Backpropagation

CSCI 601 471/671
NLP: Self-Supervised Models

https://self-supervised.cs.jhu.edu/sp2023/

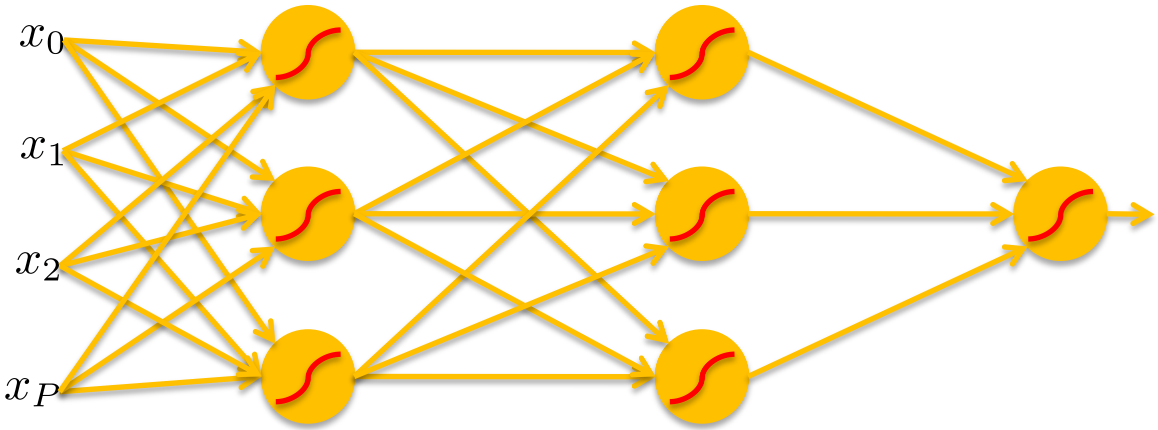
JOHNS HOPKINS UNIVERSITY

[Slide credit: Andrej Karpathy and many others]

HW update

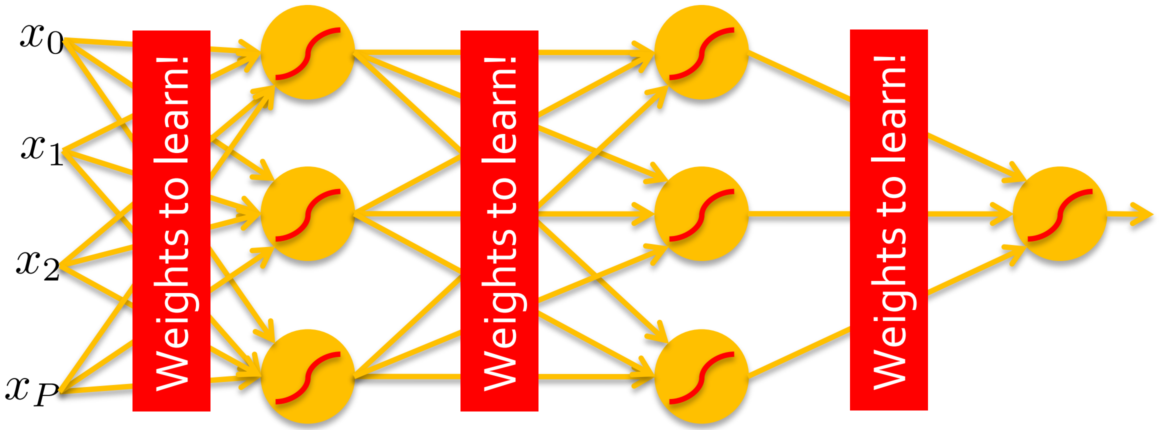
- HW1 grades are up!
  - Stats: Mean: 93.1 (std: ~5)
  - There was a mistake in grading Q4.6, but should be corrected now.
- Regrade requests can be submitted via Gradescope.
  - Please don't spam us! 🙏
- HW3 is up!
  - Focus: training neural networks

Recap: Feel Forward Neural Networks



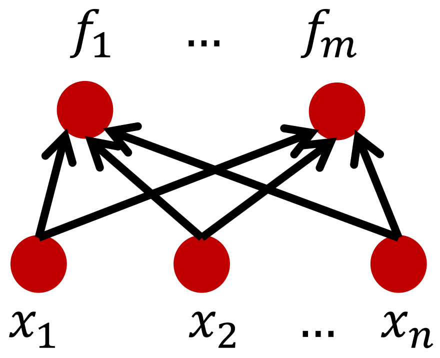
[Slide: HKUST]

Recap: Feel Forward Neural Networks



[Slide: HKUST]

Recap: Jacobian Matrix



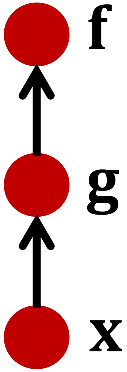
- Generalization of gradients
- Given a function with ***m* outputs** and ***n* inputs**

$$\mathbf{f}(\mathbf{x}) = [f_1(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)] \in \mathbb{R}^m$$

- It's Jacobian is an ***m* x *n* matrix** of partial derivatives:

$$\mathbf{J_f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

Recap: Chain Rule for Multivariable Functions



- Looks similar to the single-variable setup:

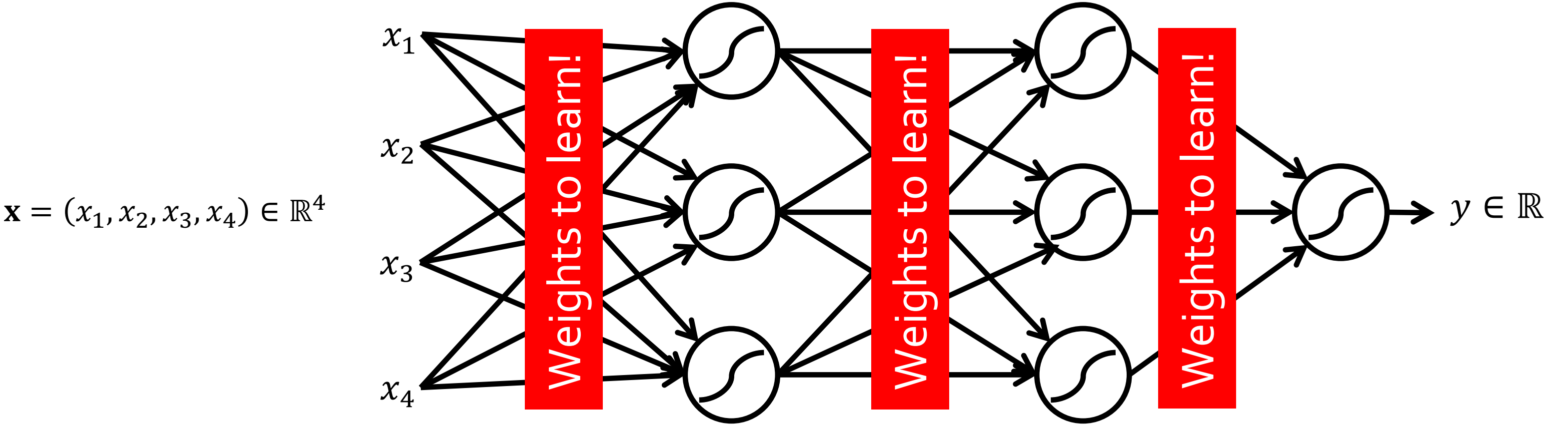
$$\mathbf{J}_{\mathbf{f}\circ\mathbf{g}}(\mathbf{x}) = \mathbf{J}_{\mathbf{f}}(\mathbf{g}(\mathbf{x}))\ \mathbf{J}_{\mathbf{g}}(\mathbf{x})$$

Note, the above statement is a **matrix** multiplication!

Function $\mathbf{f} \circ \mathbf{g}$ has *m* outputs and *d* inputs → *m* by *d* Jacobian

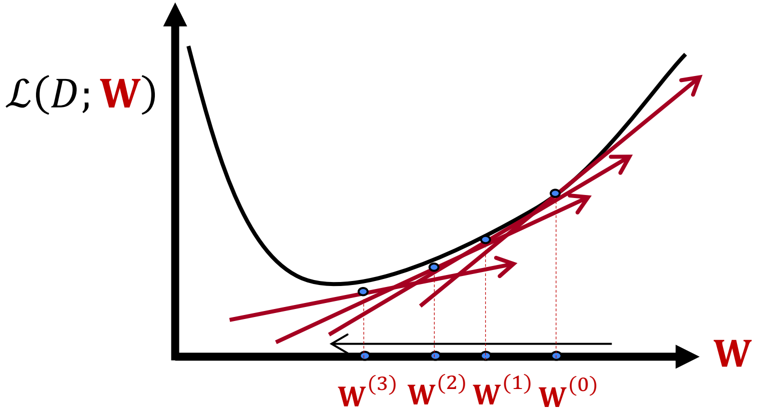
Training Neural Networks: Setup

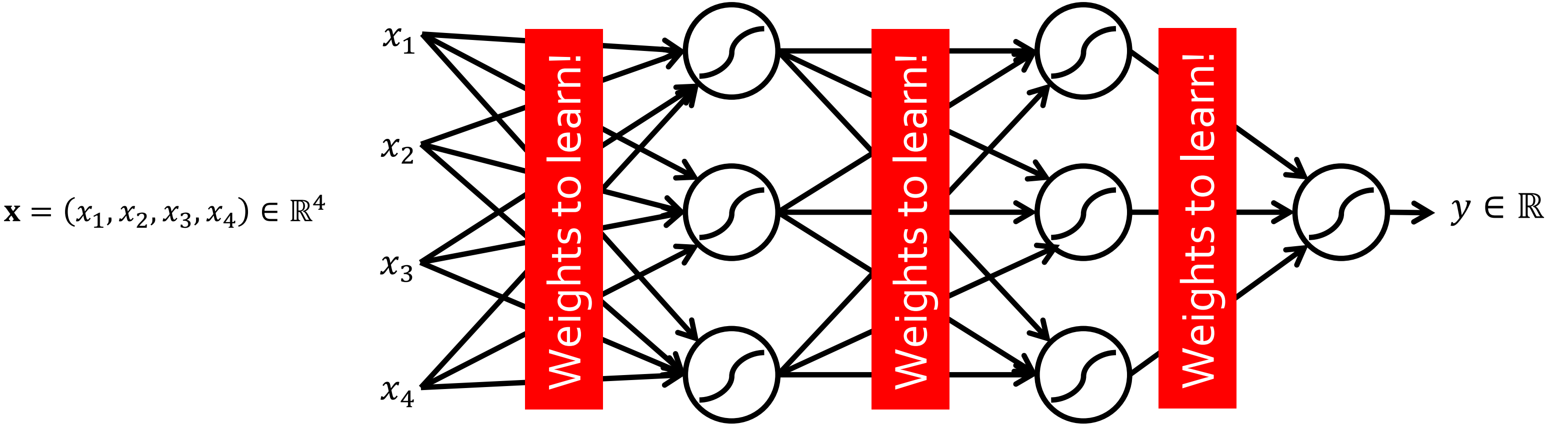
- We are given an architecture though its weights $\mathbf{W}$.
- We are given a loss function $\ell: \mathbb{R}\times\mathbb{R} \to (0, 1)$
  - $\ell(y^*, y)$ quantifies distance between an answer y^* and prediction $\mathrm{y} = \mathrm{NN}(\mathbf{x}; \mathbf{W})$ — lower is better
- Also given a training data $D = \{(\mathbf{x}_i, y_i^*)\}$
- Overall objective to optimize: $\mathcal{L}(D; \mathbf{W}) = \sum_{(\mathbf{x}_i, y_i^*)\in D} \ell(\mathrm{NN}(\mathbf{x}_i; \mathbf{W}), y_i^*)$



Training Neural Networks ~ Optimizing Parameters

- We can use gradient descent to minimizes the loss.
- At each step, the weight vector is modified in the direction that produces the steepest descent along the error surface.

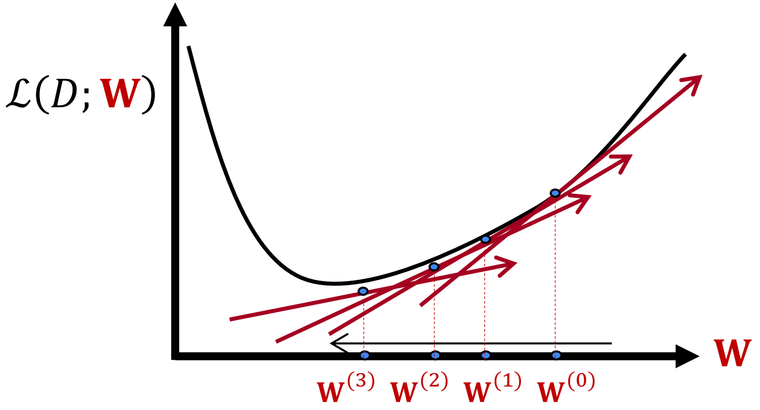




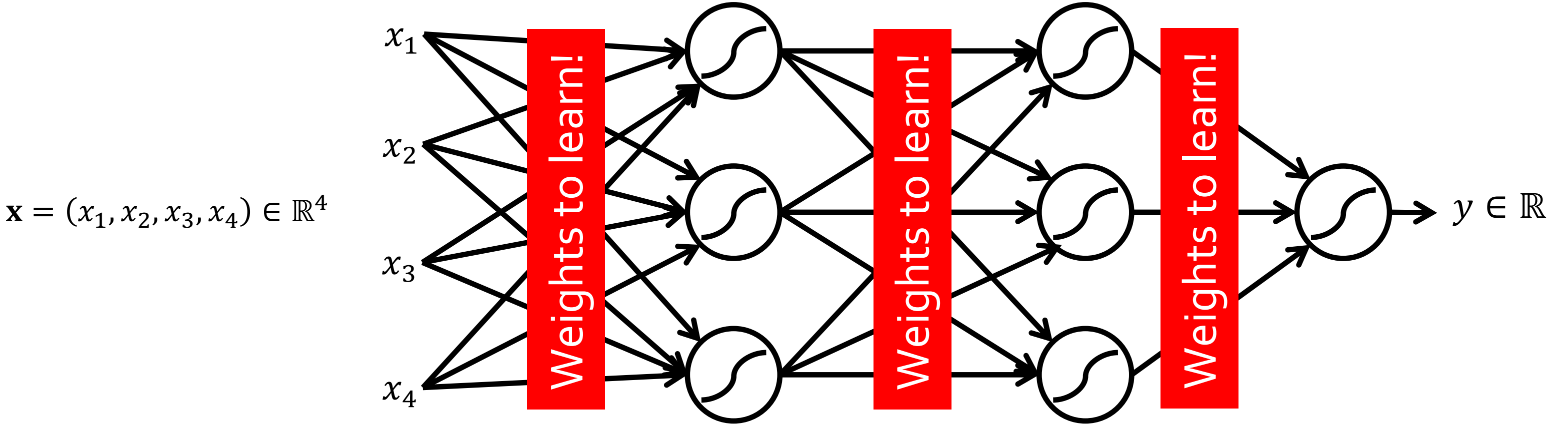
Training Neural Networks ~ Optimizing Parameters

For each sub-parameter $W_i \in \mathbf{W}$:

$$W_i^{(t+1)} = W_i^{(t)} - \alpha \frac{\partial \mathcal{L}}{\partial W_i}$$



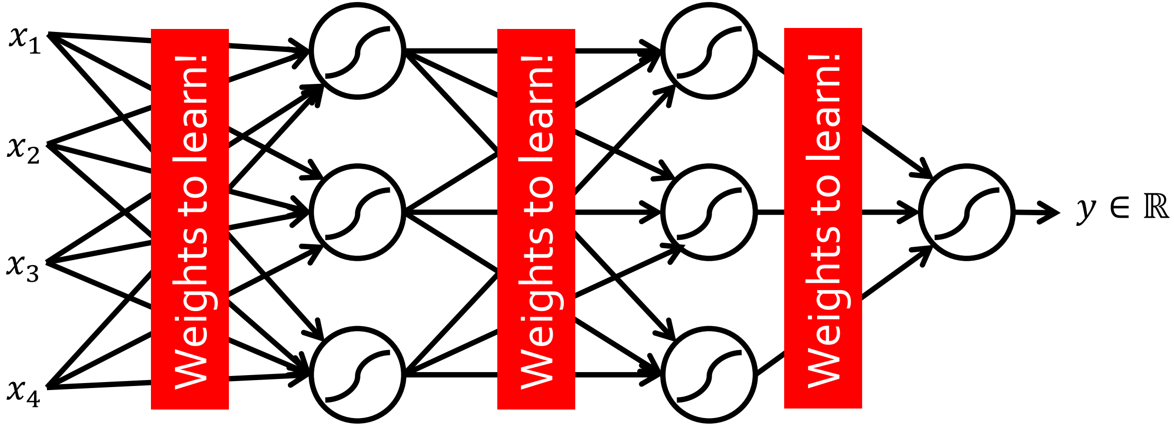
It all comes down to effectively computing $\frac{\partial \mathcal{L}}{\partial W_i}$



Training Neural Networks ~ Computing the Gradients

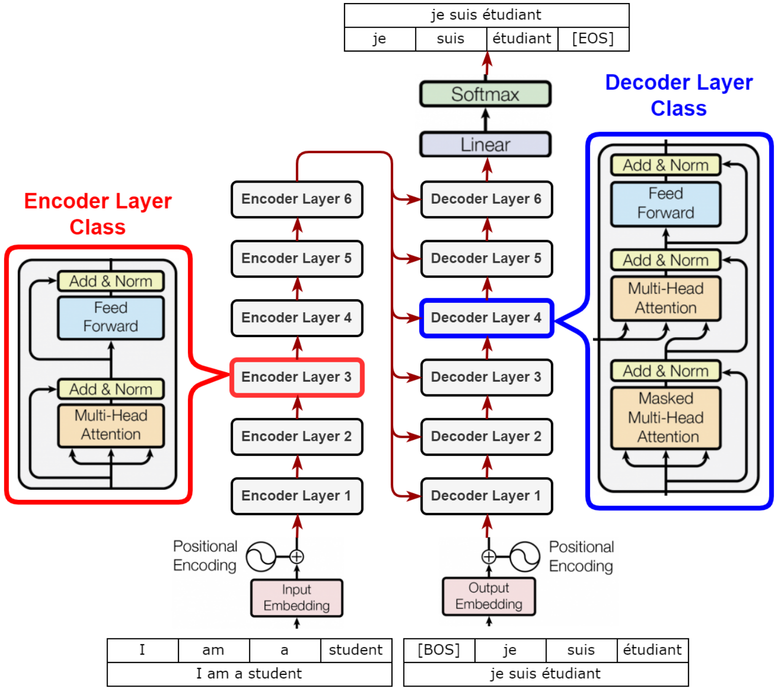
- How do you efficiently compute $\frac{\partial \mathcal{L}}{\partial W_i}$ for all parameters?
- It's easy to learn the final layer – it's just a linear unit.
- How about the weights in the earlier layers (i.e., before the final layer)?

$\mathbf{x} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$



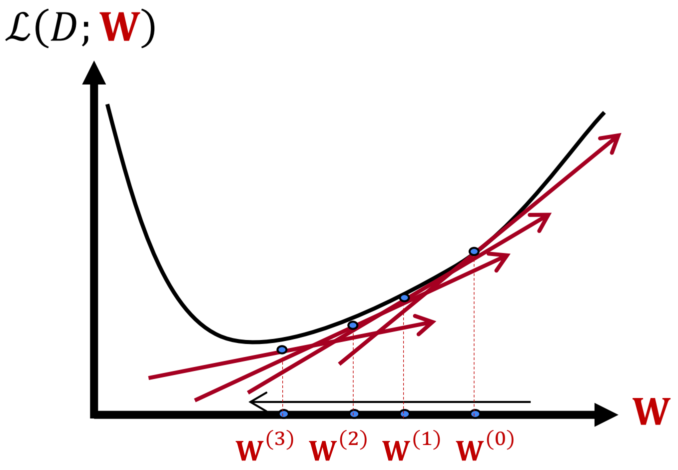
Necessity of a Principled Algorithm for Gradient Computation

- Depth gives more representational capacity to neural networks.
- However, training deep nets is not trivial.
- Even if we have analytical formula for each gradient, they can be tedious and must be repeated for each new architecture.
- The solution is “Backpropagation” algorithm!

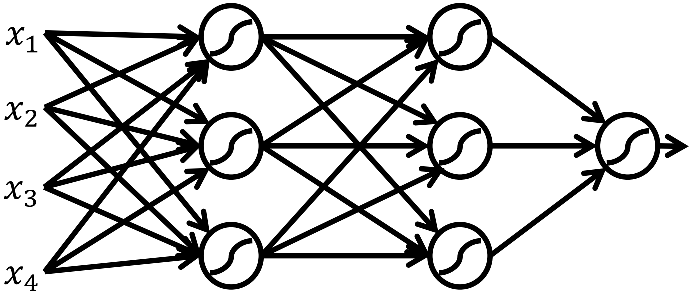


Architecture of the BERT model with over 24 layers and millions of parameters — we will study get to this model in a few weeks!

Key Intuitions Required for BP



1. Gradient Descent
   - Change the weights **W** in the direction of gradient to minimize the error function.
2. Chain Rule
   - Use the chain rule to calculate the weights of the intermediate weights
3. Dynamic Programming (Memoization)
   - Memoize the weight updates to make the updates faster.



A Generic Neural Network

- Given the following definition:

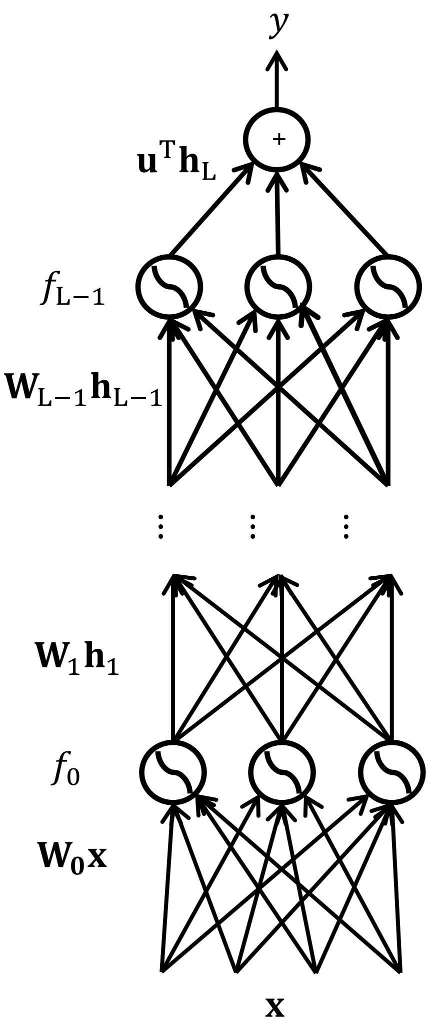
$\mathbf{x} = \mathbf{h}_0 \in \mathbb{R}^{d_0}$ (input)

$\mathbf{h}_{i+1} = f_i(\mathbf{W}_i\mathbf{h}_i) \in \mathbb{R}^{d_i}$ (hidden layer i , $0 \leq i \leq L-1$)

$y = \mathbf{u}^{\mathrm{T}}\mathbf{h}_L \in \mathbb{R}$ (output)

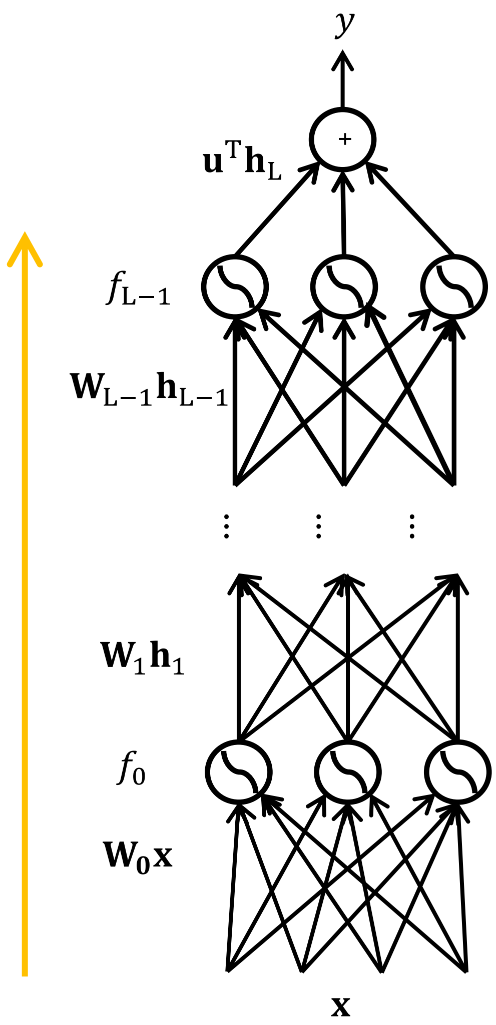
$\mathcal{L} = \ell(y, y^*) \in \mathbb{R}$ (loss)

- Trainable parameters: $\mathbf{W} = \{\mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_L, \mathbf{u}\}$



A Generic Neural Network: Forward Step

- Given some [initial] values for the parameters, we can compute the forward pass, layer by layer.
- Forward pass is basically L matrix multiplications, each followed by an activation function.
- Matrix multiplication can be done efficiently with GPUs.
  - Therefore, forward pass is somewhat fast.
- Complexity of forward pass, linear of depth $O(L)$.



A Generic Neural Network: Direct Gradients

$\mathbf{x} = \mathbf{h}_0 \in \mathbb{R}^{d_0}$ (input)	$y = \mathbf{u}^{\mathrm{T}}\mathbf{h}_L \in \mathbb{R}$ (output)
$\mathbf{h}_{i+1} = f_i(\mathbf{W}_i \mathbf{h}_i) \in \mathbb{R}^{d_i}$	$\mathcal{L} = \ell(y, y^*) \in \mathbb{R}$ (loss)
$(0 \le i \le L-1)$	$\mathbf{W} = \{\mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_L, \mathbf{u}\}$

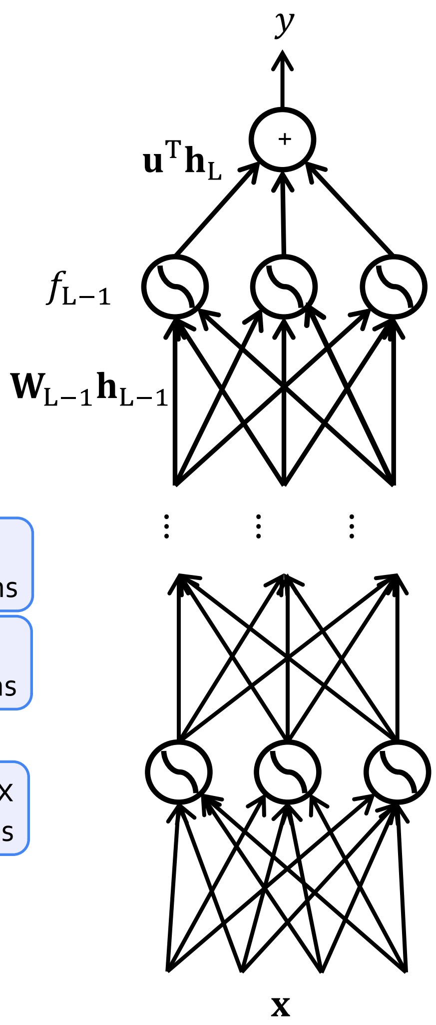
We want the gradients of $\mathcal{L}$ with respect to model parameters.

- $\nabla_{\mathcal{L}}(\mathbf{W}_{L-1}) = \left(\mathbf{J}_{\mathcal{L}}(\mathbf{W}_{L-1})\right)^{\mathrm{T}} = \left(\mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{W}_{\mathrm{L}-1})\right)^{\mathrm{T}}$ — 3 matrix multiplications
- $\nabla_{\mathcal{L}}(\mathbf{W}_{L-2}) = \left(\mathbf{J}_{\mathcal{L}}(\mathbf{W}_{L-2})\right)^{\mathrm{T}} = \left(\mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{h}_{\mathrm{L}-1})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}-1}}(\mathbf{W}_{\mathrm{L}-2})\right)^{\mathrm{T}}$ — 4 matrix multiplications
- …
- $\nabla_{\mathcal{L}}(\mathbf{W}_{0}) = \left(\mathbf{J}_{\mathcal{L}}(\mathbf{W}_{L-3})\right)^{\mathrm{T}} = \left(\mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{h}_{\mathrm{L}-1}) \dots \mathbf{J}_{\mathbf{h}_{1}}(\mathbf{W}_{0})\right)^{\mathrm{T}}$ — $L+2$ matrix multiplications

In total, how many matrix multiplications are done here?

(A) $O(L)$ (B) $O(L^2)$ (C) $O(L^3)$ (C) $O(\exp(L))$

Can we do better than this? 🤔



A Generic Neural Network: Gradients with Caching/Memoization

$$\nabla_{\mathcal{L}}(\mathbf{W}_{L-1}) = \left(\mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{W}_{\mathrm{L}-1})\right)^{\mathrm{T}} = \left(\delta_L\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{W}_{\mathrm{L}-1})\right)^{\mathrm{T}}$$

$$\nabla_{\mathcal{L}}(\mathbf{W}_{L-2}) = \left(\mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{h}_{\mathrm{L}-1})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}-1}}(\mathbf{W}_{\mathrm{L}-2})\right)^{\mathrm{T}} = \left(\delta_{L-1}\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}-1}}(\mathbf{W}_{\mathrm{L}-2})\right)^{\mathrm{T}}$$

$$\dots$$

$$\nabla_{\mathcal{L}}(\mathbf{W}_{0}) = \left(\mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{h}_{\mathrm{L}-1}) \dots \mathbf{J}_{\mathbf{h}_{1}}(\mathbf{W}_{0})\right)^{\mathrm{T}} = \left(\delta_{1}\, \mathbf{J}_{\mathbf{h}_{1}}(\mathbf{W}_{0})\right)^{\mathrm{T}}$$

- Parameter gradients depend on the gradients of the earlier layers!
- So, when computing gradients at each layer, we don't need to start from scratch!
- I can **reuse** gradients computed for higher layers for lower layers (i.e., memoization).

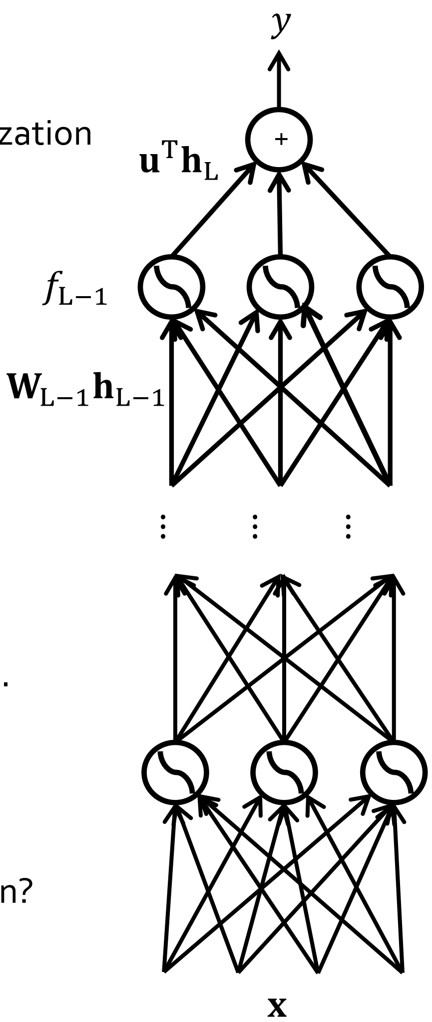
Let δ_i denote Jacobian at the output of layer i:

$$\delta_i = \mathbf{J}_{\ell}(\mathrm{y})\, \mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\, \mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{h}_{\mathrm{L}-1}) \dots \mathbf{J}_{\mathbf{h}_{i}}(\mathbf{h}_{i-1})$$

$$\delta_i = \delta_{i+1}\, \mathbf{J}_{\mathbf{h}_{i}}(\mathbf{h}_{i-1})$$

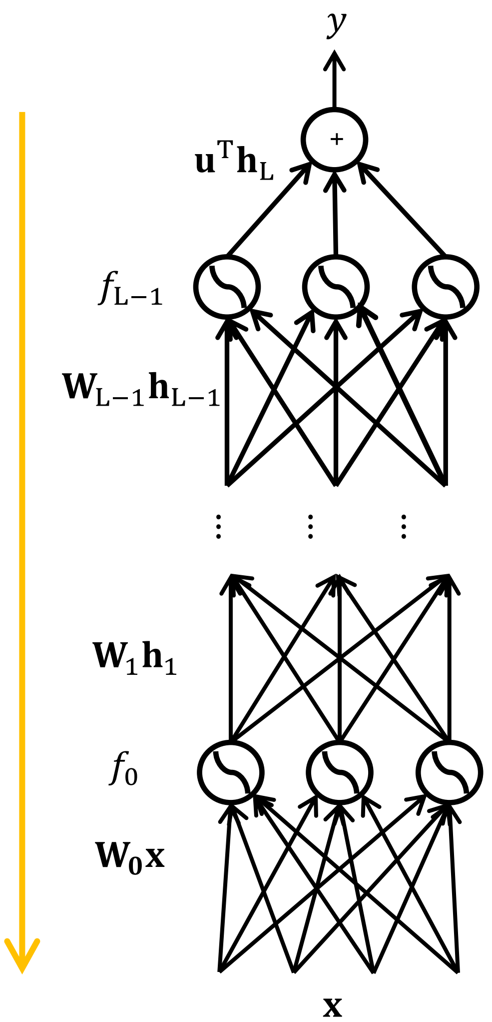
In total, how many matrix multiplications are done here when using caching/memoization?

(A) $O(L)$ (B) $O(L^2)$ (C) $O(L^3)$ (C) $O(\exp(L))$



A Generic Neural Network: Backward Step

- Backward step computes the gradients starting from the end to the beginning, layer by layer.
- Start by computing local gradients: $\mathbf{J}_{\mathbf{h}_i}(\mathbf{h}_{i-1})$
- Use then to compute upstream gradients δ_L, then δ_{L-1}, then δ_{L-2}, ….
- Use these to compute global gradients: $\nabla_{\mathcal{L}}(\mathbf{W}_i)$
- Computational cost as a function of depth:
  - With memoization, gradient computation is a **linear** function of depth L
    - (same cost as the forward process!!)
  - Without memorization, gradients computation would grow **quadratic** with L



A Generic Neural Network: Back Propagation

Initialize network parameters with random values

Loop until convergence

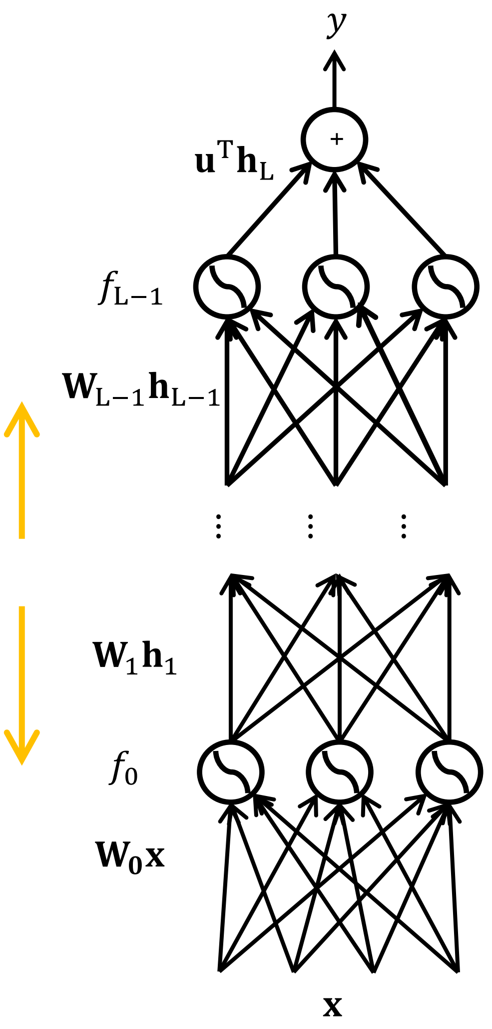
  Loop over training instances

In practice, this step is done over **batches** of instances!

i. **Forward step:**
   Start from the input and compute all the layers till the end (loss $\mathcal{L}$)

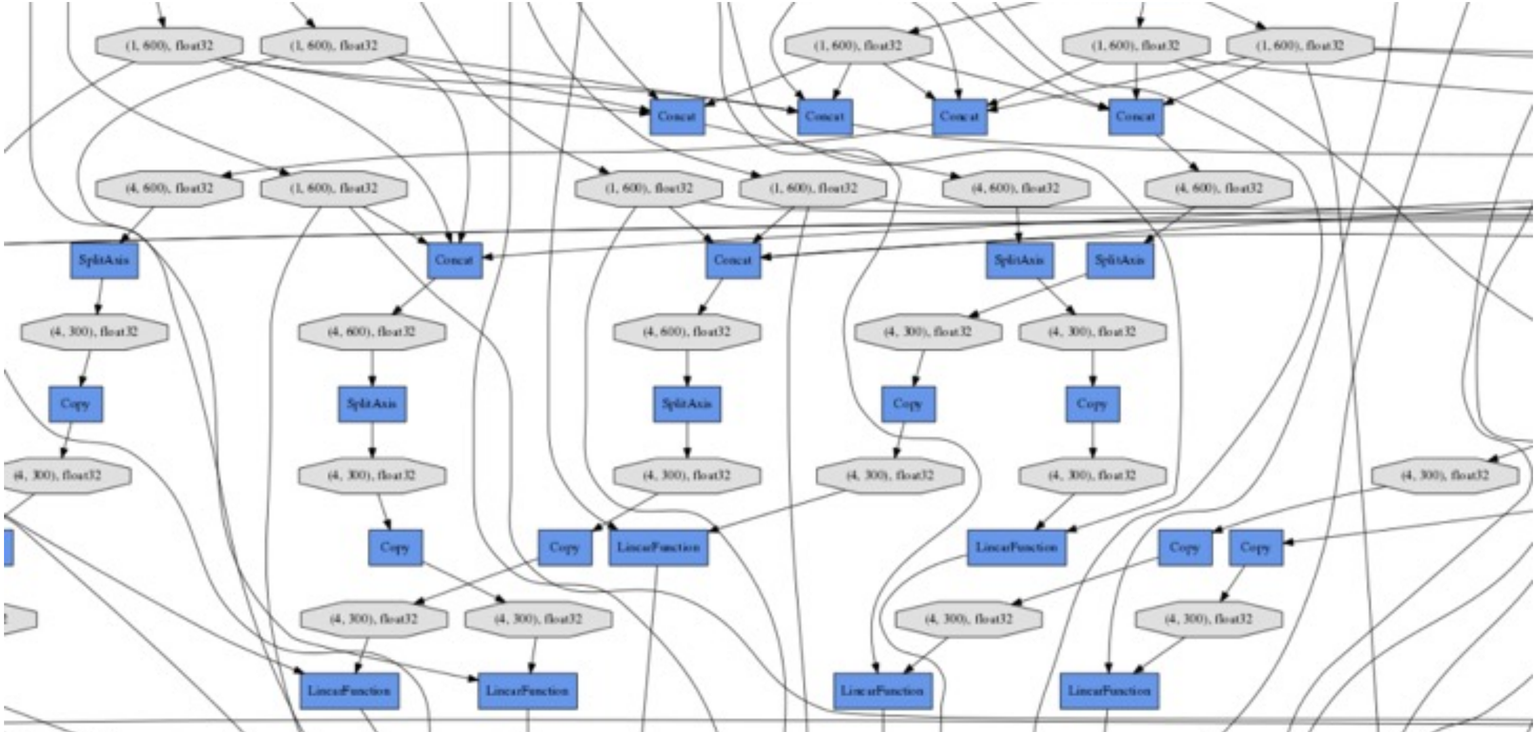
ii. **Backward step:**
   Compute local gradients, starting from the last layer
   Compute upstream gradients δ_i values, starting from the last layer
   Use δ_i values to compute global gradients $\nabla_{\mathcal{L}}(\mathbf{W}_i)$ at each layer

iii. **Gradient update:**
   Update each parameter: $\mathbf{W}_i^{(t+1)} \leftarrow \mathbf{W}_i^{(t)} - \alpha \, \nabla_{\mathcal{L}}(\mathbf{W}_i)$



Computation Graph: Example

- In reality, networks are not as regular as the previous example …

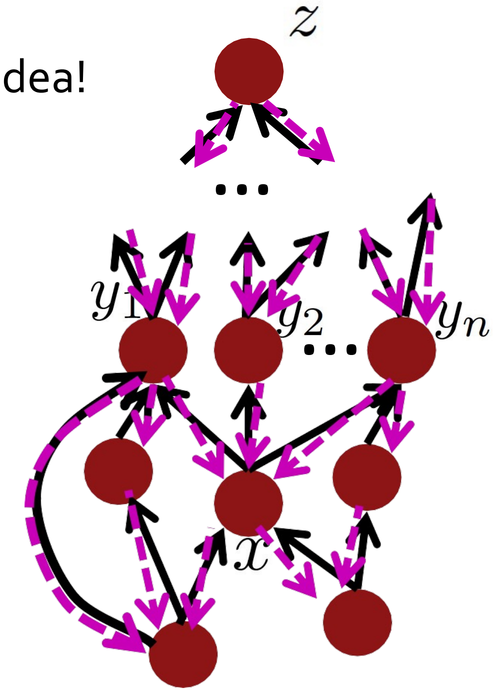


Back-Prop in General Computation Graph

- What if the network does not have a regular structure? Same idea!

- Sort the nodes in topological order (what depends on what)
- Forward-Propagation:
  - Visit nodes in topological sort order and compute value of node given predecessors
- Backward-Propagation:
  - Compute local gradients
  - Visit nodes in reverse order and compute global gradients using gradients of successors

Single scalar output

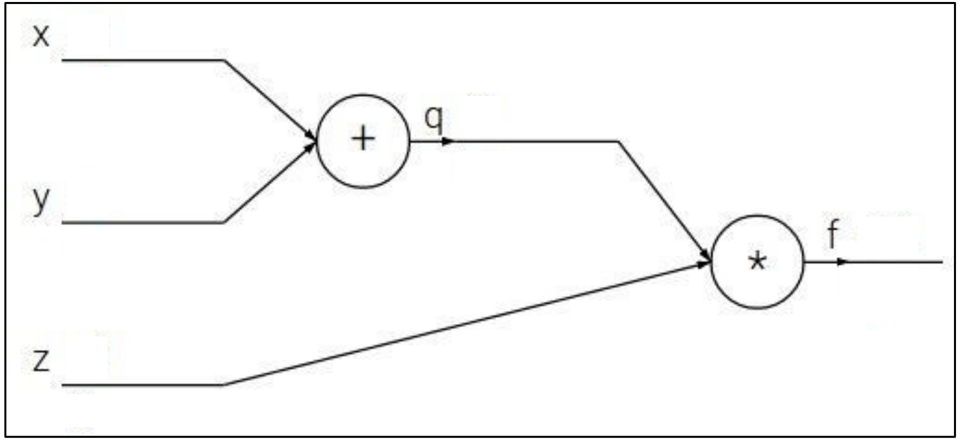


Inputs

Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4

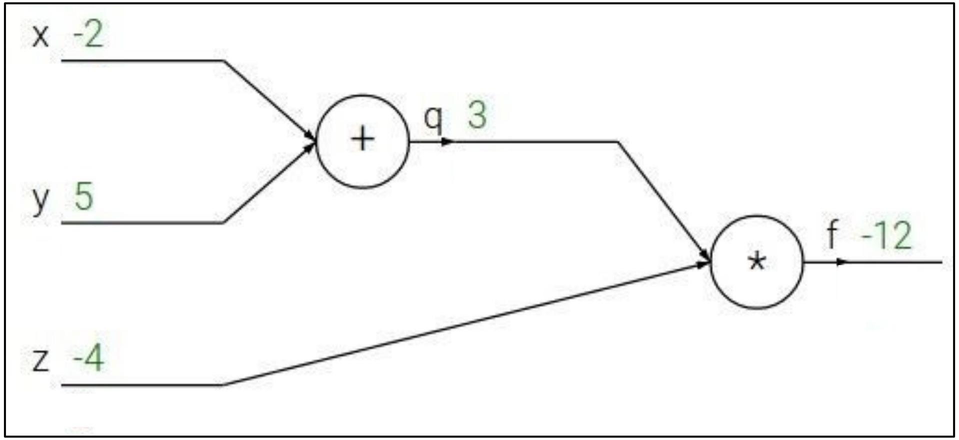


[Slide: Stanford CS231N]

Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4

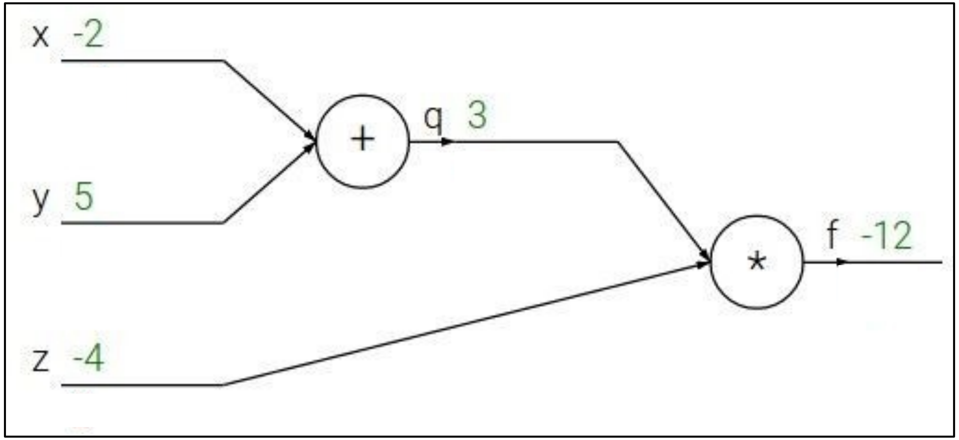


Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!



[Slide: Stanford CS231N]

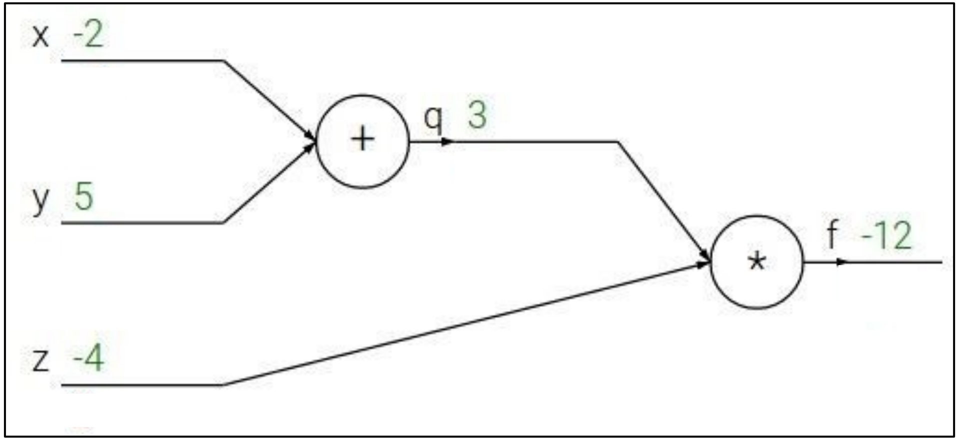
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



[Slide: Stanford CS231N]

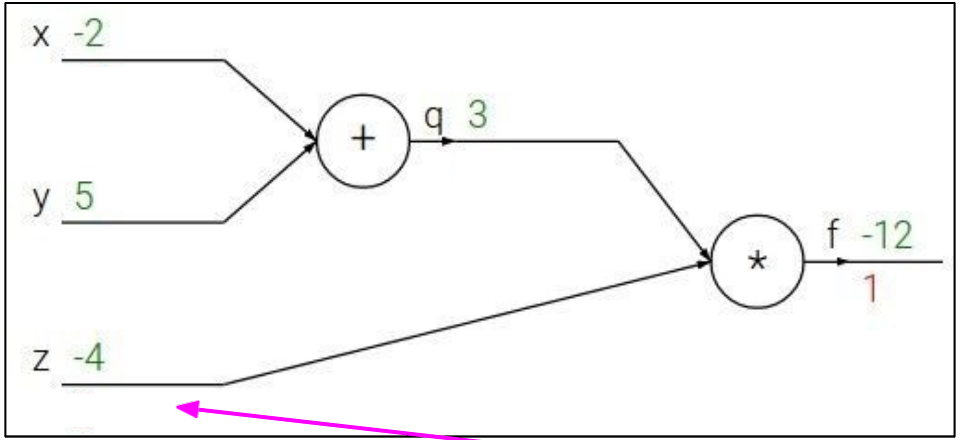
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial z}$$

[Slide: Stanford CS231N]

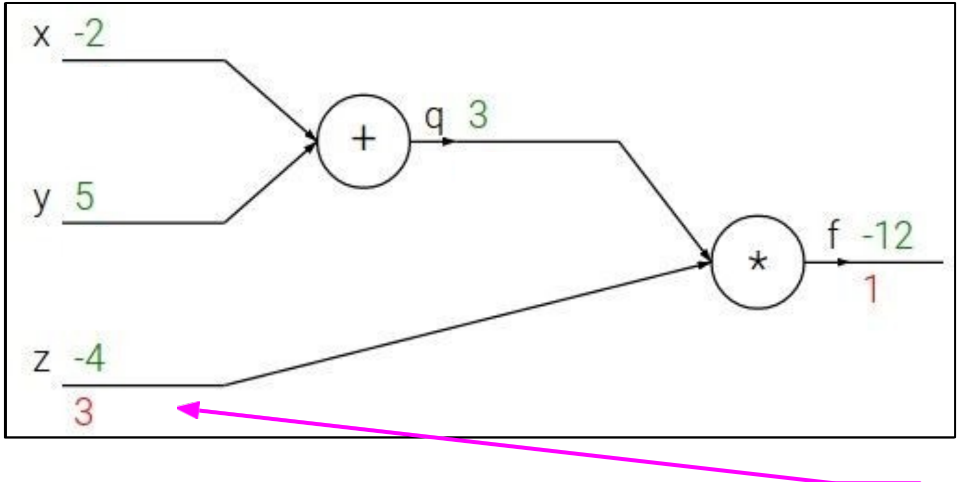
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial z}$$

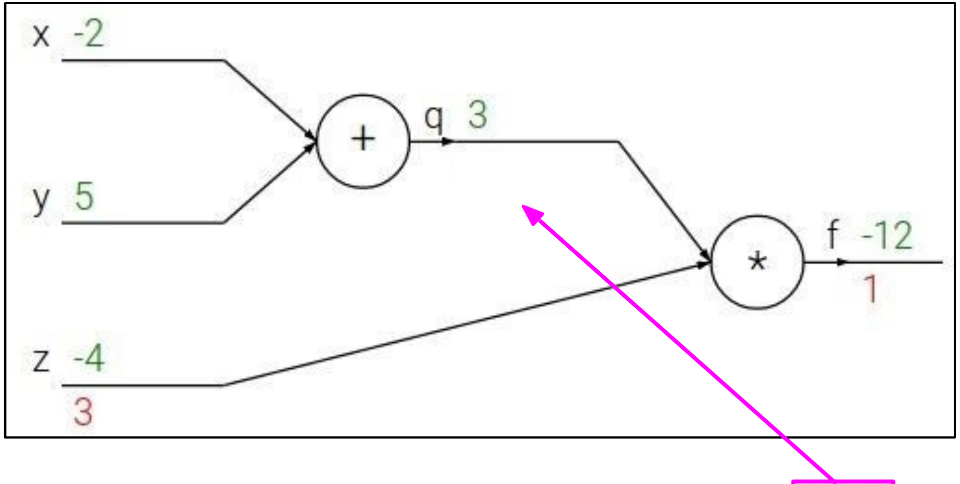
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial q}$$

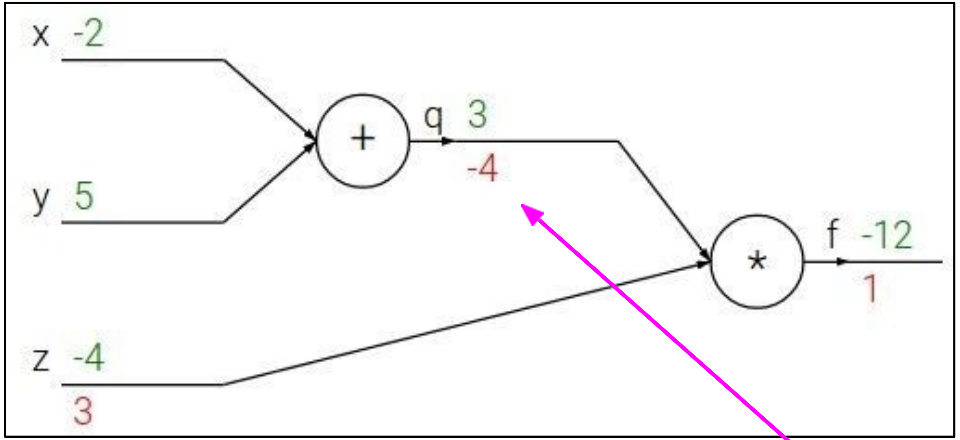
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial q}$$

[Slide: Stanford CS231N]

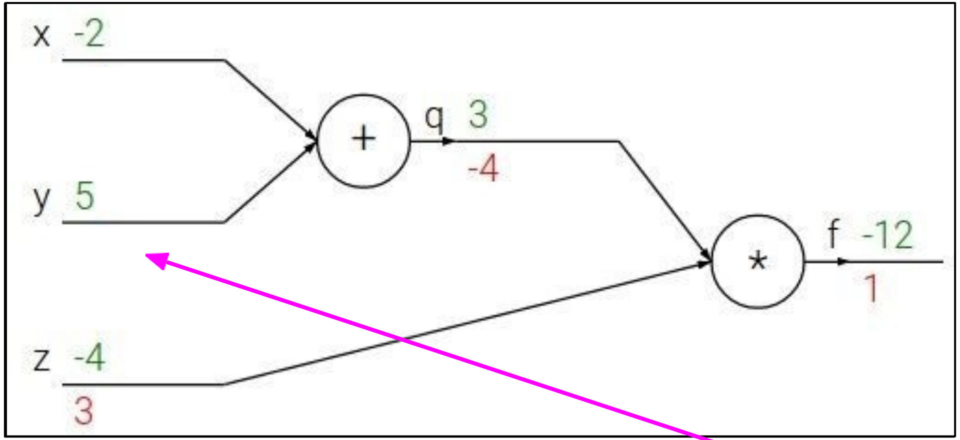
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial y}$$

[Slide: Stanford CS231N]

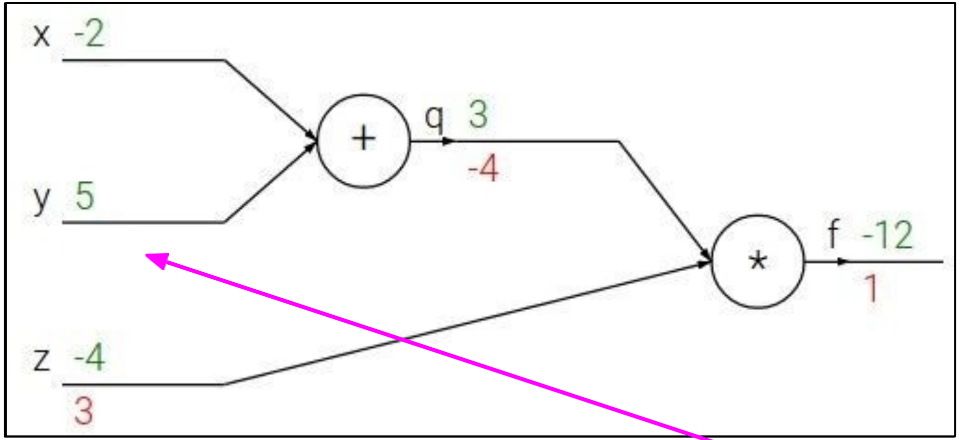
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

[Slide: Stanford CS231N]

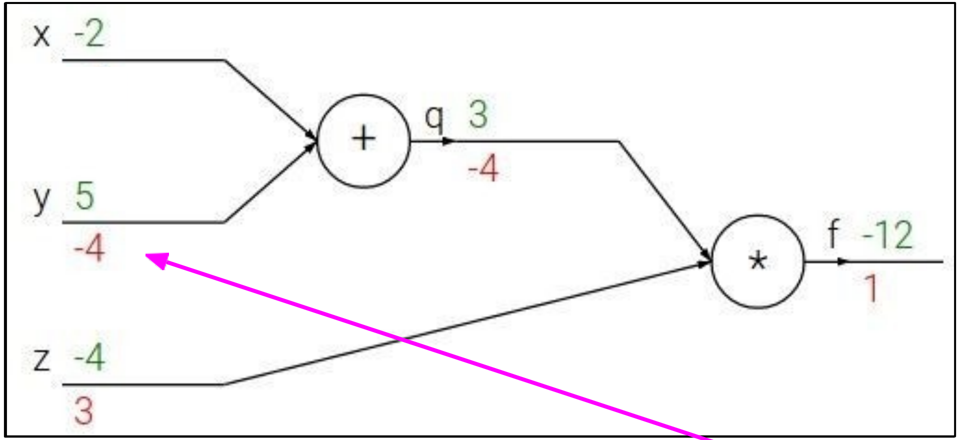
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

$$\frac{\partial f}{\partial y}$$

[Slide: Stanford CS231N]

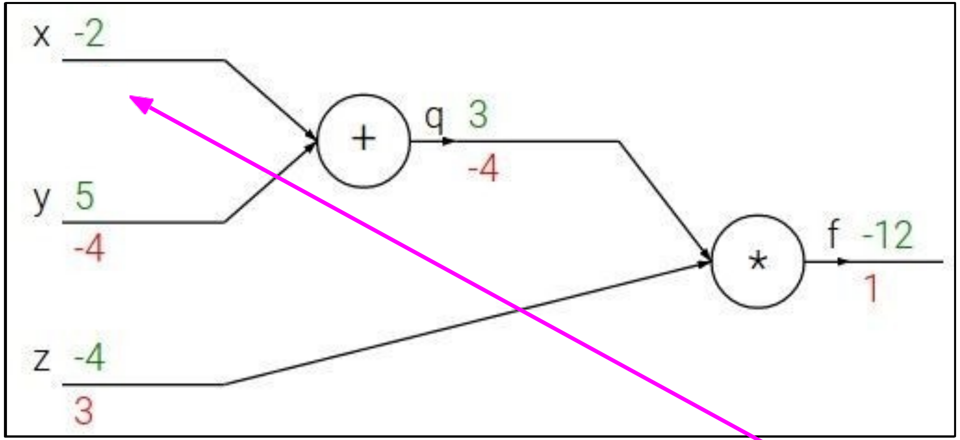
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



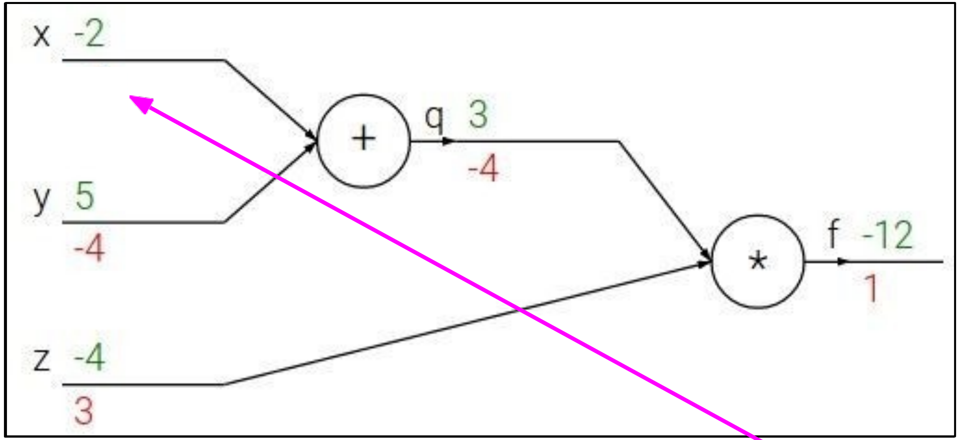
$$\frac{\partial f}{\partial x}$$

[Slide: Stanford CS231N]

Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

$$\frac{\partial f}{\partial x}$$

[Slide: Stanford CS231N]

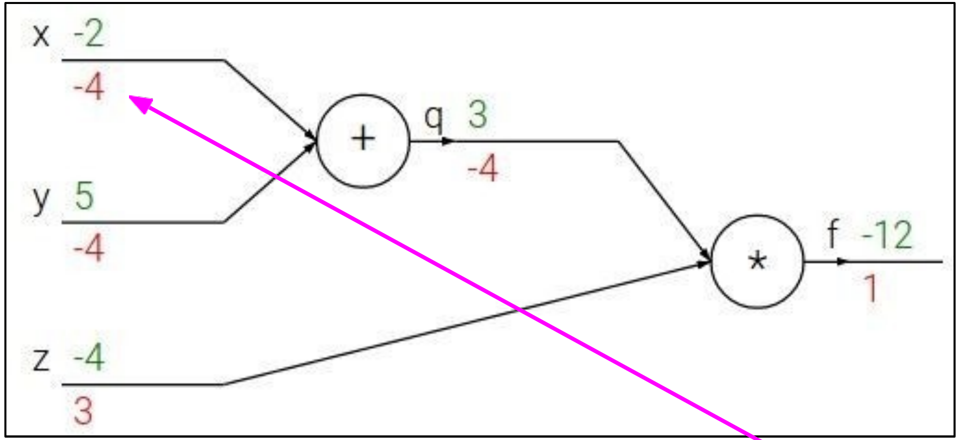
Computation Graph: An Example

$$f(x, y, z) = (x + y)z$$

- Evaluated at: x = -2, y = 5, z = -4
- Start with local gradients!

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

$$\frac{\partial f}{\partial x}$$

[Slide: Stanford CS231N]

A Generic Example

Figure from Andrej Karpathy

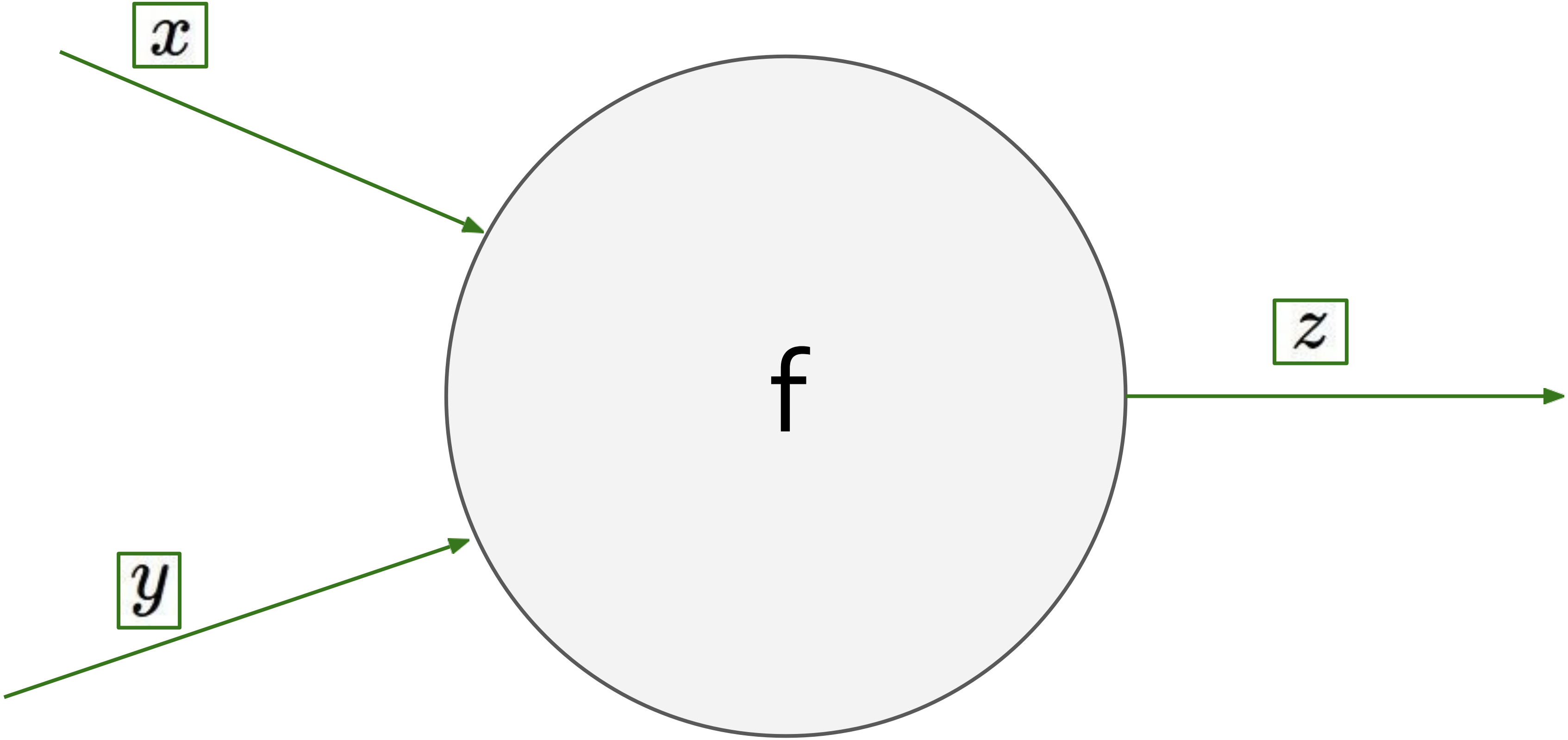


Figure from Andrej Karpathy

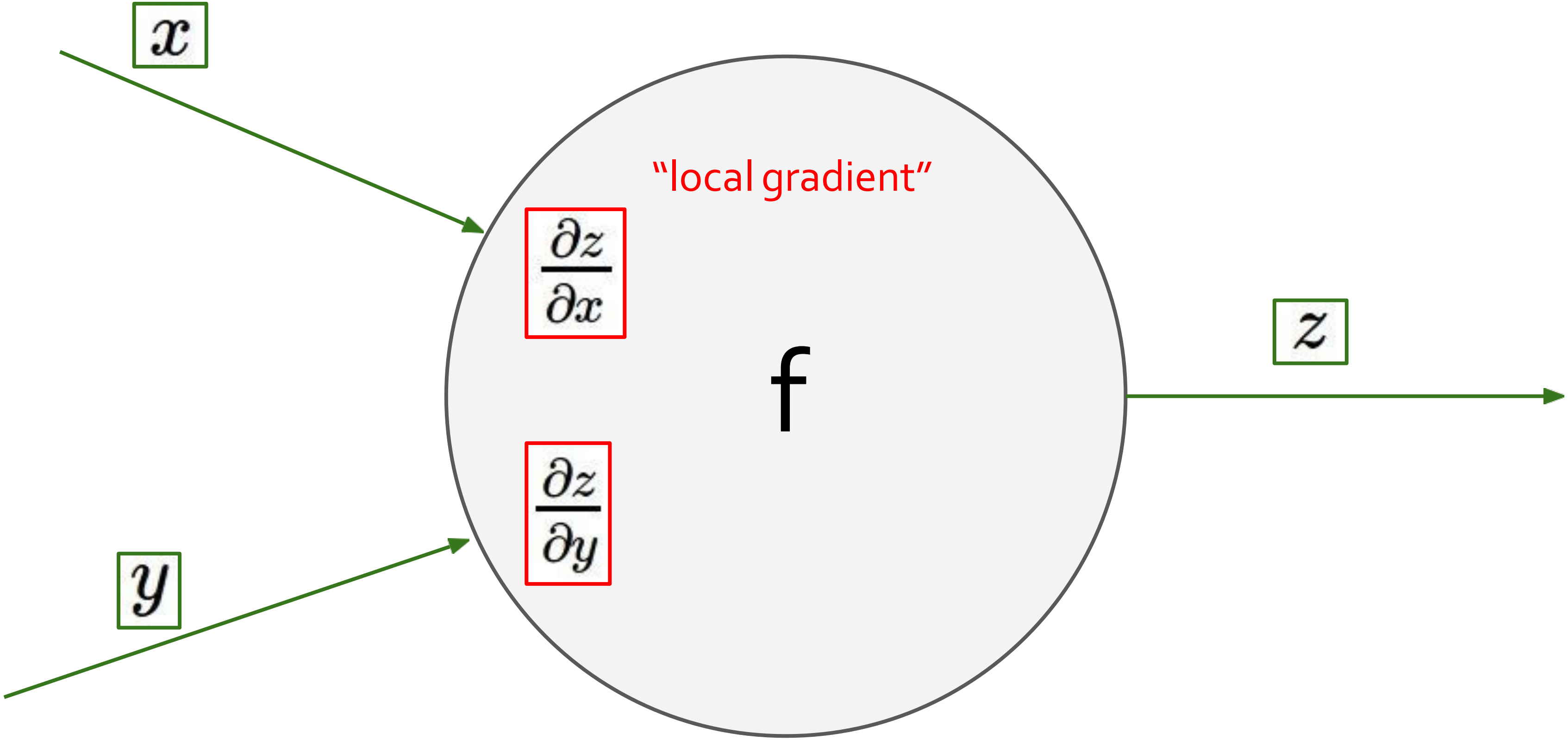


Figure from Andrej Karpathy

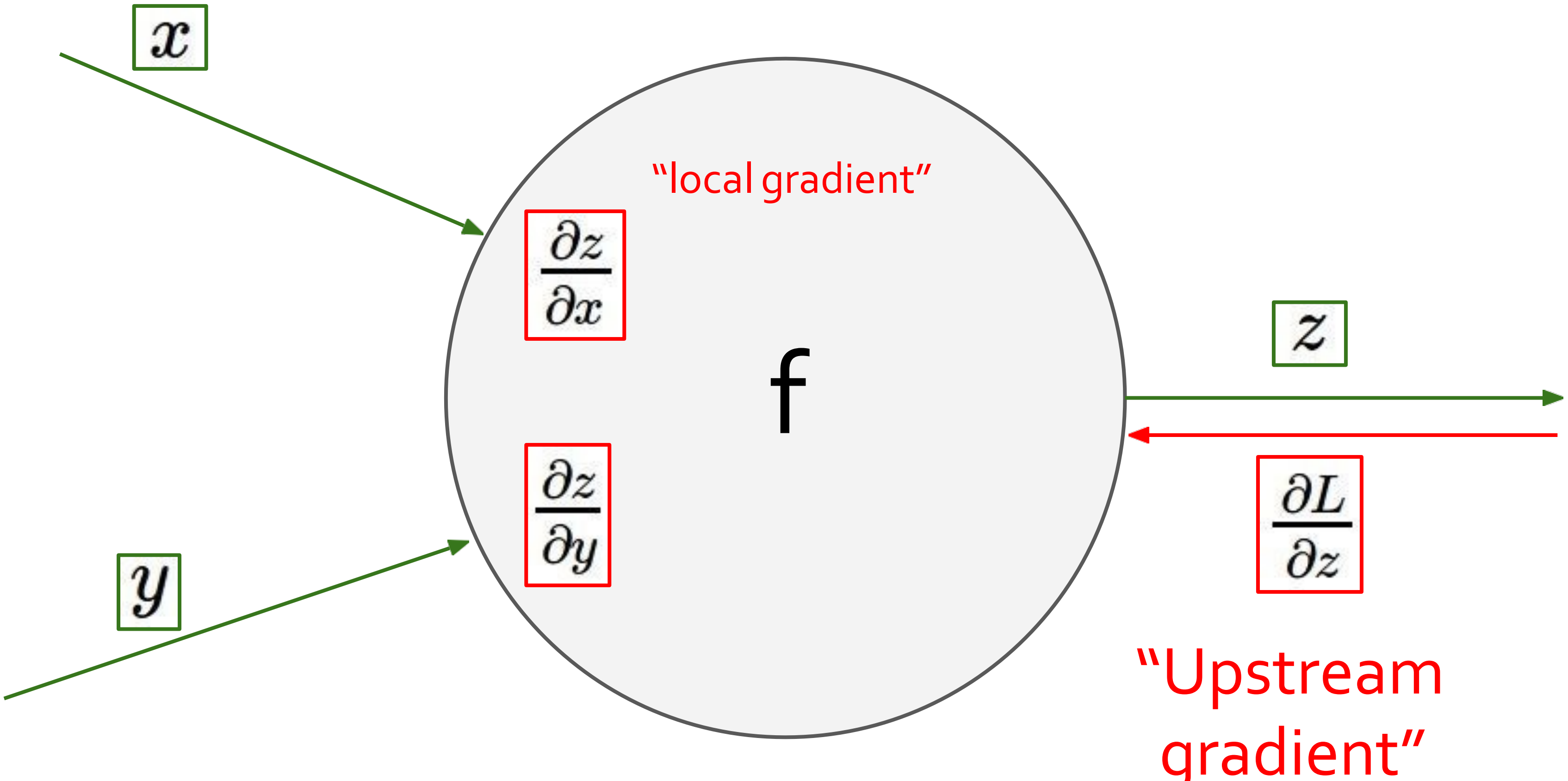


Figure from Andrej Karpathy

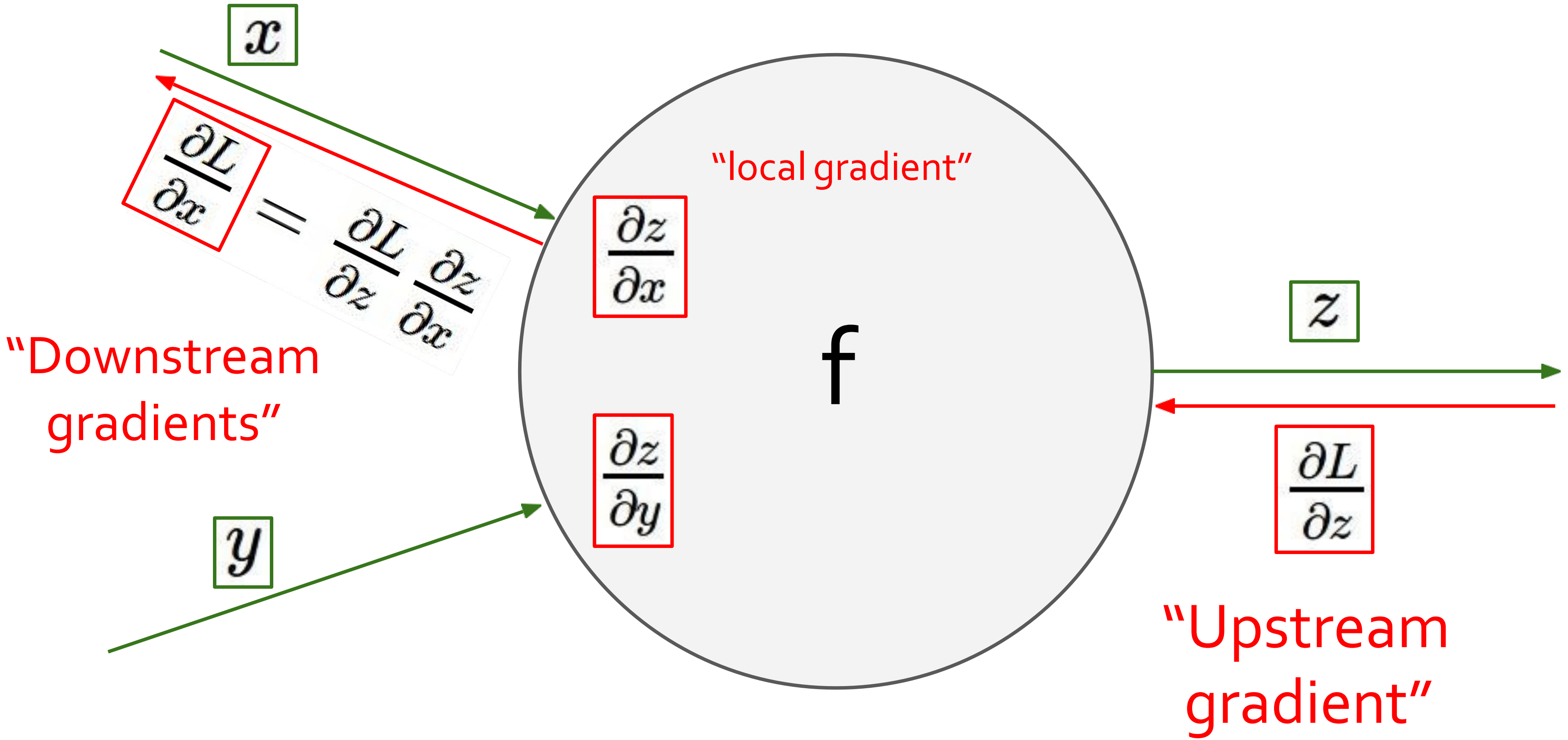


Figure from Andrej Karpathy

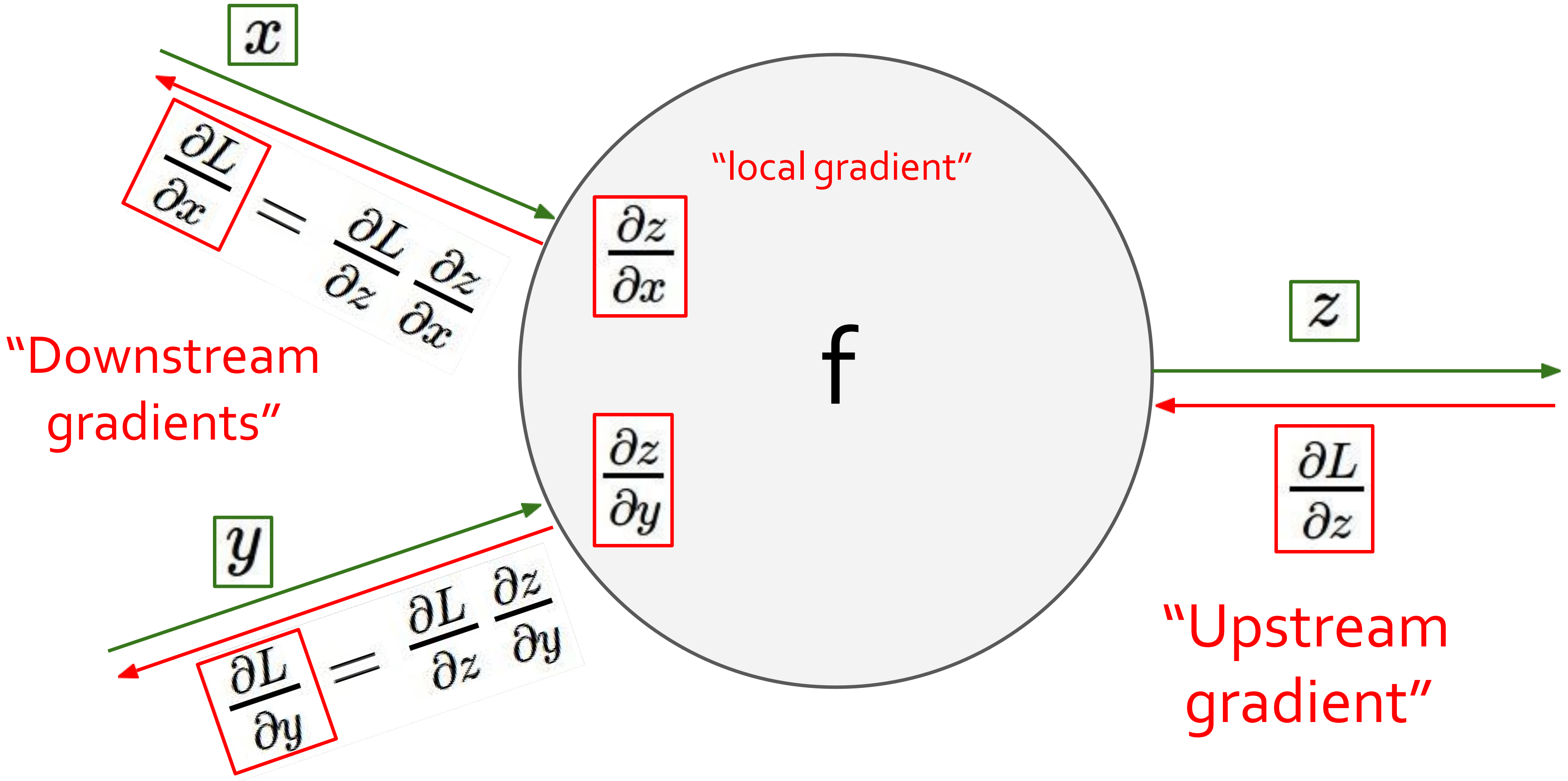


Figure from Andrej Karpathy

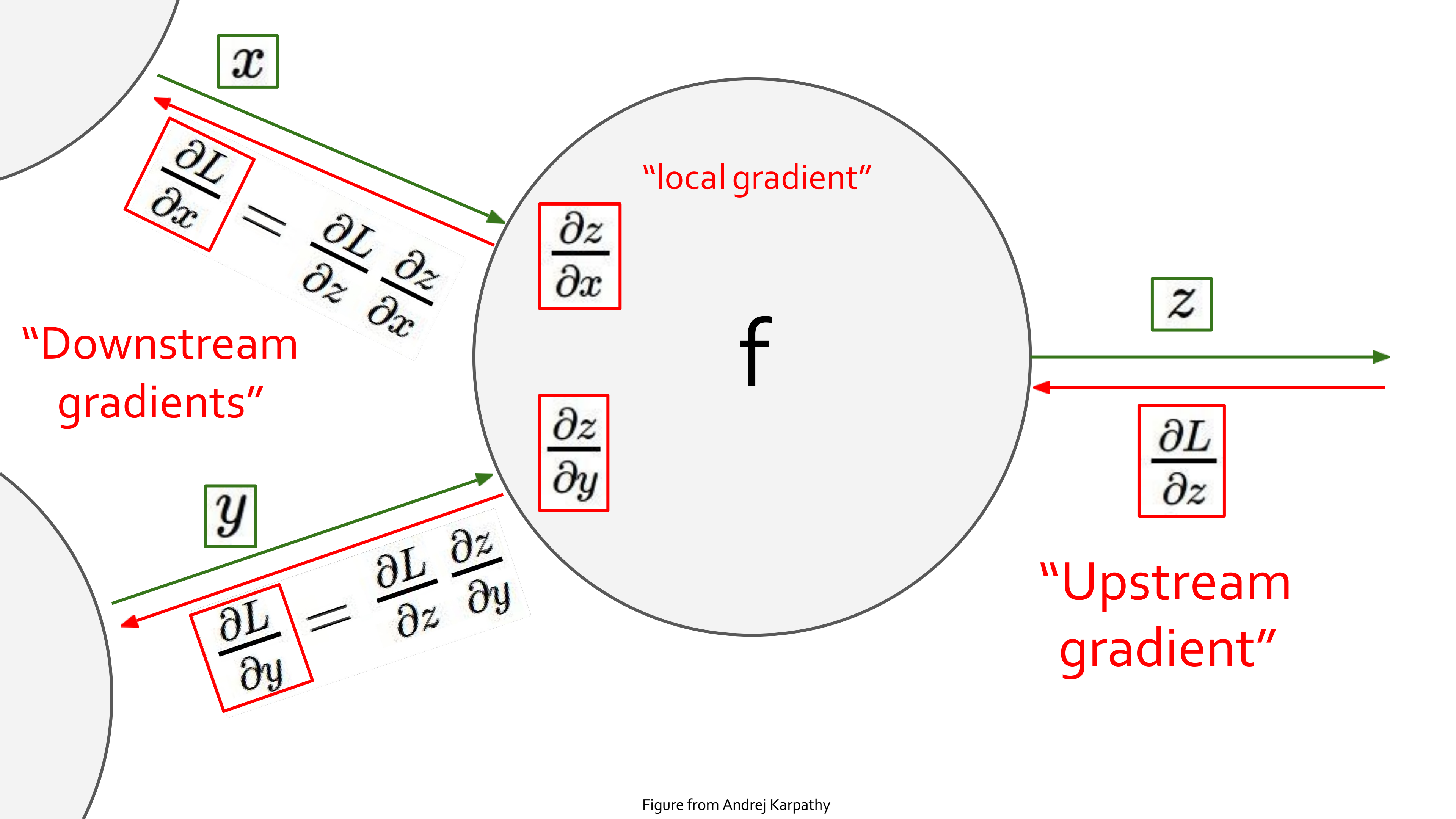


Figure from Andrej Karpathy

Demo time!

- Link: https://playground.tensorflow.org/

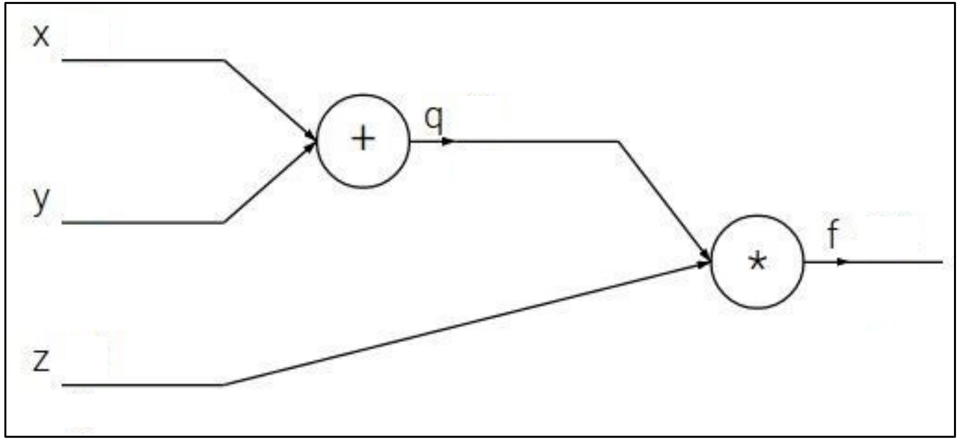
Chapter Plan

1. Feed-forward networks
2. Neural nets: brief history
3. Word2Vec as a simple neural network
4. Training neural networks: back-propagation
5. Backprop in practice

Backprop in PyTorch

$$f(x, y, z) = (x + y)z$$

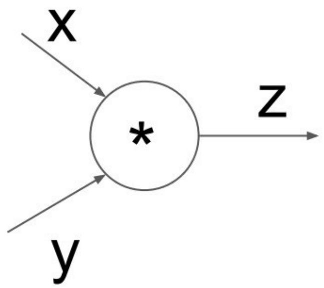
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$




```
x = torch.tensor(-2.0, requires_grad=True)
y = torch.tensor(5.0, requires_grad=True)
z = torch.tensor(-4.0, requires_grad=True)

f = (x+y)*z # Define the computation graph

f.backward() # PyTorch's internal backward gradient computation

print('Gradients after backpropagation:', x.grad, y.grad, z.grad)
```

PyTorch’s Implementation: Forward/Backward API

- PyTorch has implementation of forward/backward operations for various operators.
- Example: multiplication operator




```python
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

Need to cash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

PyTorch Operators

PyTorch's lower-level functions translate activities to graphics processor via libraries like OpenGL

pytorch / pytorch Public

Watch 1.6k | Fork 17.1k | Star 61.4k

Code | Issues 5k+ | Pull requests 789 | Actions | Projects 28 | Wiki | Security | Insights

d0a4e2e782 — pytorch / aten / src / ATen / native / vulkan / glsl / — Go to file

manuelcandales and pytorchmergebot [Vulkan] Enable copying QInt8 and QInt32 tensors from cpu to vulkan. (#... — ✓ 3297365 last month — History

..		
templates	[Pytorch][Vulkan] Templatize depth wise convolution and specialize fo...	last month
adaptive_avg_pool2d.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
add.glsl	[Vulkan] Implement arithmetic ops where one of the arguments is a ten...	5 months ago
add_.glsl	[Vulkan] Implement arithmetic ops where one of the arguments is a ten...	5 months ago
add_scalar.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
add_scalar_.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
addmm.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
avg_pool2d.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
batchnorm.glsl	[vulkan] enable prepacking for Batchnorm op (#88433)	2 months ago
buffer_to_buffer.glsl	[vulkan] Add option for buffer representations in vTensor (#87622)	2 months ago
cat_feature.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
clamp.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
clamp_.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
conv2d.glsl	[pytorch][vulkan] Add bias storage type to template (#88324)	2 months ago
conv2d_dw.glsl	[Pytorch][vulkan] Simplify depthwise conv to remove bounds compute (#...	last month
conv_transpose2d.glsl	[pytorch][vulkan] Add bias storage type to template (#88324)	2 months ago
cumsum.glsl	[Vulkan] Add cumsum op (#78554)	7 months ago
dequantize.glsl	[vulkan] implement dequantize (#81493)	5 months ago
div.glsl	[Vulkan] Implement arithmetic ops where one of the arguments is a ten...	5 months ago
div_.glsl	[Vulkan] Implement arithmetic ops where one of the arguments is a ten...	5 months ago
glu_channel.glsl	[Vulkan] Generalize GLU along channel dim to support input of any eve...	5 months ago
glu_channel_mul4.glsl	[Vulkan] Generalize GLU along channel dim to support input of any eve...	5 months ago
hardshrink.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
hardshrink_.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
hardsigmoid.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year

mul_scalar_.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
nchw_to_image.glsl	[vulkan] Enable 2D texture types (#86971)	2 months ago
nchw_to_image2d.glsl	[vulkan] Enable 2D texture types (#86971)	2 months ago
nchw_to_image_int32.glsl	[Vulkan] Enable copying QInt8 and QInt32 tensors from cpu to vulkan. (#...	last month
nchw_to_image_int8.glsl	[Vulkan] Enable copying QInt8 and QInt32 tensors from cpu to vulkan. (#...	last month
nchw_to_image_uint8.glsl	[Vulkan] Enable copying QInt8 and QInt32 tensors from cpu to vulkan. (#...	last month
permute_4d.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
quantize_per_tensor_qint32.glsl	[Vulkan] Enable QInt8 and QInt32 quantization (#89788)	last month
quantize_per_tensor_qint8.glsl	[Vulkan] Enable QInt8 and QInt32 quantization (#89788)	last month
quantize_per_tensor_quint8.glsl	[Vulkan] Enable QInt8 and QInt32 quantization (#89788)	last month
quantized_add.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_conv2d.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_conv2d_dw.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_conv2d_pw_2x2.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_div.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_mul.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_sub.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
quantized_upsample_nearest2d.glsl	[Vulkan][TCC] Fix quantized shaders (#89456)	last month
reflection_pad2d.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
replication_pad2d.glsl	[vulkan] replication_pad2d.glsl: use clamp() instead of min(max()) (#...	7 months ago
select_depth.glsl	[Vulkan] Implement select.int operator (#81771)	5 months ago
sigmoid.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
sigmoid_.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
slice_4d.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
softmax.glsl	[vulkan] Add image format qualifier to glsl files (#69330)	last year
stack_feature.glsl	[Vulkan] Implement Stack operator (#81064)	5 months ago
sub.glsl	[Vulkan] Implement arithmetic ops where one of the arguments is a ten...	5 months ago
sub_.glsl	[Vulkan] Implement arithmetic ops where one of the arguments is a ten...	5 months ago
tanh.glsl	[vulkan] Clamp tanh activation op input to preserve numerical stabili...	10 months ago
tanh_.glsl	[vulkan] Clamp tanh activation op input to preserve numerical stabili...	10 months ago

Example Activation Functions

master ▾ pytorch / aten / src / ATen / native / vulkan / glsl / sigmoid.glsl

SS-JIA [vulkan] Add image format qualifier to glsl files (#69330) ...

1 contributor

23 lines (17 sloc) | 710 Bytes

```
#version 450 core
#define PRECISION $precision
#define FORMAT    $format

layout(std430) buffer;

/* Qualifiers: layout - storage - precision - memory */

layout(set = 0, binding = 0, FORMAT) uniform PRECISION restrict writeonly image3D   uO
layout(set = 0, binding = 1)         uniform PRECISION                    sampler3D uI
layout(set = 0, binding = 2)         uniform PRECISION restrict           Block {
  ivec4 size;
} uBlock;

layout(local_size_x_id = 0, local_size_y_id = 1, local_size_z_id = 2) in;

void main() {
  const ivec3 pos = ivec3(gl_GlobalInvocationID);

  if (all(lessThan(pos, uBlock.size.xyz))) {
    imageStore(uOutput, pos, 1/(1+exp(-1*texelFetch(uInput, pos, 0))));
  }
```

master ▾ pytorch / aten / src / ATen / native / vulkan / glsl / tanh.glsl

SS-JIA [vulkan] Clamp tanh activation op input to preserve numerical stabili... ...

2 contributors

27 lines (21 sloc) | 777 Bytes

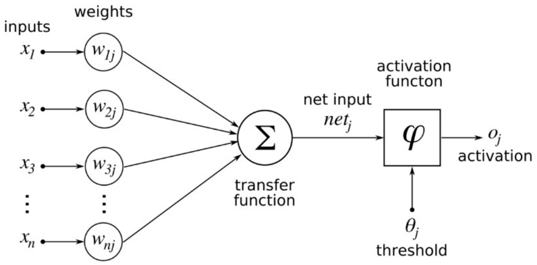
```
#version 450 core
#define PRECISION $precision
#define FORMAT    $format

layout(std430) buffer;

/* Qualifiers: layout - storage - precision - memory */

layout(set = 0, binding = 0, FORMAT) uniform PRECISION restrict writeonly image3D   uOutput;
layout(set = 0, binding = 1)         uniform PRECISION                    sampler3D uInput;
layout(set = 0, binding = 2)         uniform PRECISION restrict           Block {
  ivec4 size;
} uBlock;

layout(local_size_x_id = 0, local_size_y_id = 1, local_size_z_id = 2) in;

void main() {
  const ivec3 pos = ivec3(gl_GlobalInvocationID);

  if (all(lessThan(pos, uBlock.size.xyz))) {
    const vec4 intex = texelFetch(uInput, pos, 0);
    imageStore(
        uOutput,
        pos,
        tanh(clamp(intex, -15.0, 15.0)));
  }
```

Why Learn All These Details About Backprop?

- **Modern deep learning frameworks compute gradients for you!**
- But why take a class on compilers or systems when they are implemented for you?
  - Understanding what is going on under the hood is useful!
- Backpropagation doesn't always work perfectly out of the box
  - Understanding why is crucial for debugging and improving models

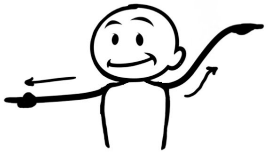
Backprop in Practice

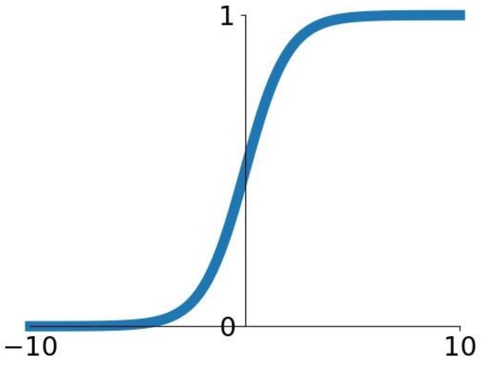
Activation Functions



- How do you choose what activation function to use?
- In general, it is problem-specific and might require trial-and-error.
- Here are some tips about popular action functions.

Activation Functions : Sigmoid





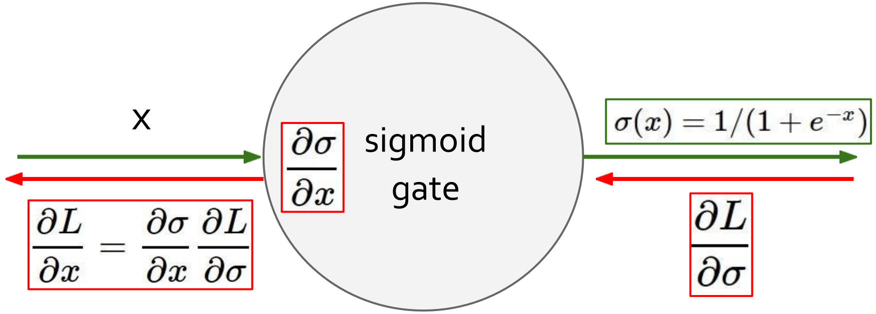
$$\sigma(x) = 1/(1+e^{-x})$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x)\,)$$

- Squashes numbers to range [0,1]
- Historically popular, interpretation as "firing rate" of a neuron

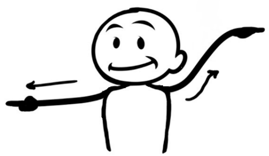
- Key limitation: Saturated neurons "kill" the gradients
- Whenever |x| > 5, the gradients are basically zero.

If all the gradients flowing back will be zero and weights will never change.

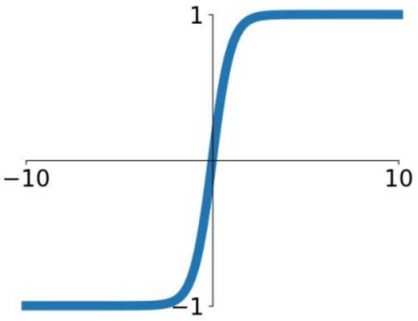


[dance figure: https://www.imaginary.org/gallery/maths-dance-moves]

Activation Functions : Tanh



- Symmetric around [-1, 1]
- Still saturates |x| > 3 and “kill” the gradients
- Zero-centered — good for stacking hidden layers

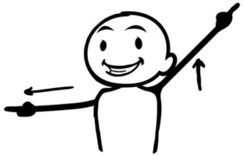


tanh(x)

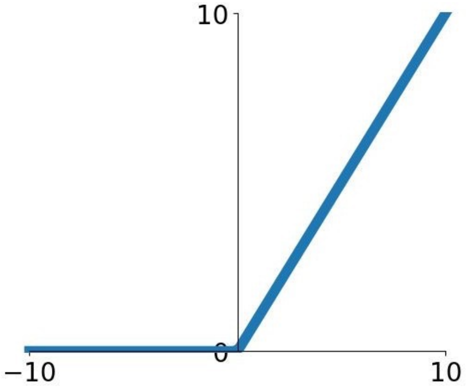
[LeCun et al., 1991]

[dance figure: https://www.imaginary.org/gallery/maths-dance-moves]

Activation Functions : ReLU



- Computationally efficient
- In practice, converges faster than sigmoid/tanh in practice
- Does not saturate (in +region) — will die less!

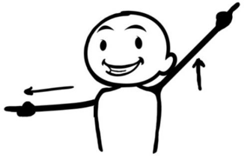


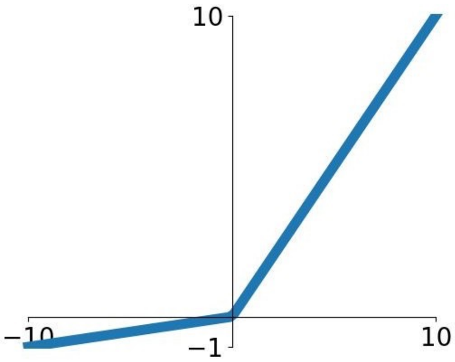
ReLU
(Rectified Linear Unit)

[Krizhevsky et al., 2012]

[dance figure: https://www.imaginary.org/gallery/maths-dance-moves]

Activation Functions : Leaky ReLU





$$f(x) = \max(0.01x, x)$$

- Does not saturate — will not die.
- Computationally efficient
- In practice it converges faster than sigmoid/tanh in practice

- Other parametrized variants:
  - Parametric Rectifier (PReLU): $f(x) = \max(\alpha x, x)$ [He et al., 2015]
  - Maxout: $\max(w_1^T x + b_1, w_2^T x + b_2)$ [Goodfellow et al., 2013]

- Provide more flexibility, though at the cost of more learnable parameters.
  - For example, Maxout doubles the number of parameters.

[dance figure: https://www.imaginary.org/gallery/maths-dance-moves]

How do You Choose What Activation Function to Use?

- In general, it is problem-specific and might require trial-and-error.

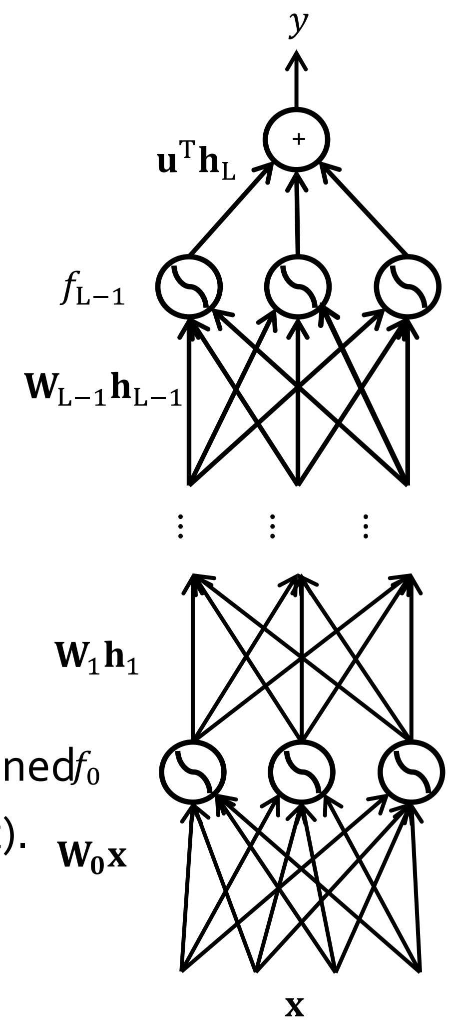
- A useful recipe:
  1. Generally, ReLU is a good activation to start with.
  2. Time/compute permitting, you can try other activations to squeeze out more performance.

Exploding/Vanishing Gradients

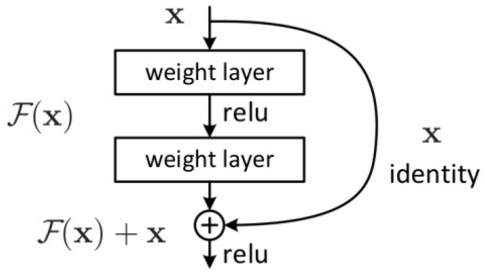
- Remember gradient computation at layer $L-k$:

$$\nabla_{\mathcal{L}}(\mathbf{W}_{L-k}) = \underbrace{\left(\mathbf{J}_{\ell}(\mathrm{y})\,\mathbf{J}_{y}(\mathbf{h}_{\mathrm{L}})\,\mathbf{J}_{\mathbf{h}_{\mathrm{L}}}(\mathbf{h}_{\mathrm{L}-1})\,\mathbf{J}_{\mathbf{h}_{\mathrm{L}-1}}(\mathbf{W}_{\mathrm{L}-2})\,\dots\,\mathbf{J}_{\mathbf{h}_{L-k+1}}(\mathbf{W}_{L-k})\right)^{\mathrm{T}}}_{\text{O(k)-many matrix multiplication}}$$

- This matrix multiplication could quickly approach
  - ∞, if the matrix elements are a large — exploding gradients.
  - 0, if the matrix elements are small — vanishing gradients.
- For those interested, convergences of matrix powers is determined by its largest eigenvalue (out of scope for this class, extra credit).
- $\infty/0$ gradients would kill learning (no flow of information).

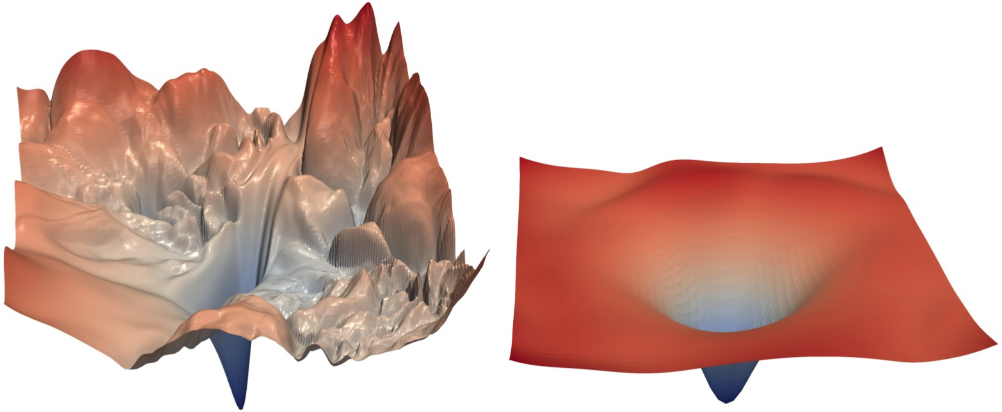


Residual Connections/Blocks



- Create direct “information highways” between layers.

- Shown to diminish the effect of vanishing/exploding gradients
  - Early in the training, there are fewer layers to propagate through.
  - The network would restore the skipped layers, as it learns richer features.
  - It is also shown to make the optimization objective smoother.
  - Fun fact: the paper introducing residual layers (He et al. 2015) is the most cited paper of century.

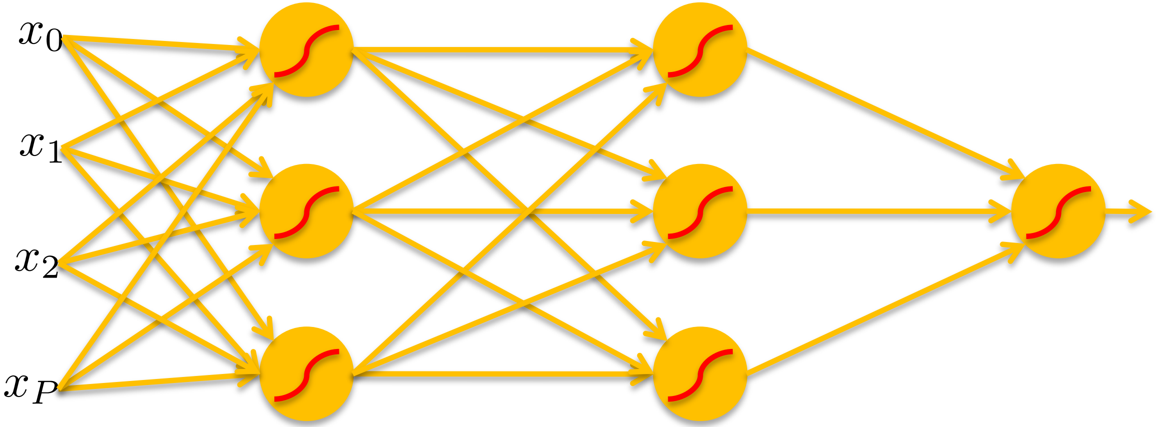


(a) without skip connections (b) with skip connections

[Li et al. “Visualizing the Loss Landscape of Neural Nets”]

Weight Initialization

- Initializing all weights with a fixed constant (e.g., 0) is a very bad idea! (why?)



- If the neurons start with the same weights, then all the neurons will follow the same gradient, and will always end up doing the same thing as one another.

Weight Initialization

- Better idea: initialize weights with random Gaussian noise.

```
x = torch.tensor.empty(3, 5)
nn.init.normal_(w)
```

- There are fancier initializations (Xavier, Kaiming, etc.) that we won't get into.

[read more here: https://pytorch.org/docs/stable/nn.init.html]

Comments on Training

- No guarantee of convergence; neural networks form non-convex functions with multiple local minima
- In practice, many large networks can be trained on large amounts of data for realistic problems.
- May be hard to set learning rate and to select number of hidden units and layers.
- Many steps (tens of thousands) may be needed for adequate training. Large data sets may require many hours of CPU
- Termination criteria: Number of epochs; Increased error on a validation set.
- To avoid local minima: several trials with different random initial weights with majority or voting techniques

Over-training Prevention

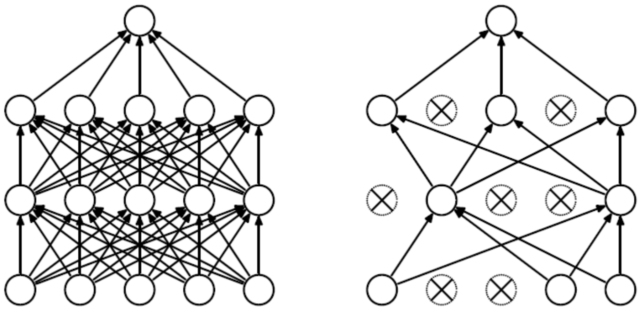
- Running too many epochs and/or a NN with many hidden layers may lead to an overfit network
- Keep a held-out validation set and evaluate accuracy after every epoch
- Early stopping: maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond that.
- To avoid losing training data to validation:
  - Use 10-fold cross-validation to determine the average number of epochs that optimizes validation performance
  - Train on the full data set using this many epochs to produce the final results

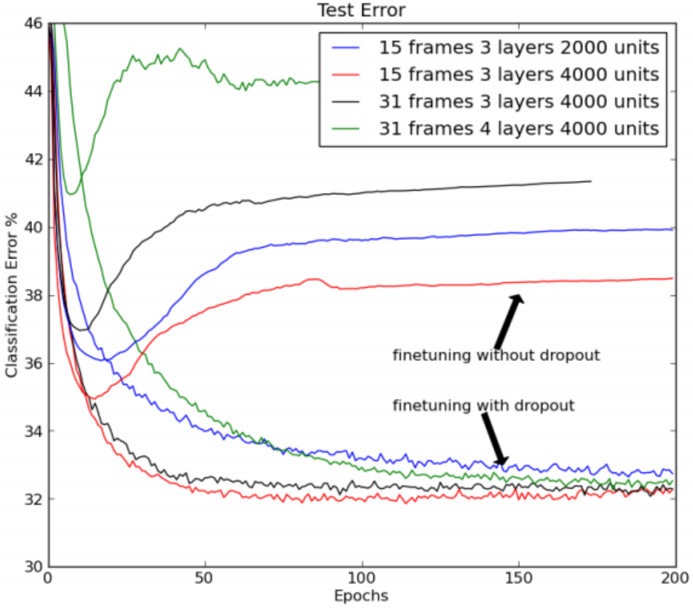
Over-fitting prevention

- Too few hidden units prevent the system from adequately fitting the data and learning the concept.
- Using too many hidden units leads to over-fitting.
- Similar cross-validation method can be used to determine an appropriate number of hidden units. (general)
- Another approach to prevent over-fitting is weight-decay: all weights are multiplied by some fraction in (0,1) after every epoch.
  - Encourages smaller weights and less complex hypothesis
  - Equivalently: change Error function to include a term for the sum of the squares of the weights in the network. (general)

Dropout training

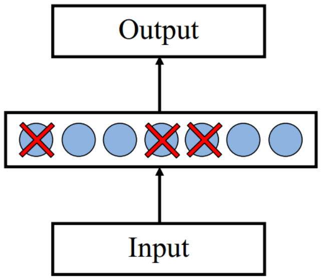
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter; 0.5 is common
- Dropout is implicitly a ensemble (average) of model that share parameters.
  - Each binary mask is one model
  - For example, an FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
  - Only ~ 10^{82} atoms in the universe ...





[Hinton et al, 2012; Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014]

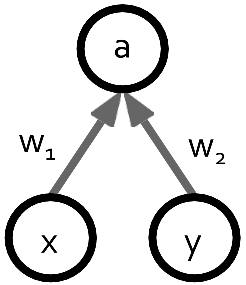
Dropout During Test Time



- The issue for test time is that Dropout adds randomization.
  - Each dropout mask would lead to a slightly different outcome.
- In ideal world, we would like to "average out" the outcome across all the possible random masks:
  - Not feasible.

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

- The alternative is to not apply dropout. Without dropout, the input values to each neuron would be higher than what was seen during the training (mismatch between train/test).
  - **Example:** Input to activation during:
    - training time: $E[a] = \frac{1}{4}(w_1x_1 + w_2x_2) + \frac{1}{4}(0+0) + \frac{1}{4}(0 + w_2x_2) + \frac{1}{4}(w_1x_1 + 0) = \frac{1}{2}(w_1x_1 + w_2x_2)$
    - test time: $E[a] = w_1x_1 + w_2x_2$



- **Solution:** scale the values proportional to dropout probability.
  - Can be applied in either testing (scaling down) or training (scaling up).

Dropout in Practice

Just call the PyTorch function!

```
dropout = nn.Dropout(p=0.2)
x = torch.randn(20, 16)
y = dropout(x)
```

It automatically

- activates the dropout for training.

```
# training step
...
model.train()
...
```

- deactivatives it during evaluations and scales the values according to its parameter.

```
# evaluate model:
...
model.eval()
...
```

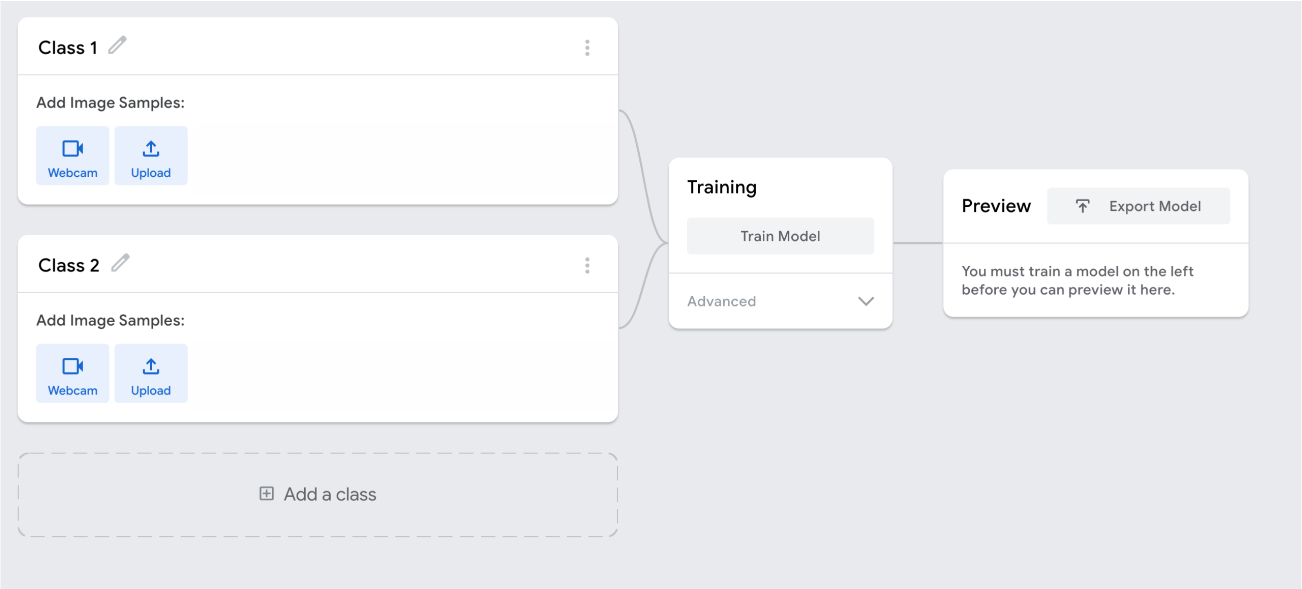
https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

The Only Time You Want to Overfit: The First Tryout

- A model with buggy implementation (e.g., incorrect gradient calculations or updates) cannot learn anything.
- Therefore, a good and easy sanity check is to see if you can overfit few examples.
    - This is really the first test you should do, before any hyperparameter tuning.
- Try to train to 100% training accuracy/performance on a small sample (<30) of training data and monitor the **training** loss trends.
    - Does it down? If not, something must be wrong.
    - Try checking the learning rate or modifying the initialization.
    - If those don't help, check the gradients.
        - If they're NaN or Inf, might indicate exploding gradients.
        - If they're zeros, might indicate vanishing gradients.

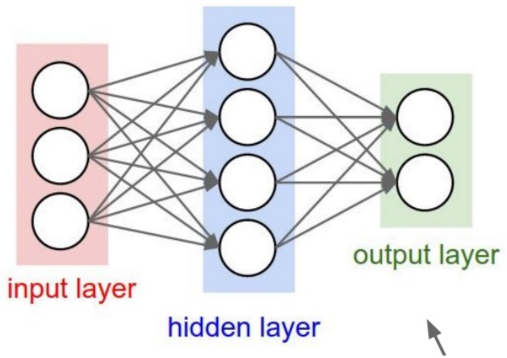
Demo Time!

- https://teachablemachine.withgoogle.com/



Chapter Summary

- Feed-forward network architecture
- Word2Vec is just a feedforward net!
  - And we can easily extend it!



- We learned Back-Prop, the most important algorithm in neural networks! 🎉
  - Recursively (and hence efficiently) apply the chain rule along computation graph

- Lots of empirical tricks for training neural networks:
  - First test: check if you can overfit.
  - Dropout
  - Be mindful of activations
  - Careful of exploding/vanishing gradients