

# CS 188: Artificial Intelligence

## Deep Learning II

Instructors: Pieter Abbeel & Anca Dragan --- University of California, Berkeley

[These slides were created by Dan Klein, Pieter Abbeel, Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

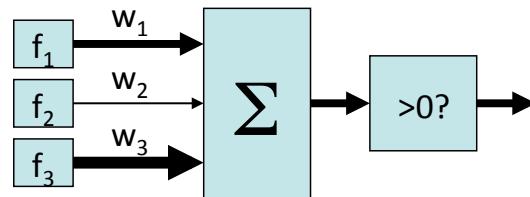
# Fun Neural Net Demo Site

---

- Demo-site:
  - <http://playground.tensorflow.org/>

# Perceptron

---



- Objective: Classification Accuracy

$$l^{\text{acc}}(w) = \frac{1}{m} \sum_{i=1}^m \left( \text{sign}(w^\top f(x^{(i)})) == y^{(i)} \right)$$

- Issue: many plateaus → how to measure incremental progress?

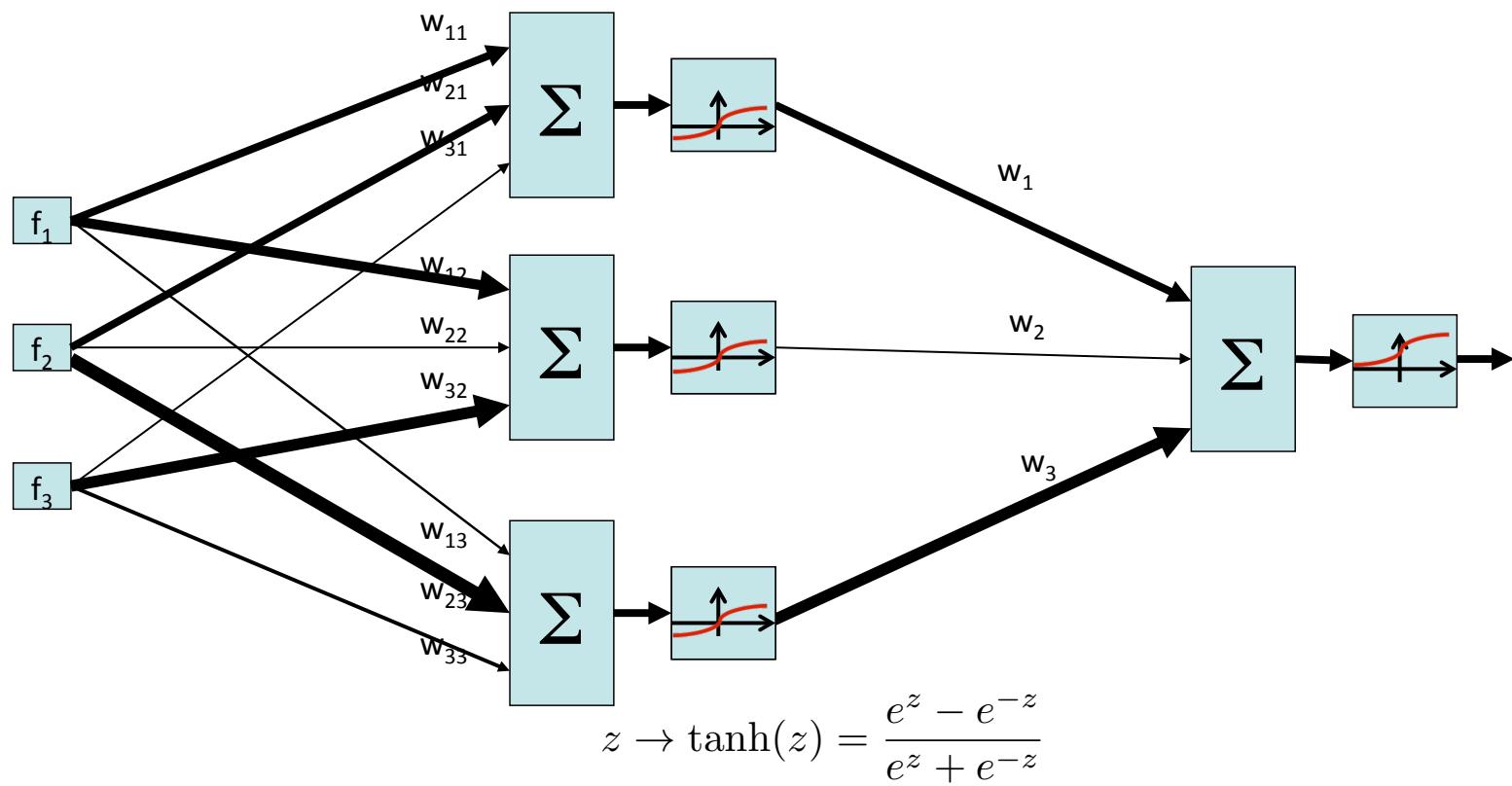
# Soft-Max

---

- Score for  $y=1$ :  $w^\top f(x)$     Score for  $y=-1$ :  $-w^\top f(x)$
- Probability of label:  
$$p(y = 1 | f(x); w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$
$$p(y = -1 | f(x); w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$
- Objective: 
$$l(w) = \prod_{i=1}^m p(y = y^{(i)} | f(x^{(i)}); w)$$
- Log: 
$$ll(w) = \sum_{i=1}^m \log p(y = y^{(i)} | f(x^{(i)}); w)$$

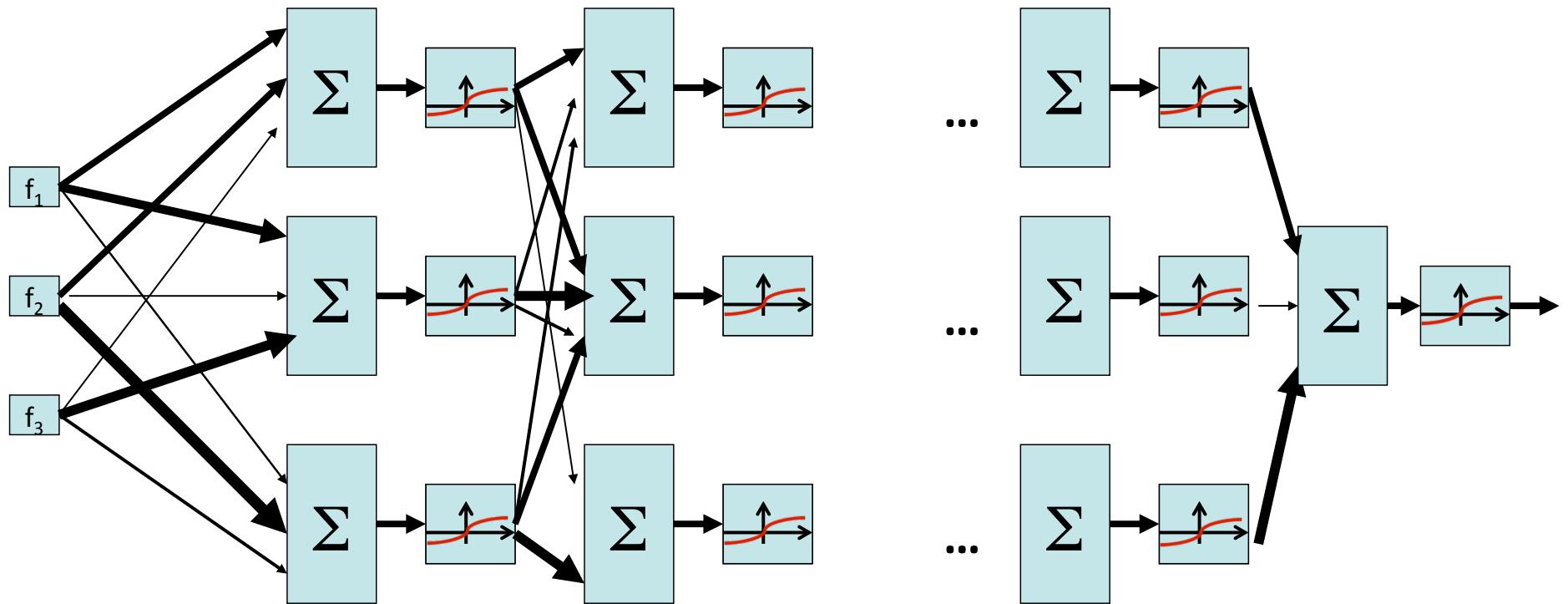
# Two-Layer Neural Network

---



# N-Layer Neural Network

---



# Multi-class Softmax

---

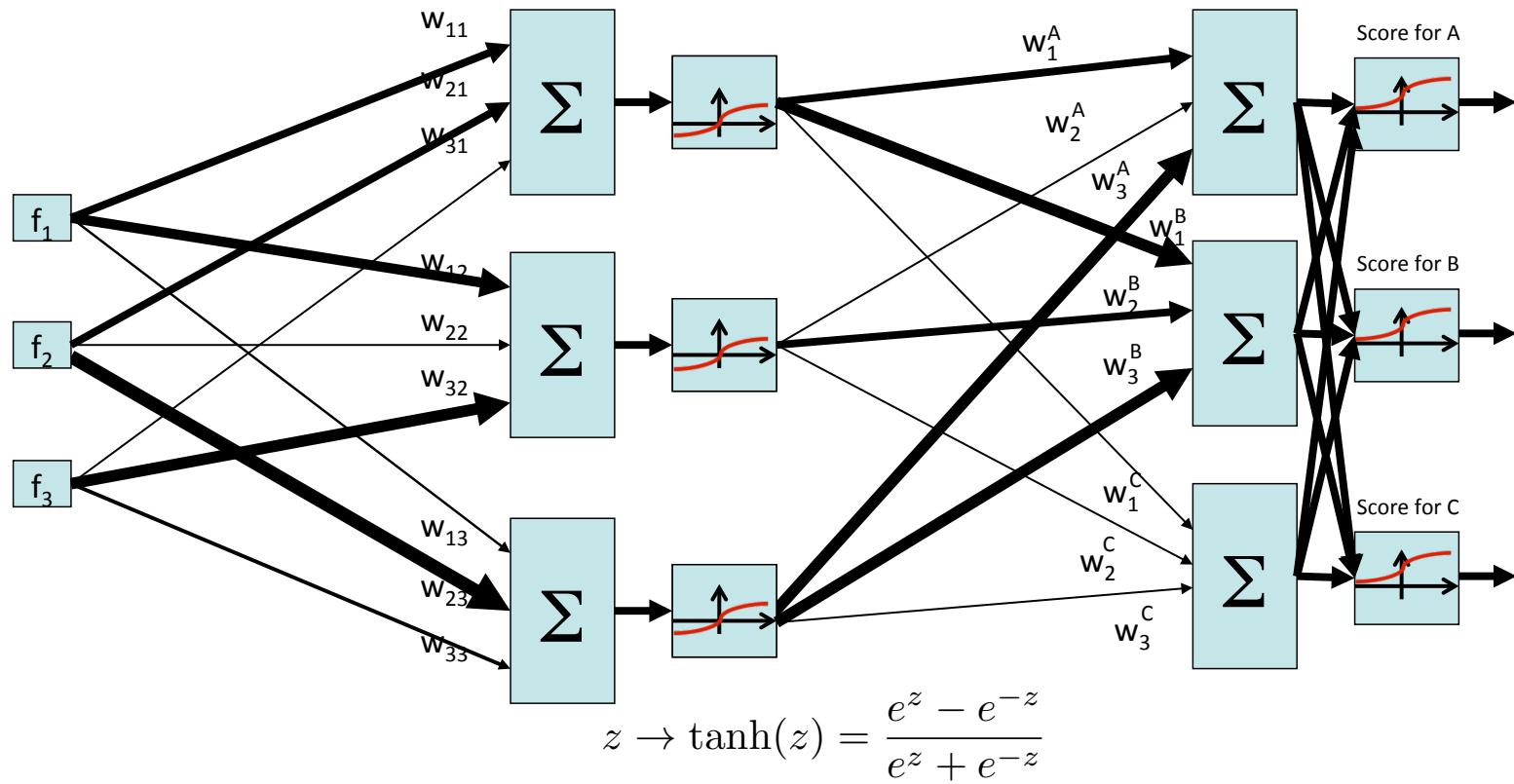
- 3-class softmax – classes A, B, C
  - 3 weight vectors:  $w_A, w_B, w_C$
- Probability of label A: (similar for B, C)

$$p(y = A | f(x); w) = \frac{e^{w_A^\top f(x)}}{e^{w_A^\top f(x)} + e^{w_B^\top f(x)} + e^{w_C^\top f(x)}}$$

- Objective:  $l(w) = \prod_{i=1}^m p(y = y^{(i)} | f(x^{(i)}; w))$
- Log:  $ll(w) = \sum_{i=1}^m \log p(y = y^{(i)} | f(x^{(i)}; w))$

# Multi-class Two-Layer Neural Network

---



# Steepest Descent

---

- Idea:
  - Start somewhere
  - Repeat: Take a step in the steepest descent direction

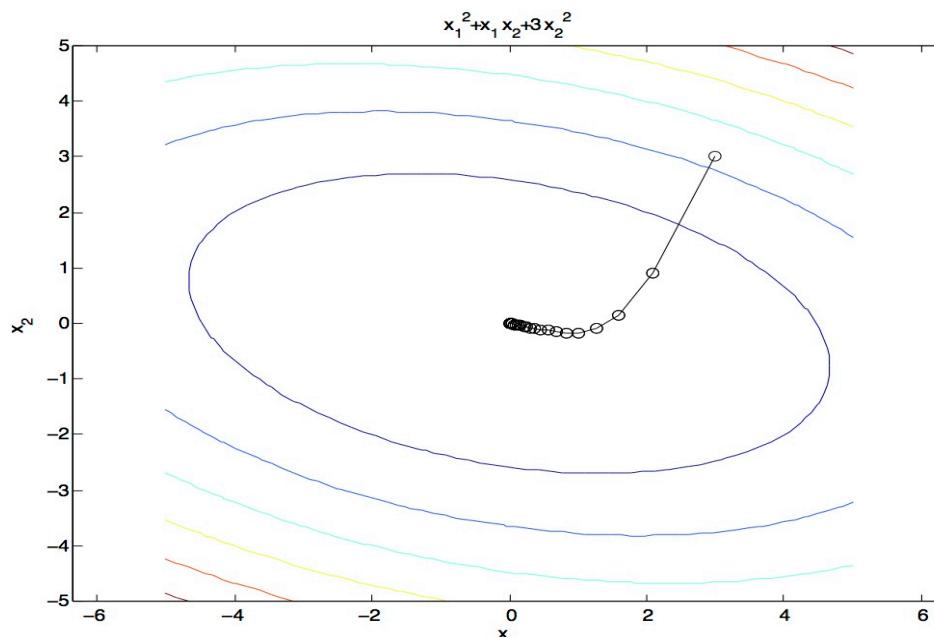


Figure source: Mathworks

# What is the Steepest Descent Direction?

---

$$\min_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \epsilon} g(w + \Delta)$$

- First-Order Taylor Expansion:  $g(w + \Delta) \approx g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$
- Steepest Descent Direction:  $\min_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \epsilon} \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$
- Recall:  $\min_{a: \|a\| \leq \epsilon} a^\top b \quad \rightarrow \quad a = -b \frac{\epsilon}{\|b\|}$
- Hence, solution:  $-\nabla g \frac{\epsilon}{\|\nabla g\|}$   $\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \end{bmatrix}$

# Generally, Steepest Direction

---

- Steepest Direction = direction of the gradient

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

# Optimization Procedure 2: Momentum

---

- Gradient Descent

- Init:  $w$
- For  $i = 1, 2, \dots$

$$w \leftarrow w - \alpha * \nabla g(w)$$

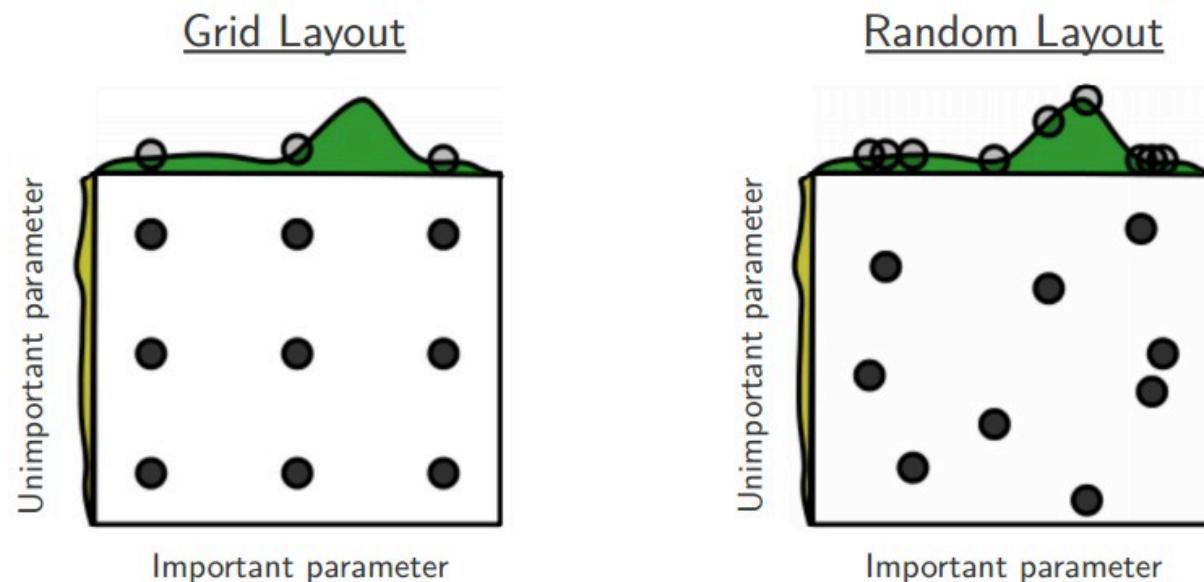
- Momentum

- Init:  $w$
- For  $i = 1, 2, \dots$

$$\begin{aligned} v &\leftarrow \mu * v - \alpha * \nabla g(w) \\ w &\leftarrow w + v \end{aligned}$$

- Physical interpretation as ball rolling down the loss function + friction (mu coefficient).
- mu = usually ~0.5, 0.9, or 0.99 (Sometimes annealed over time, e.g. from 0.5 -> 0.99)

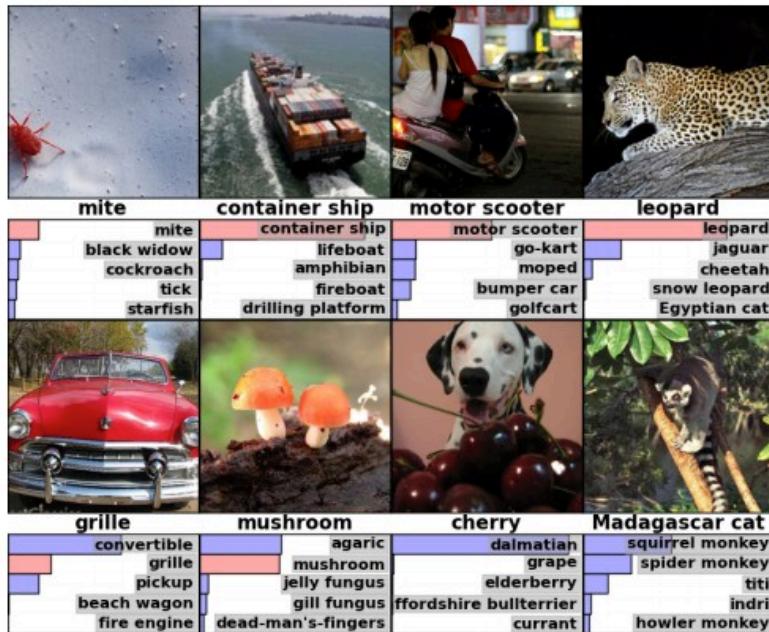
# Hyperparameters: Random Search vs. Grid Search



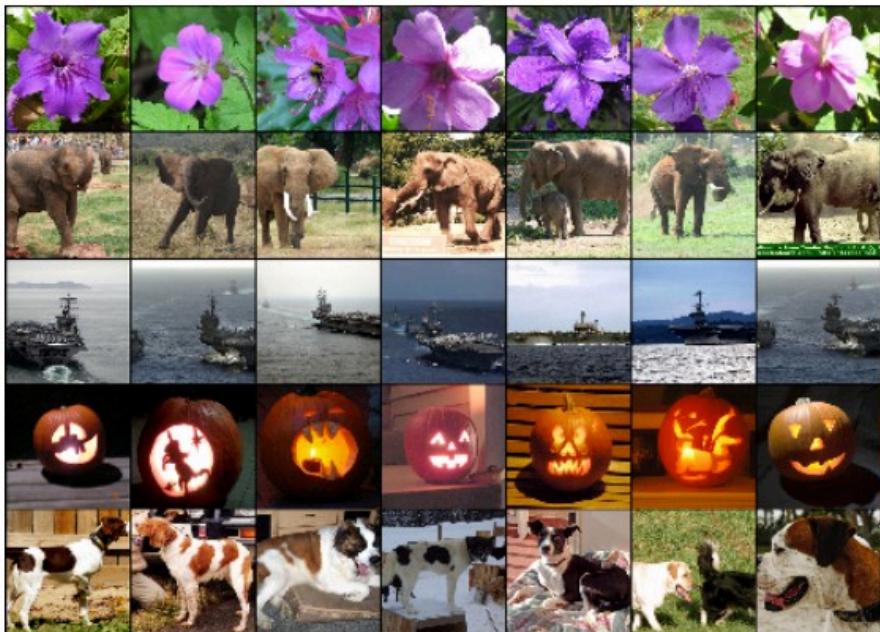
*Random Search for Hyper-Parameter Optimization*  
Bergstra and Bengio, 2012

# ConvNets are everywhere

Classification



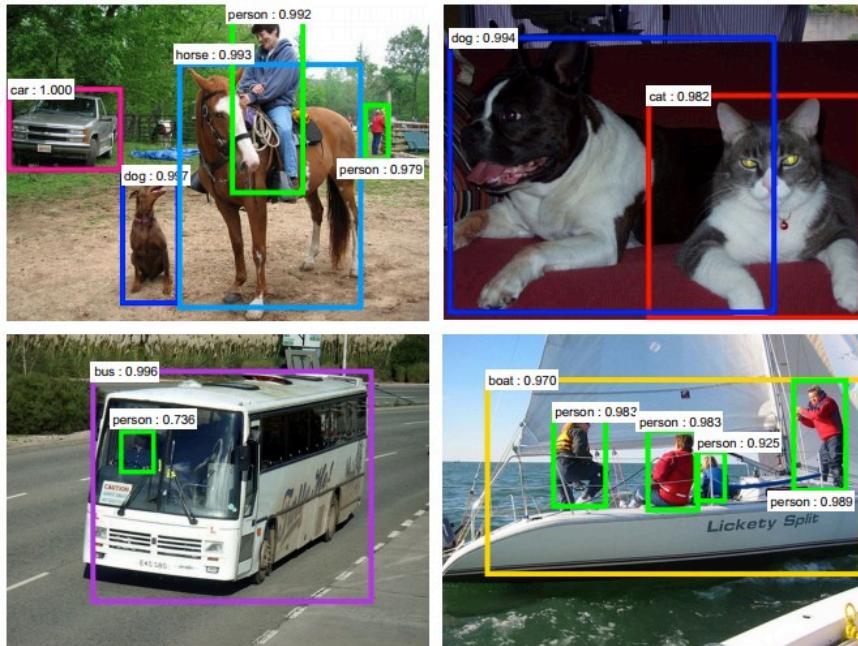
Retrieval



[Krizhevsky 2012]

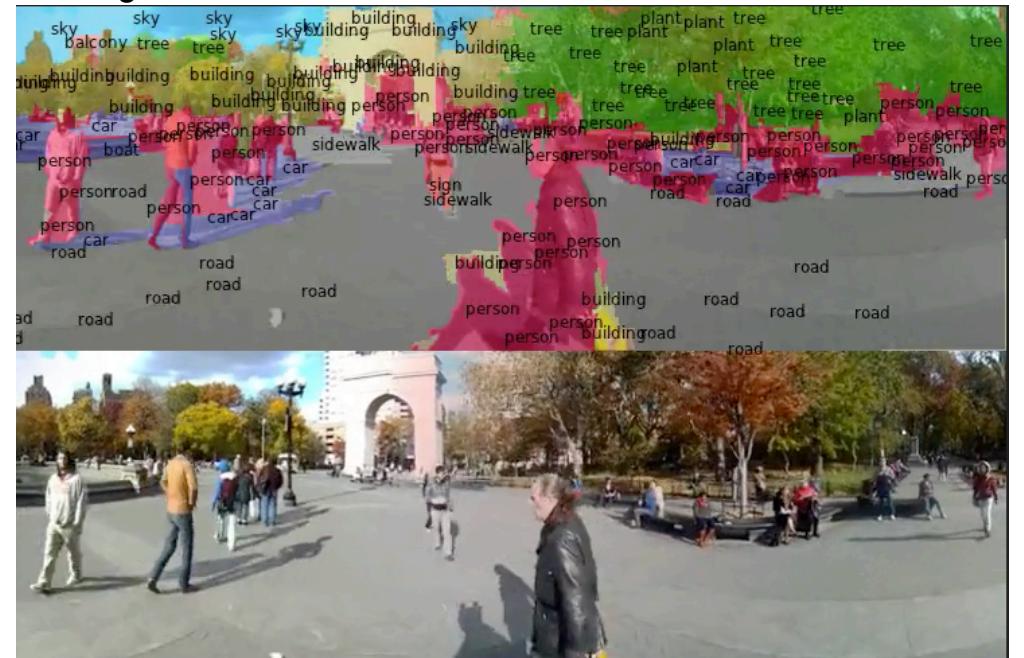
# ConvNets are everywhere

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

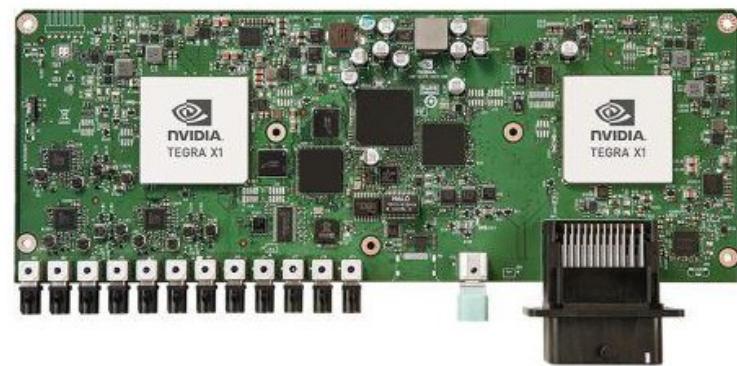


[Farabet et al., 2012]

# ConvNets are everywhere

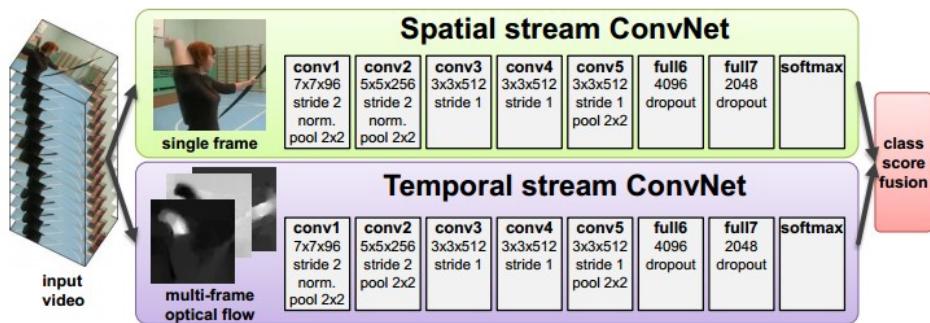
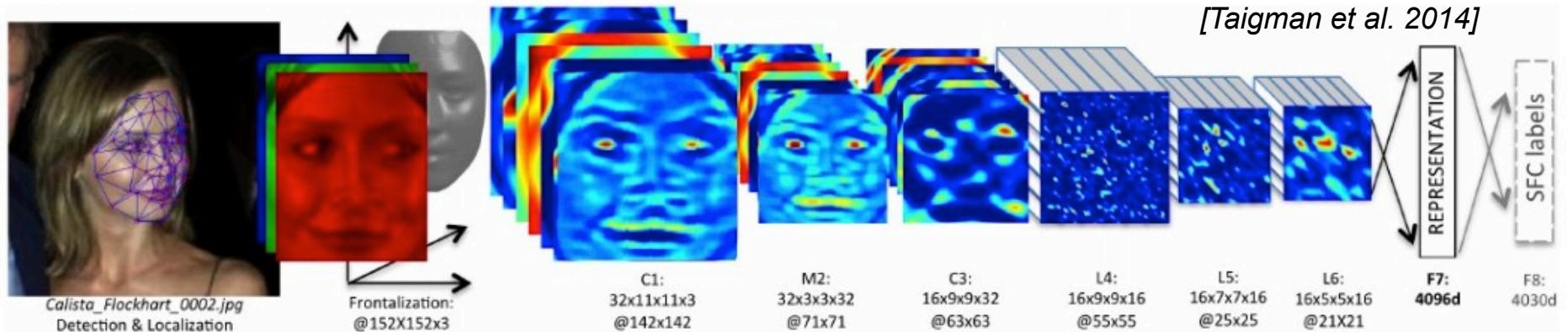


self-driving cars



NVIDIA Tegra X1

# ConvNets are everywhere



*[Goodfellow 2014]*

*[Simonyan et al. 2014]*

# ConvNets are everywhere

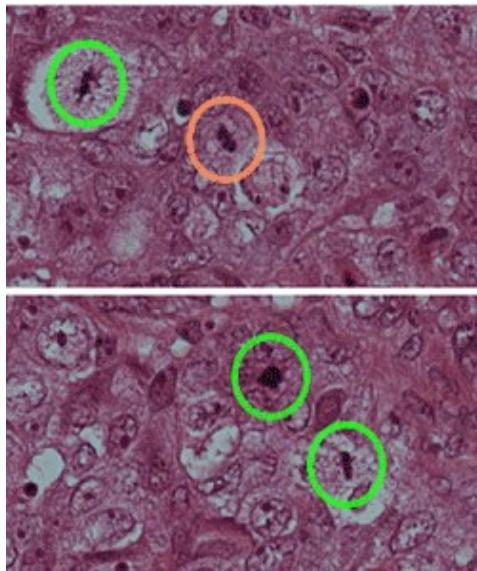


[Toshev, Szegedy 2014]



[Mnih 2013]

# ConvNets are everywhere

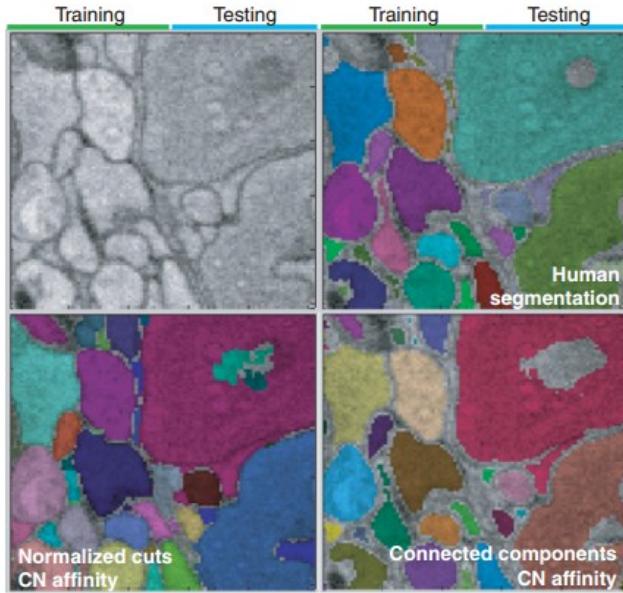


[Ciresan et al. 2013]

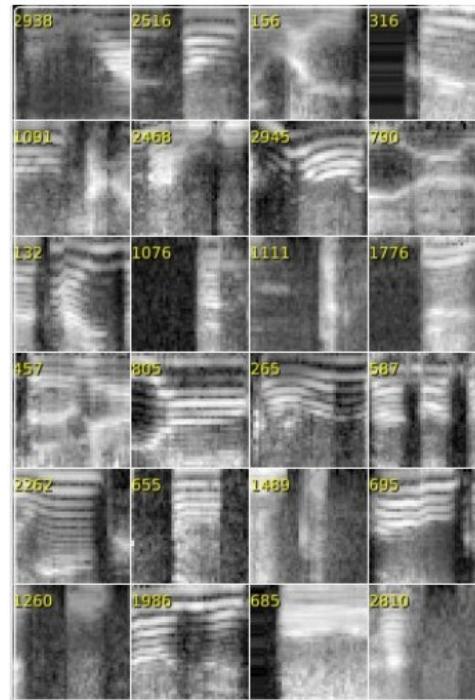


[Sermanet et al. 2011]  
[Ciresan et al.]

# ConvNets are everywhere



[Turaga et al., 2010]



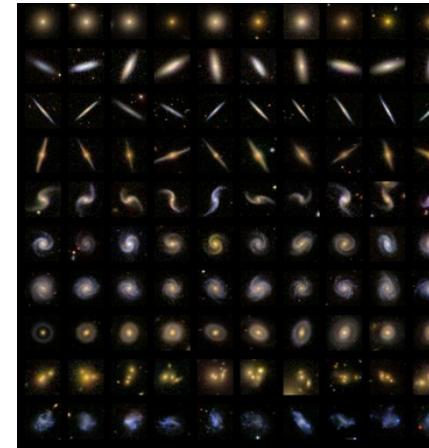
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. Two guys (one naive and one loud mouthed a\*\*) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them. Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horrorfaster buff. Definitely worth checking out.

I just saw this on a local independent station in the New York City area. The cast showed promise but when I saw the director, George Cosmatos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmatos movie I ever saw. He's like a stupid man's Michael Bay - with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

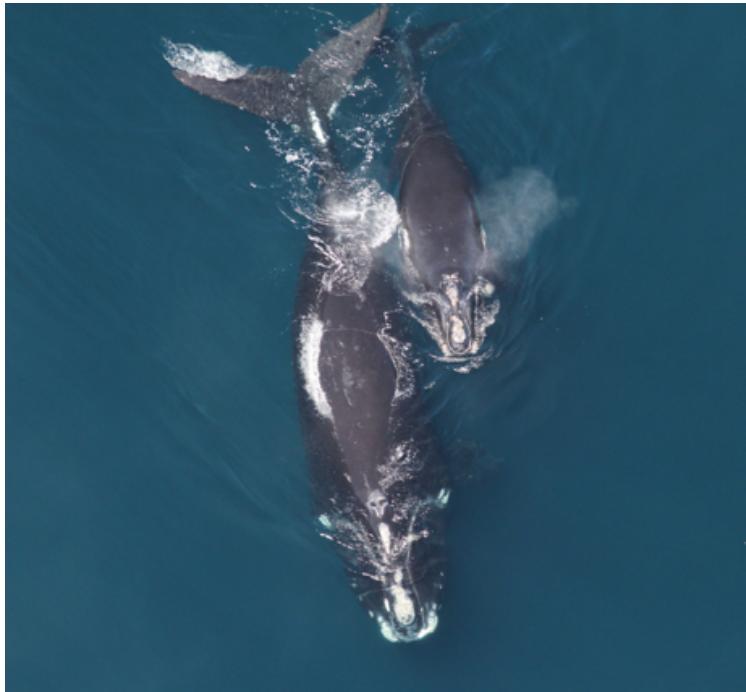
Graphics is far from the best part of the game. This is the number one best TH game in the series! Next to Underground. It deserves strong love. It is an insane game. There are massive levels, massive unlockable characters... it's just a massive game. Waste your money on this game. This is the kind of money that is wasted properly. And even though graphics suck, that's doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might, who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it. What really makes this movie work is Judd Nelson's character and the sometimes clever script. A pretty good script for a person who wrote the Final Destination films and the direction was okay. Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best. I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

[Denil et al. 2014]



# ConvNets are everywhere



*Whale recognition, Kaggle Challenge*

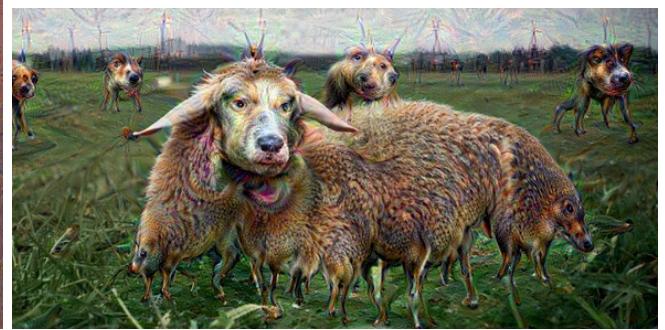
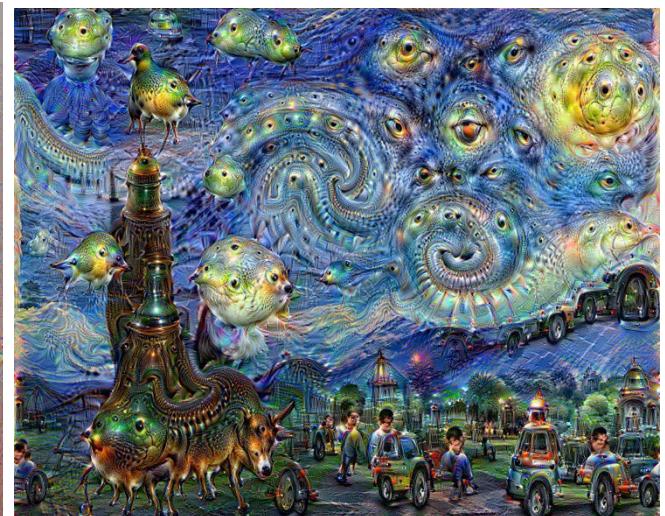
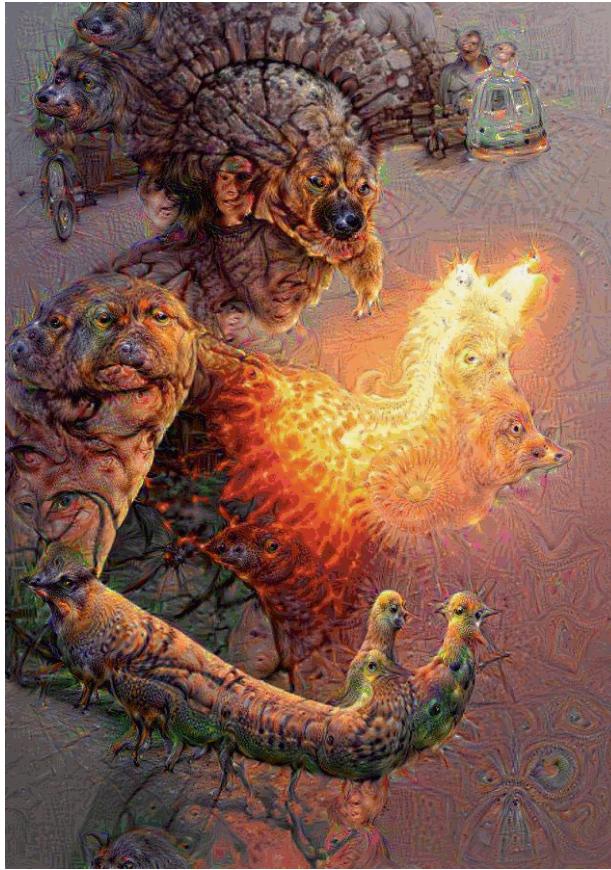
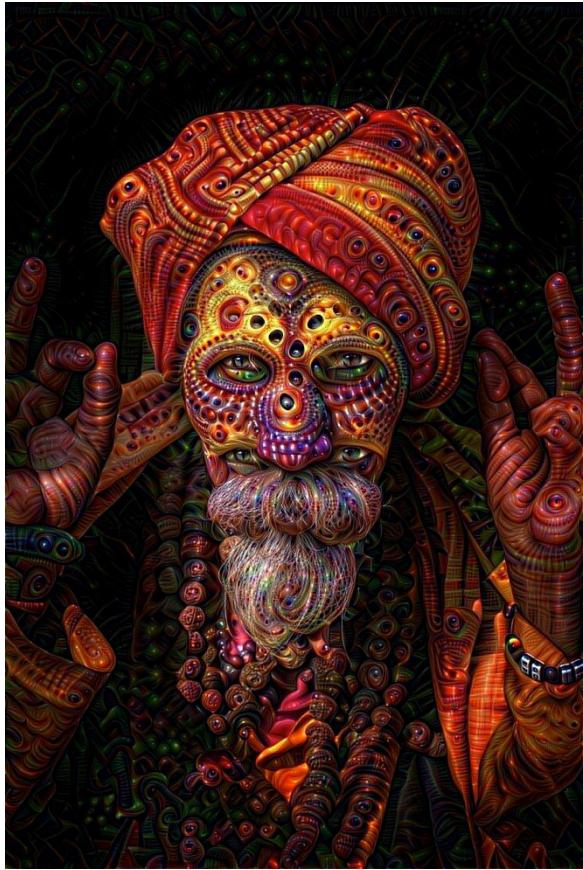


*Mnih and Hinton, 2010*

# Image Captioning

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
 A person riding a motorcycle on a dirt road.	 Two dogs play in the grass.	 A skateboarder does a trick on a ramp.	 A dog is jumping to catch a frisbee.
 A group of young people playing a game of frisbee.	 Two hockey players are fighting over the puck.	 A little girl in a pink hat is blowing bubbles.	 A refrigerator filled with lots of food and drinks.
 A herd of elephants walking across a dry grass field.	 A close up of a cat laying on a couch.	 A red motorcycle parked on the side of the road.	 A yellow school bus parked in a parking lot.

[Vinyals et al., 2015]



[reddit.com/r/deepdream](https://www.reddit.com/r/deepdream)

# Remaining Pieces

---

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient  $\nabla g(w)$ 
  - Backprop
  - Gradient checking

# Mini-batches and Stochastic Gradient Descent

---

- Typical objective:

$$ll(w) = \frac{1}{m} \sum_{i=1}^m \log p(y = y^{(i)} | f(x^{(i)}); w)$$

= average log-likelihood of label given input

$$\approx \frac{1}{k} \sum_{i=1}^k \log p(y = y^{(i)} | f(x^{(i)}); w)$$

= estimate based on mini-batch 1...k

- Mini-batch gradient descent: compute gradient on mini-batch (+ cycle over mini-batches: 1..k, k+1...2k, ... ; make sure to randomize permutation of data!)
- Stochastic gradient descent: k = 1

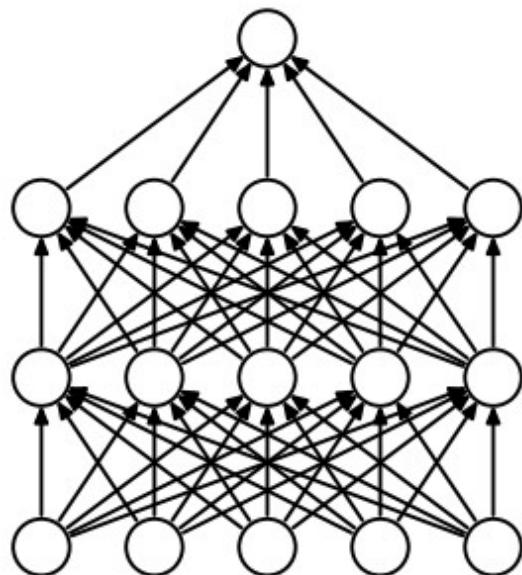
# Remaining Pieces

---

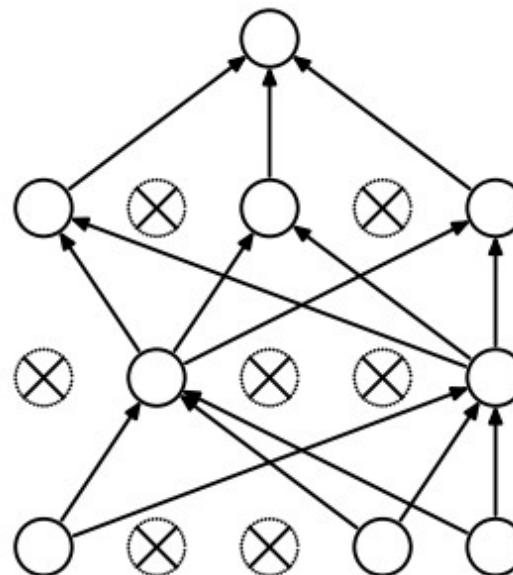
- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient  $\nabla g(w)$ 
  - Gradient checking
  - Backprop

# Regularization: Dropout

“randomly set some neurons to zero in the forward pass”



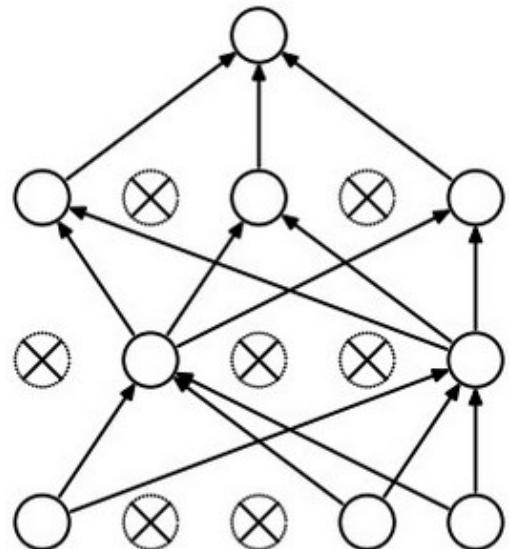
(a) Standard Neural Net



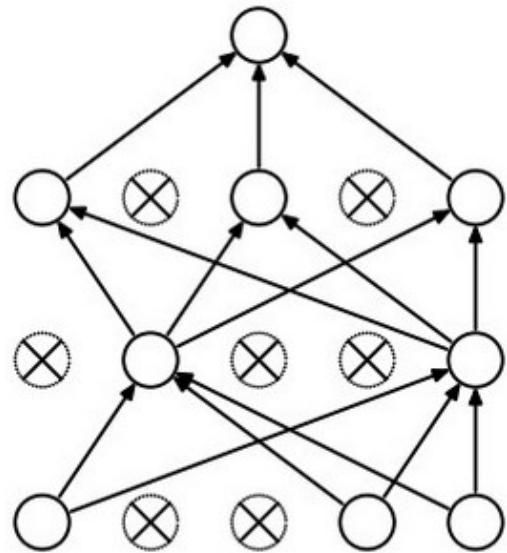
(b) After applying dropout.

[Srivastava et al., 2014]

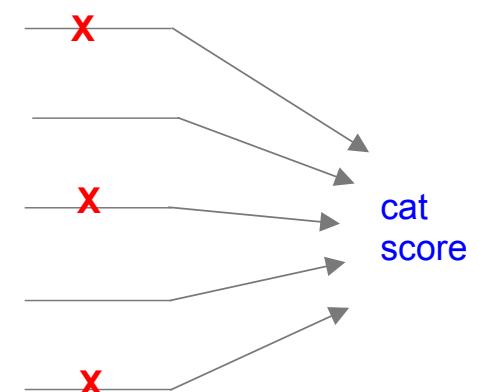
Waaaait a second...  
How could this possibly be a good idea?



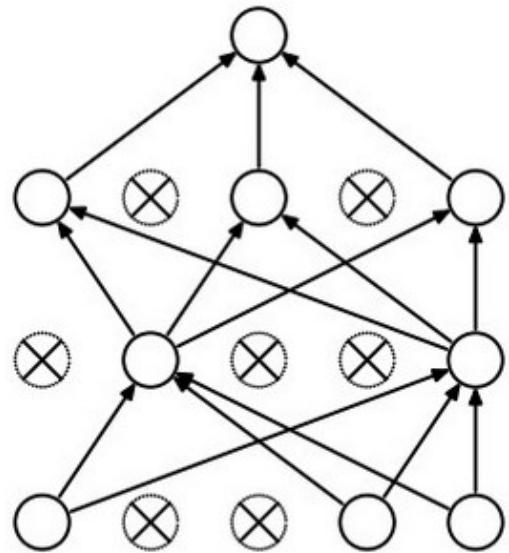
Waaaait a second...  
How could this possibly be a good idea?



Forces the network to have a redundant representation.



Waaaait a second...  
How could this possibly be a good idea?

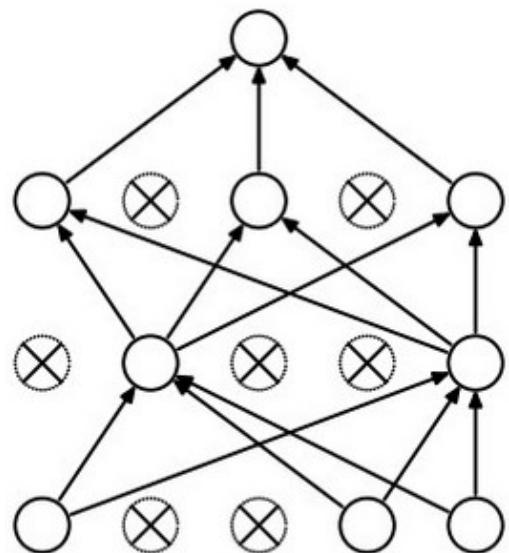


Another interpretation:

Dropout is training a large ensemble  
of models (that share parameters).

Each binary mask is one model, gets  
trained on only ~one datapoint.

At test time....



**Ideally:**

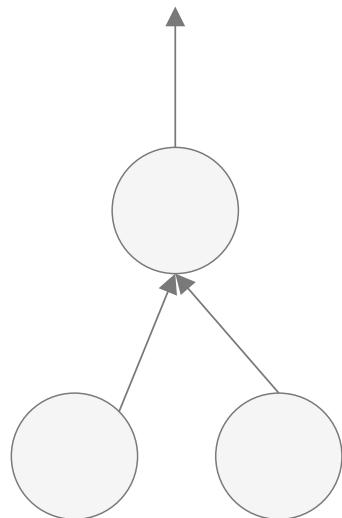
want to integrate out all the noise

**Sampling-based approximation:**

do many forward passes with  
different dropout masks, average all  
predictions

At test time....

Can in fact do this with a single forward pass! (approximately)  
Leave all input neurons turned on (no dropout).

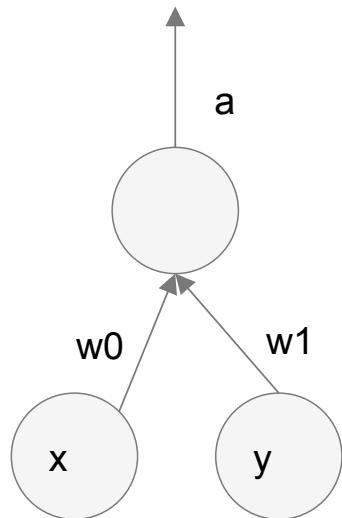


Q: Suppose that with all inputs present at test time the output of this neuron is  $x$ .

What would its output be during training time, in expectation? (e.g. if  $p = 0.5$ )

At test time....

Can in fact do this with a single forward pass! (approximately)  
Leave all input neurons turned on (no dropout).



during test:  $\mathbf{a} = \mathbf{w0} * \mathbf{x} + \mathbf{w1} * \mathbf{y}$

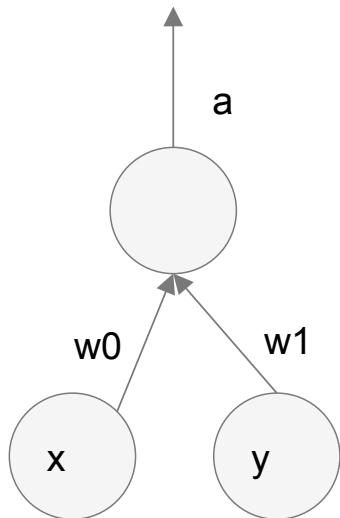
during train:

$$\begin{aligned}\mathbf{E[a]} &= \frac{1}{4} * (\mathbf{w0} * 0 + \mathbf{w1} * 0 \\ &\quad \mathbf{w0} * 0 + \mathbf{w1} * \mathbf{y} \\ &\quad \mathbf{w0} * \mathbf{x} + \mathbf{w1} * 0 \\ &\quad \mathbf{w0} * \mathbf{x} + \mathbf{w1} * \mathbf{y}) \\ &= \frac{1}{4} * (2 \mathbf{w0} * \mathbf{x} + 2 \mathbf{w1} * \mathbf{y}) \\ &= \frac{1}{2} * (\mathbf{w0} * \mathbf{x} + \mathbf{w1} * \mathbf{y})\end{aligned}$$

At test time....

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



$$\text{during test: } \mathbf{a} = \mathbf{w0}^* \mathbf{x} + \mathbf{w1}^* \mathbf{y}$$

during train:

$$\begin{aligned}\mathbf{E}[a] &= \frac{1}{4} * (\mathbf{w0}^* 0 + \mathbf{w1}^* 0 \\ &\quad \mathbf{w0}^* 0 + \mathbf{w1}^* y \\ &\quad \mathbf{w0}^* x + \mathbf{w1}^* 0 \\ &\quad \mathbf{w0}^* x + \mathbf{w1}^* y) \\ &= \frac{1}{4} * (2 \mathbf{w0}^* x + 2 \mathbf{w1}^* y) \\ &= \frac{1}{2} * (\mathbf{w0}^* x + \mathbf{w1}^* y)\end{aligned}$$

With  $p=0.5$ , using all inputs in the forward pass would inflate the activations by 2x from what the network was “used to” during training!  
=> Have to compensate by scaling the activations back down by  $\frac{1}{2}$

# Remaining Pieces

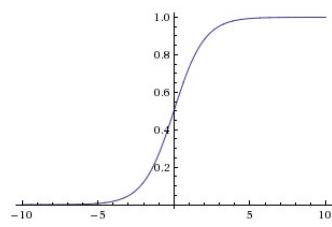
---

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient  $\nabla g(w)$ 
  - Gradient checking
  - Backprop

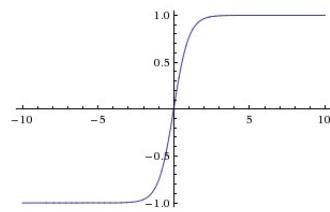
# Activation Functions

## Sigmoid

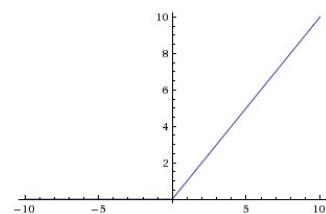
$$\sigma(x) = 1/(1 + e^{-x})$$



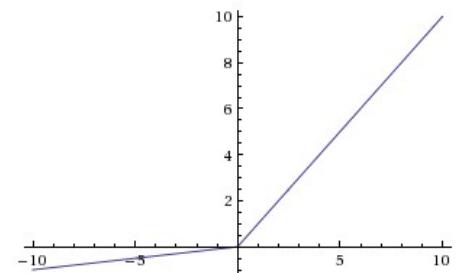
## tanh      tanh(x)



## ReLU      max(0,x)



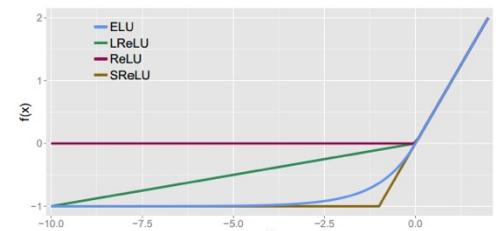
## Leaky ReLU $\max(0.1x, x)$



## Maxout      $\max(w_1^T x + b_1, w_2^T x + b_2)$

## ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

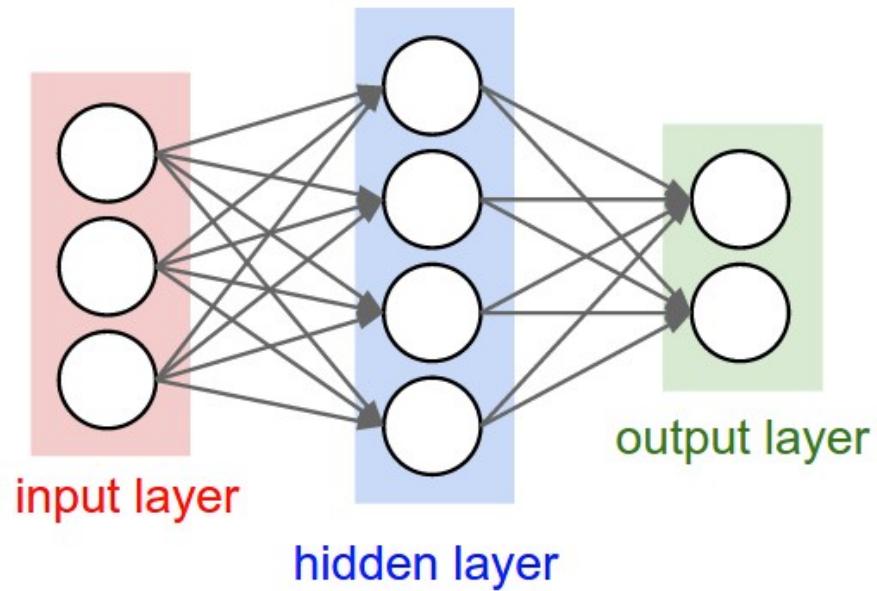


# Remaining Pieces

---

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient  $\nabla g(w)$ 
  - Gradient checking
  - Backprop

- Q: what happens when  $W=0$  init is used?



- First idea: **Small random numbers**  
(gaussian with zero mean and 1e-2 standard deviation)

```
W = 0.01* np.random.randn(D,H)
```

- First idea: **Small random numbers**  
(gaussian with zero mean and 1e-2 standard deviation)

```
W = 0.01* np.random.randn(D,H)
```

Works ~okay for small networks, but can lead to non-homogeneous distributions of activations across the layers of a network.

# Lets look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

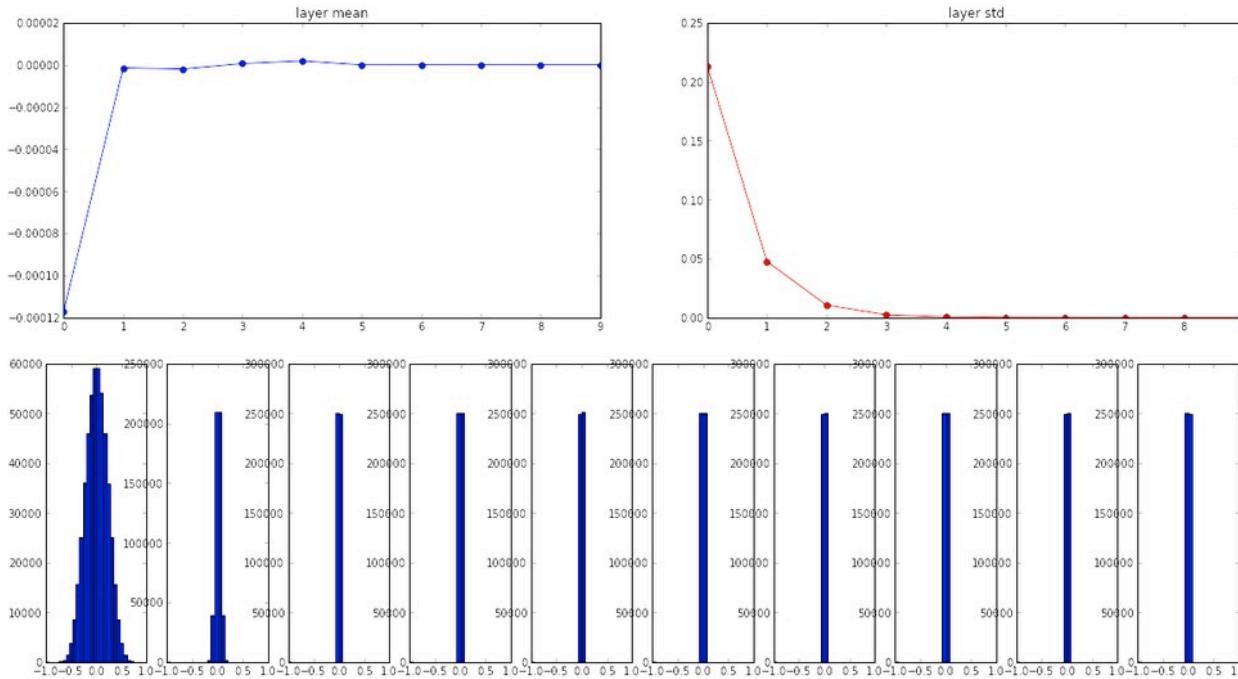
    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer

# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

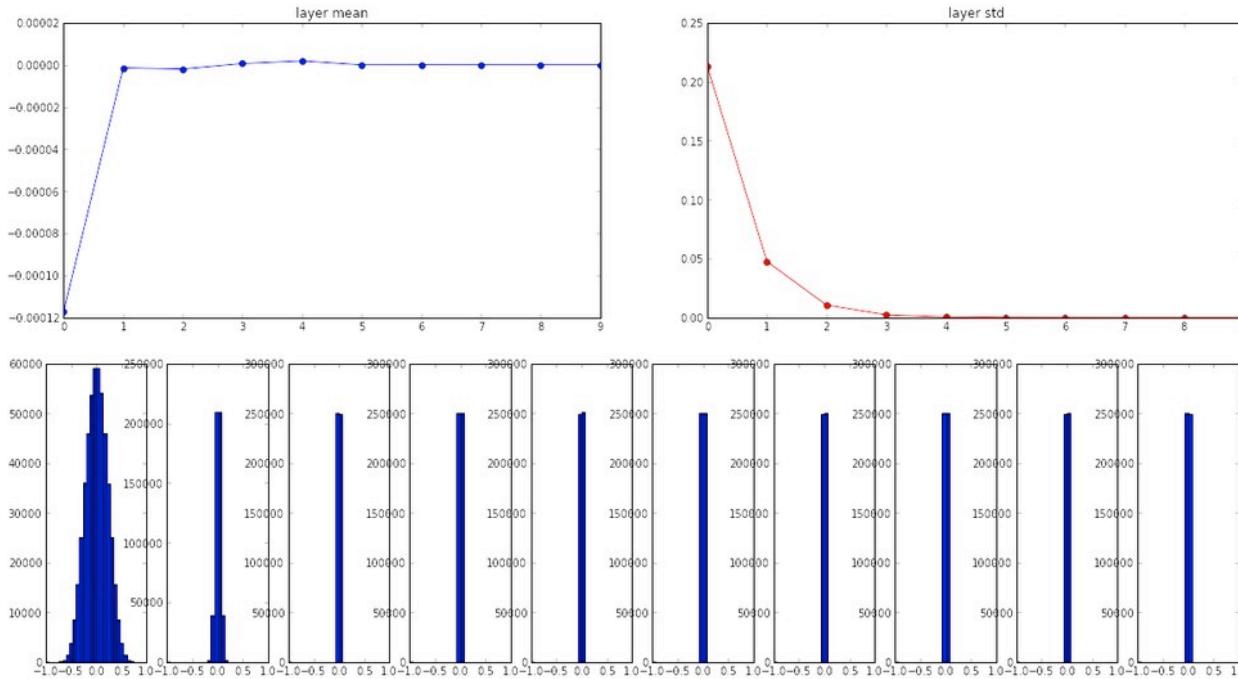
# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.0002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```



```
input layer had mean 0.000927 and std 0.998388  
hidden layer 1 had mean -0.000117 and std 0.213081  
hidden layer 2 had mean -0.000001 and std 0.047551  
hidden layer 3 had mean -0.000002 and std 0.010630  
hidden layer 4 had mean 0.000001 and std 0.000238  
hidden layer 5 had mean 0.000002 and std 0.000532  
hidden layer 6 had mean -0.000000 and std 0.000119  
hidden layer 7 had mean 0.000000 and std 0.000026  
hidden layer 8 had mean -0.000000 and std 0.000006  
hidden layer 9 had mean 0.000000 and std 0.000001  
hidden layer 10 had mean -0.000000 and std 0.000000
```



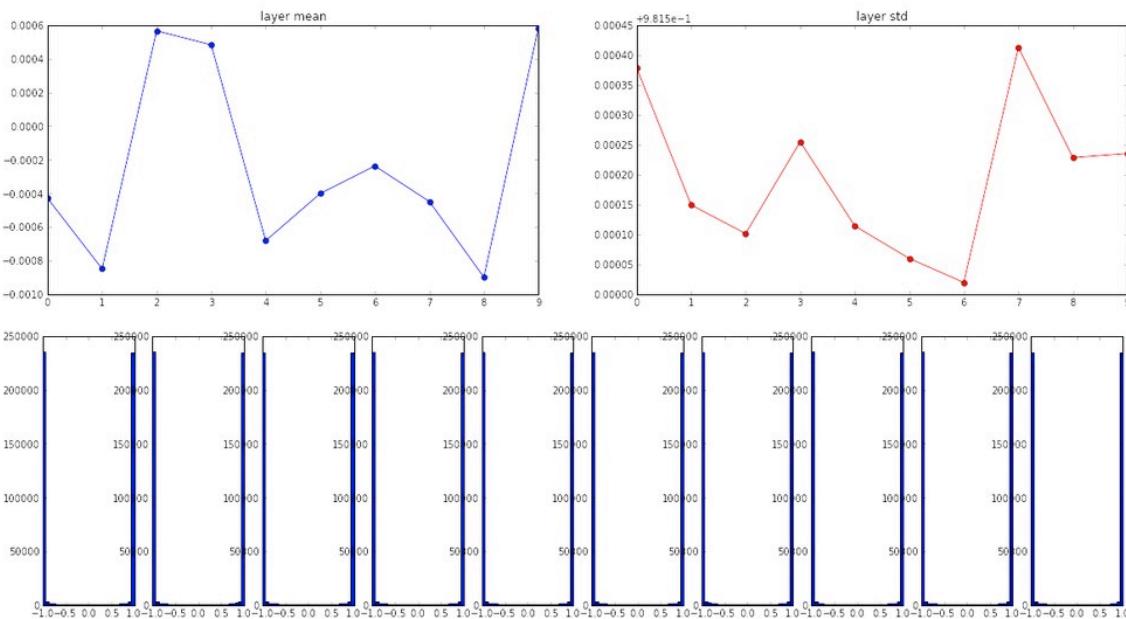
All activations  
become zero!

Q: What do the  
gradients look like?

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

\*1.0 instead of \*0.01



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

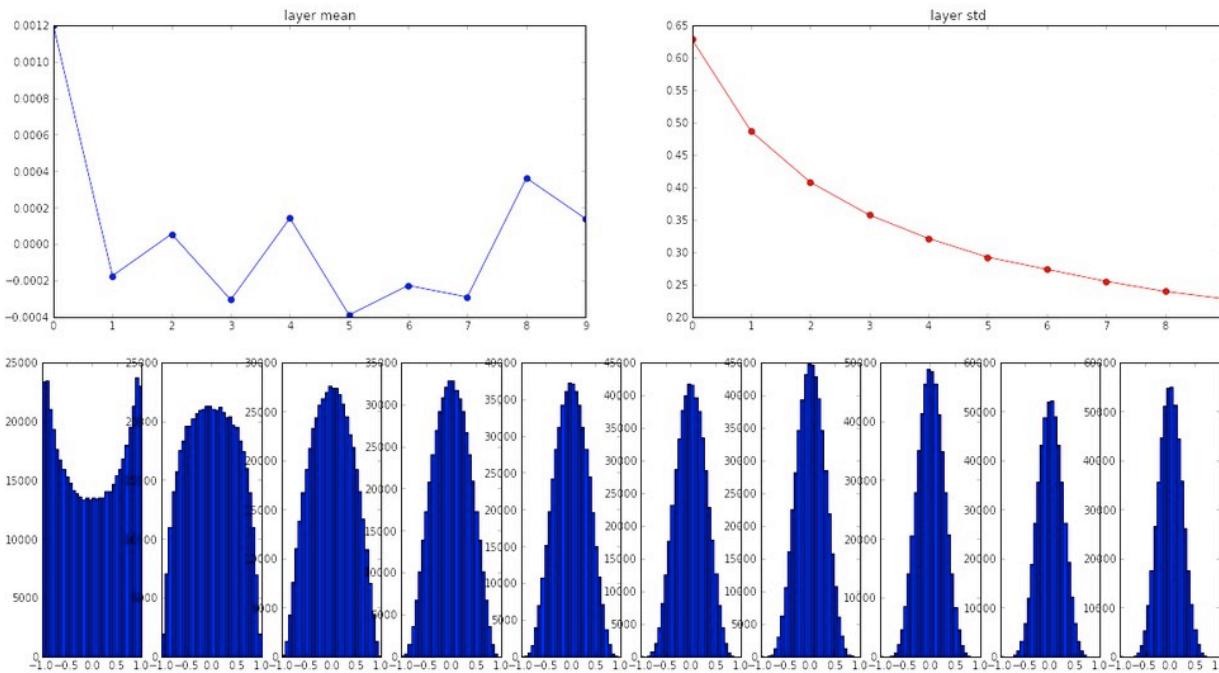
```

input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008

```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

“Xavier initialization”  
[Glorot et al., 2010]



**Reasonable initialization.**  
(Mathematical derivation assumes linear activations)

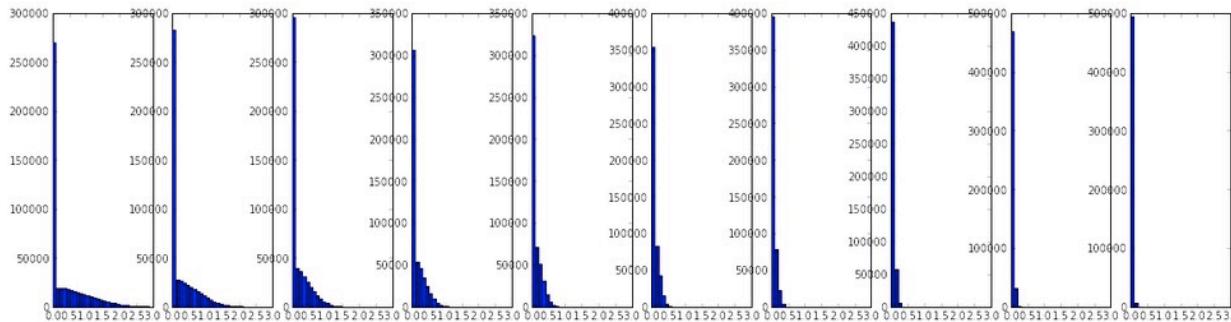
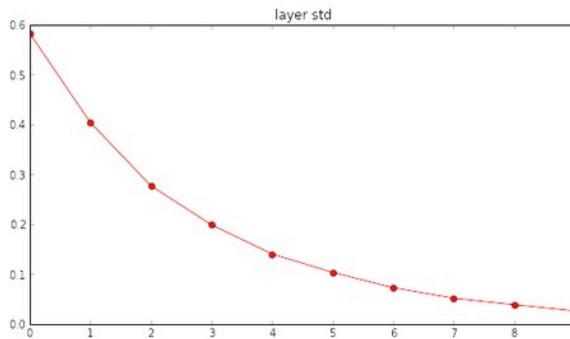
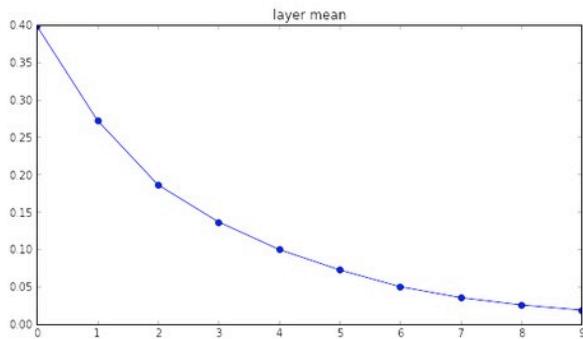
```

input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076

```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

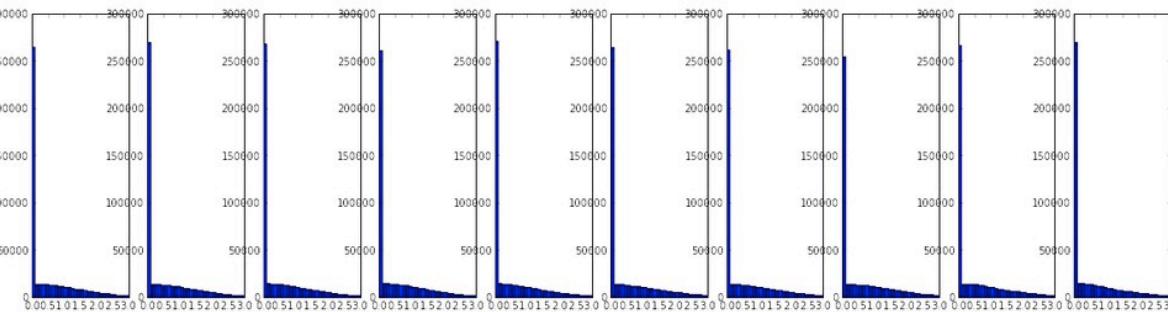
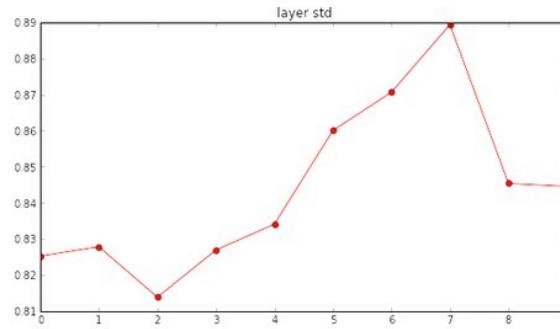
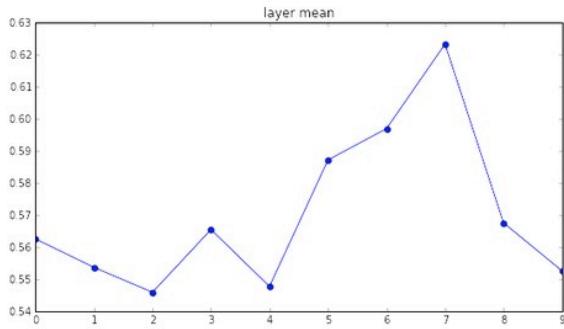
but when using the ReLU nonlinearity it breaks.



```
input layer had mean 0.000501 and std 0.999444  
hidden layer 1 had mean 0.562488 and std 0.825232  
hidden layer 2 had mean 0.553614 and std 0.827835  
hidden layer 3 had mean 0.545867 and std 0.813855  
hidden layer 4 had mean 0.565396 and std 0.826902  
hidden layer 5 had mean 0.547678 and std 0.834092  
hidden layer 6 had mean 0.587103 and std 0.860035  
hidden layer 7 had mean 0.596867 and std 0.870610  
hidden layer 8 had mean 0.623214 and std 0.889348  
hidden layer 9 had mean 0.567498 and std 0.845357  
hidden layer 10 had mean 0.552531 and std 0.844523
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

He et al., 2015  
(note additional /2)



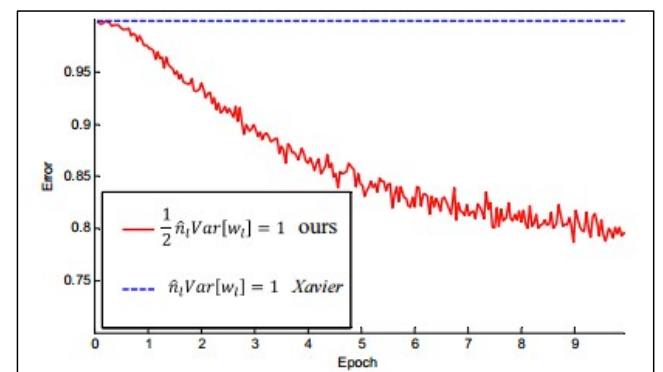
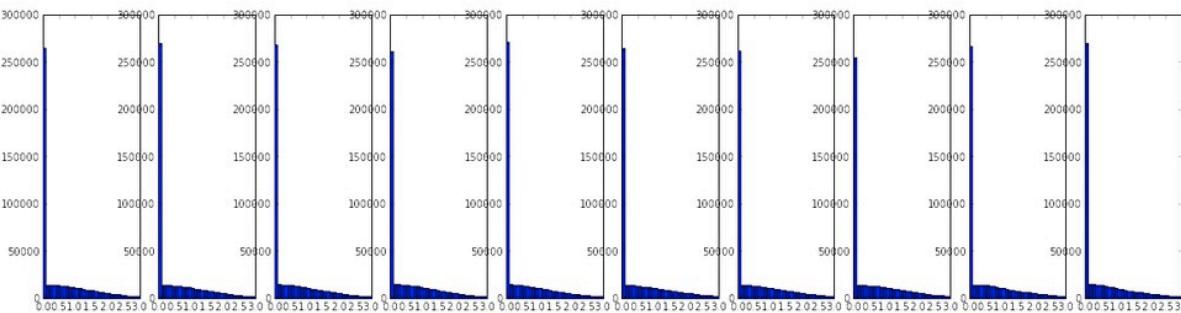
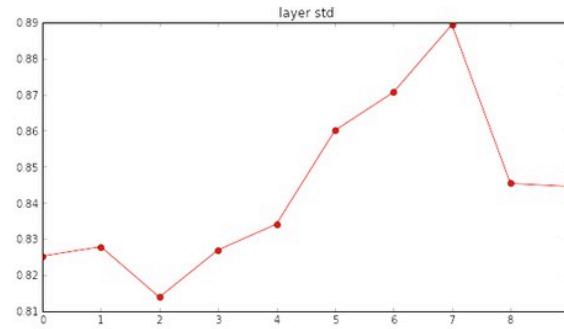
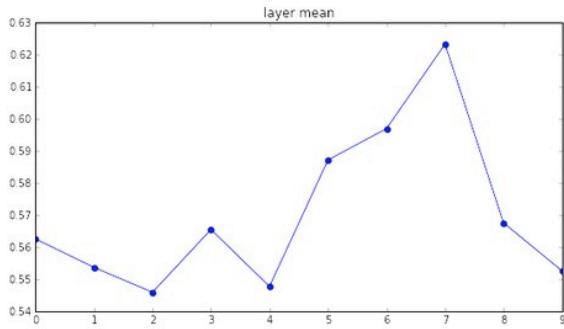
```

input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523

```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

He et al., 2015  
(note additional /2)



# Proper initialization is an active area of research...

***Understanding the difficulty of training deep feedforward neural networks***

by Glorot and Bengio, 2010

***Exact solutions to the nonlinear dynamics of learning in deep linear neural networks***

by Saxe et al, 2013

***Random walk initialization for training very deep feedforward networks***

by Sussillo and Abbott, 2014

***Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification***

by He et al., 2015

***Data-dependent Initializations of Convolutional Neural Networks***

by Krähenbühl et al., 2015

***All you need is a good init***, Mishkin and Matas, 2015

...

# Batch Normalization

[Ioffe and Szegedy, 2015]

“you want unit gaussian activations? just make them so.”

consider a batch of activations at some layer.  
To make each dimension unit gaussian, apply:

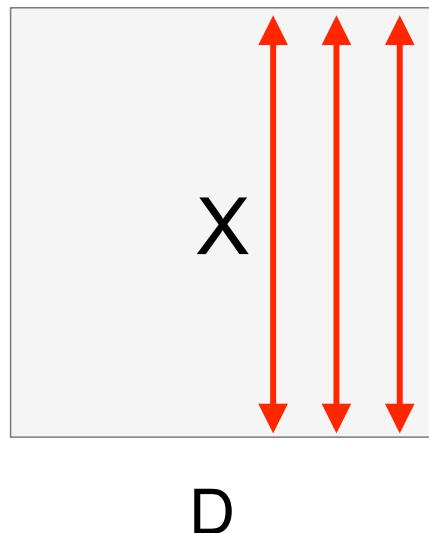
$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

this is a vanilla  
differentiable function...

# Batch Normalization

[Ioffe and Szegedy, 2015]

“you want unit gaussian activations?  
just make them so.”



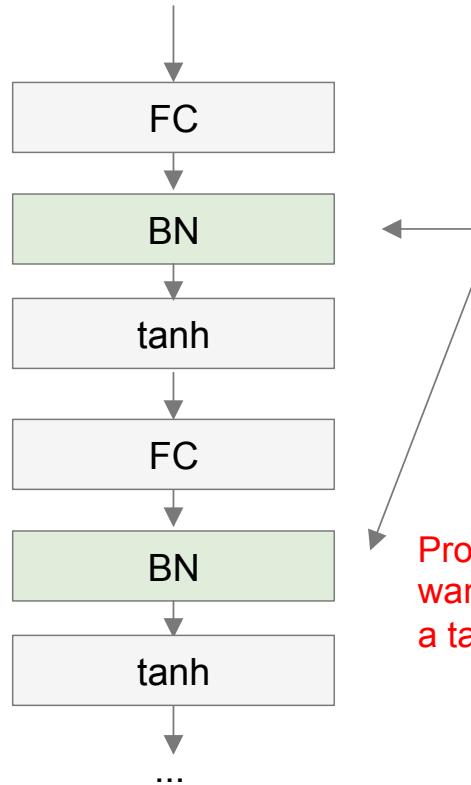
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

# Batch Normalization

[Ioffe and Szegedy, 2015]



Usually inserted after Fully Connected / (or Convolutional, as we'll see soon) layers, and before nonlinearity.

Problem: do we necessarily want a unit gaussian input to a tanh layer?

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

# Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \text{E}[x^{(k)}]$$

to recover the identity mapping.

# Batch Normalization

[Ioffe and Szegedy, 2015]

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

# Batch Normalization

[Ioffe and Szegedy, 2015]

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

**Note:** at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

# Remaining Pieces

---

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient  $\nabla g(w)$ 
  - Gradient checking
  - Backprop

# Gradient Descent

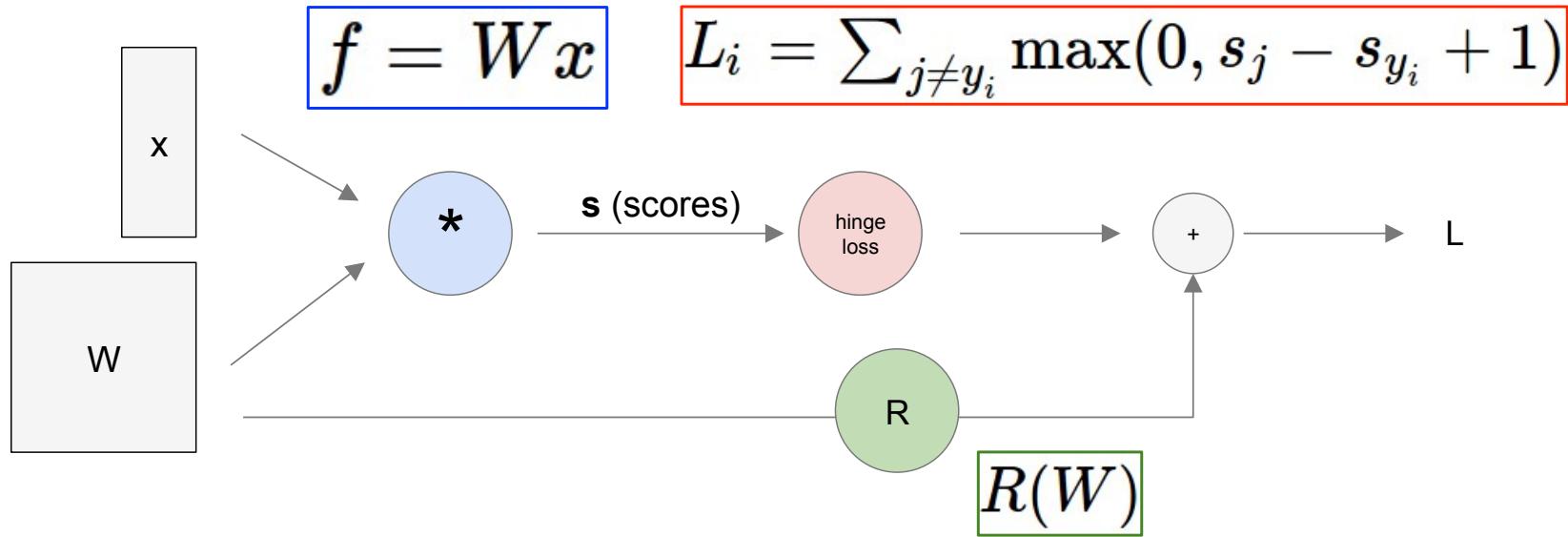
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

**Numerical gradient:** slow :, approximate :, easy to write :)

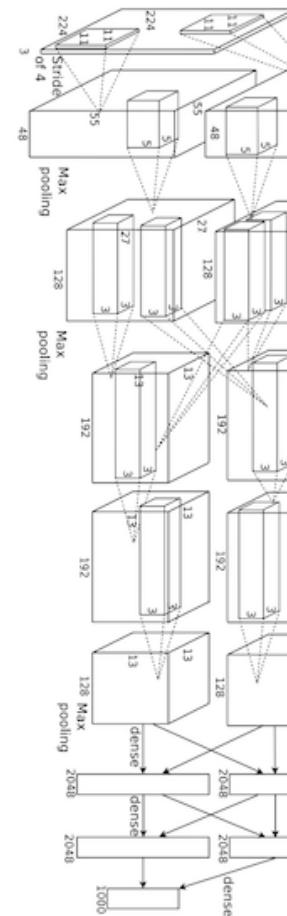
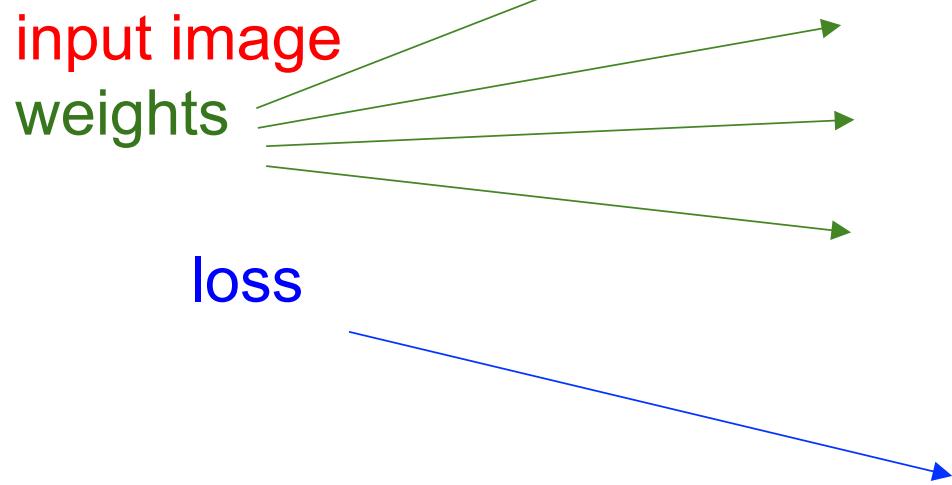
**Analytic gradient:** fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient

# Computational Graph



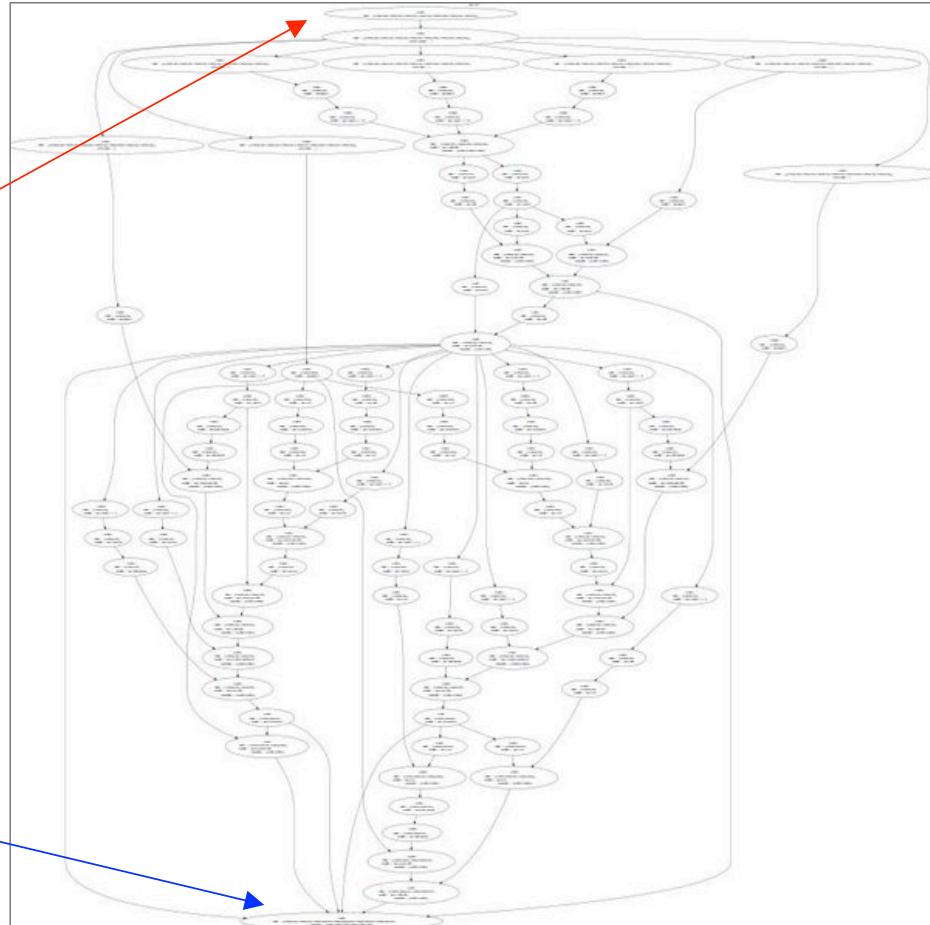
# Convolutional Network (AlexNet)



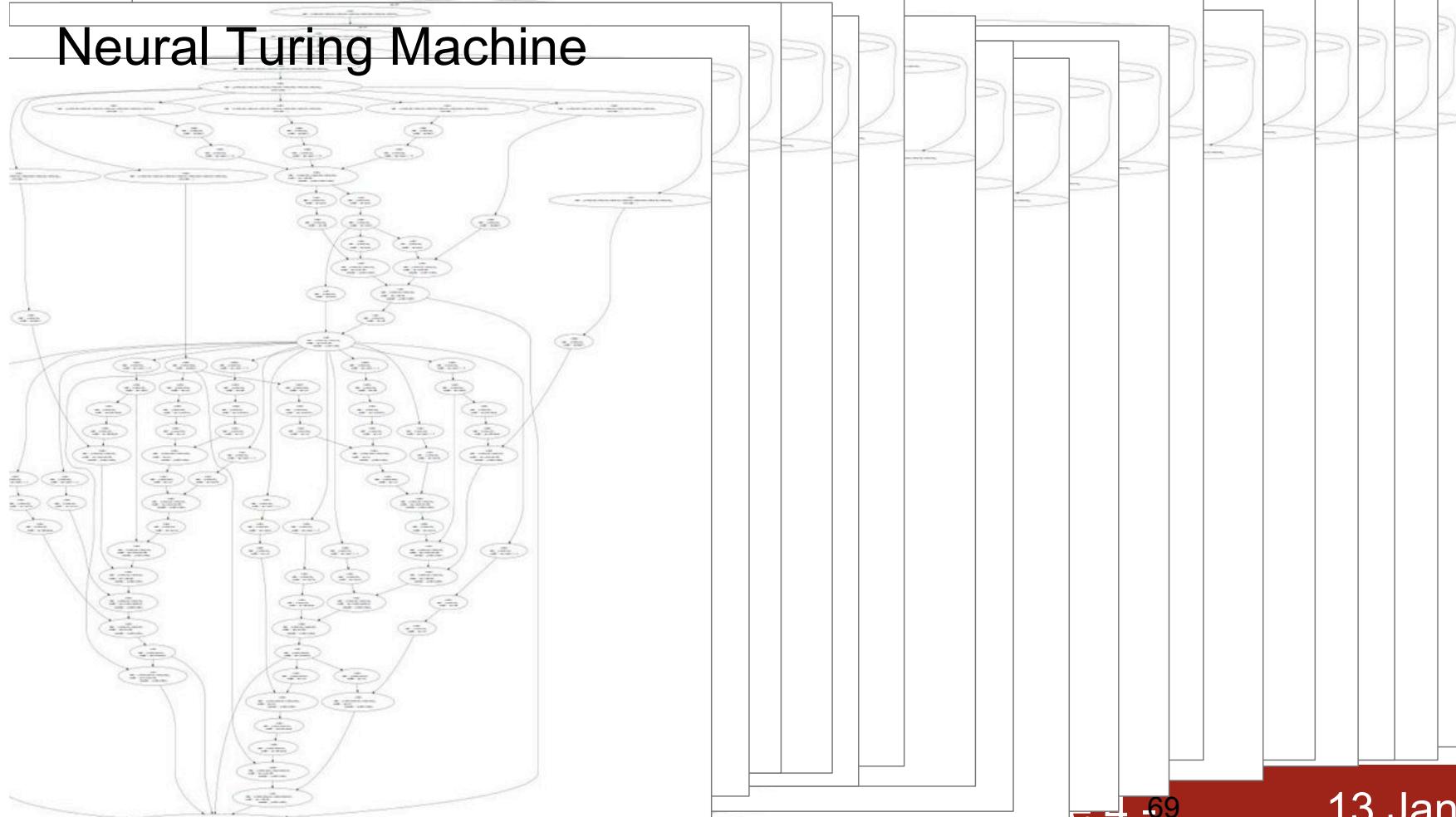
# Neural Turing Machine

input tape

loss

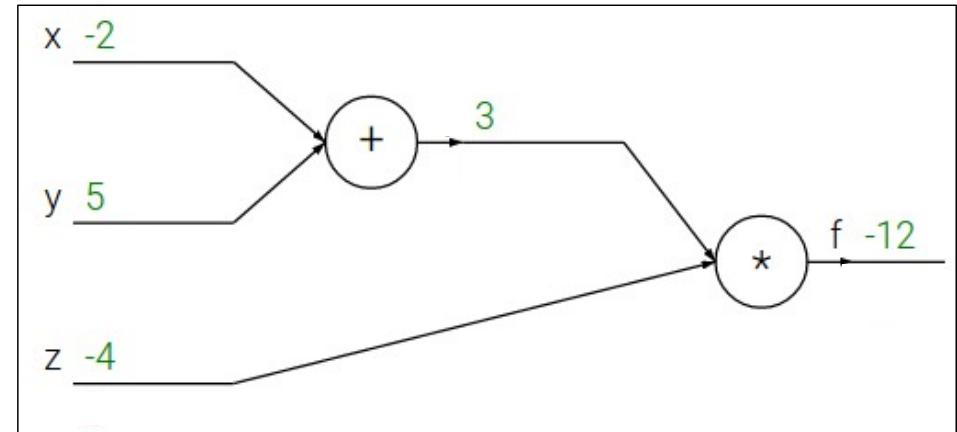


# Neural Turing Machine



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

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



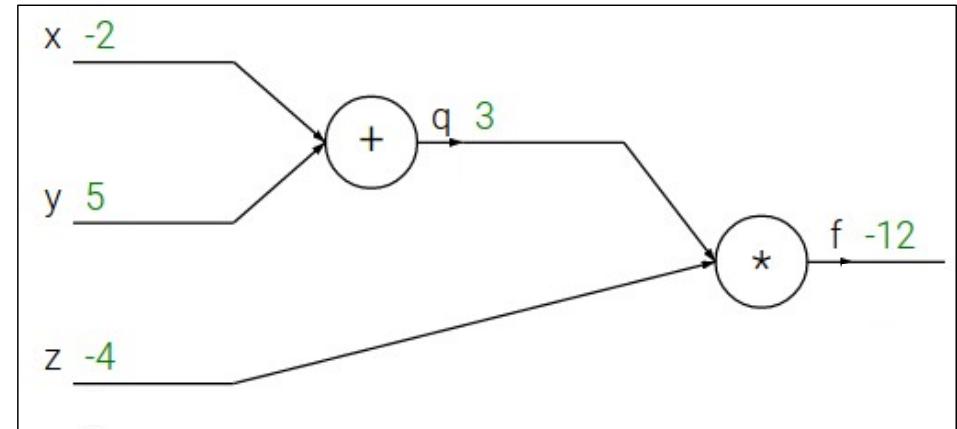
$$f(x, y, z) = (x + y)z$$

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

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

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

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

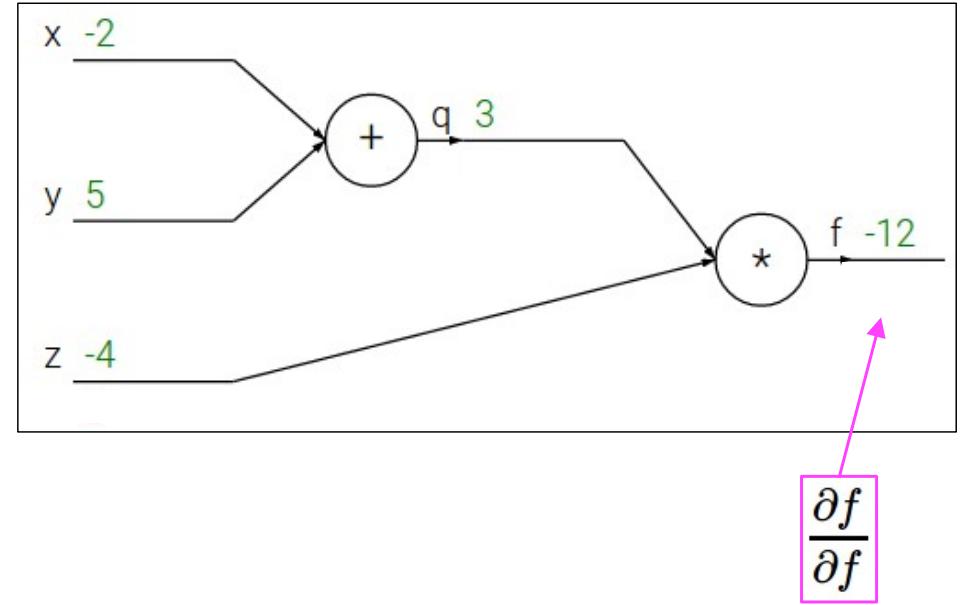


$$f(x, y, z) = (x + y)z$$

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

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

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



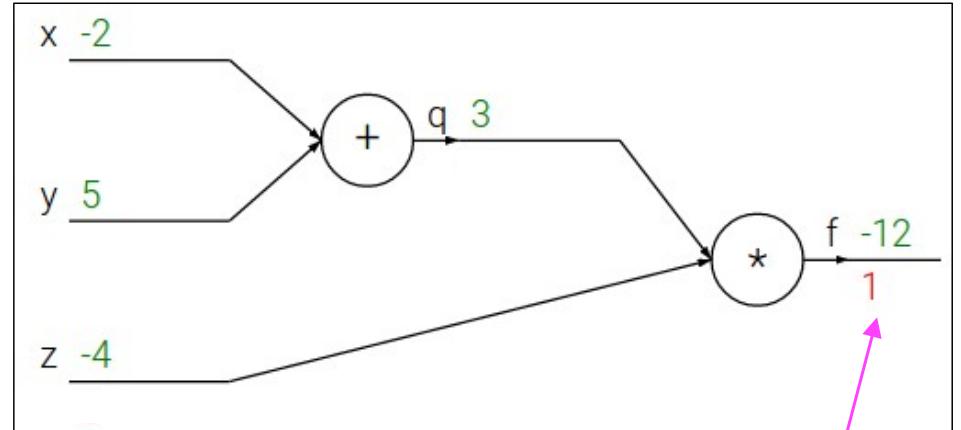
Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

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

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

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



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

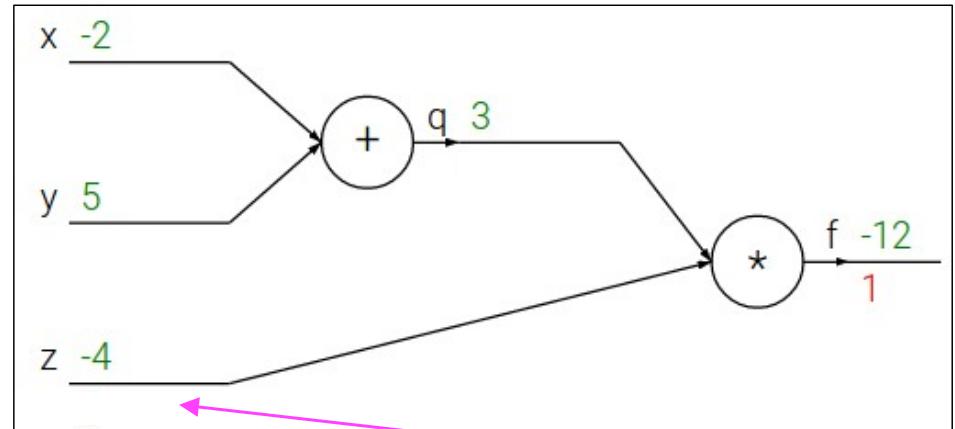
$$f(x, y, z) = (x + y)z$$

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

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

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

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



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

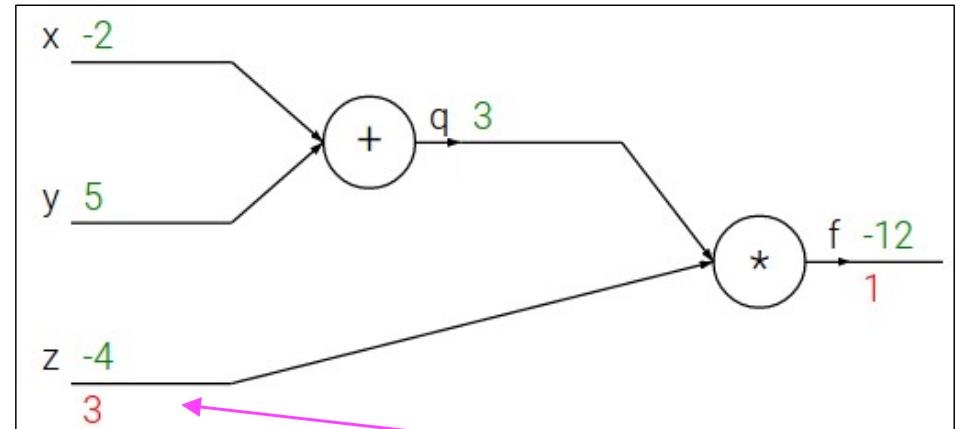
$$f(x, y, z) = (x + y)z$$

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

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

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

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



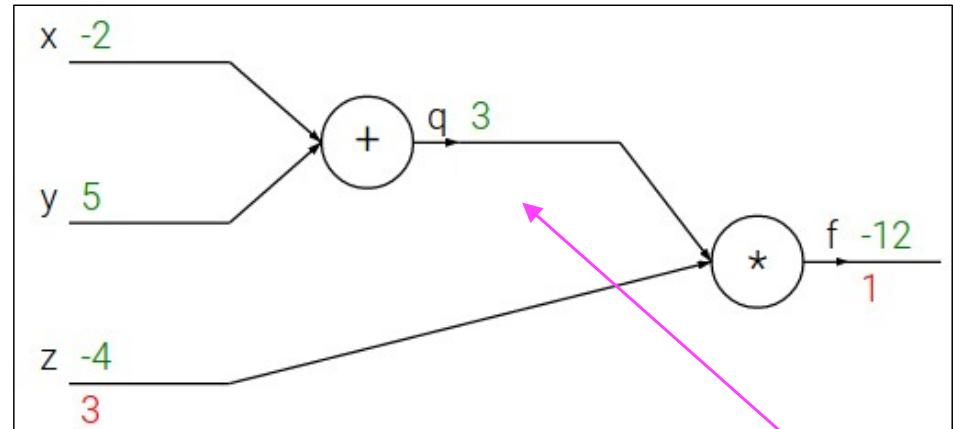
$$\frac{\partial f}{\partial z}$$

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

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

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

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



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

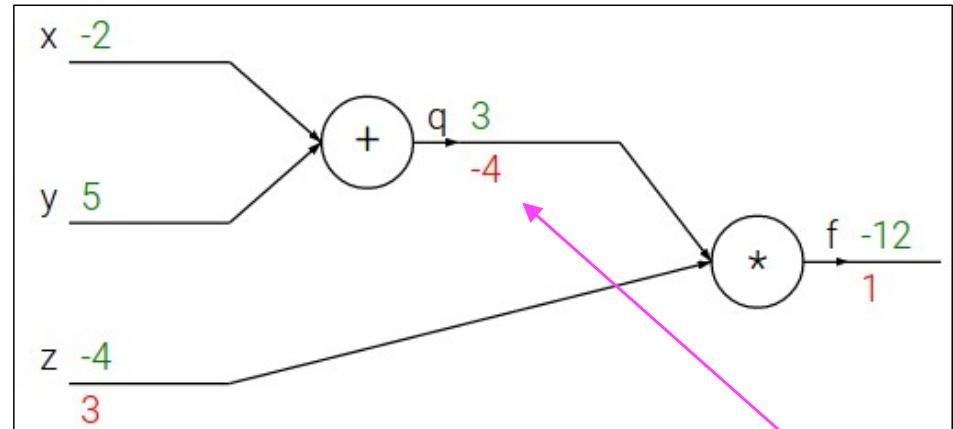
Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

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

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

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



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

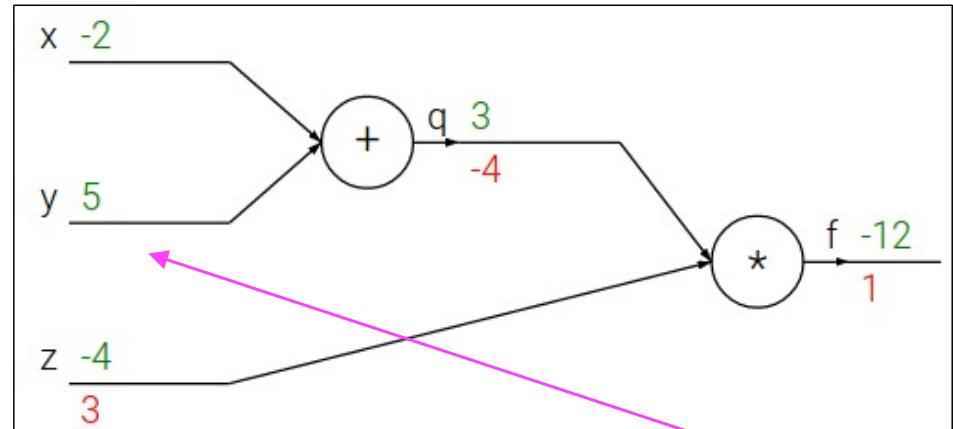
$$f(x, y, z) = (x + y)z$$

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

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

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

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



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

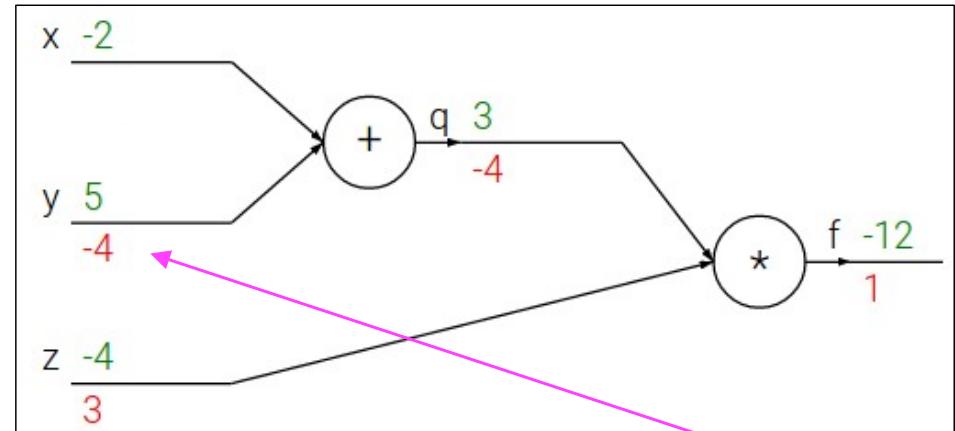
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Chain rule:

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

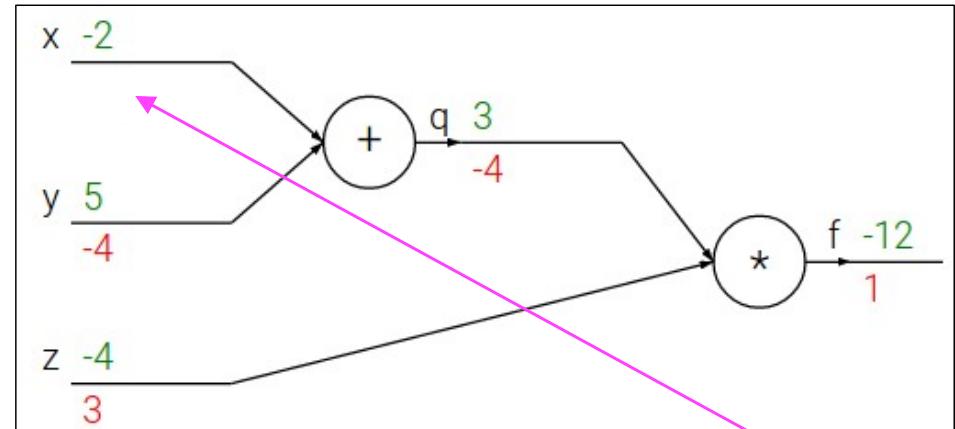
$$\frac{\partial f}{\partial y}$$

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

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

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

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



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

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

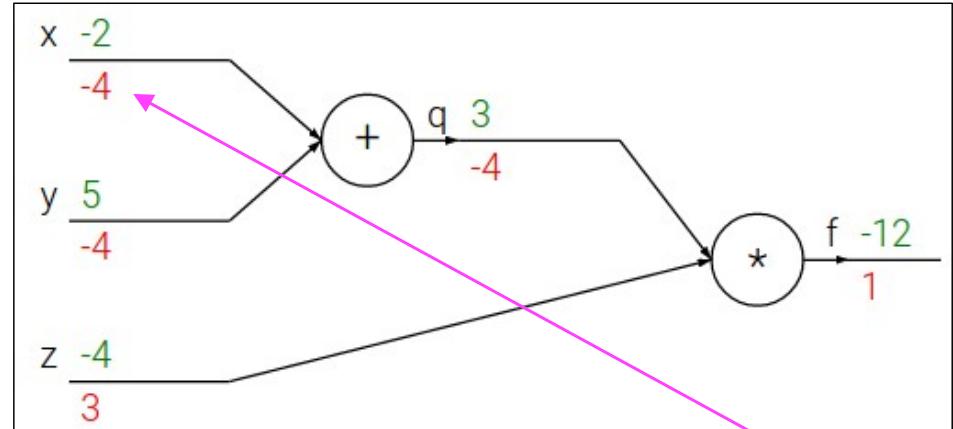
$$f(x, y, z) = (x + y)z$$

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

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

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

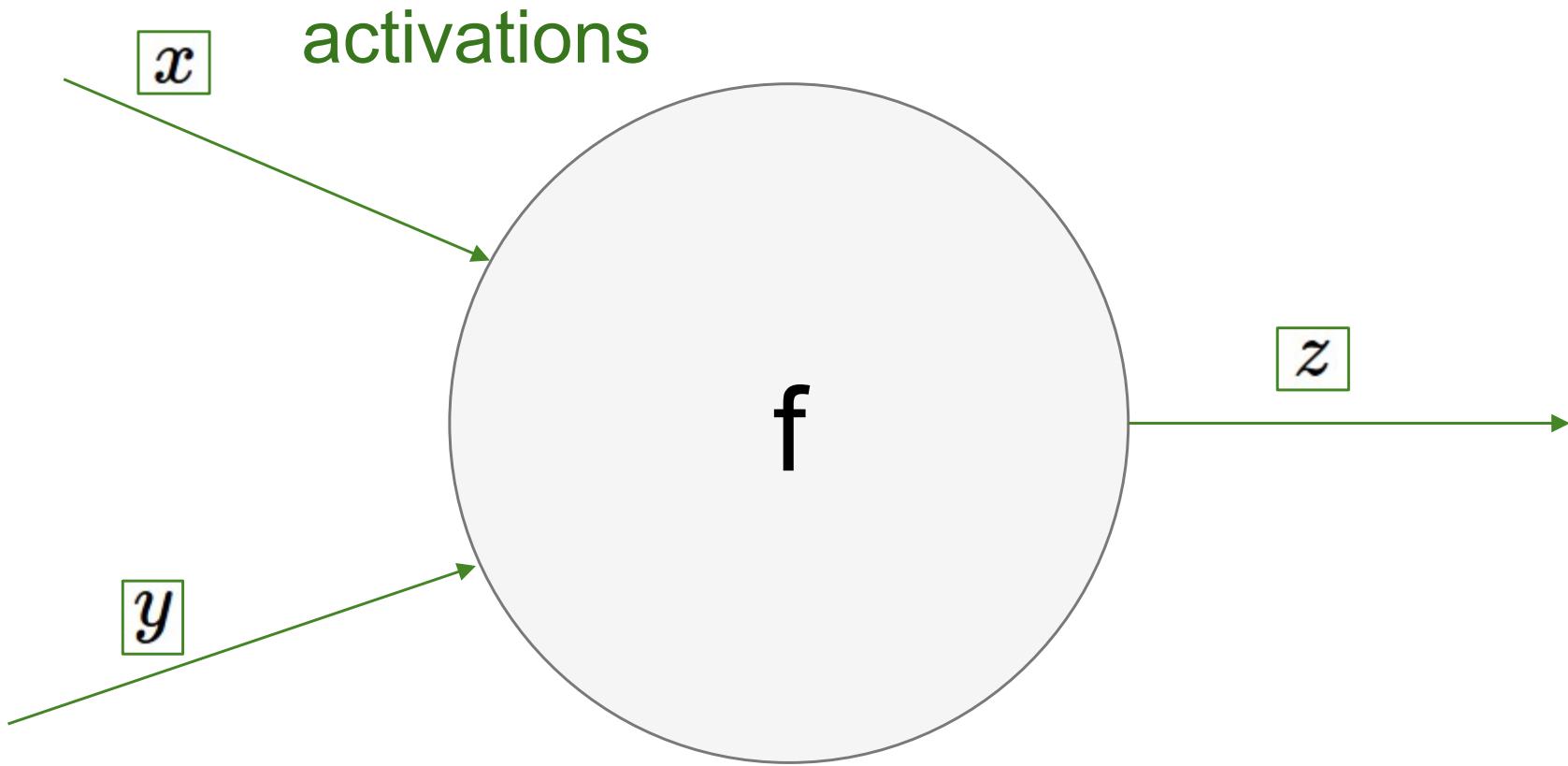
Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

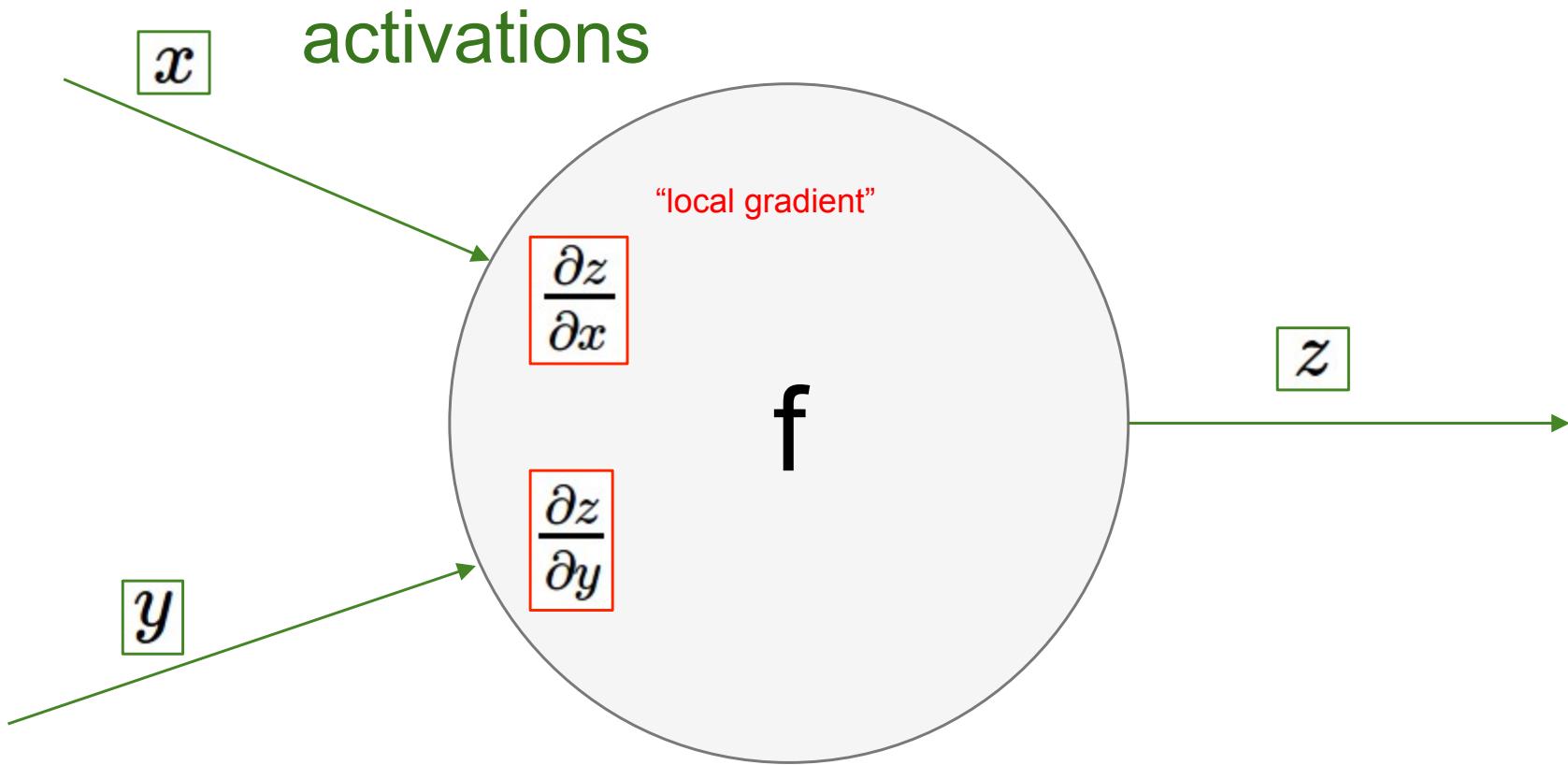


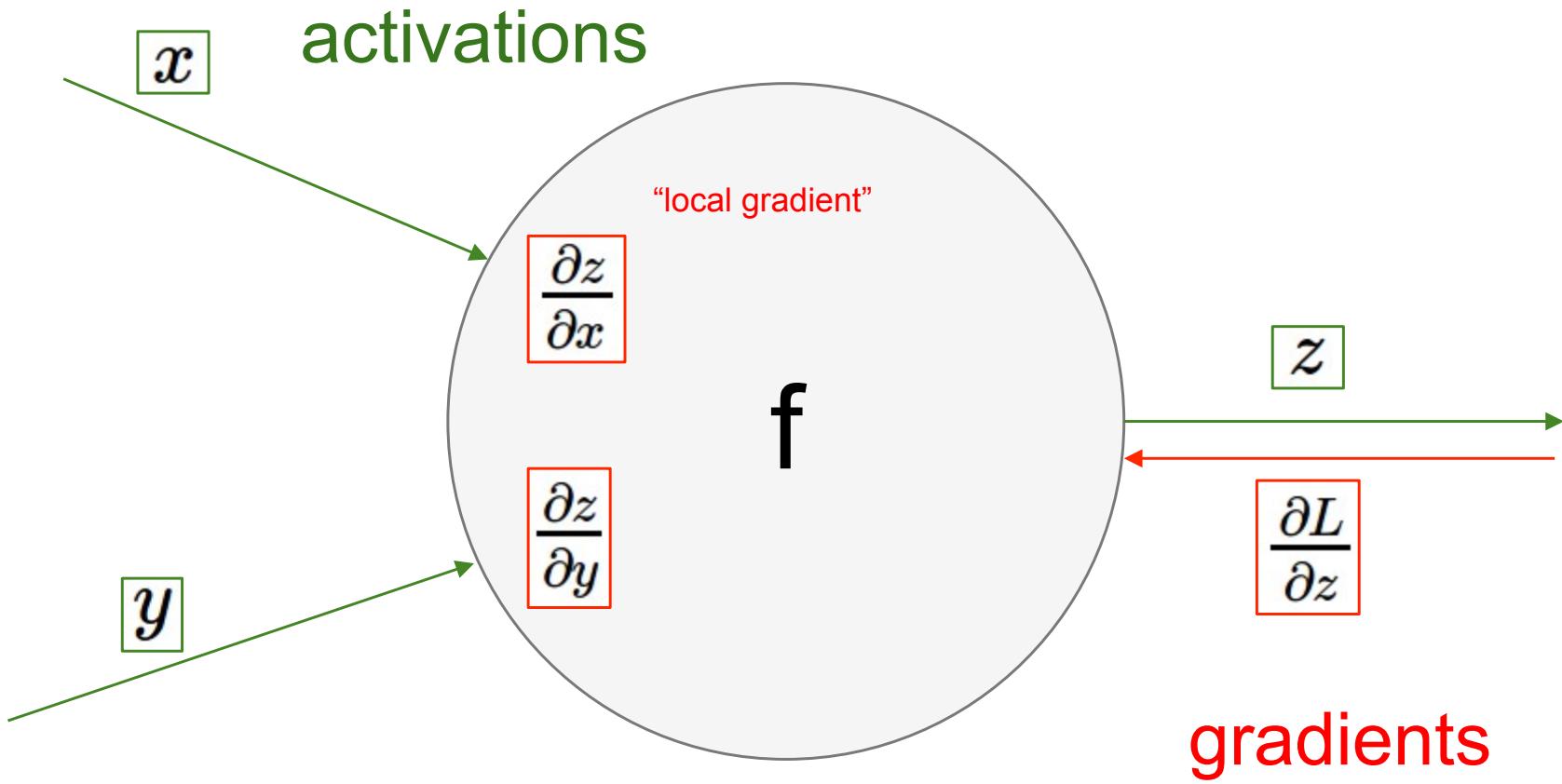
Chain rule:

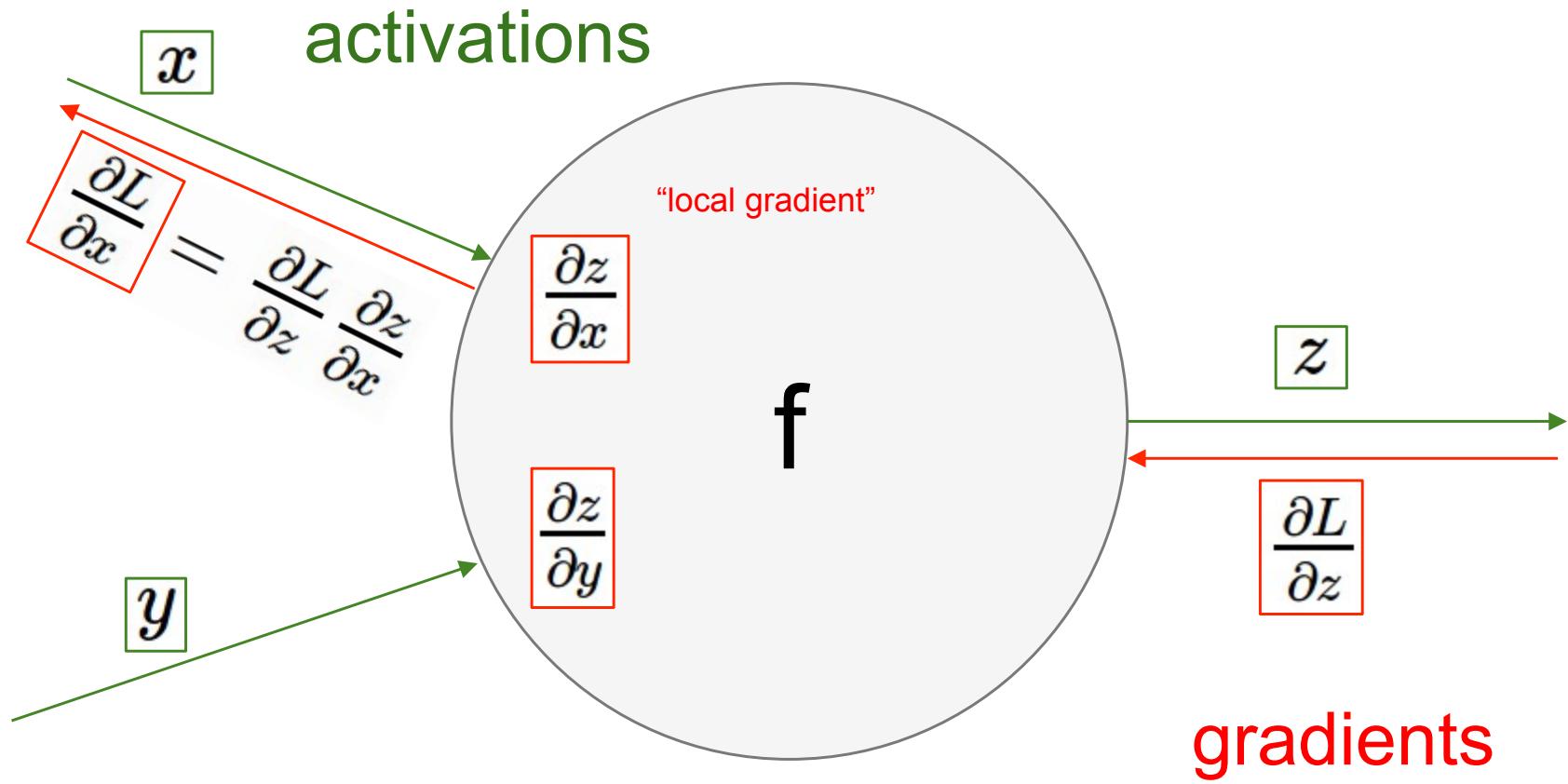
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

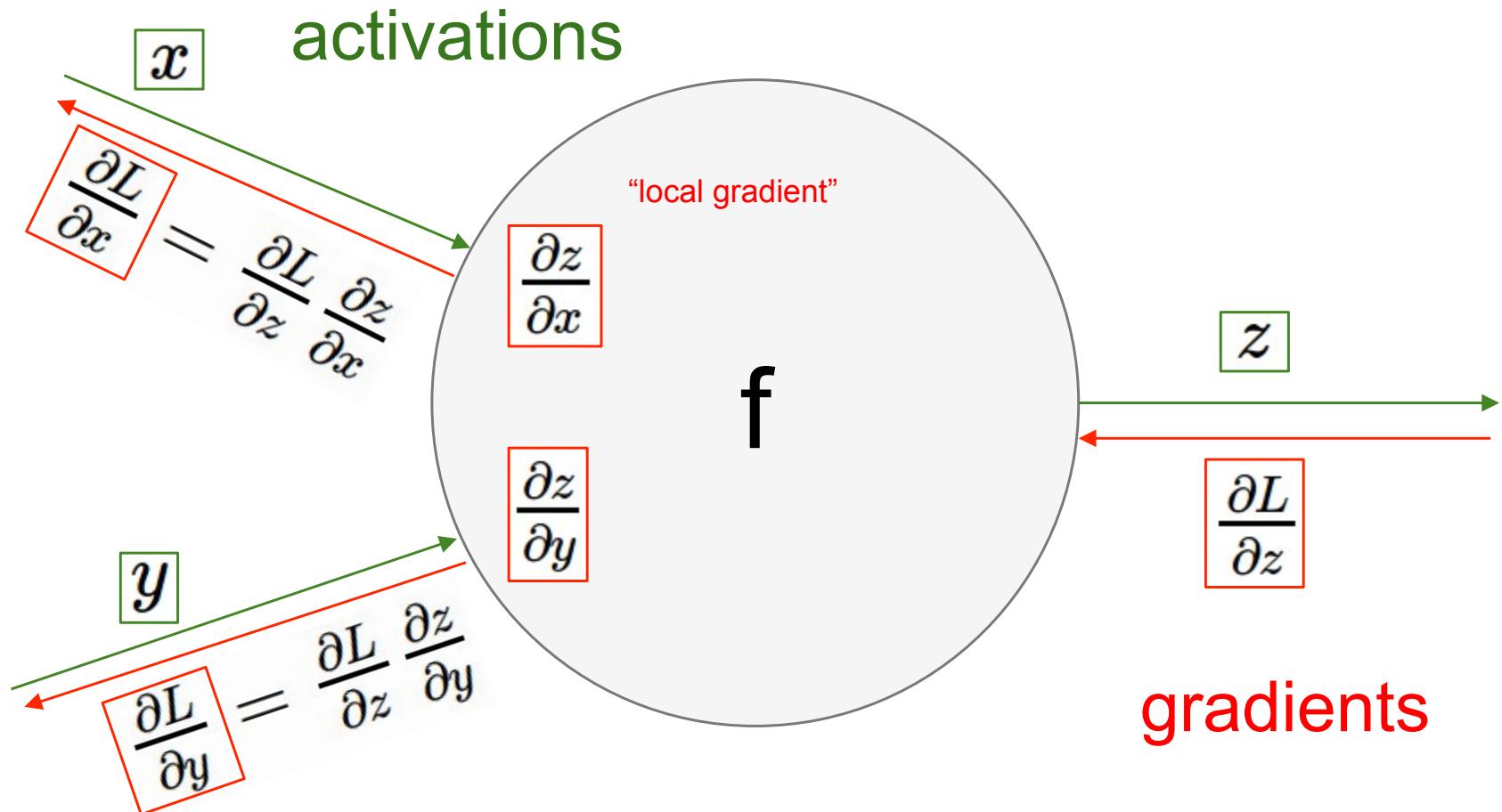
$$\frac{\partial f}{\partial x}$$

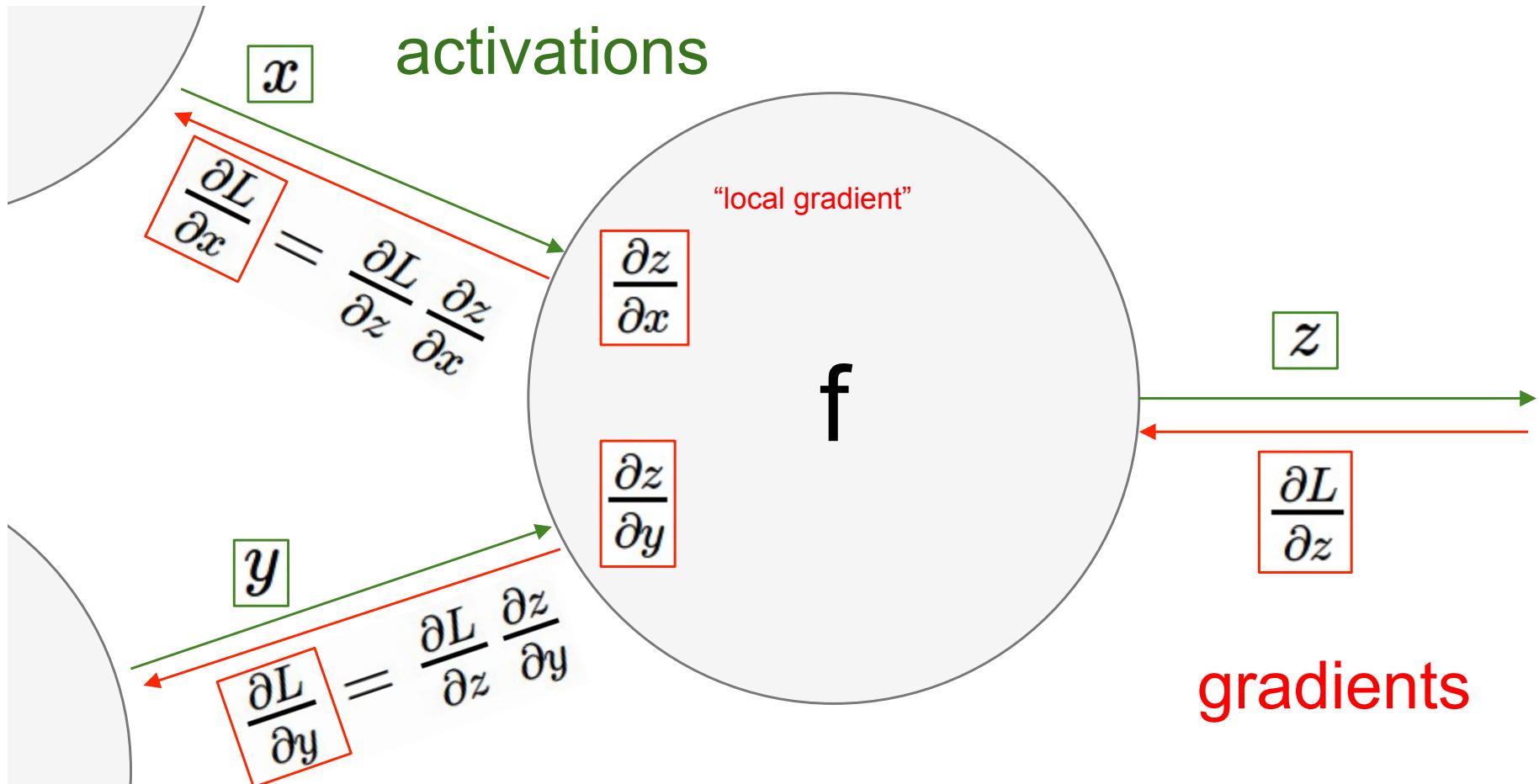






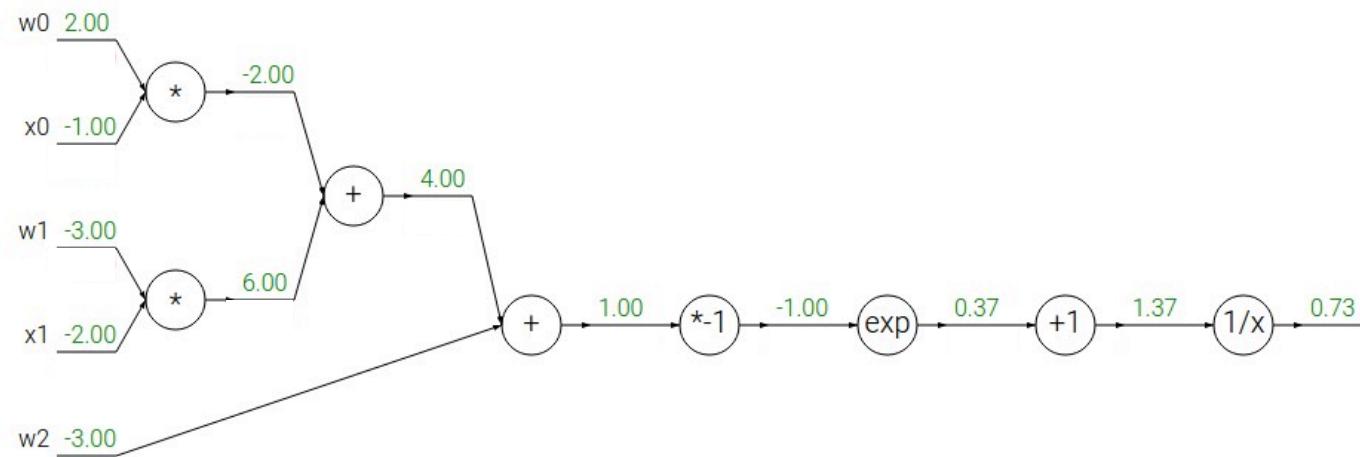




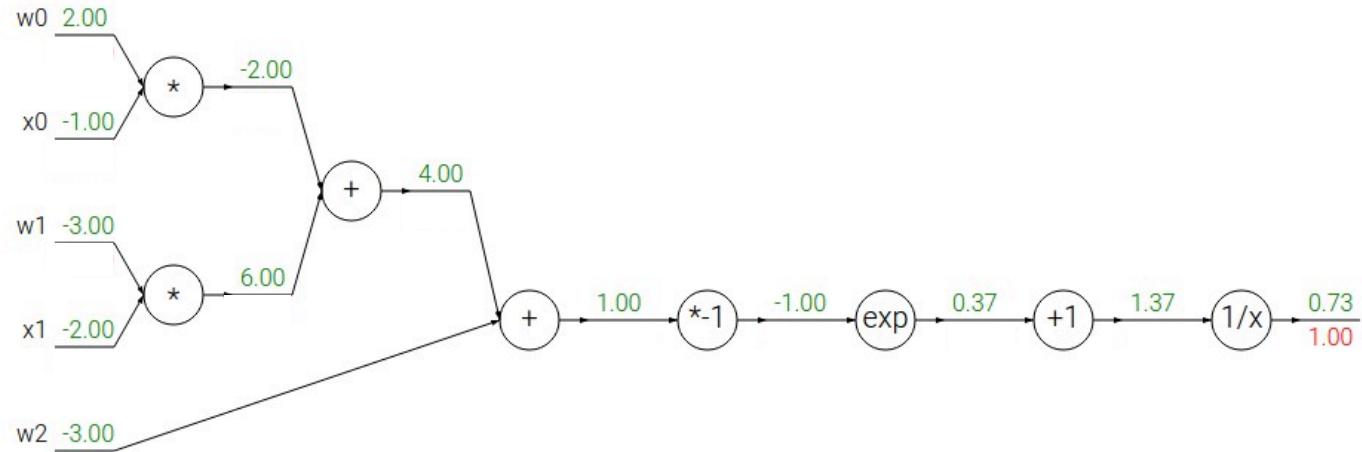


Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

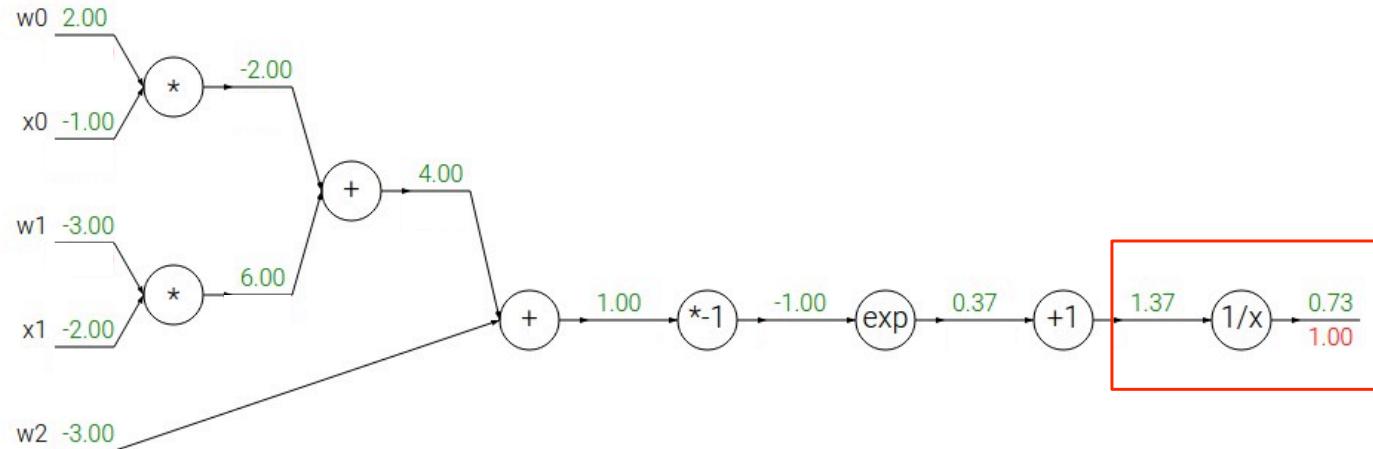
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

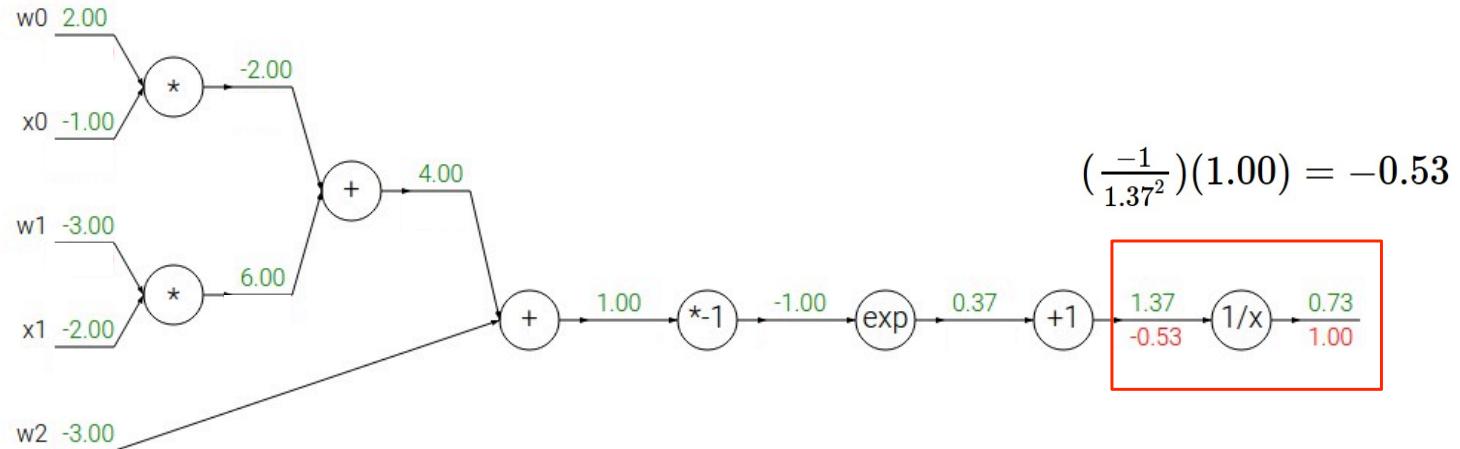
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

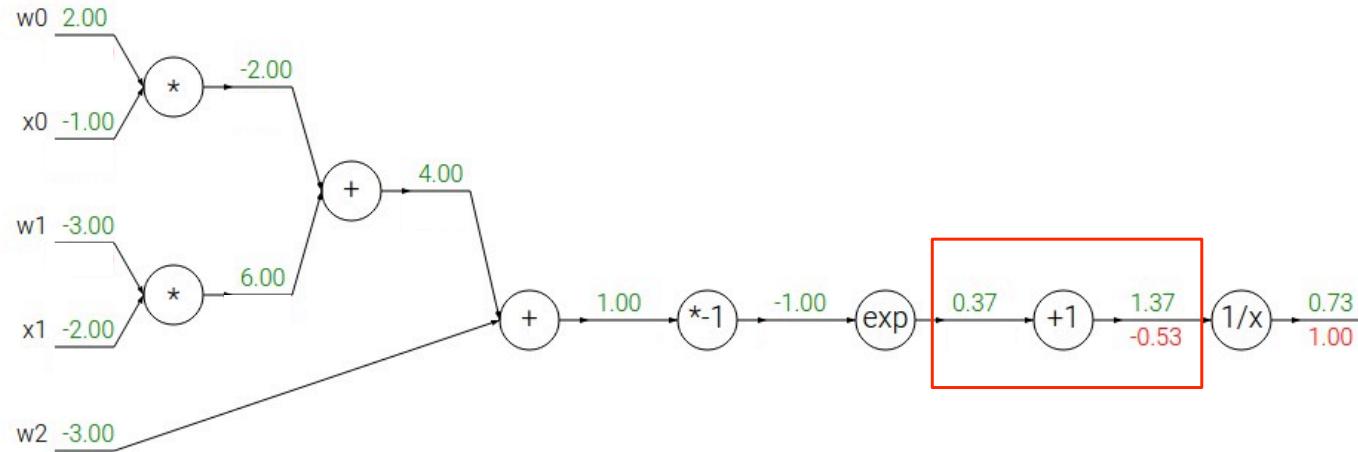
$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

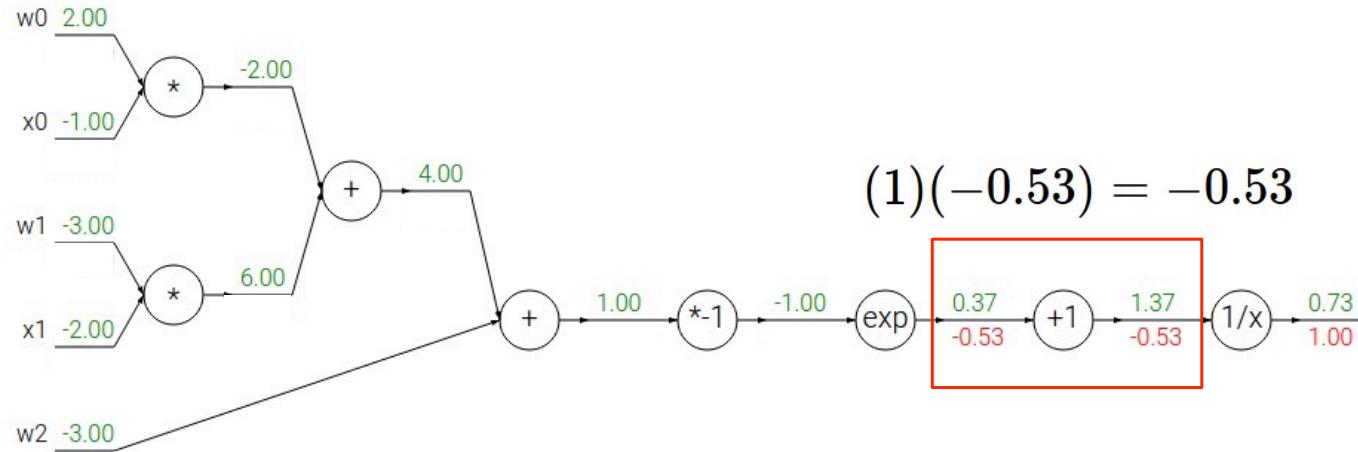
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

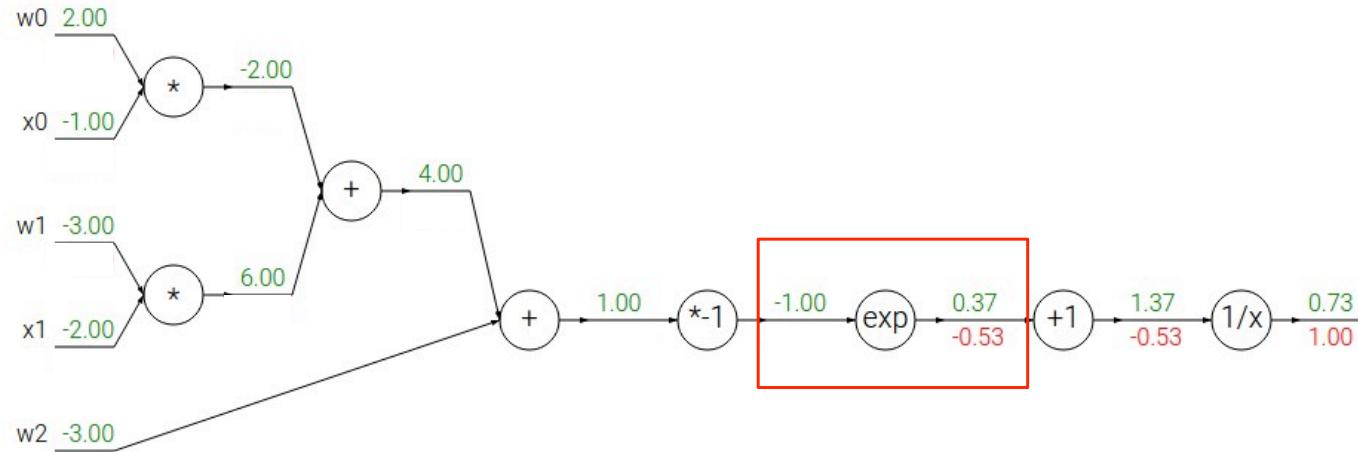
Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$(1)(-0.53) = -0.53$$

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$	$ $	$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$		$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



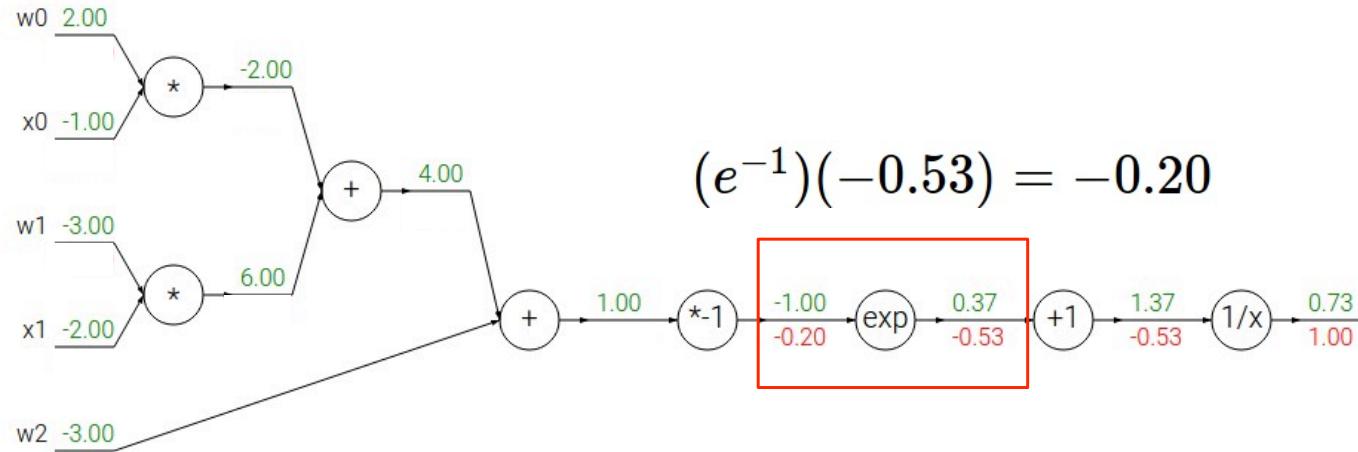
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



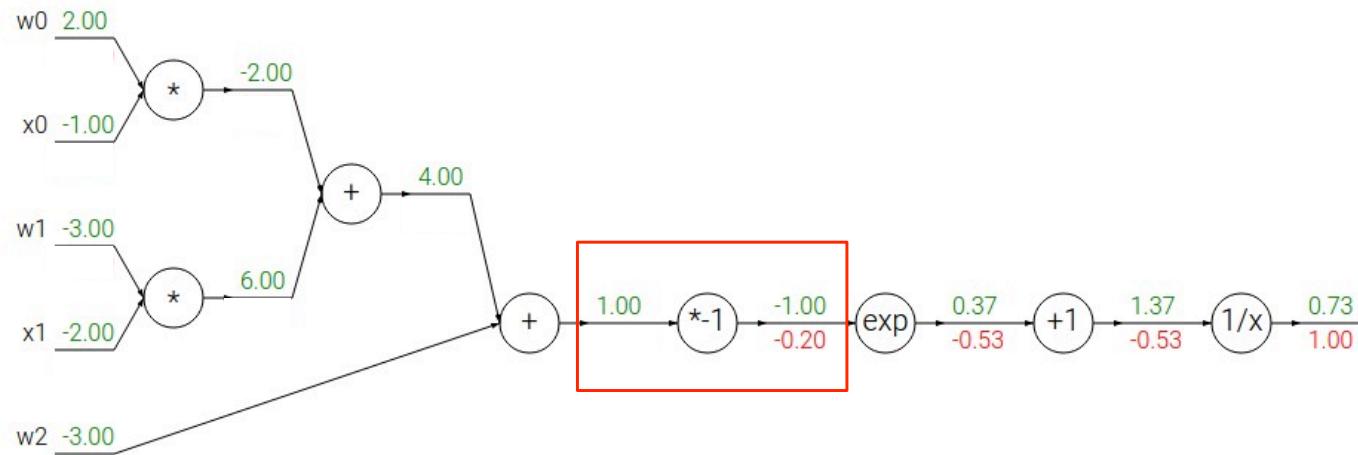
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

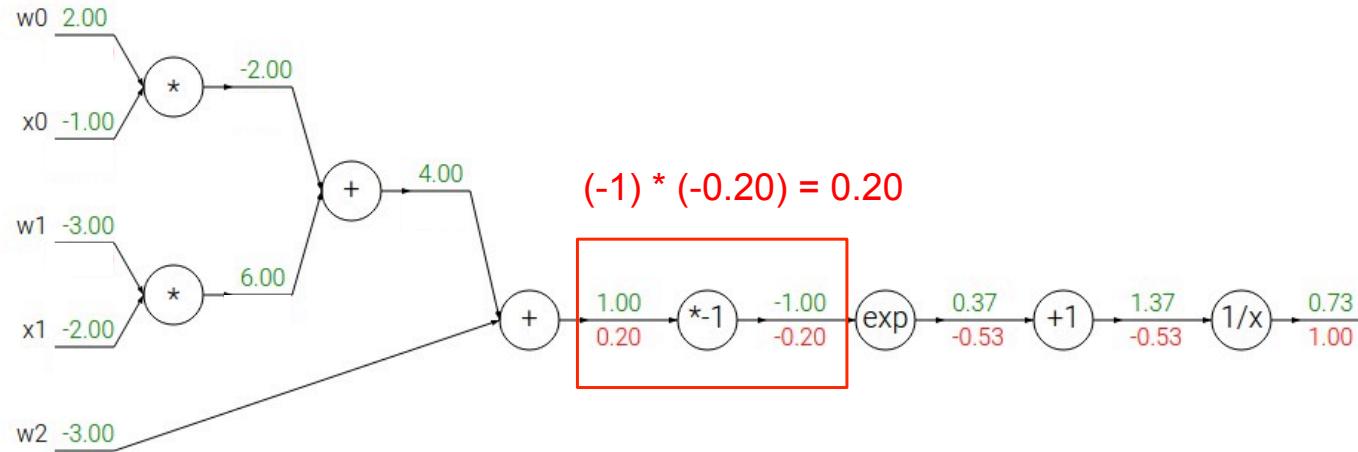
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

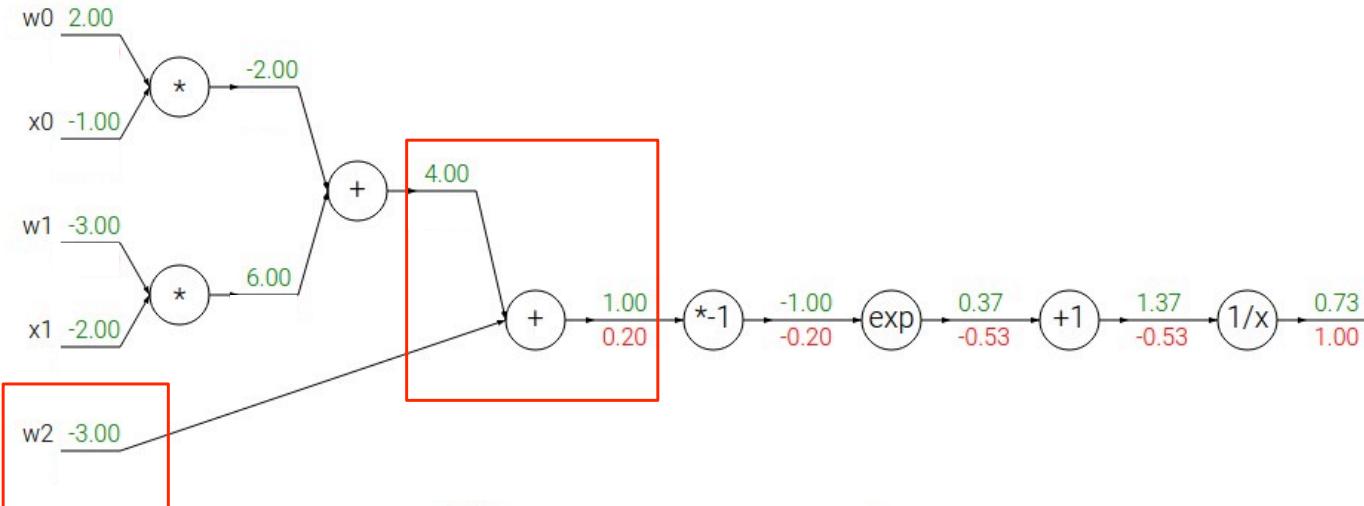
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

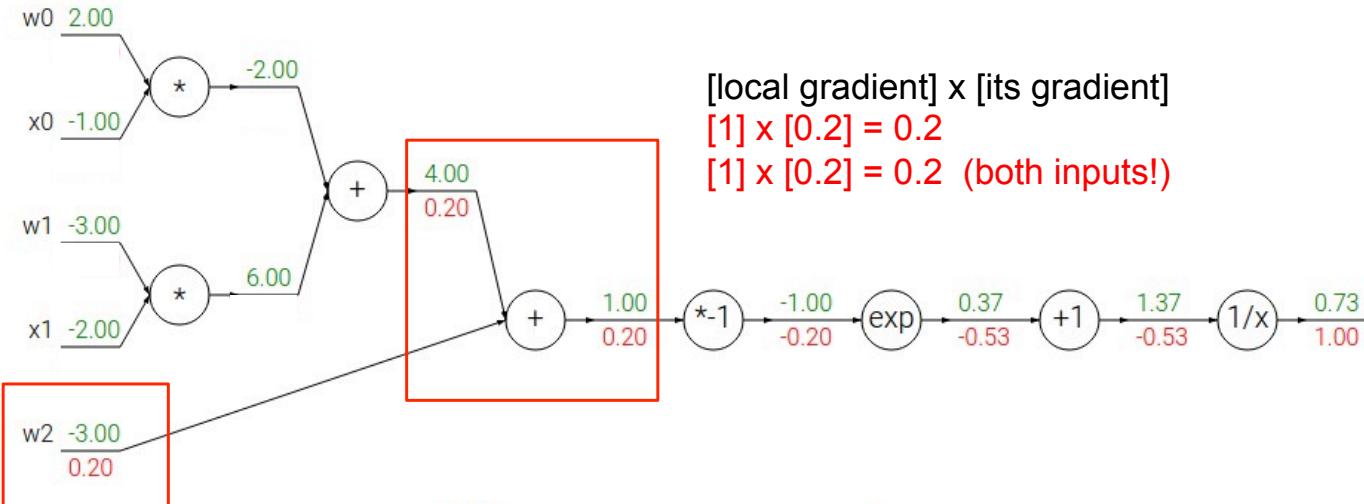
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:  $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

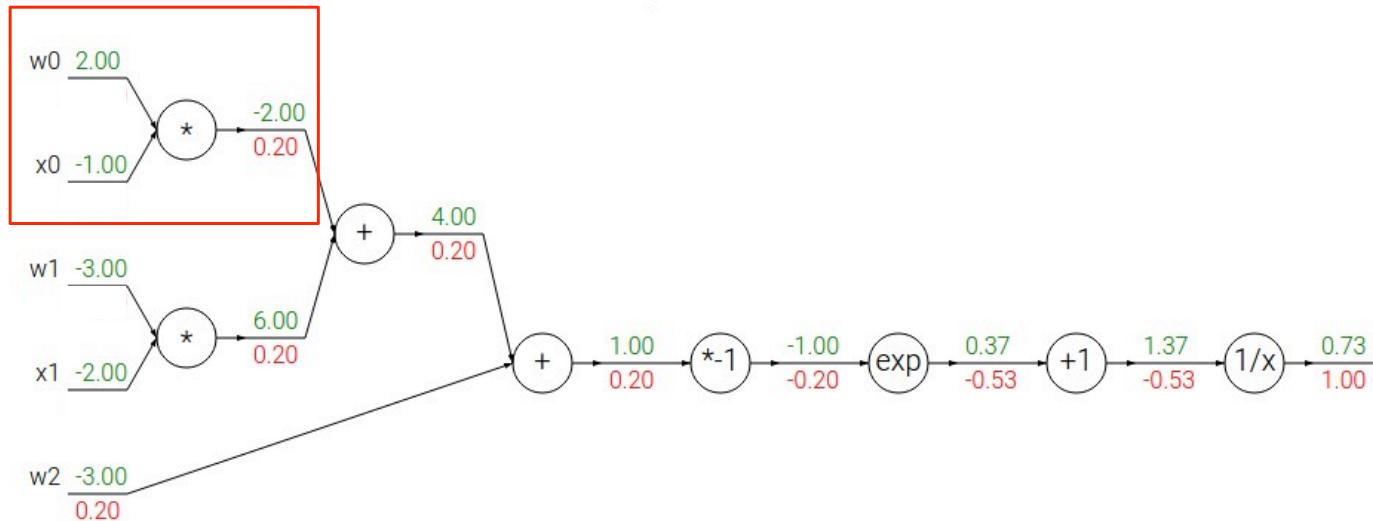
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

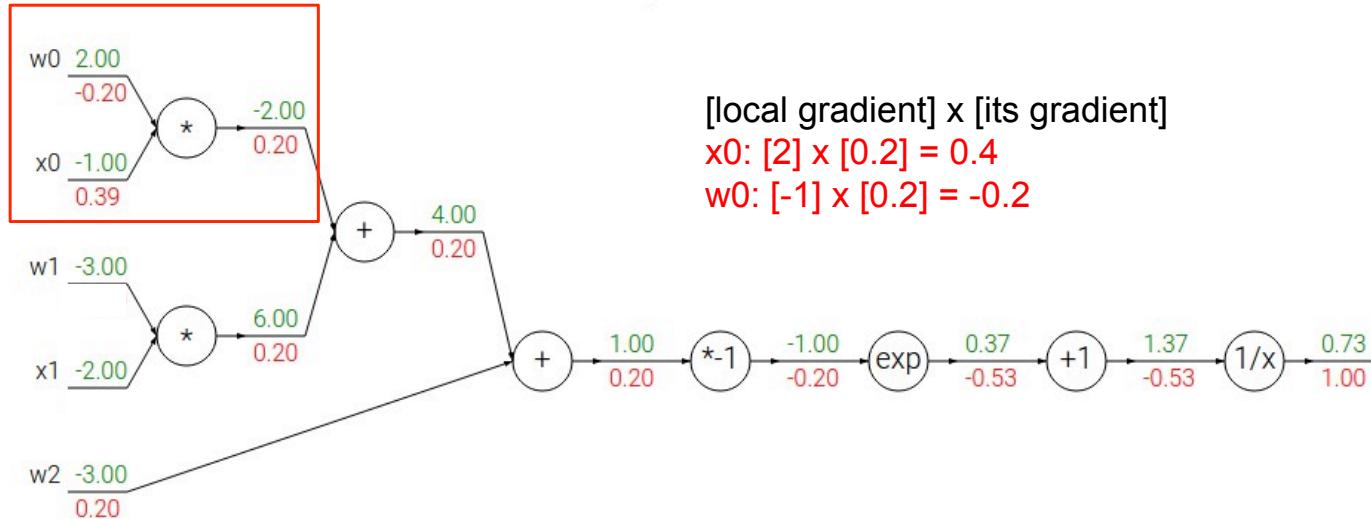
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

→

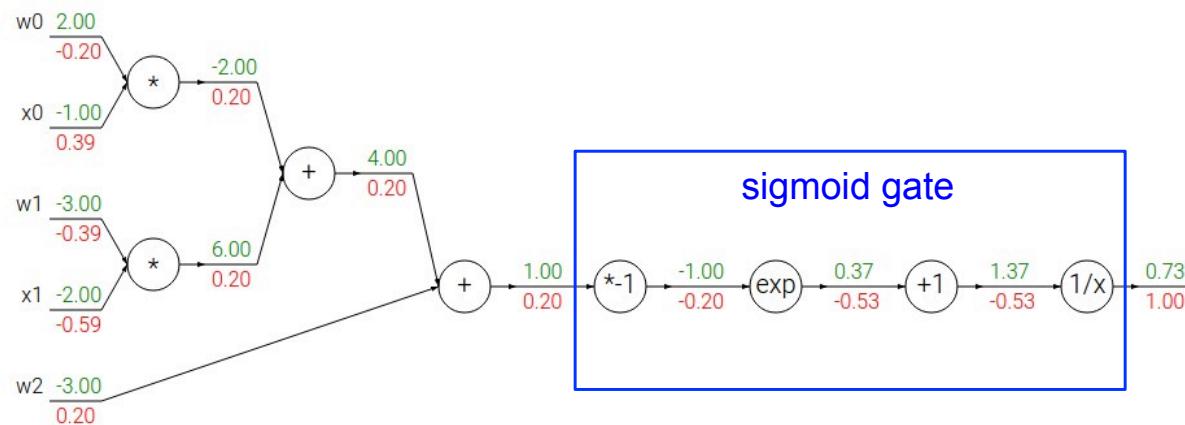
$$\frac{df}{dx} = 1$$

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

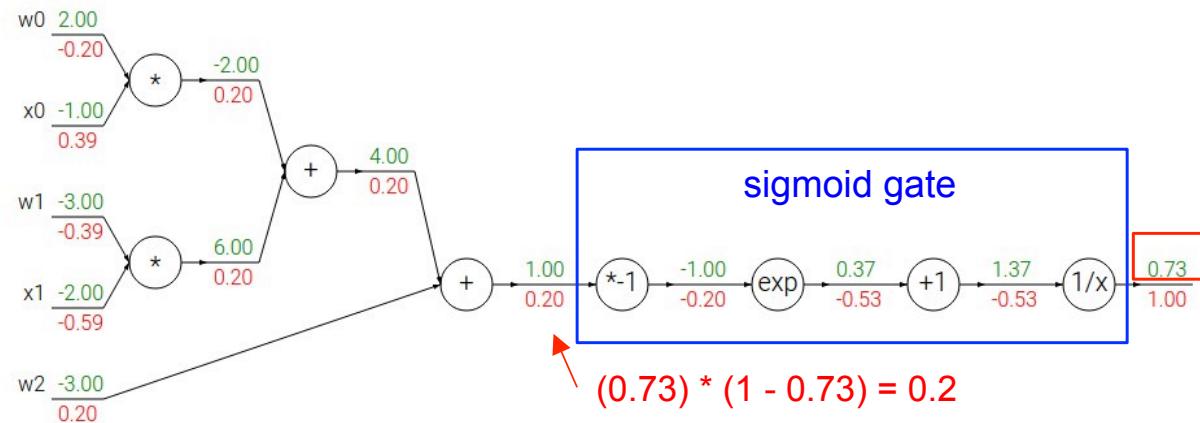


$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

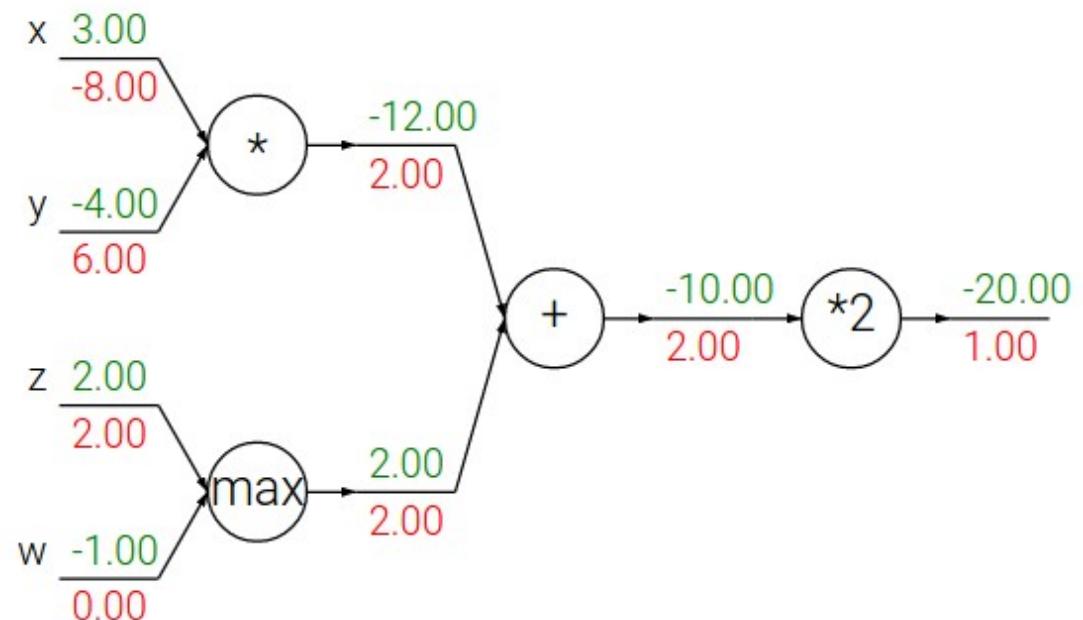


## Patterns in backward flow

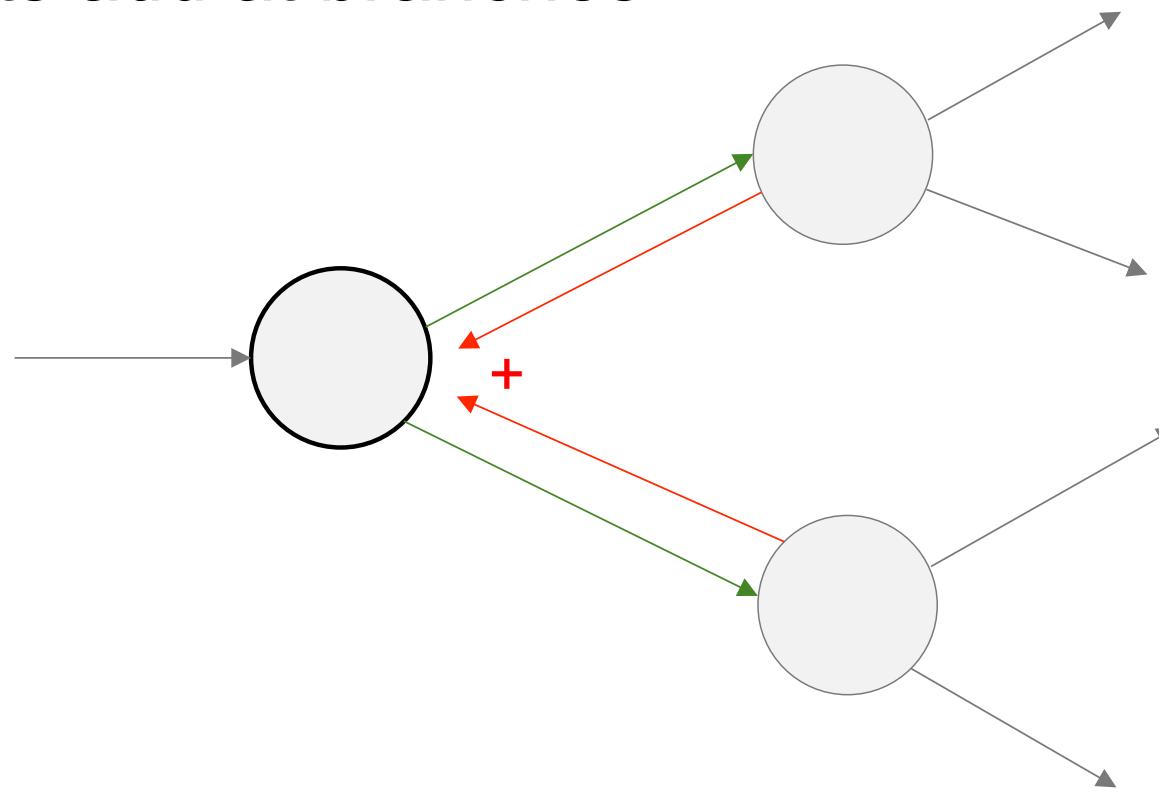
**add** gate: gradient distributor

**max** gate: gradient router

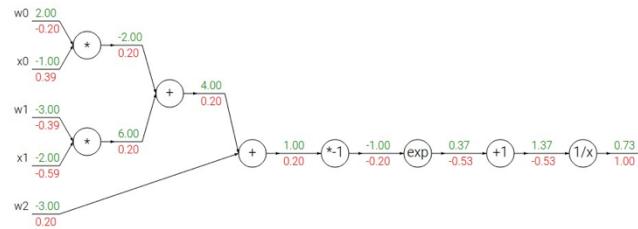
**mul** gate: gradient...?



# Gradients add at branches



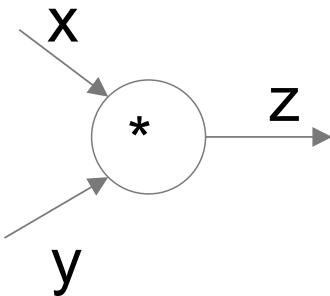
# Implementation: forward/backward API



Graph (or Net) object. (*Rough psuedo code*)

```
class ComputationalGraph(object):
    ...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

# Implementation: forward/backward API



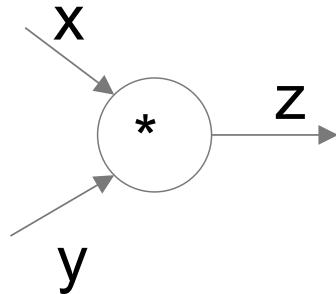
(x,y,z are scalars)

```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        return z  
    def backward(dz):  
        # dx = ... #todo  
        # dy = ... #todo  
        return [dx, dy]
```

$$\frac{\partial L}{\partial x}$$

$$\frac{\partial L}{\partial z}$$

# Implementation: forward/backward API



```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        self.x = x # must keep these around!  
        self.y = y  
        return z  
    def backward(dz):  
        dx = self.y * dz # [dz/dx * dL/dz]  
        dy = self.x * dz # [dz/dy * dL/dz]  
        return [dx, dy]
```

( $x, y, z$  are scalars)



# Deep Learning Frameworks

---

TensorFlow (in your Project 6!)

Theano

Torch

CAFFE

Computation Graph Toolkit (CGT)

# That's it for ML

---

Up next: Advanced Applications!