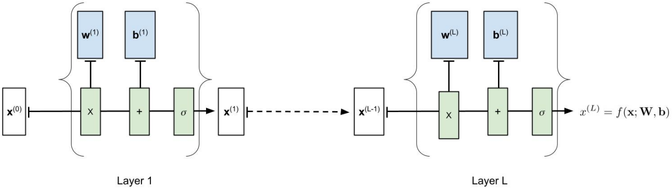
Deep Learning

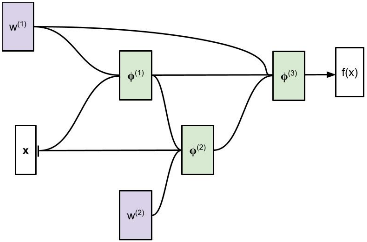
3.6 Backprop beyond MLP and Autograd

Dr. Konda Reddy Mopuri
kmopuri@iittp.ac.in
Dept. of CSE, IIT Tirupati

Beyond MLP

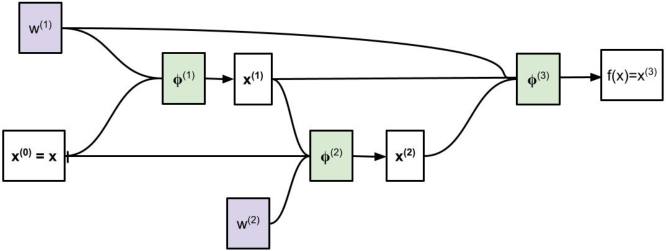
1. We can generalize MLP





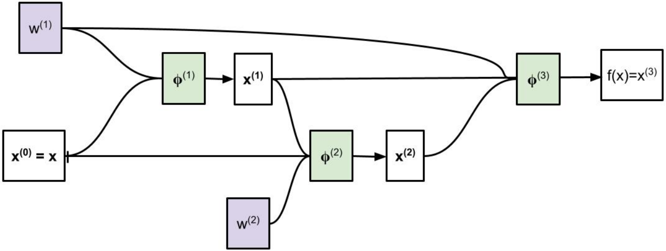
To an arbitrary Directed Acyclic Graph (DAG)

Forward pass in the computational graph



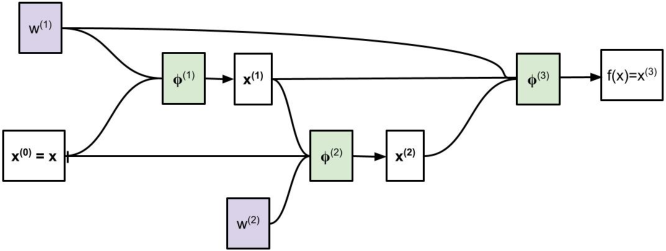
1. $x^{(0)} = x$

Forward pass in the computational graph



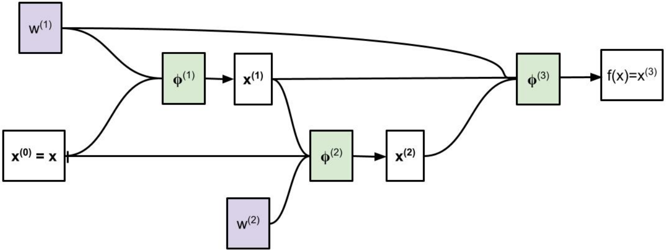
1. $x^{(0)} = x$
2. $x^{(1)} = \phi^{(1)}(x^{(0)}; w^{(1)})$

Forward pass in the computational graph



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

Forward pass in the computational graph



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

Notation: Jacobian of a general transformation

1.

$$\text{if } (a_1 \dots a_Q) = \phi(b_1 \dots b_R) \text{ then we use the notation} \tag{1}$$

$$\left[\frac{\partial a}{\partial b}\right] = J_\phi^T = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \dots & \frac{\partial a_Q}{\partial b_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial b_R} & \dots & \frac{\partial a_Q}{\partial b_R} \end{bmatrix} \tag{2}$$

Notation: Jacobian of a general transformation

1.

$$\text{if } (a_1 \dots a_Q) = \phi(b_1 \dots b_R) \text{ then we use the notation} \tag{1}$$

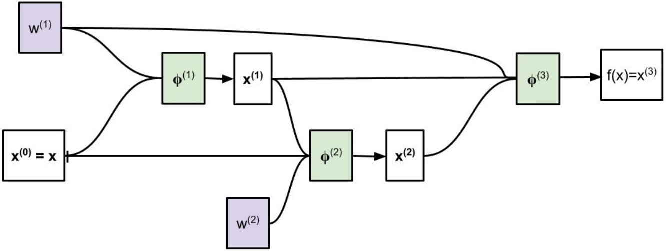
$$\left[\frac{\partial a}{\partial b}\right] = J_\phi^T = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \cdots & \frac{\partial a_Q}{\partial b_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial b_R} & \cdots & \frac{\partial a_Q}{\partial b_R} \end{bmatrix} \tag{2}$$

2.

$$\text{if } (a_1 \dots a_Q) = \phi(b_1 \dots b_R; c_1 \dots c_S) \text{ then we use the notation} \tag{3}$$

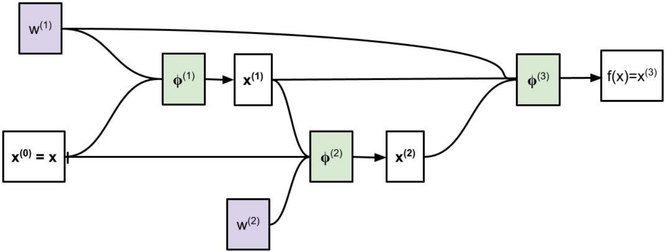
$$\left[\frac{\partial a}{\partial c}\right] = J_{\phi|c}^T = \begin{bmatrix} \frac{\partial a_1}{\partial c_1} & \cdots & \frac{\partial a_Q}{\partial c_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial C_S} & \cdots & \frac{\partial a_Q}{\partial c_S} \end{bmatrix} \tag{4}$$

Backward pass



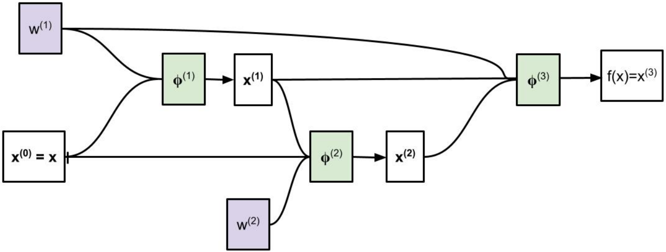
1. From the loss equation, we can compute $\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$

Backward pass



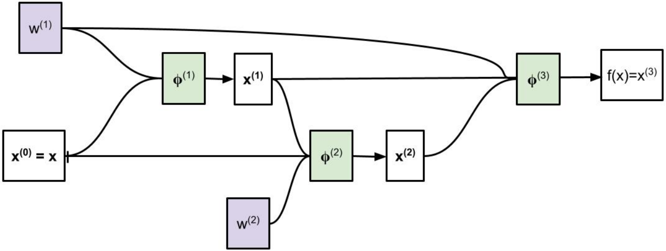
1. From the loss equation, we can compute $\left[\frac{\partial \ell}{\partial x^{(3)}}\right]$
2. $$\left[\frac{\partial \ell}{\partial x^{(2)}}\right] = \left[\frac{\partial x^{(3)}}{\partial x^{(2)}}\right]\left[\frac{\partial \ell}{\partial x^{(3)}}\right] = J^T_{\phi^{(3)}|x^{(2)}}\left[\frac{\partial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

Backward pass



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

Backward pass

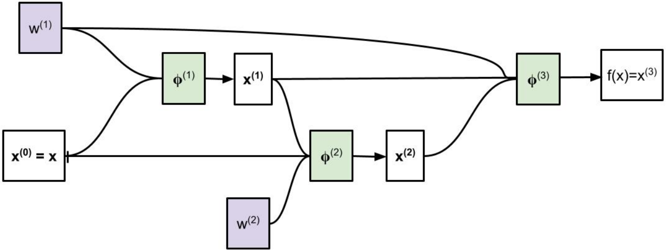


1

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

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

Backward pass



1

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

2

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

Developing DNN architectures

1. Developing large architectures from scratch is tedious

Developing DNN architectures

1. Developing large architectures from scratch is tedious
2. DL frameworks facilitate with libraries for

Developing DNN architectures

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

Developing DNN architectures

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

Developing DNN architectures

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

Autograd

Gradient Computation

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

Gradient Computation

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

Autograd

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

Autograd

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

Autograd in PyTorch

1. A tensor has the Boolean field '`requires_grad`'

Autograd in PyTorch

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

Autograd in PyTorch

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

Autograd

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

Backward()

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

Backward()

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

Backward()

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

Backward()

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

Backward()

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

torch.no_grad()

1. Switches the autograd machinery off

torch.no_grad()

1. Switches the autograd machinery off
2. Useful for operations such as parameter updation

detach()

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

`detach()`

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

detach()

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

Some Notes

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

Some Notes

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

Some Notes

1. By default, autograd deletes the computational graph after it is evaluated
2. `retain_graph` indicates to keep it
3. Graph can compute on the gradient tensor also, and Autograd can compute higher-order derivatives

Some Notes

1. By default, autograd deletes the computational graph after it is evaluated
2. `retain_graph` indicates to keep it
3. Graph can compute on the gradient tensor also, and Autograd can compute higher-order derivatives
4. Specified with `create_graph = True`

Demo

▸ Colab Notebook: Backword()