Deep Learning for Computer Vision

Dr. Konda Reddy Mopuri
Mehta Family School of Data Science and Artificial Intelligence
IIT Guwahati
Aug-Dec 2022

Recap

- Gradient of a scalar valued function $f(\mathbf{x})$: $\mathbf{x} \rightarrow \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_D}\right)^T$

Recap

- Gradient of a scalar valued function $f(\mathbf{x})$: $\mathbf{x} \rightarrow \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_D}\right)^T$
- Gradient of a vector valued function $\mathbf{f}(\mathbf{x})$ is called Jacobian:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Gradient descent on MLP

- Loss is $\mathcal{L}(W, \mathbf{b}) = \sum_n l(f(x_n; W, \mathbf{b}), y_n)$

Gradient descent on MLP

- Loss is $\mathcal{L}(W, \mathbf{b}) = \sum_n l(f(x_n; W, \mathbf{b}), y_n)$
- For applying Gradient descent, we need gradient of individual sample loss with respect to all the model parameters

$$l_n = l(f(x_n; W, \mathbf{b}), y_n)$$

$\frac{\partial l_n}{\partial W_{i,j}^{(l)}}$ and $\frac{\partial l_n}{\partial \mathbf{b}_i^{(l)}}$

Forward pass operation

$$x^{(0)} = x \xrightarrow{W^{(1)},\mathbf{b}^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{W^{(2)},\mathbf{b}^{(2)}} s^{(2)} \dots x^{(L-1)} \xrightarrow{W^{(L)},\mathbf{b}^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} = f(x; W, \mathbf{b})$$

Formally, $x^{(0)} = x, f(x; W, \mathbf{b}) = x^{(L)}$

$$\forall l = 1, \dots, L \quad \begin{cases} s^{(l)} & = W^{(l)} x^{(l-1)} + \mathbf{b}^{(l)} \\ x^{(l)} & = \sigma(s^{(l)}) \end{cases}$$

Chain rule of differential calculus

- Core concept of backpropagation

Chain rule of differential calculus

- Core concept of backpropagation
- $$(g \circ f)'(x) = g'(f(x)) \cdot f'(x)$$

Chain rule of differential calculus

- Core concept of backpropagation
- $$(g \circ f)'(x) = g'(f(x)) \cdot f'(x)$$
- $$\frac{\partial}{\partial x} g(f(x)) = \left. \frac{\partial g(a)}{\partial a} \right|_{a=f(x)} \cdot \frac{\partial f(x)}{\partial x}$$

Chain rule of differential calculus

- Core concept of backpropagation
- $$(g \circ f)'(x) = g'(f(x)) \cdot f'(x)$$
- $$\frac{\partial}{\partial x} g(f(x)) = \frac{\partial g(a)}{\partial a}\bigg|_{a=f(x)} \cdot \frac{\partial f(x)}{\partial x}$$
- $$J_{f_N \circ f_{N-1} \circ \ldots f_1(x)} = J_{f_N(f_{N-1}(\ldots f_1(x)))} \cdot J_{f_{N-1}(f_{N-2}(\ldots f_1(x)))} \cdot \cdots \cdot J_{f_2(f_1(x))} \cdot J_{f_1(x)}$$

  $J_{f(x)}$ is Jacobian of f computed at x.

Consider a specific Layer

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$

Consider a specific Layer

- $x^{(l-1)} \xrightarrow{W^{(l)}, \mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $x_i^{(l)} = \sigma(s_i^{(l)})$

Consider a specific Layer

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $x_i^{(l)} = \sigma(s_i^{(l)})$
- Since $s^{(l)}$ influences loss $\mathcal{L}$ through only $x^{(l)}$,

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)})$$

Consider a specific Layer

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $x_i^{(l)} = \sigma(s_i^{(l)})$
- Since $s^{(l)}$ influences loss $\mathcal{L}$ through only $x^{(l)}$,

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)})$$

- $$s_i^{(l)} = \Sigma_j W_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$$

Consider a specific Layer

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $x_i^{(l)} = \sigma(s_i^{(l)})$
- Since $s^{(l)}$ influences loss $\mathcal{L}$ through only $x^{(l)}$,

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)})$$

- $$s_i^{(l)} = \Sigma_j W_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$$

- Since $x^{(l-1)}$ influences the loss $\mathcal{L}$ only through $s^{(l)}$,

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \Sigma_i \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial x_j^{(l-1)}} = \Sigma_i \frac{\partial \ell}{\partial s_i^{(l)}} W_{i,j}^{(l)}$$

We need gradients wrt parameters W and b

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$

We need gradients wrt parameters W and b

- $x^{(l-1)} \xrightarrow{W^{(l)}, \mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $W_{i,j}^{(l)}$ and $\mathbf{b}^{(l)}$ influence the loss through $s^{(l)}$ via $s_i^{(l)} = \Sigma_j W_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$,

We need gradients wrt parameters W and b

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $W_{i,j}^{(l)}$ and $\mathbf{b}^{(l)}$ influence the loss through $s^{(l)}$ via $s_i^{(l)} = \Sigma_j W_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$,
- $$\frac{\partial \ell}{\partial W_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial W_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)} \tag{1}$$

We need gradients wrt parameters W and b

- $x^{(l-1)} \xrightarrow{W^{(l)},\mathbf{b}^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$
- $W_{i,j}^{(l)}$ and $\mathbf{b}^{(l)}$ influence the loss through $s^{(l)}$ via $s_i^{(l)} = \Sigma_j W_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$,
- $$\frac{\partial \ell}{\partial W_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial W_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)} \tag{1}$$
- $$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \tag{2}$$

Summary of Backprop

- From the definition of loss, obtain $\frac{\partial l}{\partial x_i^{(l)}}$

Summary of Backprop

- From the definition of loss, obtain $\frac{\partial l}{\partial x_i^{(l)}}$
- Recursively compute the loss derivatives wrt the activations

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)}) \text{ and } \frac{\partial \ell}{\partial x_j^{(l-1)}} = \Sigma_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}$$

Summary of Backprop

- From the definition of loss, obtain $\frac{\partial l}{\partial x_i^{(l)}}$
- Recursively compute the loss derivatives wrt the activations

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)}) \text{ and } \frac{\partial \ell}{\partial x_j^{(l-1)}} = \Sigma_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}$$

- Then wrt the parameters
$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)} \text{ and } \frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}$$

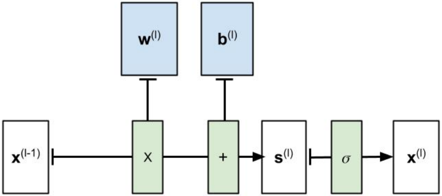
Jocobian in Tensorial form

- $\psi : \mathcal{R}^N \rightarrow \mathcal{R}^M$ then $\left[\frac{\partial \psi}{\partial x}\right] = \begin{bmatrix} \frac{\partial \psi_1}{\partial x_1} & \cdots & \frac{\partial \psi_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi_M}{\partial x_1} & \cdots & \frac{\partial \psi_M}{\partial x_N} \end{bmatrix}$

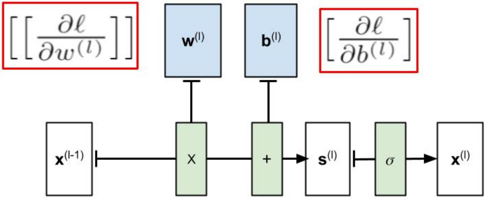
Jocobian in Tensorial form

- $\psi : \mathcal{R}^N \rightarrow \mathcal{R}^M$ then $\left[\frac{\partial \psi}{\partial x}\right] = \begin{bmatrix} \frac{\partial \psi_1}{\partial x_1} & \cdots & \frac{\partial \psi_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi_M}{\partial x_1} & \cdots & \frac{\partial \psi_M}{\partial x_N} \end{bmatrix}$
- $\psi : \mathcal{R}^{N \times M} \rightarrow \mathcal{R}$ then $\left[\left[\frac{\partial \psi}{\partial x}\right]\right] = \begin{bmatrix} \frac{\partial \psi}{\partial w_{1,1}} & \cdots & \frac{\partial \psi}{\partial w_{1,M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi}{\partial w_{N,1}} & \cdots & \frac{\partial \psi}{\partial w_{N,M}} \end{bmatrix}$

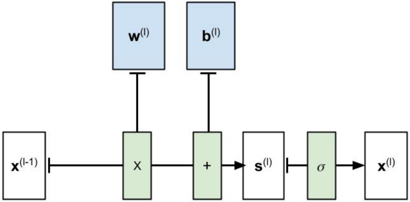
Forward Pass



Goal of Backward Pass

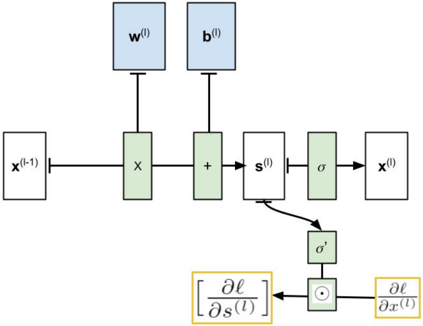


Begin from succeeding layer

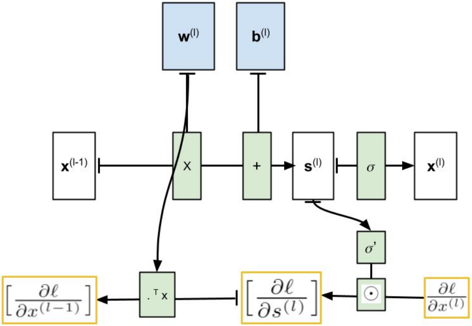


$$\boxed{\frac{\partial \ell}{\partial x^{(l)}}}$$

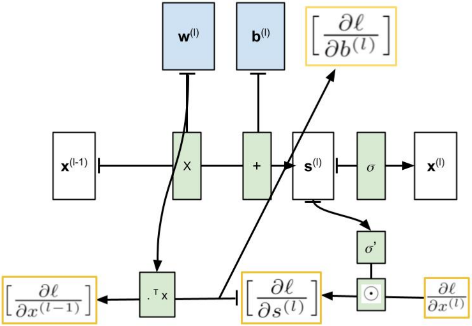
Begin from succeeding layer



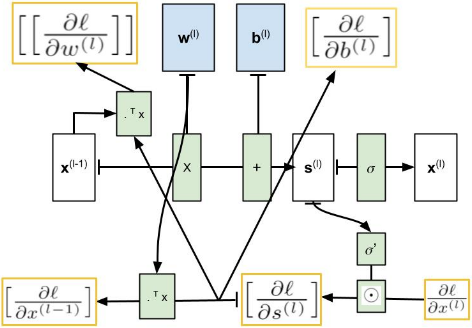
Begin from succeeding layer



Begin from succeeding layer



Begin from succeeding layer



Update the parameters

- $W^{(l)} = W^{(l)} - \eta \left[\left[\frac{\partial \ell}{\partial w^{(l)}} \right] \right]$ and $\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \eta \left[\frac{\partial \ell}{\partial b^{(l)}} \right]$

Observations

- BP is basically simple: applying chain rule iteratively

Observations

- BP is basically simple: applying chain rule iteratively
- It can be expressed in tensorial form (similar to the forward pass)

Observations

- BP is basically simple: applying chain rule iteratively
- It can be expressed in tensorial form (similar to the forward pass)
- Heavy computations are with the linear operations

Observations

- BP is basically simple: applying chain rule iteratively
- It can be expressed in tensorial form (similar to the forward pass)
- Heavy computations are with the linear operations
- Nonlinearities go into simple element wise operations

Observations

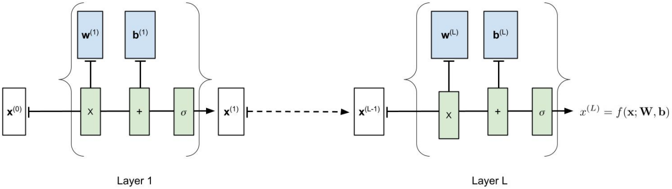
- BP is basically simple: applying chain rule iteratively
- It can be expressed in tensorial form (similar to the forward pass)
- Heavy computations are with the linear operations
- Nonlinearities go into simple element wise operations
- In an untreated situation, BP Needs all the intermediate layer results to be in memory

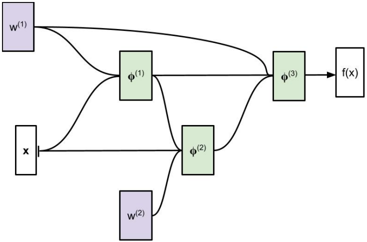
Observations

- BP is basically simple: applying chain rule iteratively
- It can be expressed in tensorial form (similar to the forward pass)
- Heavy computations are with the linear operations
- Nonlinearities go into simple element wise operations
- In an untreated situation, BP Needs all the intermediate layer results to be in memory
- Takes twice the computations of forward pass

Beyond MLP

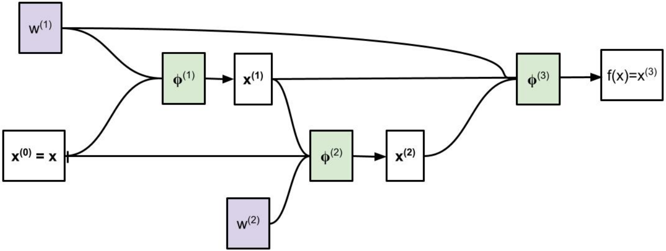
- We can generalize MLP





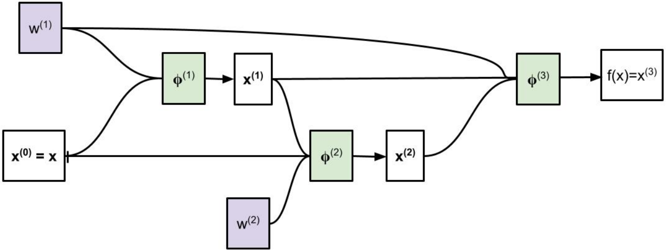
To an arbitrary Directed Acyclic Graph (DAG)

Forward pass in the computational graph



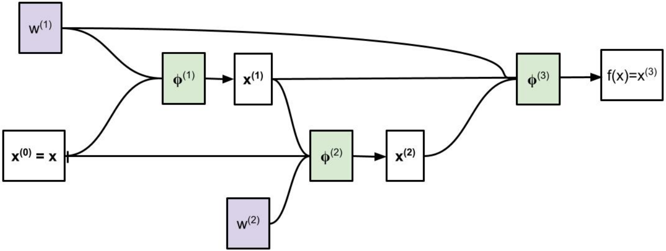
- $x^{(0)} = x$

Forward pass in the computational graph



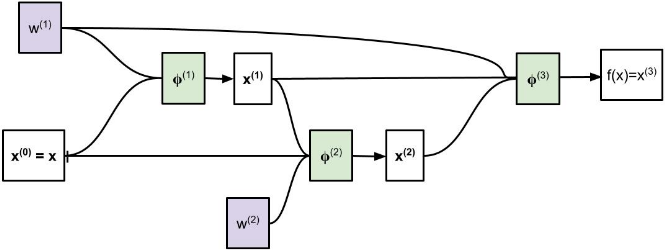
- $x^{(0)} = x$
- $x^{(1)} = \phi^{(1)}(x^{(0)}; w^{(1)})$

Forward pass in the computational graph



- $x^{(0)} = x$
- $x^{(1)} = \phi^{(1)}(x^{(0)}; w^{(1)})$
- $x^{(2)} = \phi^{(2)}(x^{(0)}, x^{(1)}; w^{(2)})$

Forward pass in the computational graph



- $x^{(0)} = x$
- $x^{(1)} = \phi^{(1)}(x^{(0)}; w^{(1)})$
- $x^{(2)} = \phi^{(2)}(x^{(0)}, x^{(1)}; w^{(2)})$
- $f(x) = x^{(3)} = \phi^{(3)}(x^{(1)}, x^{(2)}; w^{(1)})$

Notation: Jacobian of a general transformation

- $$\text{if } (a_1 \dots a_Q) = \phi(b_1 \dots b_R) \text{ then we use the notation} \tag{3}$$

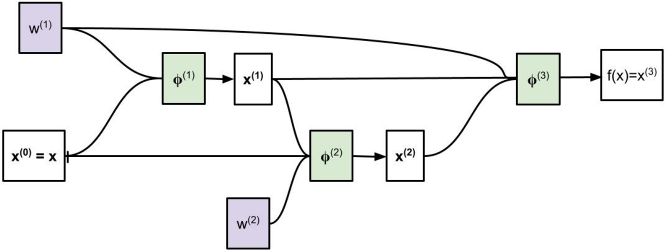
$$\left[\frac{\partial a}{\partial b}\right] = J_\phi^T = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \cdots & \frac{\partial a_Q}{\partial b_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial b_R} & \cdots & \frac{\partial a_Q}{\partial b_R} \end{bmatrix} \tag{4}$$

Notation: Jacobian of a general transformation

- $$\text{if } (a_1 \ldots a_Q) = \phi(b_1 \ldots b_R) \text{ then we use the notation} \tag{3}$$
  $$\left[\frac{\partial a}{\partial b}\right] = J_\phi^T = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \cdots & \frac{\partial a_Q}{\partial b_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial b_R} & \cdots & \frac{\partial a_Q}{\partial b_R} \end{bmatrix} \tag{4}$$

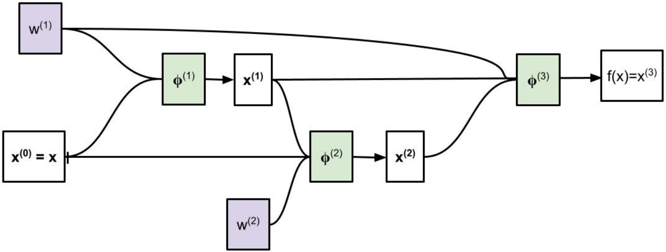
- $$\text{if } (a_1 \ldots a_Q) = \phi(b_1 \ldots b_R; c_1 \ldots c_S) \text{ then we use the notation} \tag{5}$$
  $$\left[\frac{\partial a}{\partial c}\right] = J_{\phi|c}^T = \begin{bmatrix} \frac{\partial a_1}{\partial c_1} & \cdots & \frac{\partial a_Q}{\partial c_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial C_S} & \cdots & \frac{\partial a_Q}{\partial c_S} \end{bmatrix} \tag{6}$$

Backward pass



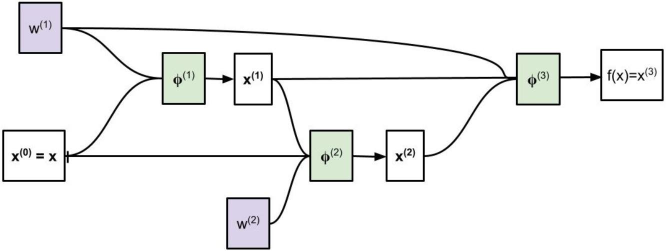
- From the loss equation, we can compute $\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$

Backward pass



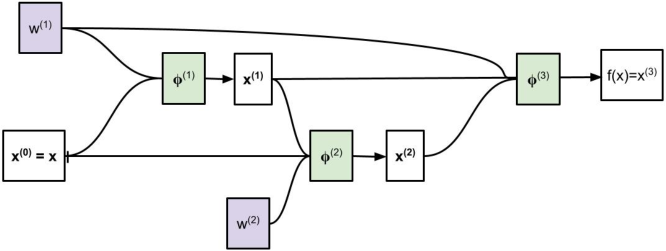
- From the loss equation, we can compute $\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$
- $$\left[\frac{\partial \ell}{\partial x^{(2)}}\right] = \left[\frac{\partial x^{(3)}}{\partial x^{(2)}}\right]\left[\frac{\partial \ell}{\partial x^{(3)}}\right] = J^T_{\phi^{(3)}|x^{(2)}}\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$$

Backward pass



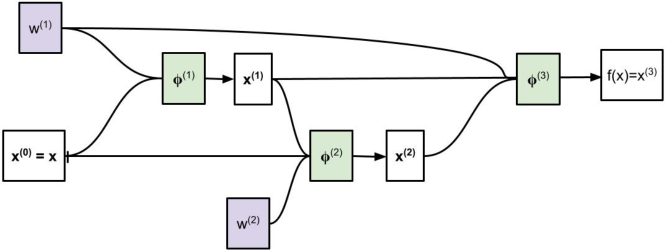
- From the loss equation, we can compute $\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$
- $$\left[\frac{\partial \ell}{\partial x^{(2)}}\right] = \left[\frac{\partial x^{(3)}}{\partial x^{(2)}}\right]\left[\frac{\partial \ell}{\partial x^{(3)}}\right] = J^T_{\phi^{(3)}|x^{(2)}}\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$$
- $$\begin{aligned}\left[\frac{\partial \ell}{\partial x^{(1)}}\right] &= \left[\frac{\partial x^{(3)}}{\partial x^{(1)}}\right]\left[\frac{\partial \ell}{\partial x^{(3)}}\right] + \left[\frac{\partial x^{(2)}}{\partial x^{(1)}}\right]\left[\frac{\partial \ell}{\partial x^{(2)}}\right] \\ &= J^T_{\phi^{(3)}|x^{(1)}}\left[\frac{\partial \ell}{\partial x^{(3)}}\right] + J^T_{\phi^{(2)}|x^{(1)}}\left[\frac{\partial \ell}{\partial x^{(2)}}\right]\end{aligned}$$

Backward pass



- $$\left[\frac{\partial \ell}{\partial w^{(1)}}\right] = \left[\frac{\partial x^{(3)}}{\partial w^{(1)}}\right]\left[\frac{\partial \ell}{\partial x^{(3)}}\right] + \left[\frac{\partial x^{(1)}}{\partial w^{(1)}}\right]\left[\frac{\partial \ell}{\partial x^{(1)}}\right]$$
  $$= J^T_{\phi^{(3)}|w^{(1)}}\left[\frac{\partial \ell}{\partial x^{(3)}}\right] + J^T_{\phi^{(1)}|w^{(1)}}\left[\frac{\partial \ell}{\partial x^{(1)}}\right]$$

Backward pass



- $$\left[\frac{\partial \ell}{\partial w^{(1)}}\right] = \left[\frac{\partial x^{(3)}}{\partial w^{(1)}}\right]\left[\frac{\partial \ell}{\partial x^{(3)}}\right] + \left[\frac{\partial x^{(1)}}{\partial w^{(1)}}\right]\left[\frac{\partial \ell}{\partial x^{(1)}}\right]$$
$$= J^T_{\phi^{(3)}|w^{(1)}}\left[\frac{\partial \ell}{\partial x^{(3)}}\right] + J^T_{\phi^{(1)}|w^{(1)}}\left[\frac{\partial \ell}{\partial x^{(1)}}\right]$$

- $$\left[\frac{\partial \ell}{\partial w^{(2)}}\right] = \left[\frac{\partial x^{(2)}}{\partial w^{(2)}}\right]\left[\frac{\partial \ell}{\partial x^{(2)}}\right] = J^T_{\phi^{(2)}|w^{(2)}}\left[\frac{\partial \ell}{\partial x^{(2)}}\right]$$

Developing DNN architectures

- Developing large architectures from scratch is tedious

Developing DNN architectures

- Developing large architectures from scratch is tedious
- DL frameworks facilitate with libraries for

Developing DNN architectures

- Developing large architectures from scratch is tedious
- DL frameworks facilitate with libraries for
  - tensor operators

Developing DNN architectures

- Developing large architectures from scratch is tedious
- DL frameworks facilitate with libraries for
  - tensor operators
  - mechanisms to combine them into DAGs

Developing DNN architectures

- Developing large architectures from scratch is tedious
- DL frameworks facilitate with libraries for
  - tensor operators
  - mechanisms to combine them into DAGs
  - automatically differentiate them

Autograd

Gradient Computation

- PyTorch automatically constructs on-the-fly graph to compute gradient of any wrt any tensor

Gradient Computation

- PyTorch automatically constructs on-the-fly graph to compute gradient of any wrt any tensor
- Via autograd

Autograd

- Easy to use syntax: only need to define the sequence of forward pass operations

Autograd

- Easy to use syntax: only need to define the sequence of forward pass operations
- Flexible: Computational graph can be dynamic, so is the forward pass

Autograd in PyTorch

- A tensor has the Boolean field '`requires_grad`'

Autograd in PyTorch

- A tensor has the Boolean field '`requires_grad`'
- PyTorch knows if it has to compute gradients wrt this tensor or not

Autograd in PyTorch

- A tensor has the Boolean field '`requires_grad`'
- PyTorch knows if it has to compute gradients wrt this tensor or not
- Default is False
- `requires_grad_()` function can be used to set to any value

Autograd

- `torch.autograd.grad(o/p,i/p)` returns gradients of outputs wrt the inputs

Backward()

- `Tensor.backward()` accumulates the gradients of all the leaf nodes in the graph

Backward()

- `Tensor.backward()` accumulates the gradients of all the leaf nodes in the graph
- `Tensor.grad` field accumulates these gradient

Backward()

- `Tensor.backward()` accumulates the gradients of all the leaf nodes in the graph
- `Tensor.grad` field accumulates these gradient
- Standard function used to train the models.

Backward()

- `Tensor.backward()` accumulates the gradients of all the leaf nodes in the graph
- `Tensor.grad` field accumulates these gradient
- Standard function used to train the models.
- Since it ACCUMULATES the gradients, one may need to set `Tensor.grad` to zero before calling it

Backward()

- `Tensor.backward()` accumulates the gradients of all the leaf nodes in the graph
- `Tensor.grad` field accumulates these gradient
- Standard function used to train the models.
- Since it ACCUMULATES the gradients, one may need to set `Tensor.grad` to zero before calling it
- Accumulation is helpful (e.g. sum of losses, or sum over different mini-batches, etc.)

`torch.no_grad()`

- Switches the autograd machinery off

torch.no_grad()

- Switches the autograd machinery off
- Useful for operations such as parameter updation

`detach()`

- Creates a tensor which only shares data but doesn't require gradient computation

`detach()`

- Creates a tensor which only shares data but doesn't require gradient computation
- Not connected to the current graph

`detach()`

- Creates a tensor which only shares data but doesn't require gradient computation
- Not connected to the current graph
- Used when gradient should not be propagated beyond a variable, or to update the leaf nodes in the graph

Some Notes

- By default, autograd deletes the computational graph after it is evaluated

Some Notes

- By default, autograd deletes the computational graph after it is evaluated
- `retain_graph` indicates to keep it

Some Notes

- By default, autograd deletes the computational graph after it is evaluated
- `retain_graph` indicates to keep it
- Autograd can compute higher-order derivatives

Some Notes

- By default, autograd deletes the computational graph after it is evaluated
- `retain_graph` indicates to keep it
- Autograd can compute higher-order derivatives
- Specified with `create_graph = True`

Demo

▸ Colab Notebook: Backword()