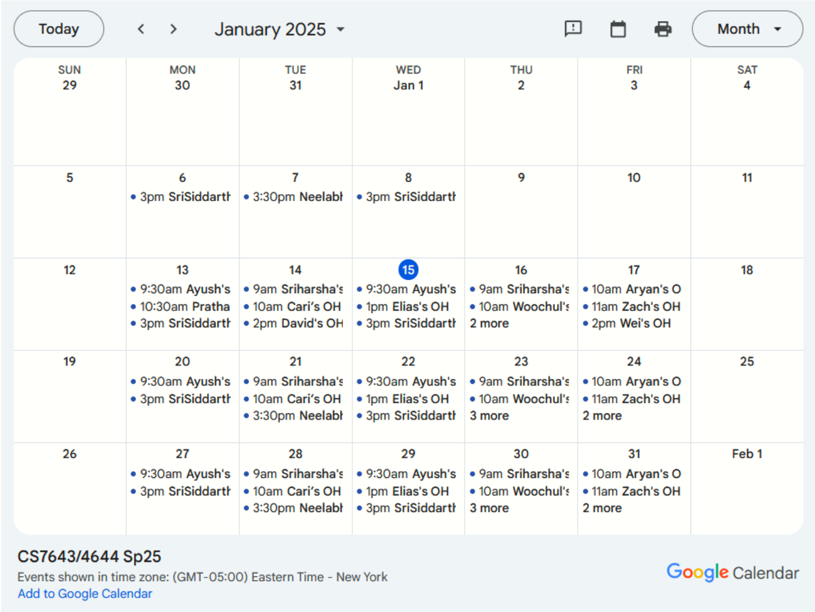
Topics:

- Gradient Descent
- Neural Networks

CS 4644-DL / 7643-A
ZSOLT KIRA

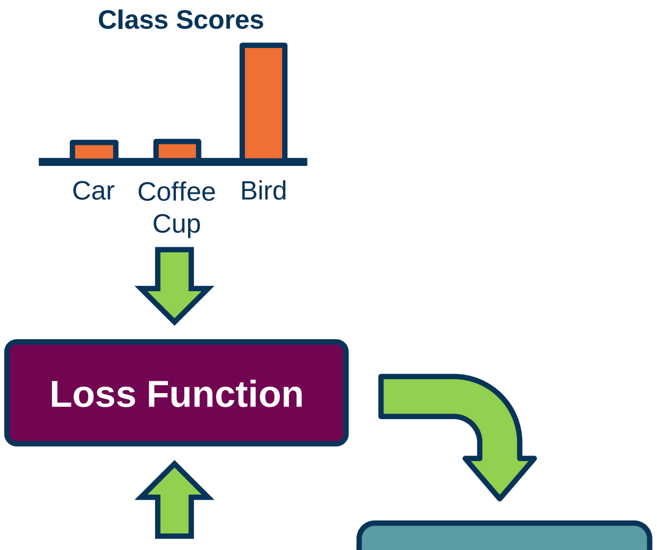
- **Assignment 1 out!**
  - **Due Jan 31st (with grace period Feb 2nd)**
  - Start now, start now, start now!
  - Start now, start now, start now!
  - Start now, start now, start now!

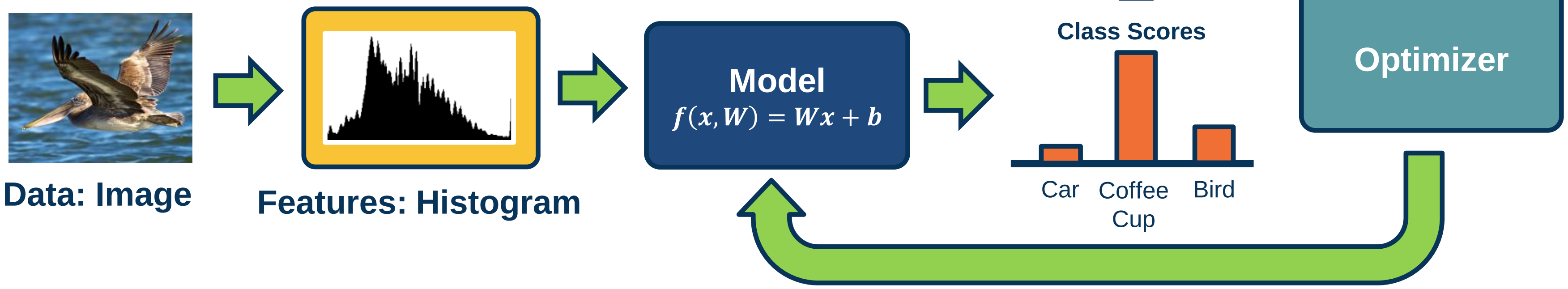


- **Piazza**
  - Be active!!!

- **Office hours**
  - Lots of special topics TBA (e.g. Assignment 1, Matrix Calculus, etc.)
  - My OH Thursdays 3:30-4:30pm ET
- Note: Course will start to get math heavy!
- Matrix calculus for deep learning

- **Input** (and representation)
- **Functional form** of the model
  - Including parameters
- **Performance measure** to improve
  - Loss or objective function
- **Algorithm** for finding best parameters
  - Optimization algorithm





Components of a Parametric Model

Several issues with scores:

- Not very interpretable (no bounded value)

We often want **probabilities**

- More interpretable
- Can relate to probabilistic view of machine learning

We use the **softmax** function to convert scores to probabilities

$$\boldsymbol{s} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{W}) \quad \textbf{Scores}$$

$$= \boldsymbol{W}\boldsymbol{x}$$

$$\boldsymbol{P}(\boldsymbol{Y} = \boldsymbol{k} | \boldsymbol{X} = \boldsymbol{x}) = \frac{\boldsymbol{e}^{\boldsymbol{s}_k}}{\sum_j \boldsymbol{e}^{\boldsymbol{s}_j}} \quad \textbf{Softmax Function}$$

Performance Measure: Cross-Entropy

- If we use the softmax function to convert scores to probabilities, the right loss function to use is **cross-entropy**
- Can be derived by looking at the distance between two probability distributions (output of model and ground truth)
- Can also be derived from a maximum likelihood estimation perspective

$$s = f(x, W)$$ **Scores**

$$P(Y = k | X = x) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ **Softmax Function**

$$L_i = -\log P(Y = y_i | X = x_i)$$

Maximize log-prob of correct class = Maximize the log likelihood = Minimize the negative log likelihood

- We can find the steepest descent direction by computing the **derivative (gradient):**

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

- Steepest descent direction is the **negative gradient**
- **Intuitively:** Measures how the function changes as the argument a changes by a small step size
  - As step size goes to zero
- **In Machine Learning:** Want to know how the **loss function** changes **as weights** are varied
  - Can consider each parameter separately by taking **partial derivative** of loss function with respect to that parameter

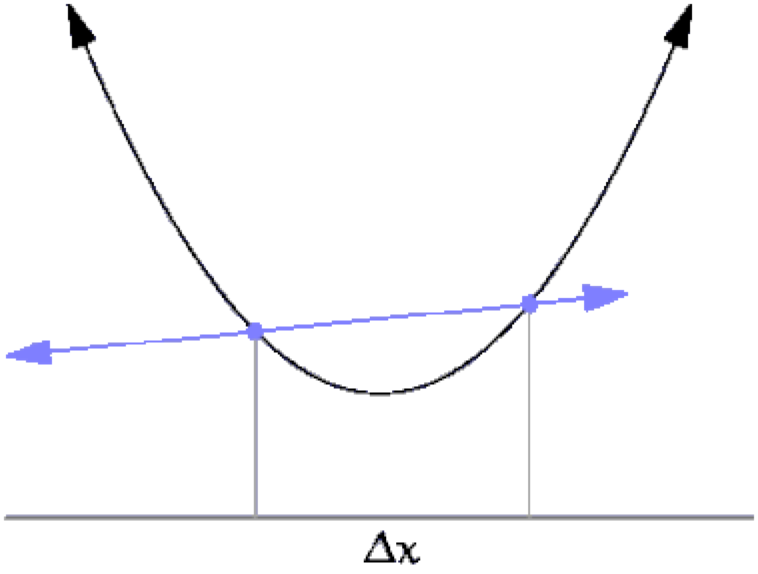


Image and equation from: https://en.wikipedia.org/wiki/Derivative#/media/File:Tangent_animation.gif

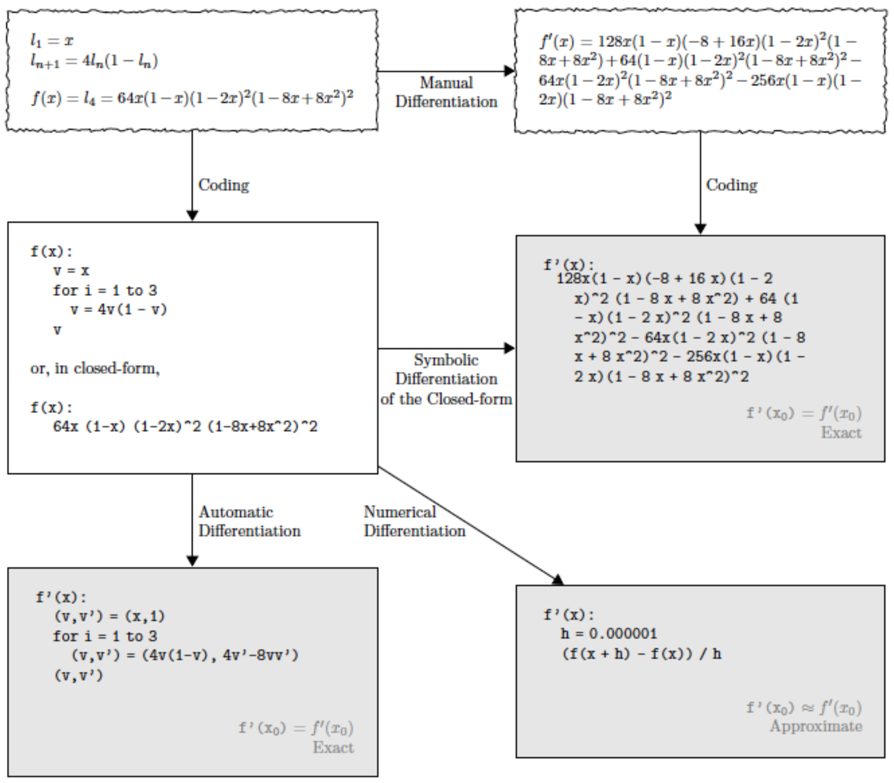
Optimization: Gradient Descent

- Input**: Vector**
- Functional form of the model: **Softmax(Wx)**
- Performance measure to improve: **Cross-Entropy**
- Algorithm for finding best parameters: **Gradient Descent**
  - Compute $\frac{\partial L}{\partial w_i}$
  - Update Weights $w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$

We know how to compute the **model output and loss function**

Several ways to compute $\frac{\partial L}{\partial w_i}$

- Manual differentiation
- Symbolic differentiation
- Numerical differentiation
- Automatic differentiation



current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:	**W + h** (first dim)**:**	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[?,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

?,
?,…]

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	**0.6**,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

(1.25353 - 1.25347)/0.0001
= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	**0**,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

(1.25347 - 1.25347)/0.0001
= 0

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical vs Analytic Gradients

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient.
This is called a **gradient check.**

For some functions, we can analytically derive the partial derivative

Example:

Function

$$f(w, x_i) = w^T x_i$$

Loss

$$\sum_{i=1}^{N} (y_i - w^T x_i)^2$$

(Assume w and $\mathbf{x_i}$ are column vectors, so same as $w \cdot x_i$)

Dataset: N examples (indexed by i)

Update Rule

$$w_j \leftarrow w_j + 2\alpha \sum_{i=1}^{N} \delta_i x_{ij}$$

Derivation of Update Rule

L= $\sum_{i=1}^{N} (y_i - w^T x_i)^2$

Gradient descent tells us we should update w as follows to minimize L:

$$w_j \leftarrow w_j - \alpha \frac{\partial L}{\partial w_j}$$

So what's $\frac{\partial L}{\partial w_j}$?

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial}{\partial w_j} (y_i - w^T x_i)^2$$

$$= \sum_{i=1}^{N} 2(y_i - w^T x_i) \frac{\partial}{\partial w_j} (y_i - w^T x_i)$$

$$= -2 \sum_{i=1}^{N} \delta_i \frac{\partial}{\partial w_j} w^T x_i$$

…where…
$\delta_i = y_i - w^T x_i$

$$= -2 \sum_{i=1}^{N} \delta_i \frac{\partial}{\partial w_j} \sum_{k=1} w_k x_{ik}$$

$$= -2 \sum_{i=1}^{N} \delta_i x_{ij}$$

If we add a **non-linearity (sigmoid),** derivation is more complex

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

First, one can derive that: $\sigma'(x) = \sigma(x)(1-\sigma(x))$

$$\mathrm{f}(\mathrm{x}) = \sigma\left(\sum_k w_k x_k\right)$$

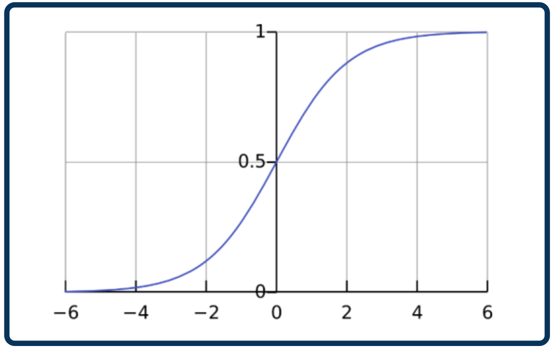
$$\mathrm{L} = \sum_i \left(y_i - \sigma\left(\sum_k w_k x_{ik}\right)\right)^2$$

$$\frac{\partial L}{\partial w_j} = \sum_i 2\left(y_i - \sigma\left(\sum_k w_k x_{ik}\right)\right)\left(-\frac{\partial}{\partial w_j}\sigma\left(\sum_k w_k x_{ik}\right)\right)$$

$$= \sum_i -2\left(y_i - \sigma\left(\sum_k w_k x_{ik}\right)\right)\sigma'\left(\sum_k w_k x_{ik}\right)\frac{\partial}{\partial w_j}\sum_k w_k x_{ik}$$

$$= \sum_i -2\delta_i \sigma(\mathrm{d}_i)(1-\sigma(\mathrm{d}_i))x_{ij}$$

where $\delta_i = y_i - \mathrm{f}(x_i)$ $\quad d_i = \sum w_k x_{ik}$



The sigmoid perception update rule:

$$w_j \leftarrow w_j + 2\alpha \sum_{k=1}^{N} \delta_i \sigma_i (1-\sigma_i) x_{ij}$$

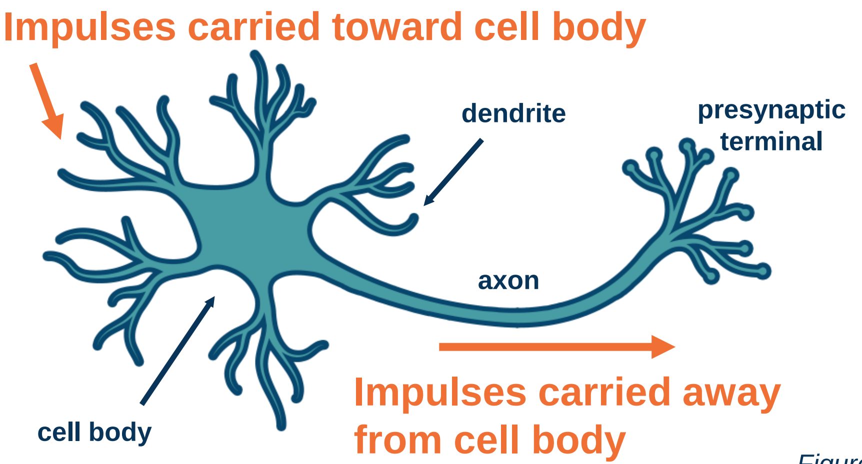
where $\quad \sigma_i = \sigma\left(\sum_{j=1}^{d} w_j x_{ij}\right)$

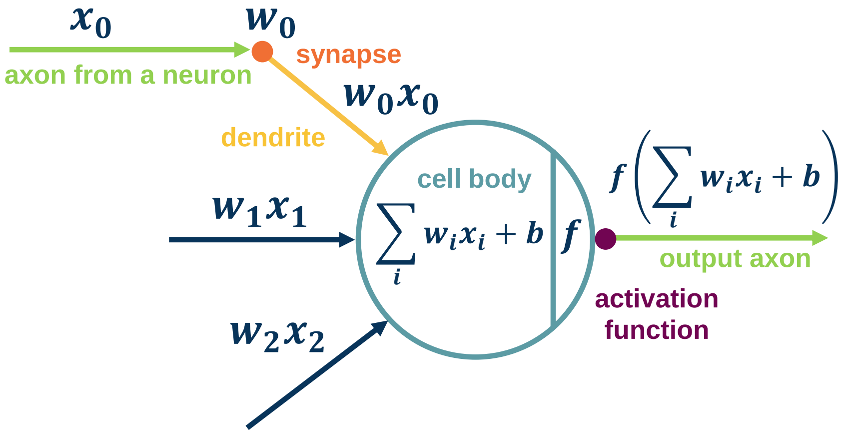
$$\delta_i = y_i - \sigma_i$$

Neural Network View of a Linear Classifier

A simple **neural network** has similar structure as our linear classifier:

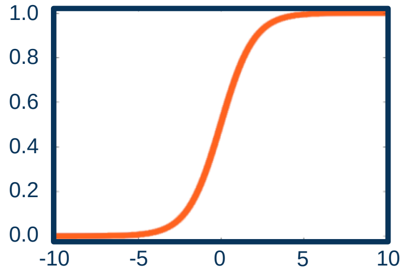
- A neuron takes input (firings) from other neurons **(-> input to linear classifier)**
- The inputs are summed in a weighted manner **(-> weighted sum)**
  - Learning is through a modification of the weights
- If it receives enough input, it "fires" (threshold or if weighted sum plus bias is high enough)





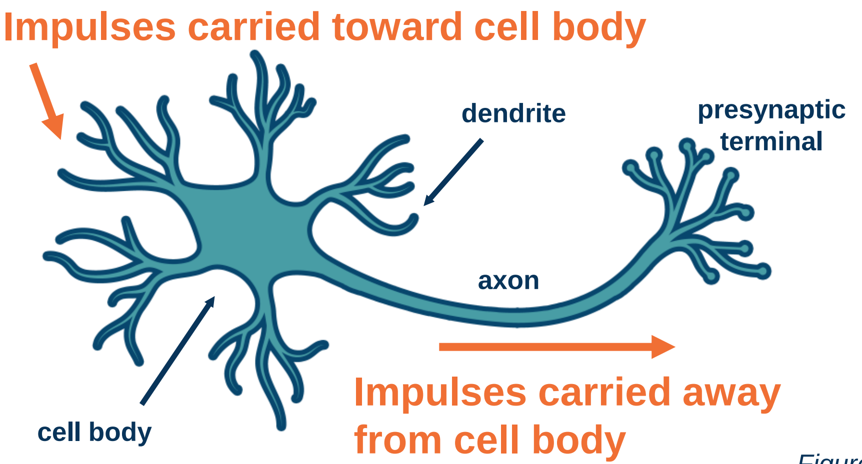
Figures adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

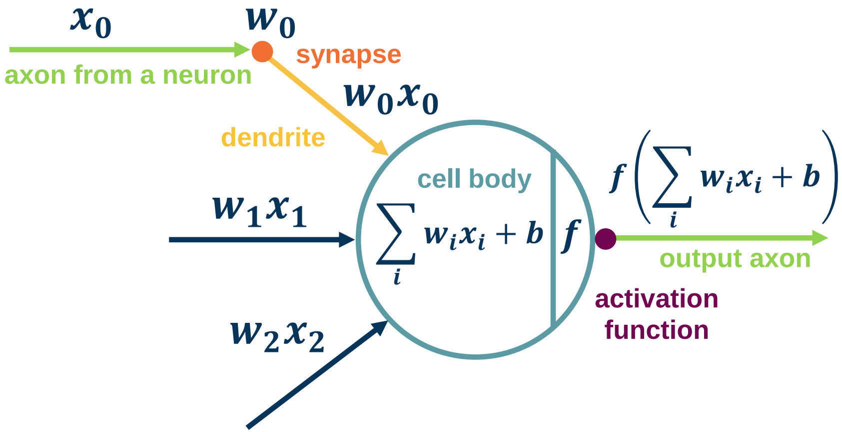
As we did before, the output of a neuron can be modulated by a non-linear function (e.g. sigmoid)



Sigmoid Activation Function

$$\frac{1}{1+e^{-x}}$$





Figures adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Adding Non-Linearities

We can have **multiple** neurons connected to the same input

Corresponds to a multi-class classifier

- Each output node outputs the score for a class

$$f(x, W) = \sigma(Wx + b) \quad \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b2 \\ w_{21} & w_{22} & \cdots & w_{3m} & b3 \end{bmatrix}$$

- Often called fully connected layers
  - Also called a linear *projection layer*

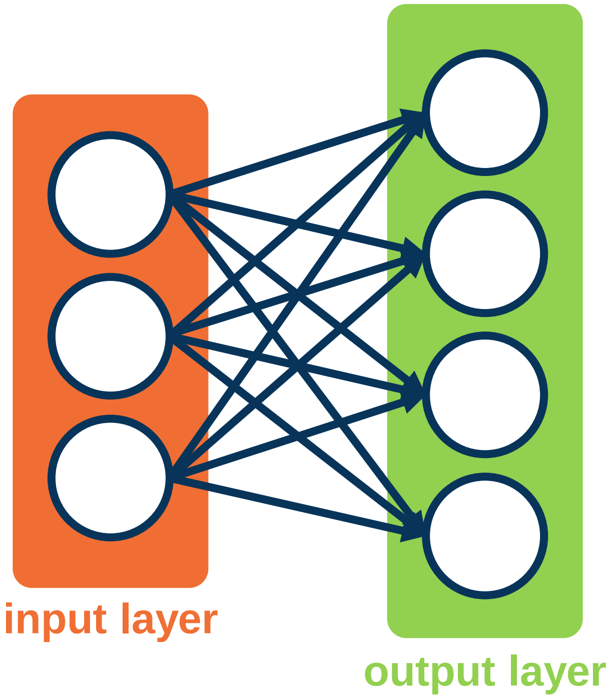


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Connecting Many Neurons

- Each input/output is a **neuron (node)**
- A linear classifier (+ optional non-linearity) is called a **fully connected** layer
- Connections are represented as **edges**
- Output of a particular neuron is referred to as **activation**
- This will be expanded as we view computation in a neural network as a **graph**

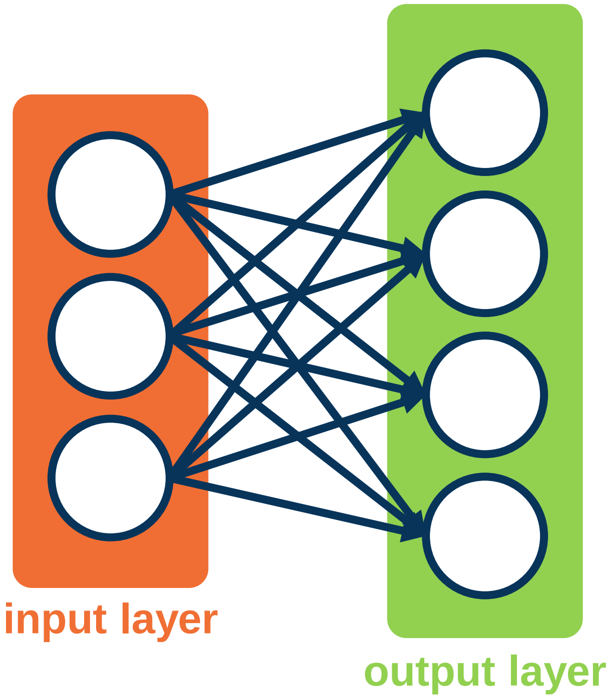


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Neural Network Terminology

We can **stack** multiple layers together

- Input to second layer is output of first layer

Called a **2-layered neural network** (input is not counted)

Because the middle layer is neither input or output, and we don’t know what their values represent, we call them **hidden** layers

- We will see that they end up learning effective features

This **increases** the representational power of the function!

- Two layered networks can represent any continuous function

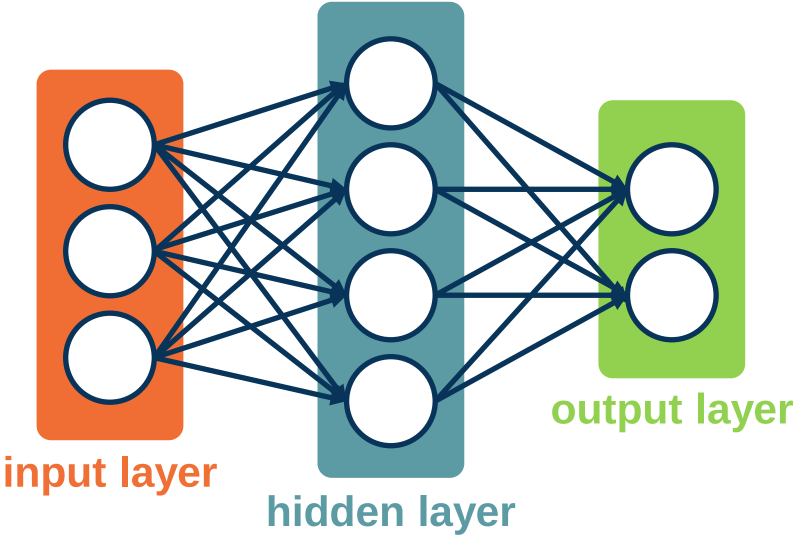
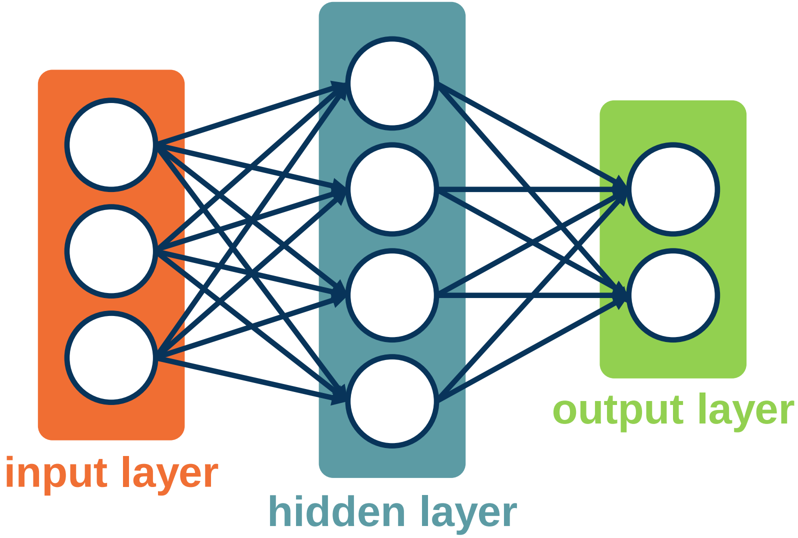


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

The same two-layered neural network **corresponds to adding another weight matrix**

- We will prefer the linear algebra view, but use some terminology from neural networks (& biology)



$$\boldsymbol{x} \quad \boldsymbol{W_1} \quad \boldsymbol{W_2}$$

$$=$$

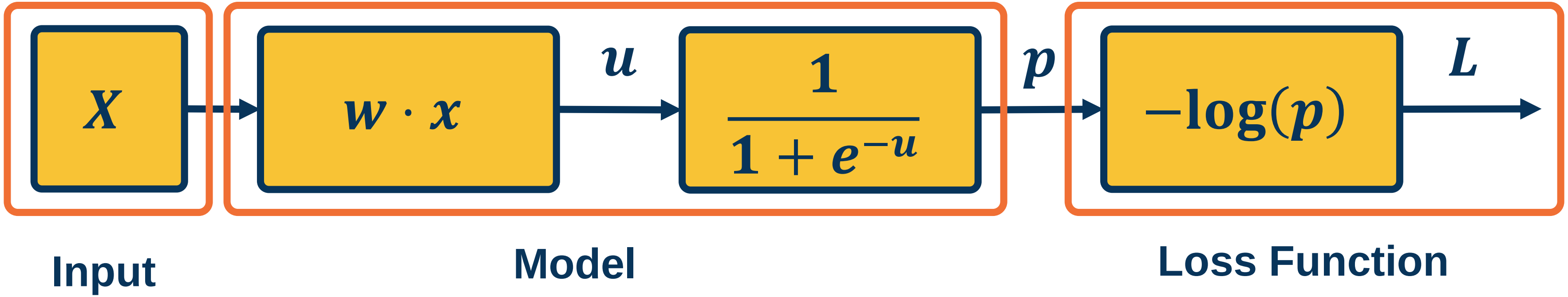
$$\boldsymbol{f(x, W_1, W_2) = \sigma(W_2 \sigma(W_1 x))}$$

Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

A **classifier** can be broken down into:

- Input
- A function of the input
- A loss function

It's all just one function that can be **decomposed** into building blocks

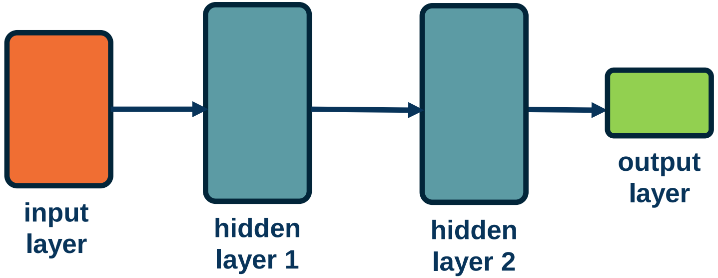


Large (deep) networks can be built by adding more and more layers

Three-layered neural networks can represent **any function**

- The number of nodes could grow unreasonably (exponential or worse) with respect to the complexity of the function

We will show them **without edges**:



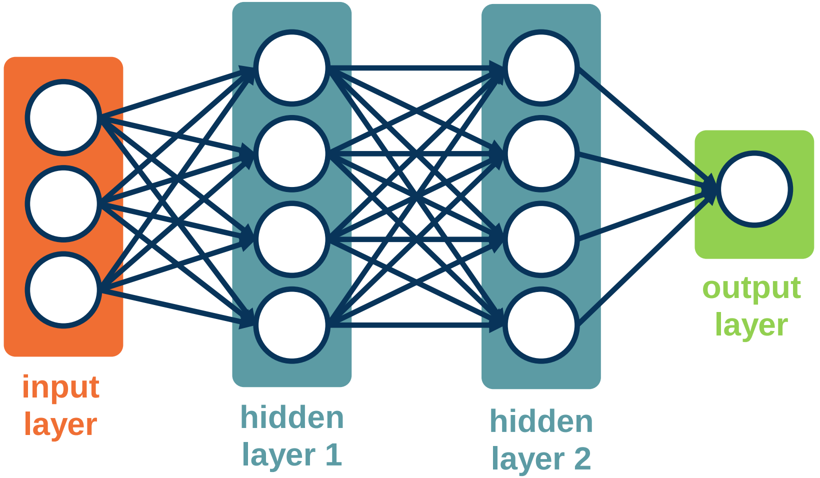


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Adding More Layers!

Computation Graphs

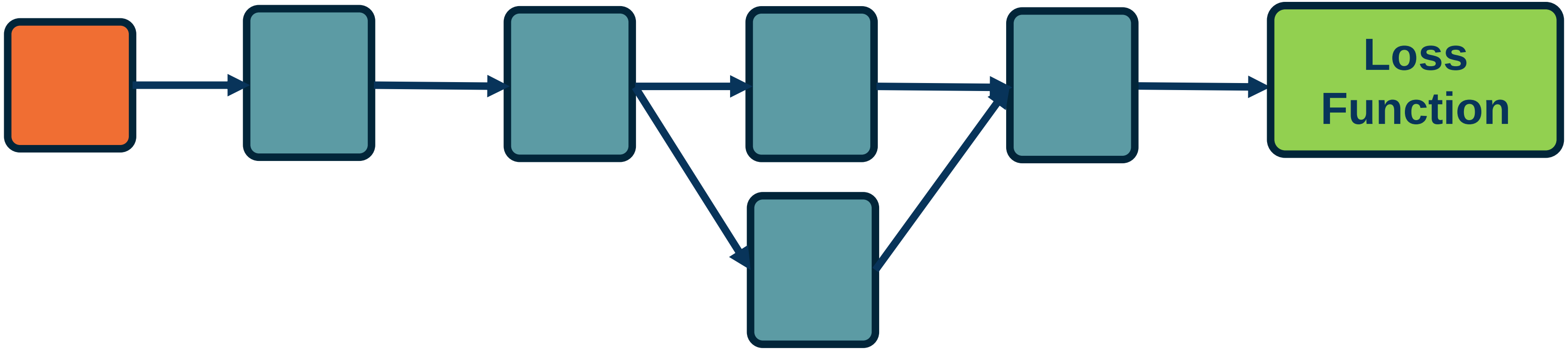
Functions can be made **arbitrarily complex** (subject to memory and computational limits), e.g.:

$$f(x, W) = \sigma(W_5\sigma(W_4\sigma(W_3\sigma(W_2\sigma(W_1x))$$

We can use **any type of differentiable function (layer)** we want!

- At the end, **add the loss function**

Composition can have **some structure**



Adding Even More Layers

Compositionality

The world is **compositional**!

We want our **model** to reflect this

Empirical and theoretical evidence that it makes **learning complex functions easier**

Note that **prior state of art engineered features** often had this compositionality as well

VISION

pixels → edge → texton → motif → part → object

SPEECH

sample → spectral band → formant → motif → phone → word

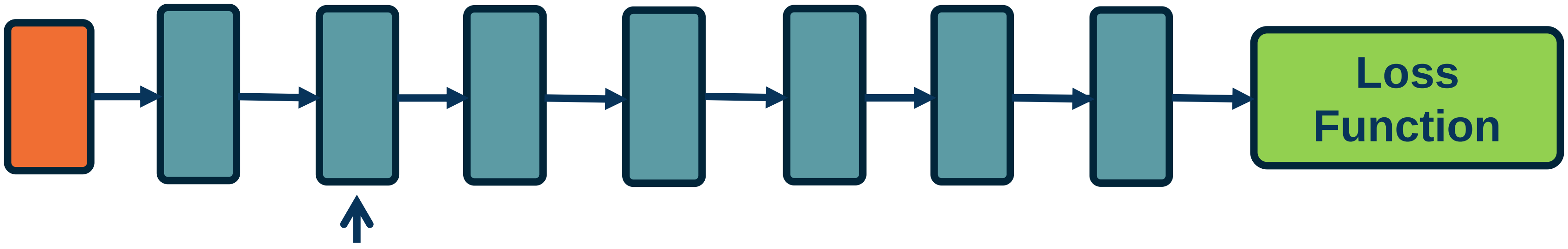
NLP

character → word → NP/VP/.. → clause → sentence → story

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

- **Pixels -> edges -> object parts -> objects**

- We are learning **complex models** with significant amount of parameters (millions or billions)
- How do we compute the gradients of the **loss** (at the end) with respect to **internal** parameters?
- Intuitively, want to understand how **small changes** in weight deep inside **are propagated** to affect the **loss function** at the end



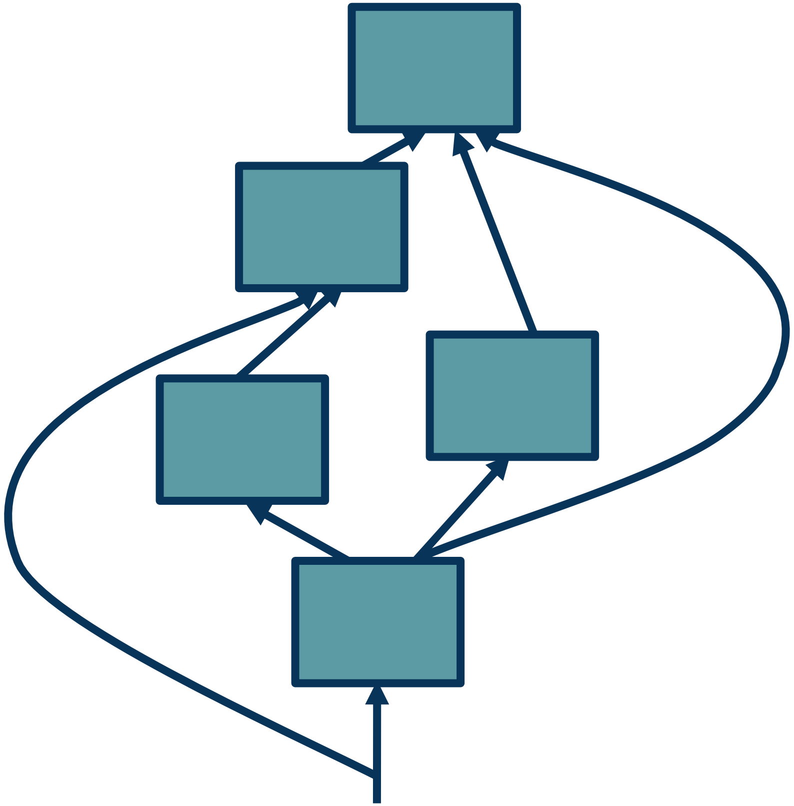
$\frac{\partial L}{\partial w_i}$?

To develop a general algorithm for this, we will view the function as a **computation graph**

Graph can be any **directed acyclic graph (DAG)**

- Modules must be differentiable to support gradient computations for gradient descent

A **training algorithm** will then process this graph, **one module at a time**

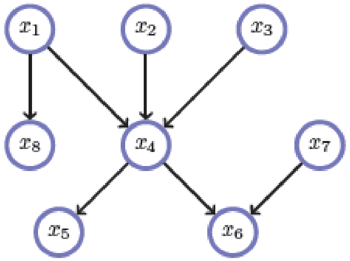


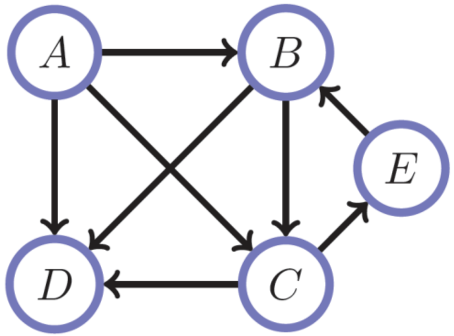
Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

A General Framework

Directed Acyclic Graphs (DAGs)

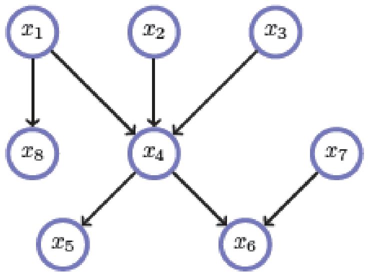
- Exactly what the name suggests
  - Directed edges
  - No (directed) cycles
  - Underlying undirected cycles okay

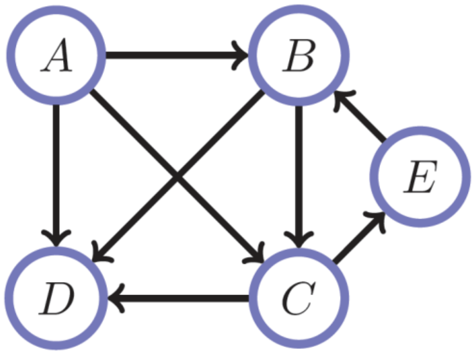




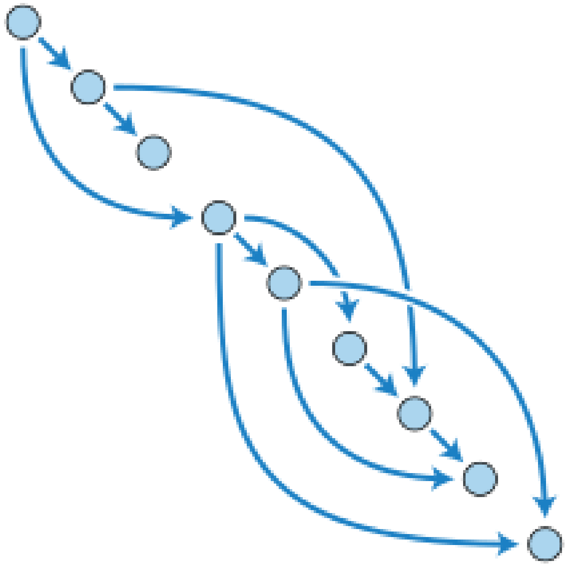
Directed Acyclic Graphs (DAGs)

- Concept
  - Topological Ordering





Directed Acyclic Graphs (DAGs)



Backpropagation

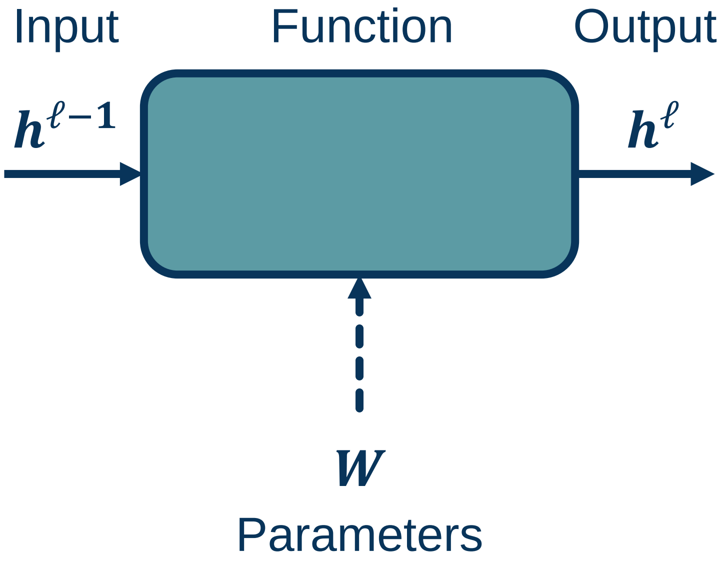
Given this computation graph, the training algorithm will:

- Calculate the current model's outputs (called the **forward pass**)
- Calculate the gradients for each module (called the **backward pass**)

Backward pass is a recursive algorithm that:

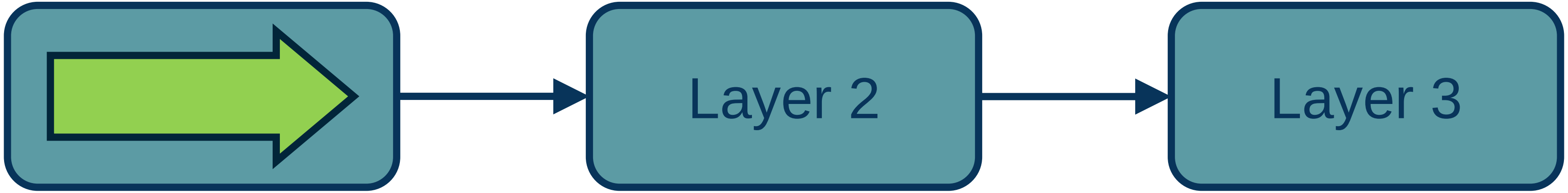
- Starts at **loss function** where we know how to calculate the gradients
- Progresses back through the modules
- Ends in the **input layer** where we do not need gradients (no parameters)

This algorithm is called **backpropagation**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

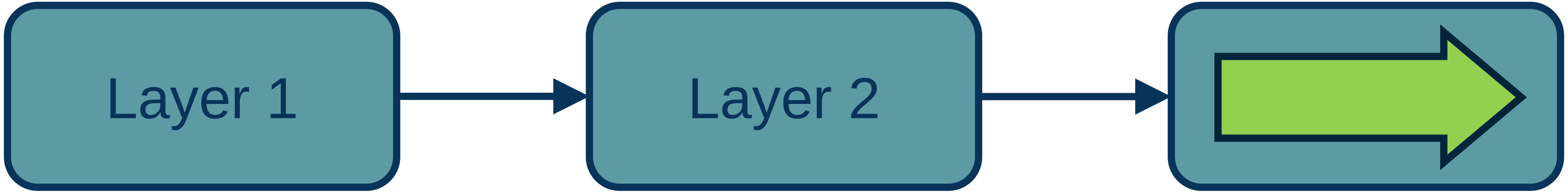


Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**



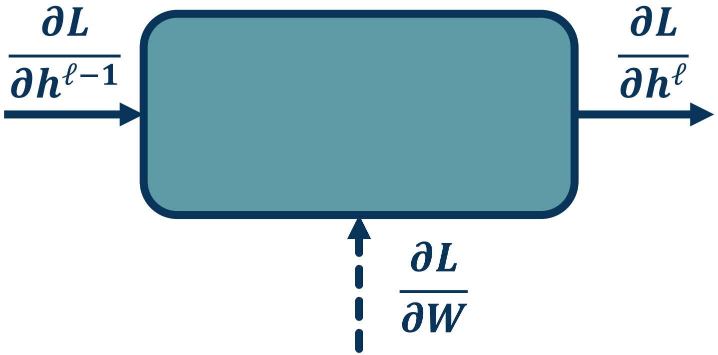
Note that we must store the **intermediate outputs of all layers**!

- This is because we will need them to **compute the gradients** (the gradient equations will have terms with the output values in them)

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

In the **backward pass**, we seek to calculate the gradients of the loss with respect to the module's parameters

- Assume that we have the gradient of the loss with respect to the **module's outputs** (given to us by upstream module)
- We will also pass the gradient of the loss with respect to the **module's inputs**
  - This is not required for update the module's weights, but passes the gradients back to the previous module

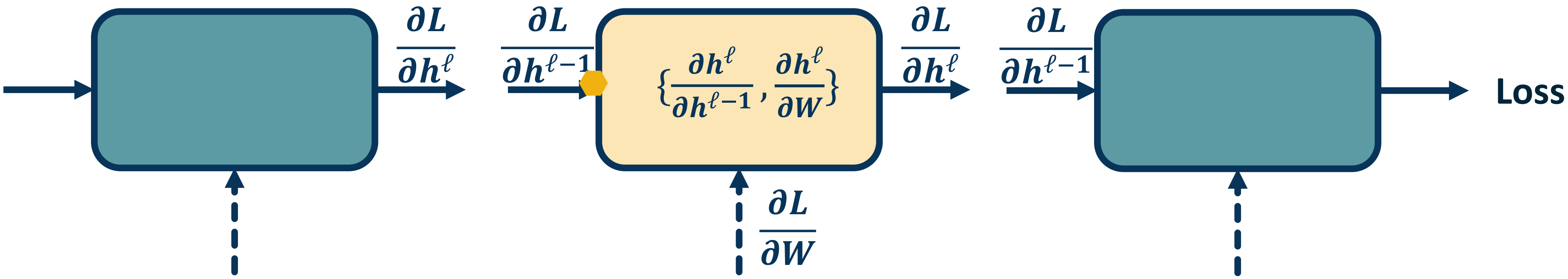


Problem:

- We are given: $\frac{\partial L}{\partial h^{\ell}}$
- We can compute local gradients: $\{\frac{\partial h^{\ell}}{\partial h^{\ell-1}}, \frac{\partial h^{\ell}}{\partial W}\}$
- Compute: $\{\frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W}\}$

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

- We want to compute: $\left\{\frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W}\right\}$



- We will use the *chain rule* to do this:

Chain Rule: $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$

- We can compute **local gradients**: $\{\frac{\partial \boldsymbol{h}^{\ell}}{\partial \boldsymbol{h}^{\ell-1}}, \frac{\partial \boldsymbol{h}^{\ell}}{\partial \boldsymbol{W}}\}$
- This is just the **derivative of our function** with respect to its parameters and inputs!

Example: If $\boldsymbol{h}^{\ell} = \boldsymbol{W}\boldsymbol{h}^{\ell-1}$

then $\frac{\partial \boldsymbol{h}^{\ell}}{\partial \boldsymbol{h}^{\ell-1}} = \boldsymbol{W}$

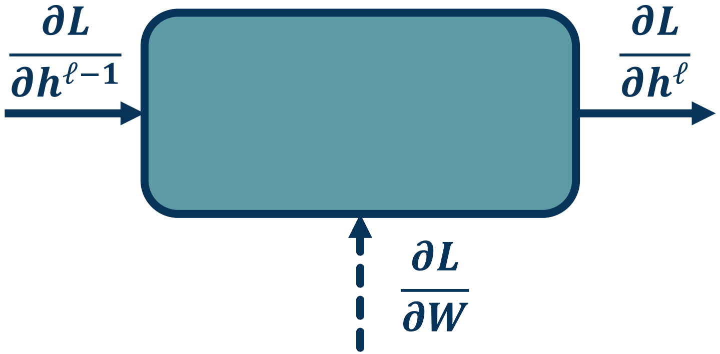
and $\frac{\partial \boldsymbol{h}_i^{\ell}}{\partial \boldsymbol{w}_i} = \boldsymbol{h}^{\ell-1,T}$

- We will use the **chain rule** to compute: $\{\frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W}\}$

- **Gradient of loss w.r.t. inputs:** $\frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial h^{\ell-1}}$

Given by upstream module **(upstream gradient)**

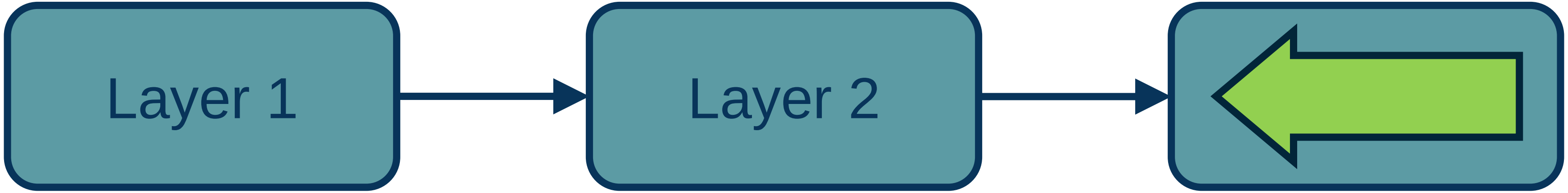
- **Gradient of loss w.r.t. weights:** $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial W}$



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

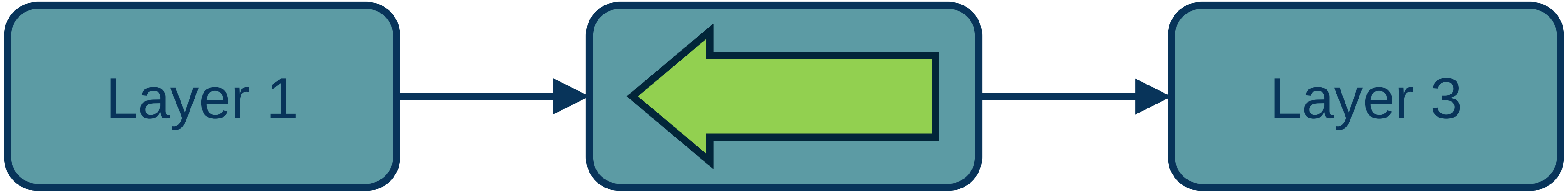
Step 2: Compute Gradients wrt parameters: **Backward Pass**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

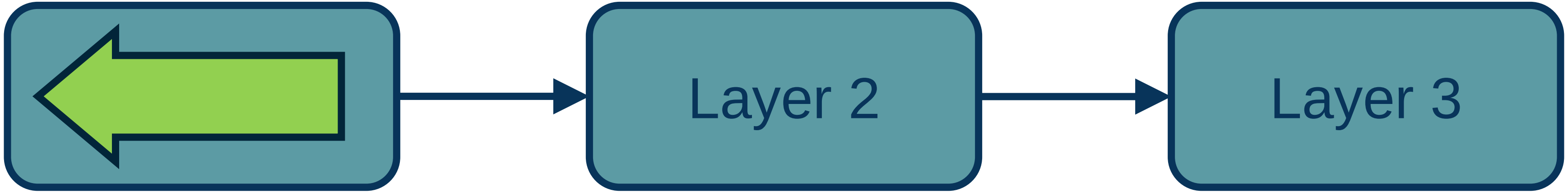
Step 2: Compute Gradients wrt parameters: **Backward Pass**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**

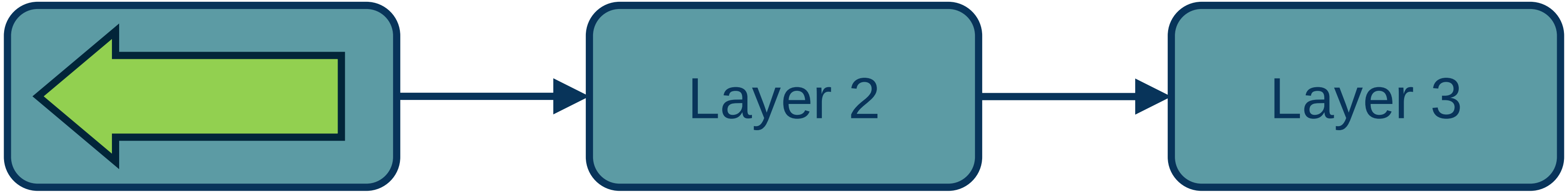


Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**

Step 3: Use **gradient** to update **all parameters** at the end



$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

Backpropagation is the application of gradient descent to a computation graph via the chain rule!

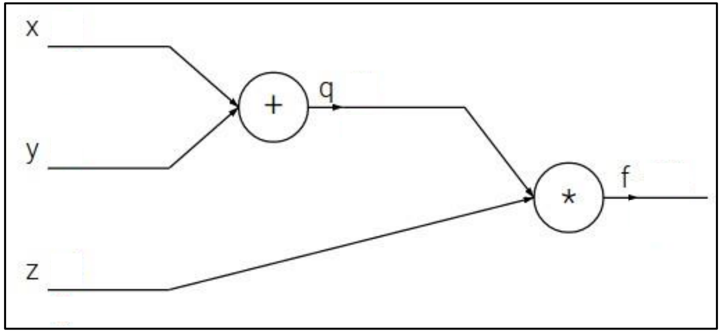
Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Backpropagation: a simple example

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

Backpropagation: a simple example

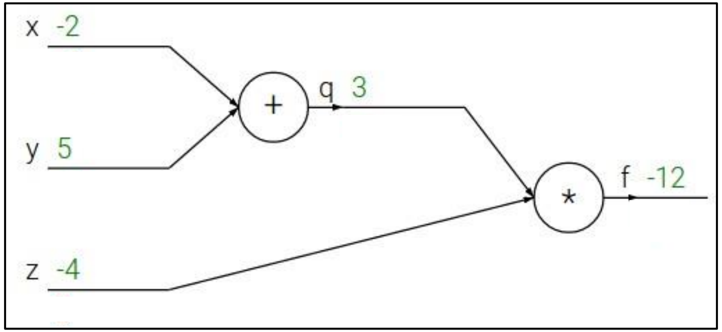
$$f(x, y, z) = (x + y)z$$



Backpropagation: a simple example

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

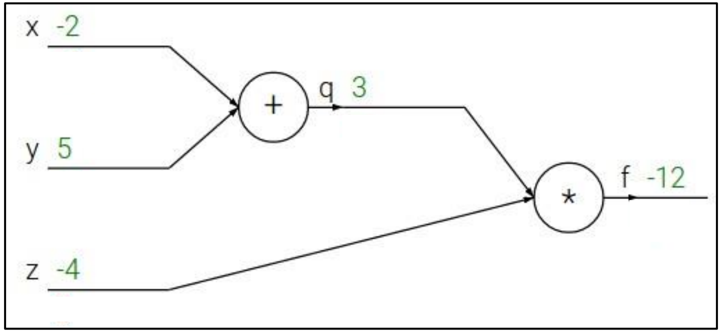
e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4



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

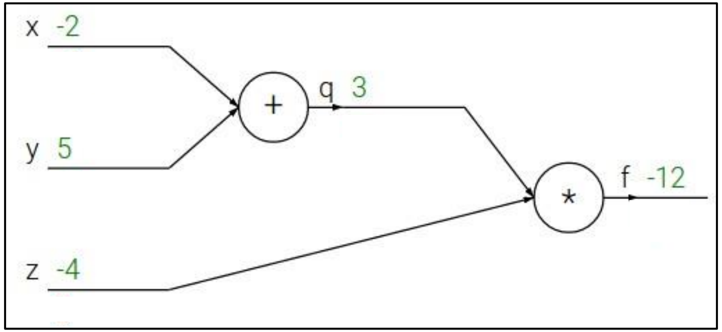
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4

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

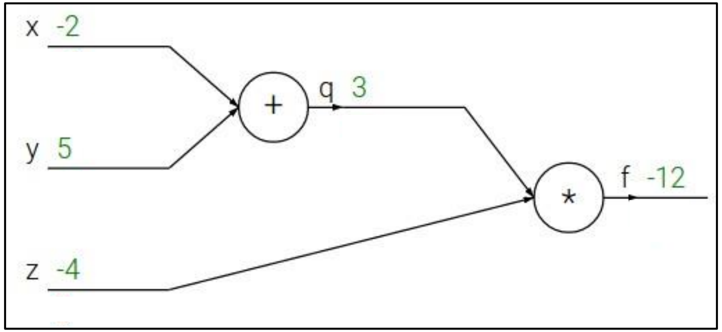


Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4



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

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

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

Backpropagation: a simple example

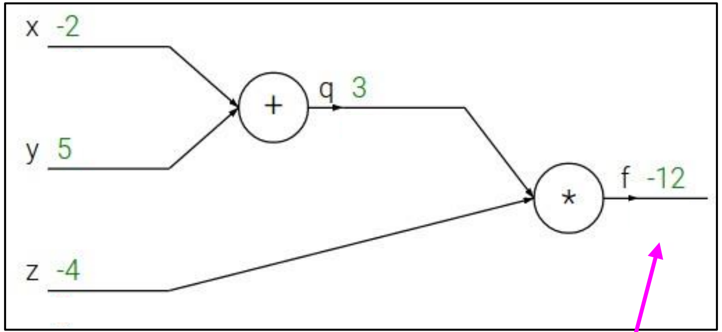
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Backpropagation: a simple example

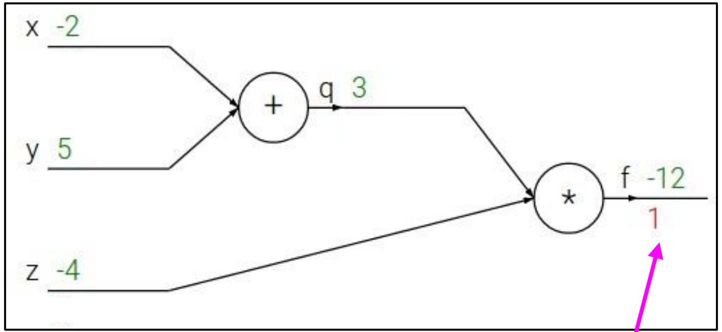
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Backpropagation: a simple example

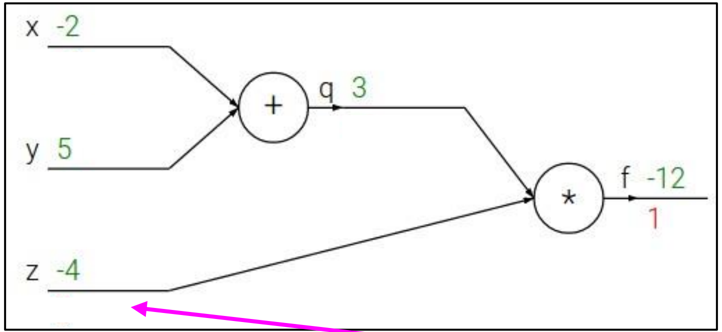
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Backpropagation: a simple example

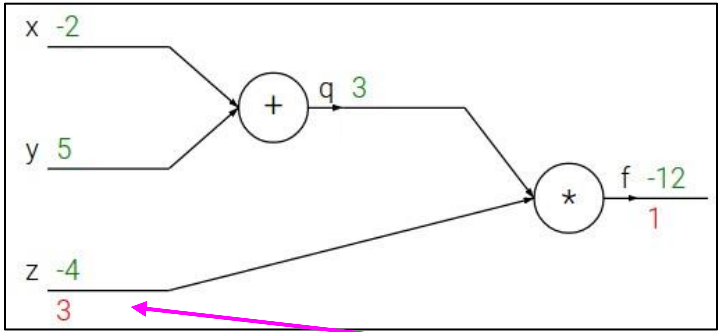
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



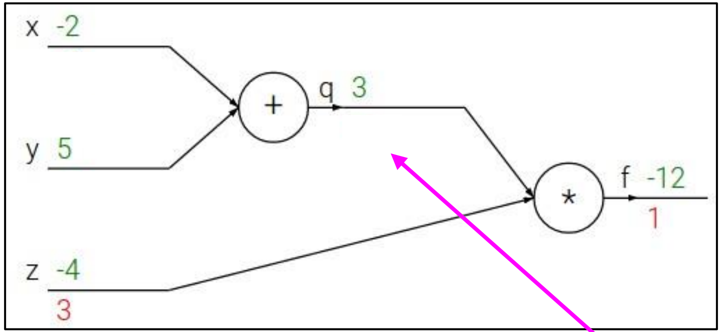
$$\frac{\partial f}{\partial z}$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Backpropagation: a simple example

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

e.g. x = -2, y = 5, z = -4



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

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

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

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

Backpropagation: a simple example

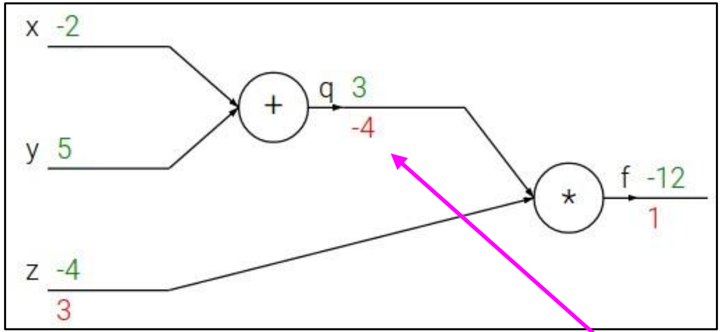
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Backpropagation: a simple example

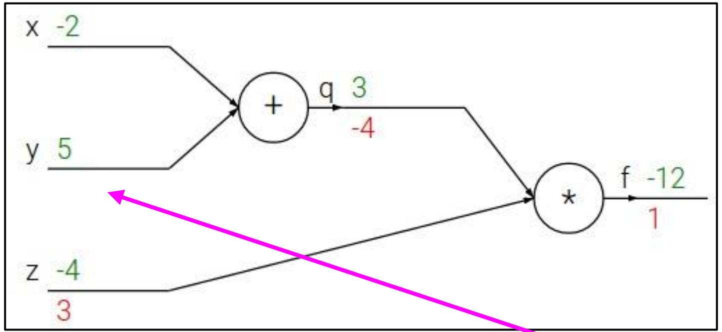
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

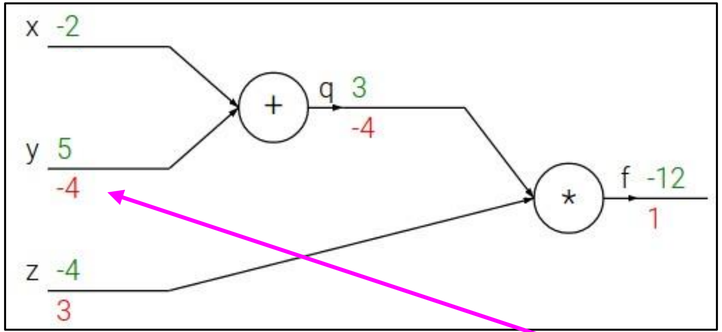
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

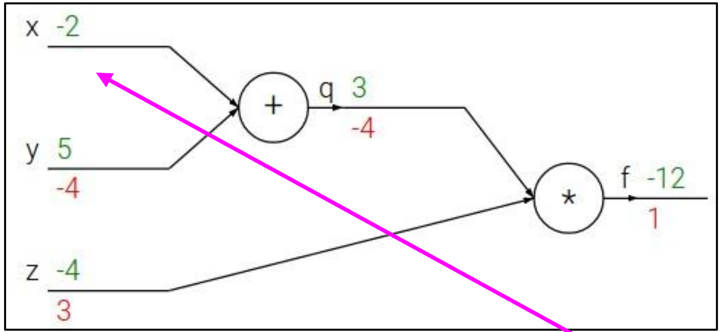
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

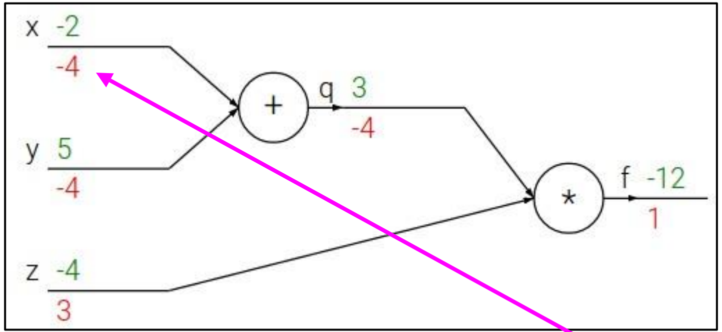
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

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



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

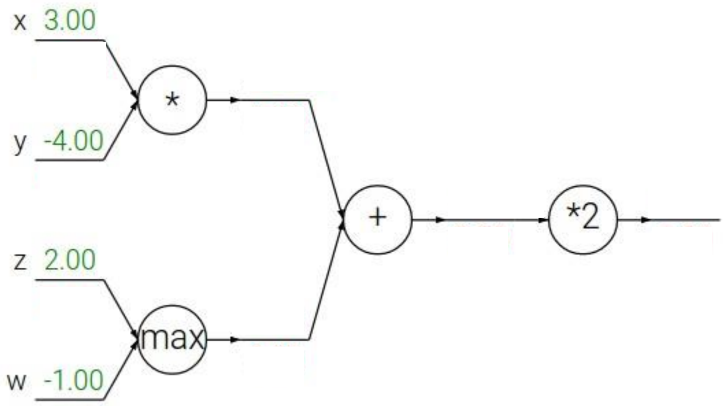
Chain rule:

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

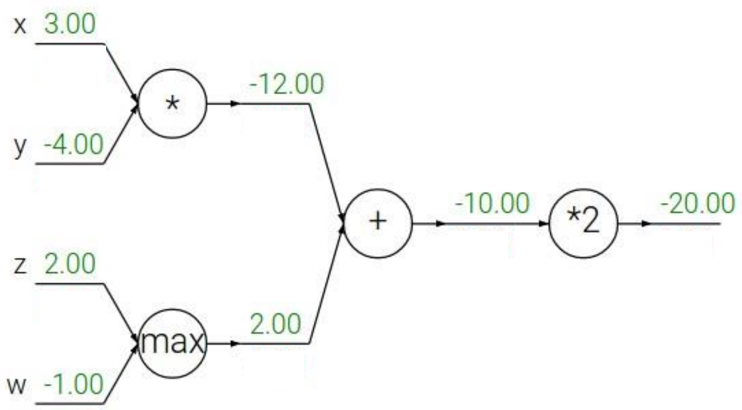
Upstream gradient

Local gradient

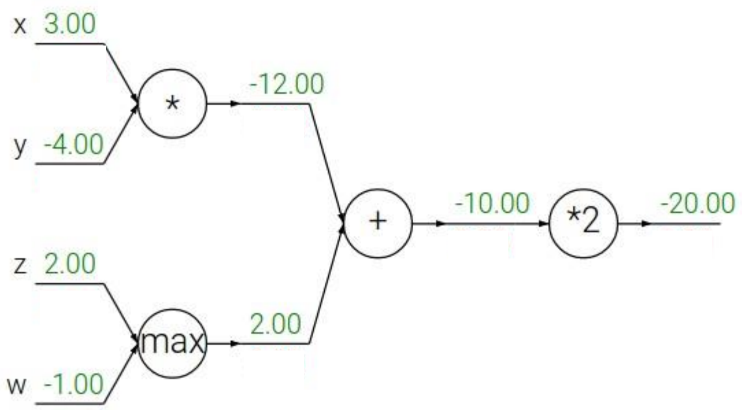
Backpropagation: a simple example



Backpropagation: a simple example

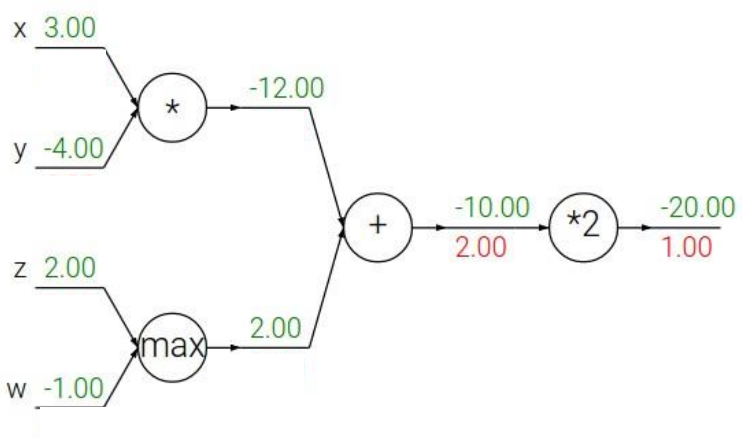


Patterns in backward flow



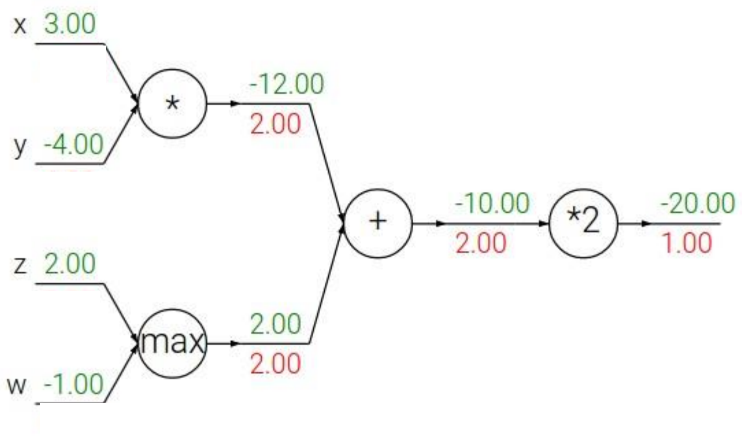
Patterns in backward flow

Q: What is an **add** gate?



Patterns in backward flow

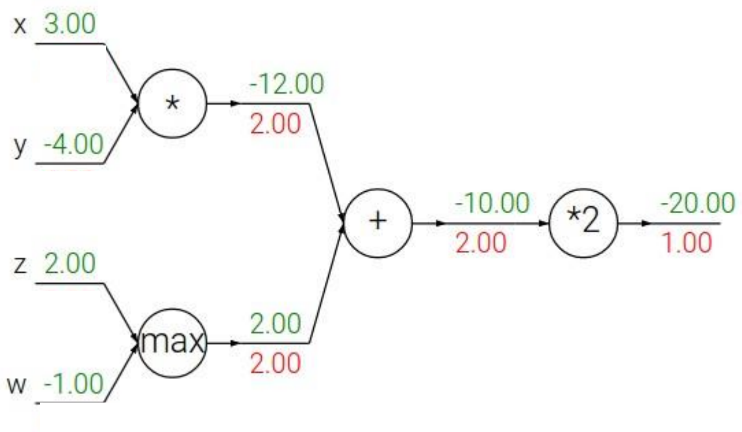
add gate: gradient distributor



Patterns in backward flow

add gate: gradient distributor

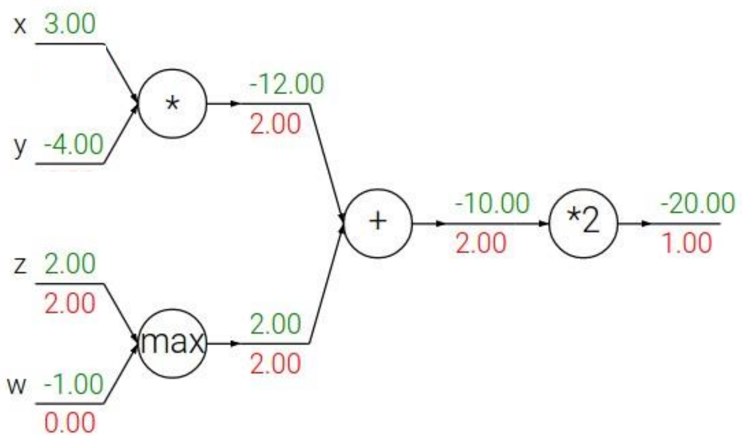
Q: What is a **max** gate?



Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

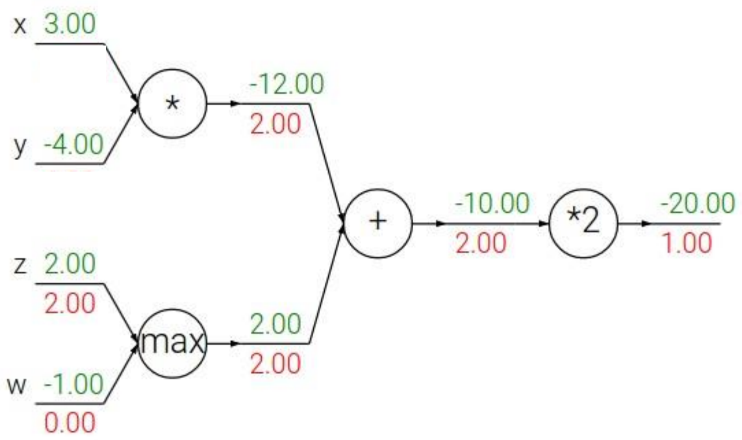


Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

Q: What is a **mul** gate?

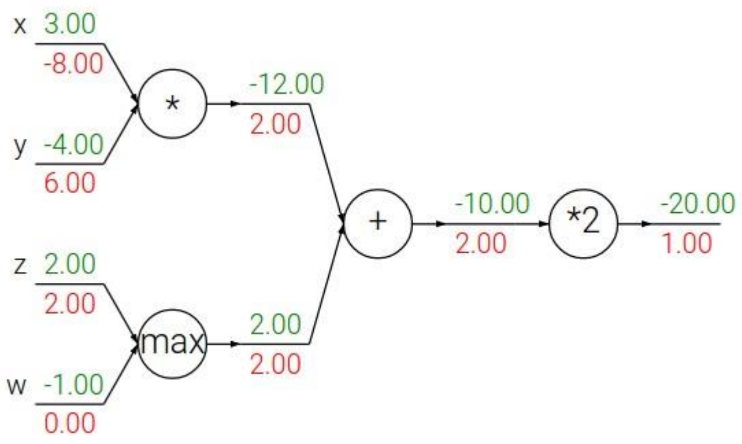


Patterns in backward flow

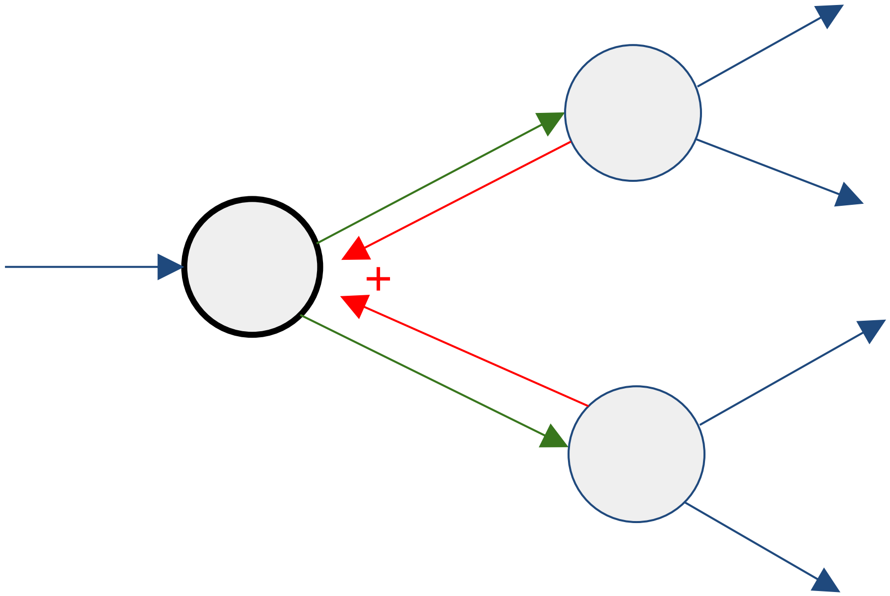
add gate: gradient distributor

max gate: gradient router

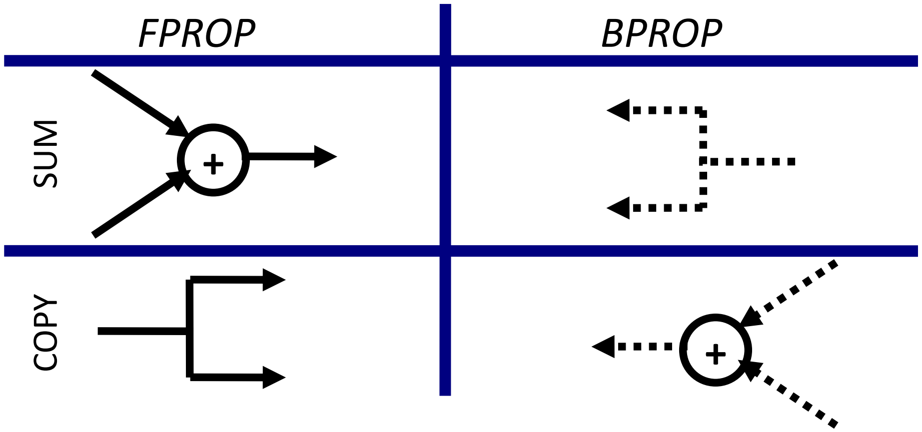
mul gate: gradient switcher



Gradients add at branches



Duality in Fprop and Bprop



Deep Learning = Differentiable Programming

- Computation = Graph
  - Input = Data + Parameters
  - Output = Loss
  - Scheduling = Topological ordering

- What do we need to do?
  - Generic code for representing the graph of modules
  - Specify modules (both forward and backward function)
  - Backpropagation implementation on the graph