow \frac{\partial J}{\partial \mathbf{p}}$$

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

$$\mathbf{g} \leftarrow \mathbf{g}\frac{\partial \mathbf{s}}{\partial \mathbf{z}} = \mathbf{g}\, I_C \quad \leftarrow \frac{\partial J}{\partial \mathbf{z}}$$

Then

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



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

1. Let

$$\mathbf{g} = -\mathbf{y}^T \text{diag}(\mathbf{p})^{-1} \left(\text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T\right) = -(\mathbf{y} - \mathbf{p})^T$$ ←easy to show this last simplification

2. The gradient of J w.r.t. the bias vector is the $1 \times C$ 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 $\mathcal{D}$.

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

- The gradients we need to compute are

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

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

Gradient Computations for a mini-batch

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

- Compute gradient of $L(\mathcal{D}^{(t)}, W, \mathbf{b})$ w.r.t. $W, \mathbf{b}$:
  - Set all entries in $\frac{\partial L}{\partial \mathbf{b}}$ and $\frac{\partial L}{\partial W}$ to zero.
  - for each $(\mathbf{x}, y) \in \mathcal{D}^{(t)}$
    1. Evaluate $\mathbf{p} = \text{SoftMax}(W\mathbf{x} + \mathbf{b})$
    2. Let
    $$\mathbf{g} = -(\mathbf{y} - \mathbf{p})^T$$
    3. Add gradient of $l(\mathbf{x}, y, W, \mathbf{b})$ w.r.t. $\mathbf{b}$
    $$\frac{\partial L}{\partial \mathbf{b}} \mathrel{+}= \mathbf{g}$$
    4. Add gradient of $l(\mathbf{x}, y, W, \mathbf{b})$ w.r.t. W:
    $$\frac{\partial L}{\partial W} \mathrel{+}= \mathbf{g}^T \mathbf{x}^T$$
  - Divide by the number of entries in $\mathcal{D}^{(t)}$:
  $$\frac{\partial L}{\partial W} \mathrel{/}= |\mathcal{D}^{(t)}|, \qquad \frac{\partial L}{\partial \mathbf{b}} \mathrel{/}= |\mathcal{D}^{(t)}|$$
- Add the gradient for the regularization term
$$\frac{\partial J}{\partial W} = \frac{\partial L}{\partial W} + 2\lambda W, \qquad \frac{\partial J}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{b}}$$

Efficient Gradient Computations for a mini-batch

- Let $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{n_b}, \mathbf{y}_{n_b})\}$ be the data in the mini-batch $\mathcal{D}^{(t)}$.
  - Gather all $\mathbf{x}_i$'s from the batch into a matrix, similarly for $\mathbf{y}_i$'s

$$\mathbf{X}_{\text{batch}} = \begin{pmatrix} \uparrow & & \uparrow \\ \mathbf{x}_1 & \cdots & \mathbf{x}_{n_b} \\ \downarrow & & \downarrow \end{pmatrix}, \quad \mathbf{Y}_{\text{batch}} = \begin{pmatrix} \uparrow & & \uparrow \\ \mathbf{y}_1 & \cdots & \mathbf{y}_{n_b} \\ \downarrow & & \downarrow \end{pmatrix}$$

  - Complete the **forward pass**

$$\mathbf{P}_{\text{batch}} = \text{SoftMax}\left(W\mathbf{X}_{\text{batch}} + \mathbf{b}\mathbf{1}_{n_b}^T\right)$$

← SoftMax applied independently to each column

  - Complete the **backward pass**

    1. Set

$$\mathbf{G}_{\text{batch}} = -\left(\mathbf{Y}_{\text{batch}} - \mathbf{P}_{\text{batch}}\right)$$

    2. Then

$$\frac{\partial L}{\partial W} = \frac{1}{n_b}\mathbf{G}_{\text{batch}}\mathbf{X}_{\text{batch}}^T, \qquad \frac{\partial L}{\partial \mathbf{b}} = \frac{1}{n_b}\mathbf{G}_{\text{batch}}\mathbf{1}_{n_b}$$

- Add the gradient for the regularization term

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