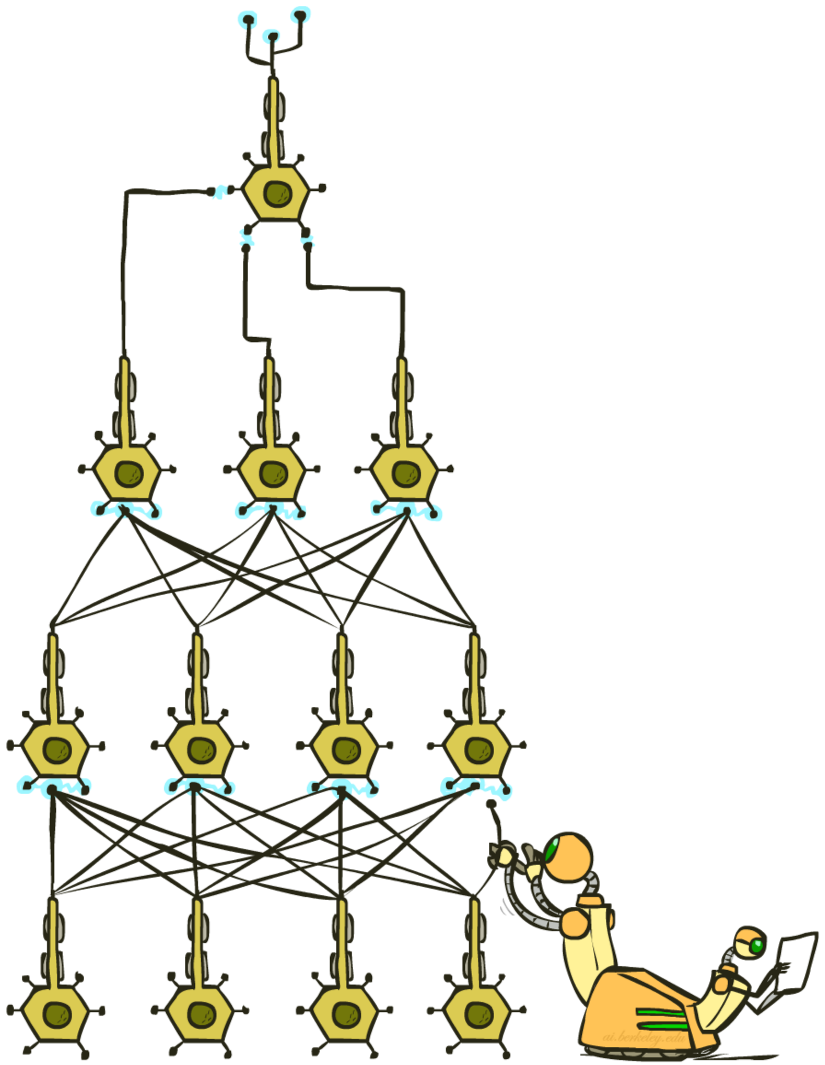


CS 343: Artificial Intelligence

Deep Learning

Profs. Peter Stone and Yuke Zhu — The University of Texas at Austin

[These slides based on those of Dan Klein, Pieter Abbeel, Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Good morning colleagues!

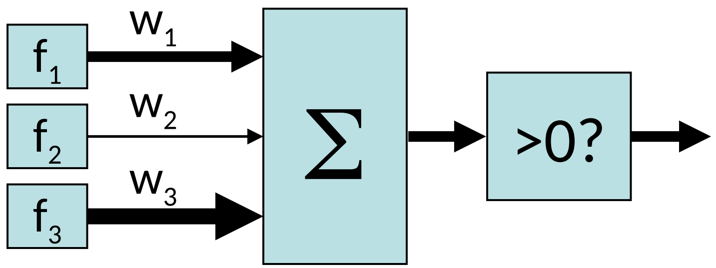
- Past due:
  - HW1-7: Search, CSPs, Games, MDP, RL, Bayes Nets, Particle Filters/VPI
  - 11 reading responses: AI100 report; 10 Textbook readings
  - P0,1,2,3: tutorial, Search, Multiagent RL
  - Midterm
- Upcoming EdX Homeworks
  - HW8: Naive Bayes and Perceptrons – due Monday 4/19 at 11:59 pm
  - HW9: Neural Networks – due Monday 4/26 at 11:59 pm
- Upcoming programming projects
  - P4: Bayes Nets – due Wednesday 4/14 at 11:59pm
  - P5: Particle Filters – due Wednesday 4/21 at 11:59pm
  - Contest (Capture the flag): Qualification due 4/28; Finals 5/3 (extra credit)
- Readings: SVMs, Kernels, and Clustering – Due Monday 4/19 at 9:30am

Good morning colleagues!

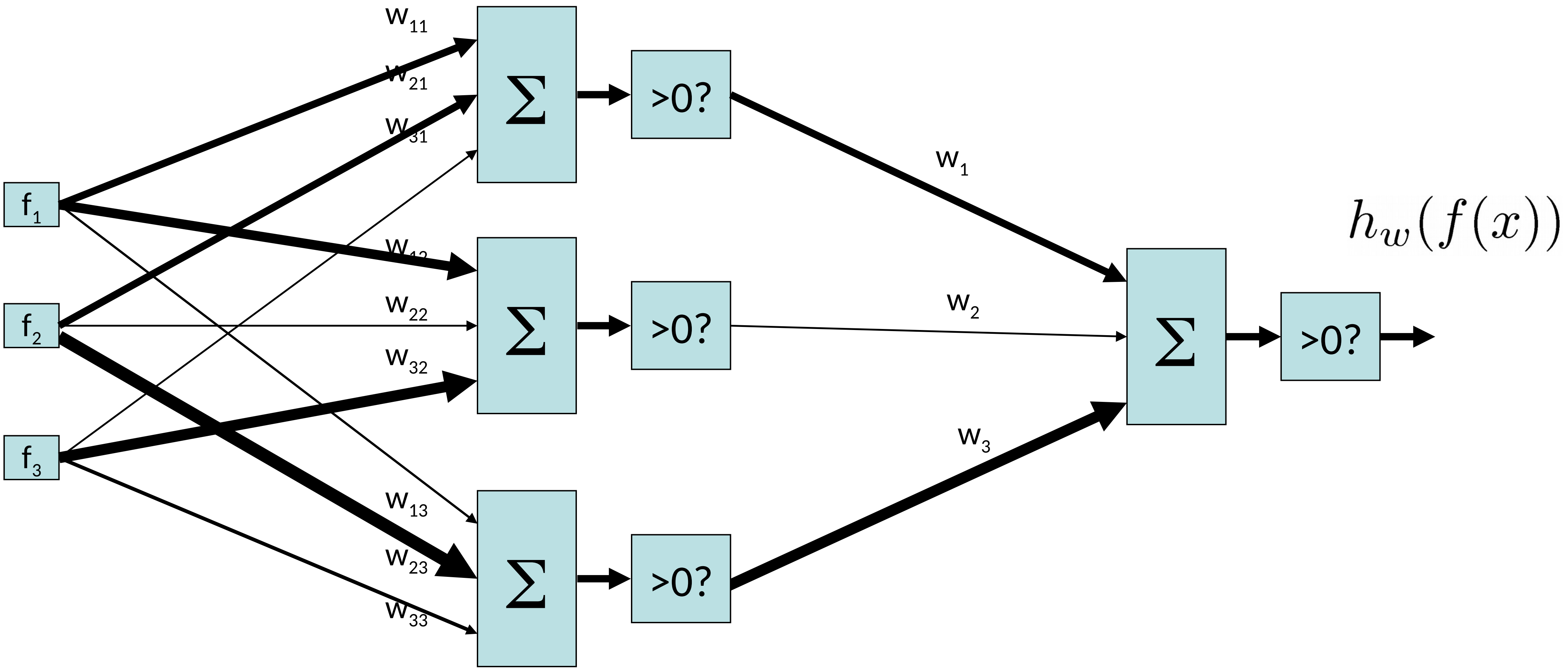
- Midterm grades

- Some context:
  - Deep learning = neural networks
  - AI <> Deep Learning
  - But…it's definitely an important area to know about these days
  - Applications other than vision and natural language processing?
    - Robotics
    - Fraud detection
    - Game playing (e.g. go, Starcraft)
    - Election predictions

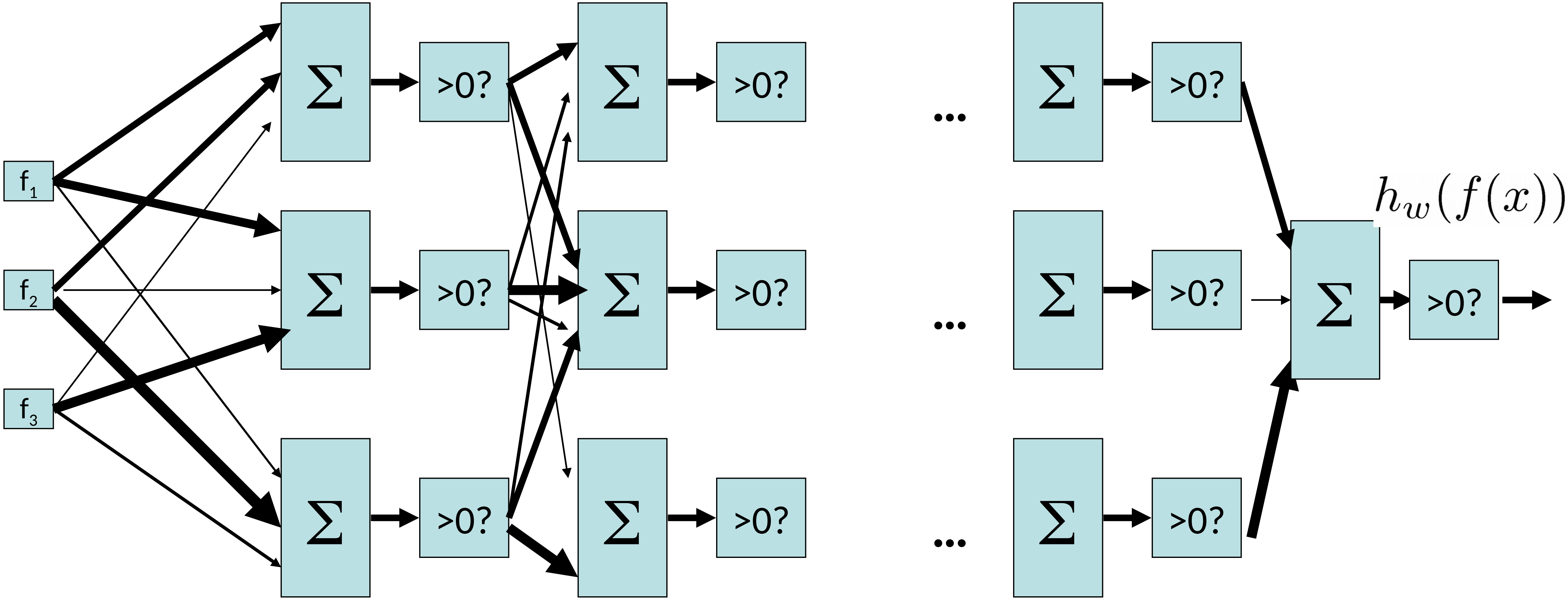
Perceptron



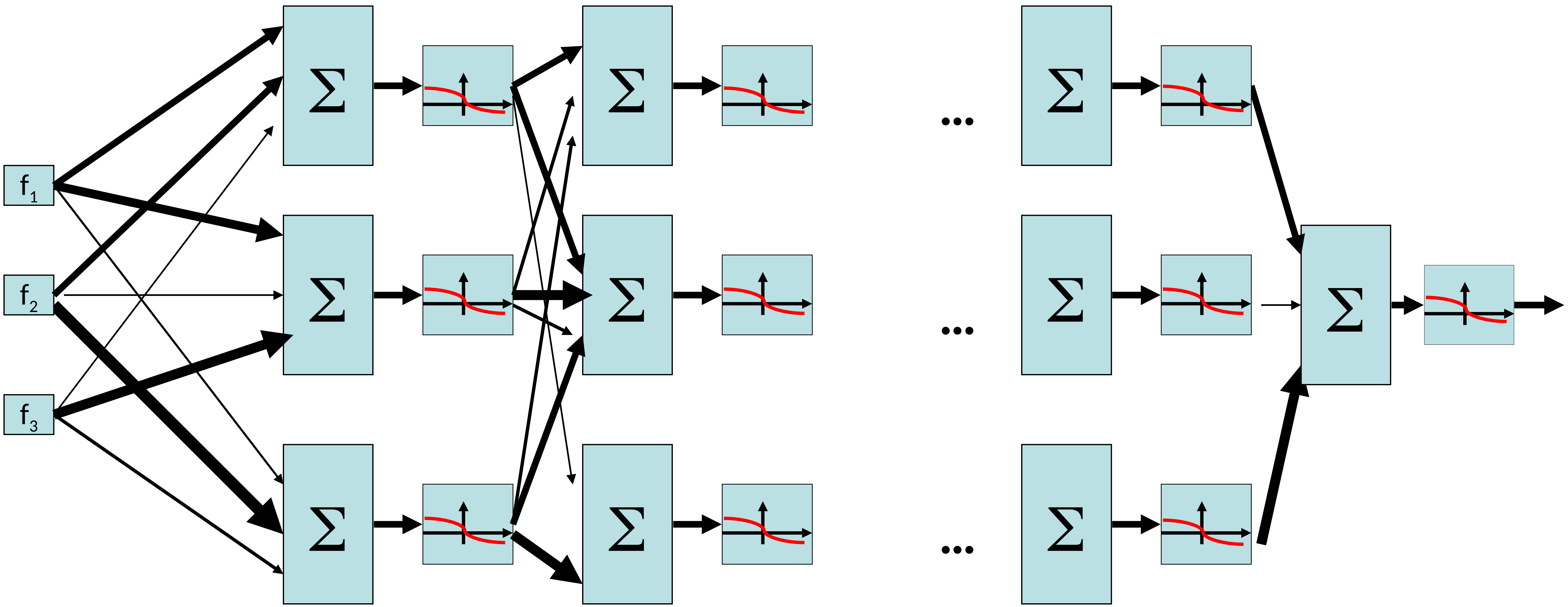
Two-Layer Perceptron Network



N-Layer Perceptron Network



N-Layer Neural Network



Some of Your Questions

- What's the relationship between a weight and a gradient?
  - How does gradient ascent work?
    - *[Policy Grad RL slides]*
    - *[Gradient ascent problem]*
  - Can gradient ascent be changed to find a global maximum? (Michael Labarca)
- What's the purpose of the activation function?
- What exactly is backpropagation?
  - *[Gradient computation problem]*
- Are there other ways to train NNs? (Tyler Miller)
  - NEAT

Test Your Understanding

- Data sufficiency problem

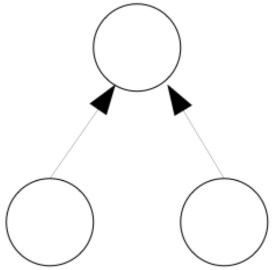
- Practice problem in breakout rooms
- Talk about each subproblem individually

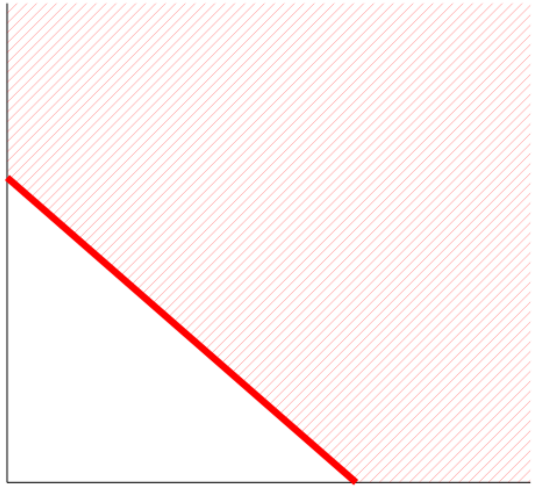
Some of Your Questions

- What’s the purpose of the activation function?
- Why are leaky RELU units better than sigmoids or tanh? (Cyrus Mahdavi)
- *[Representation capacity problem]*

What Can be Done with Non-Linear (e.g., Threshold) Units?

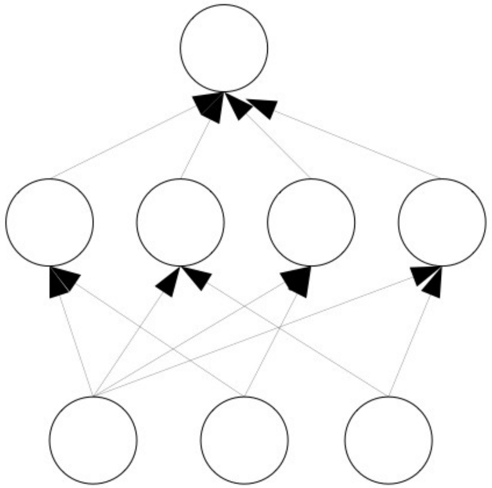
1 layer of trainable weights

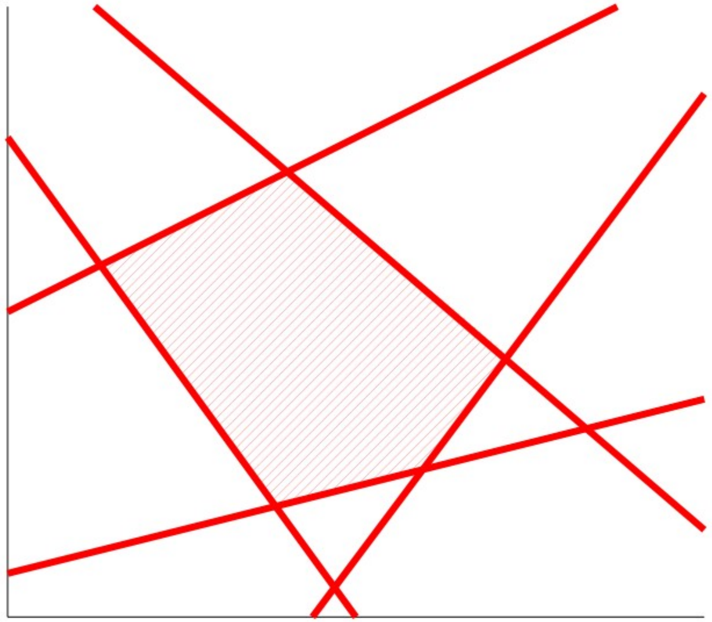




separating hyperplane

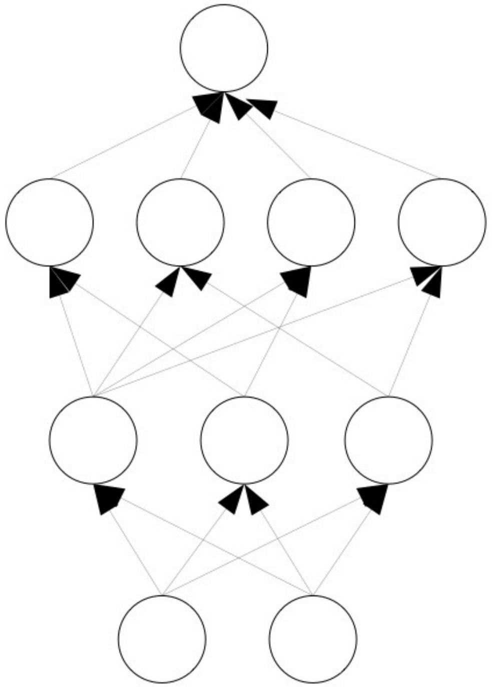
2 layers of trainable weights

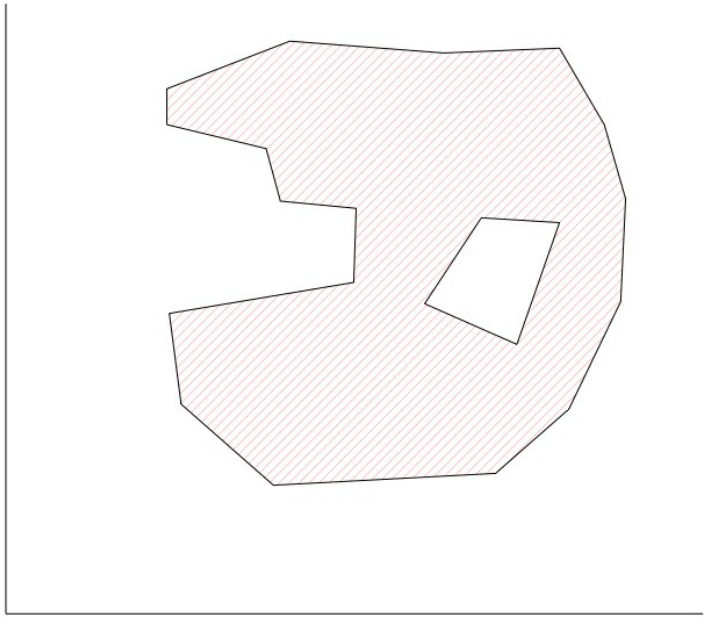




convex polygon region

3 layers of trainable weights



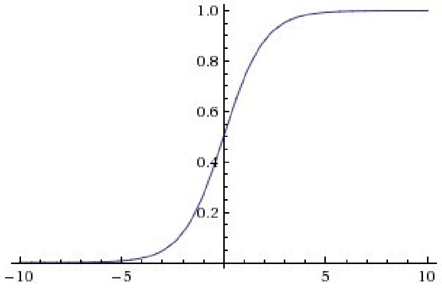


composition of polygons:
non convex regions

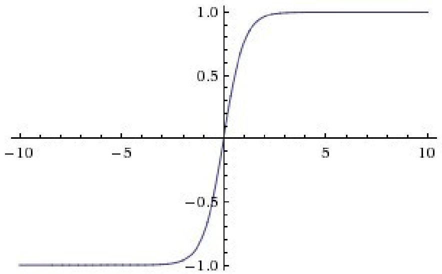
Activation Functions

Sigmoid

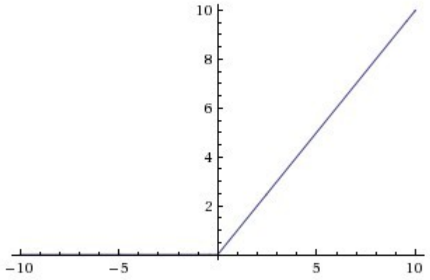
$$\sigma(x) = 1/(1+e^{-x})$$



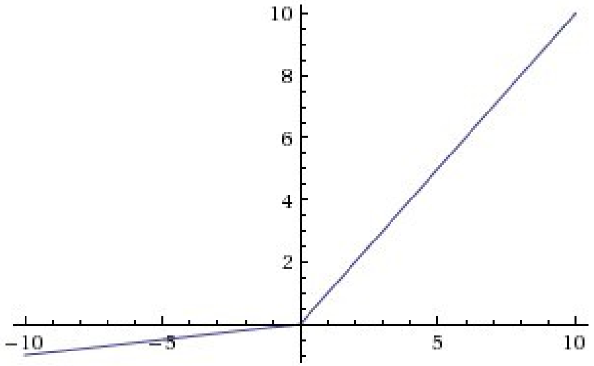
tanh tanh(x)



ReLU max(0,x)



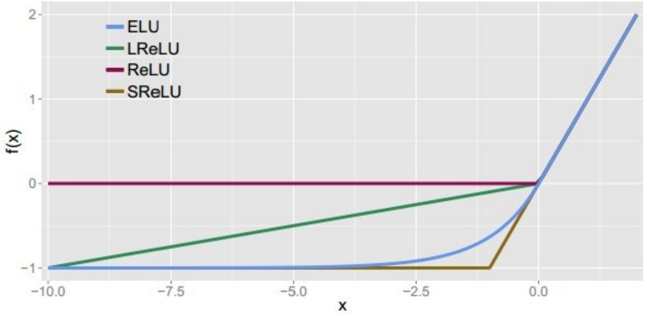
Leaky ReLU
max(0.1x, x)



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

ELU

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



Some of Your Questions

- What are hyperparameters and how do you tune them?
  - Activation units
  - Learning rate
  - Momentum parameter
  - Dropout parameters
  - Normalization scheme
  - Number of layers and units (architecture)

- How do you design architectures?
  - Neural architecture search *[NEAT+Q slides]*
  - Can hyperparameters be made learned parameters? (Conrad Li)
  - What's the difference between adding layers vs. widening a layer? (Michael Labarca)
  - Does batch normalization correct for bad initialization? (Yuhan Zheng)

Some of Your Questions

- Differences between brain’s NN and artificial NNs design? (Pranooha Veeramachaneni)
    - Are NNs designed based on intuition from brain structures? (Rudraksh Garg)
- What are the limitations of NNs? (Jack Si)
    - How far can we go with deeper and larger networks? (Trong Lv)
- When shouldn’t you use NNs? (Ethan Houston)

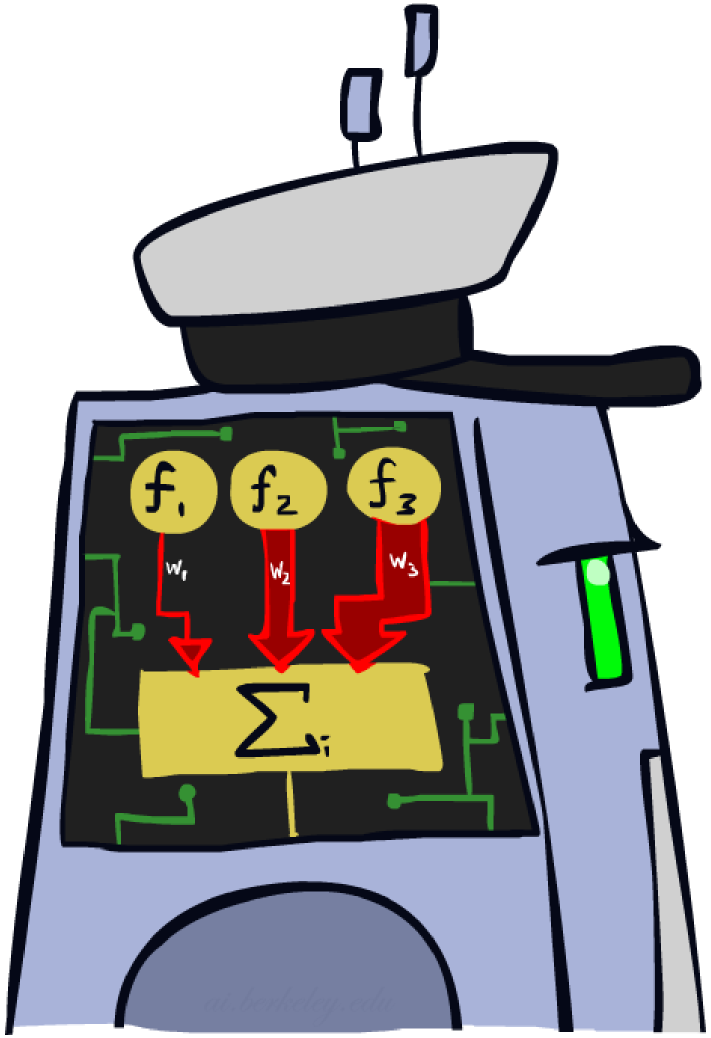
Some of Your Questions

- Computation requirements of NNs vs. traditional vision classification (Vijay Vuyyuru)
  - Training vs. testing
  - Depends on hardware
  - Why does deep learning work so much better on GPUs? (Cameron Doggett)
- Why manual features favored in past, and only recently NNs favored? (Nalin Mahajan)
- If image recognition is so good, why do some websites still require you to identify images to check if you're a robot? (Jessica Ma)

Some of Your Questions

- Are there good ways of introducing human knowledge into NNs? Or is that missing the point? (Tyler Miller)
    - Neurosymbolic systems

Review: Linear Classifiers



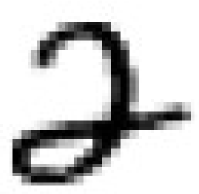
Feature Vectors

x $\quad$ $f(x)$ $\quad$ y

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

```
# free        : 2
YOUR_NAME     : 0
MISSPELLED    : 2
FROM_FRIEND   : 0
...
```

SPAM
or
+

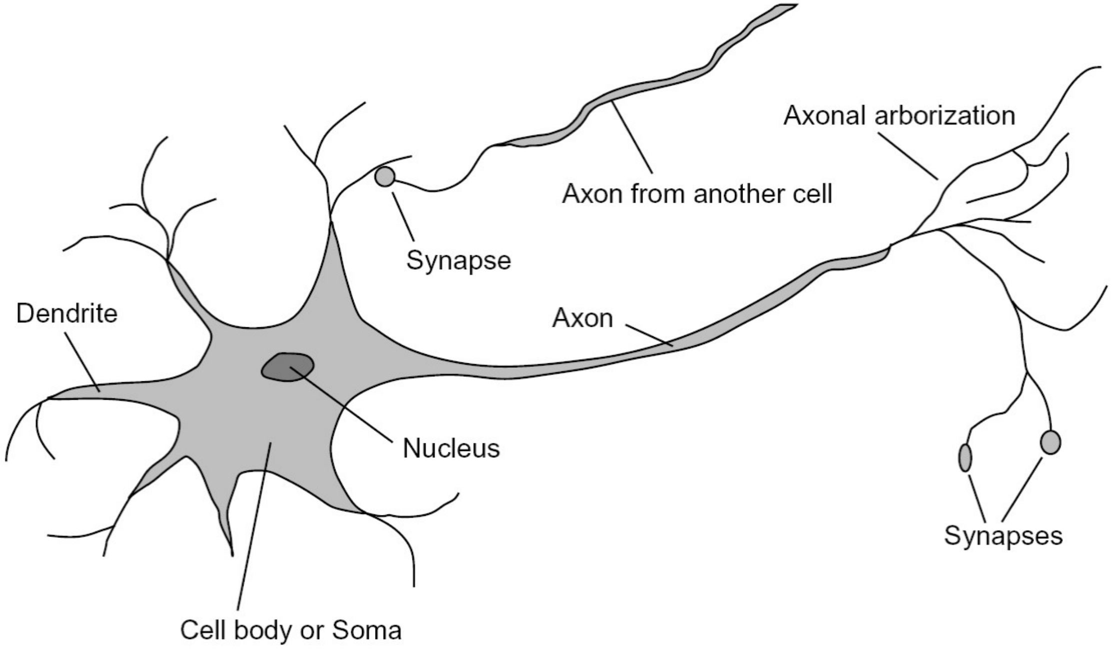


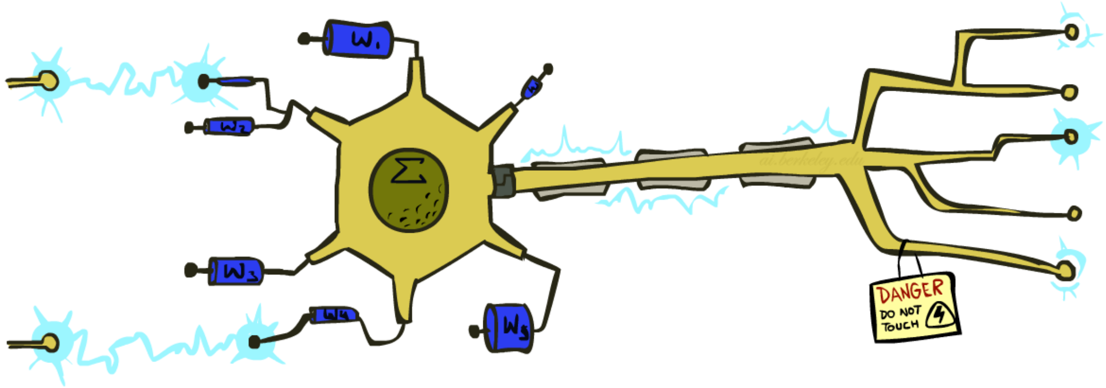
```
PIXEL-7,12  : 1
PIXEL-7,13  : 0
...
NUM_LOOPS   : 1
...
```

“2”

Some (Simplified) Biology

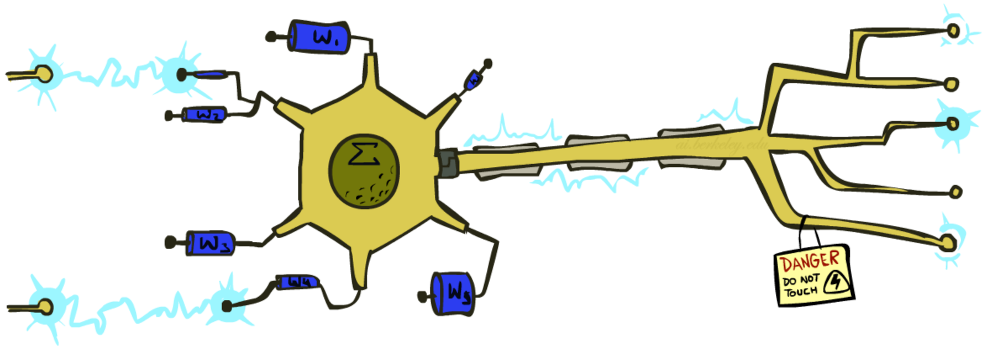
- Very loose inspiration: human neurons





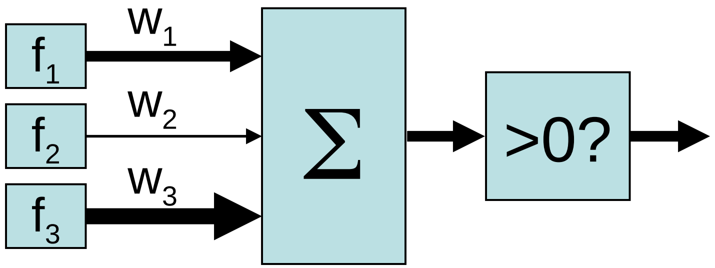
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

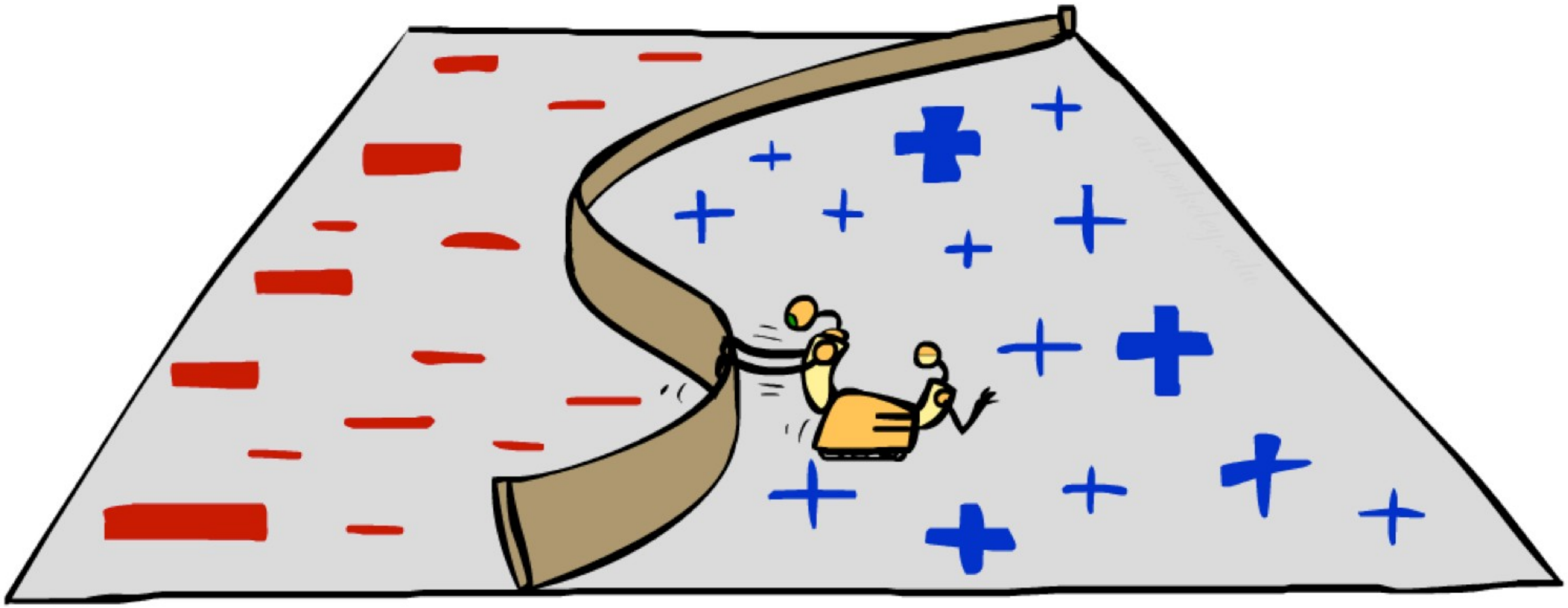


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  -

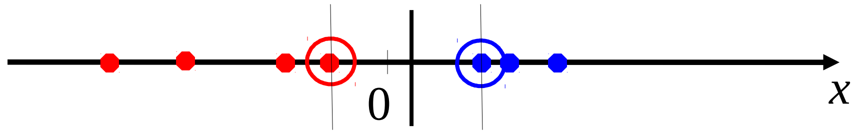


Non-Linearity

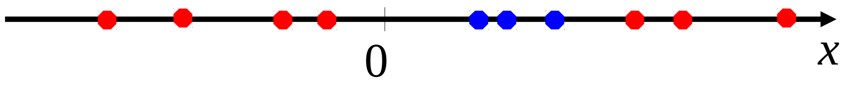


Non-Linear Separators

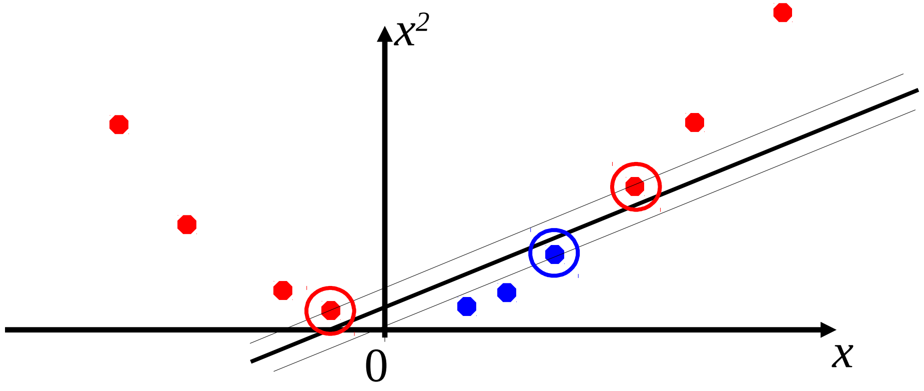
- Data that is linearly separable works out great for linear decision rules:



- But what are we going to do if the dataset is just too hard?



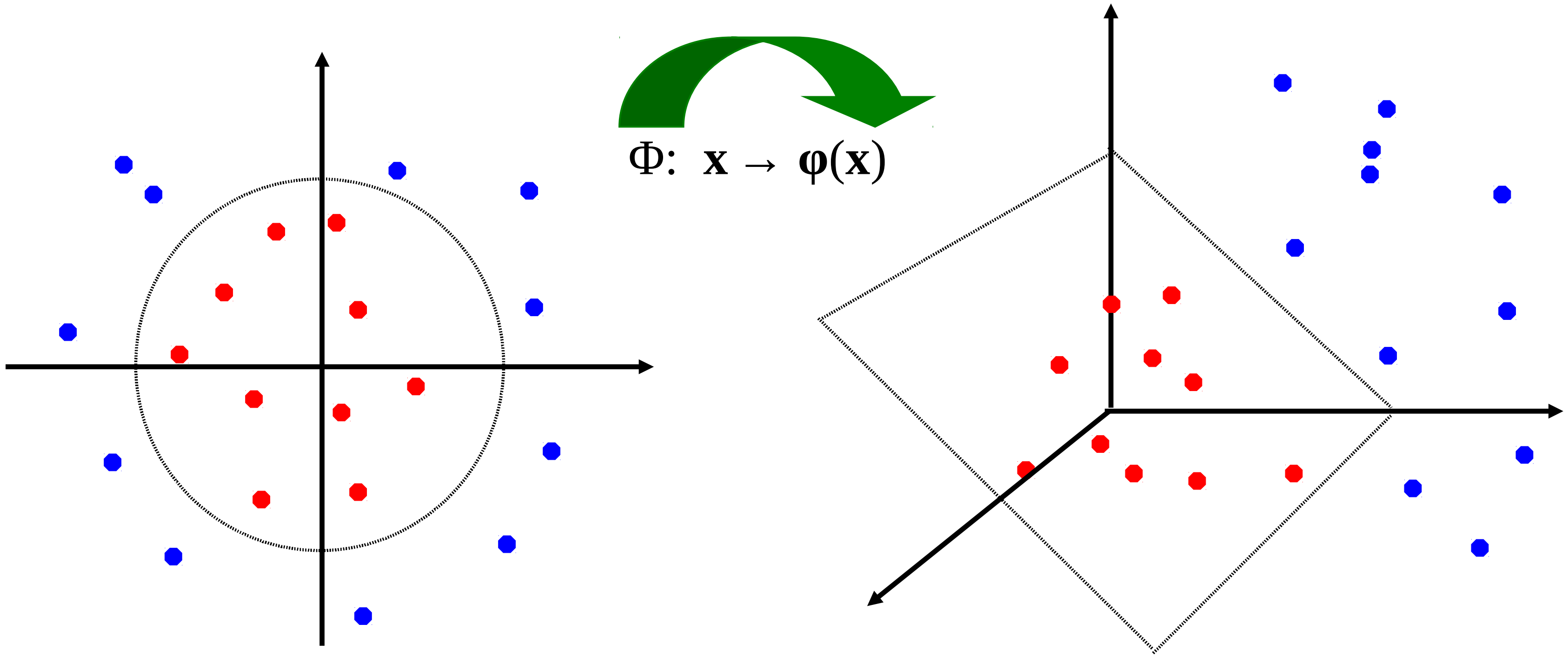
-



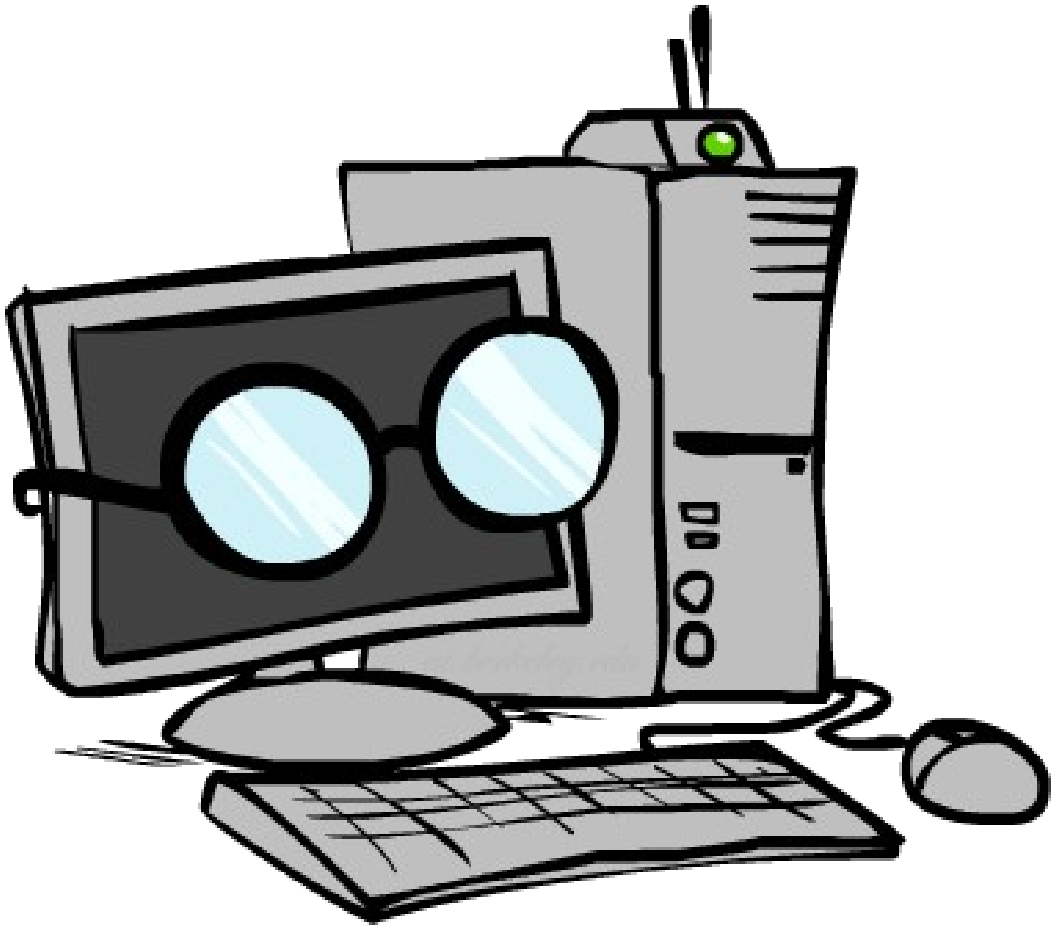
This and next slide adapted from Ray Mooney, UT

Non-Linear Separators

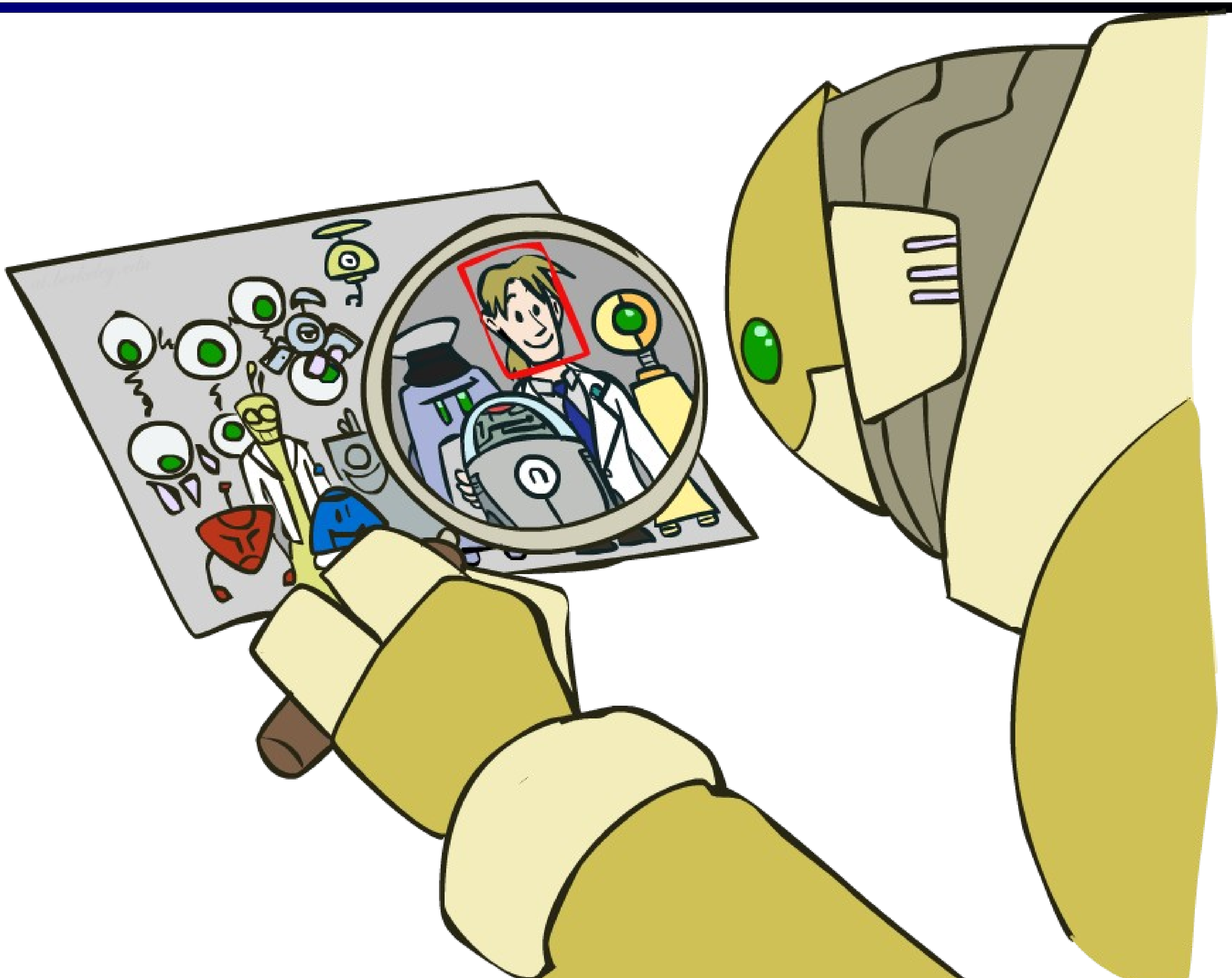
- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



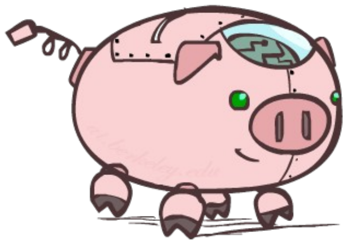
Computer Vision

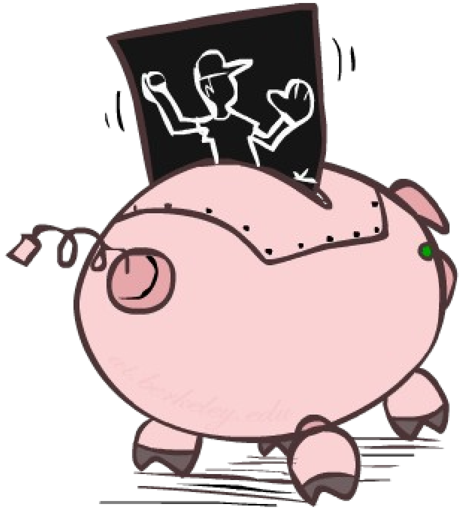


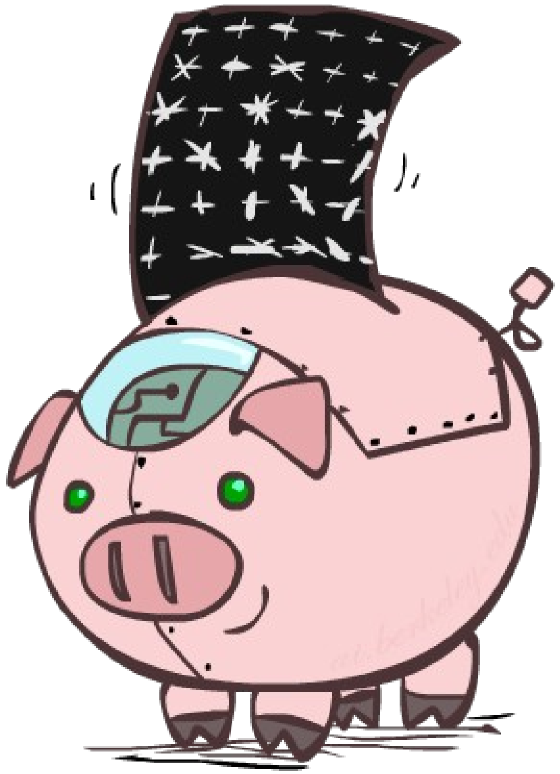
Object Detection



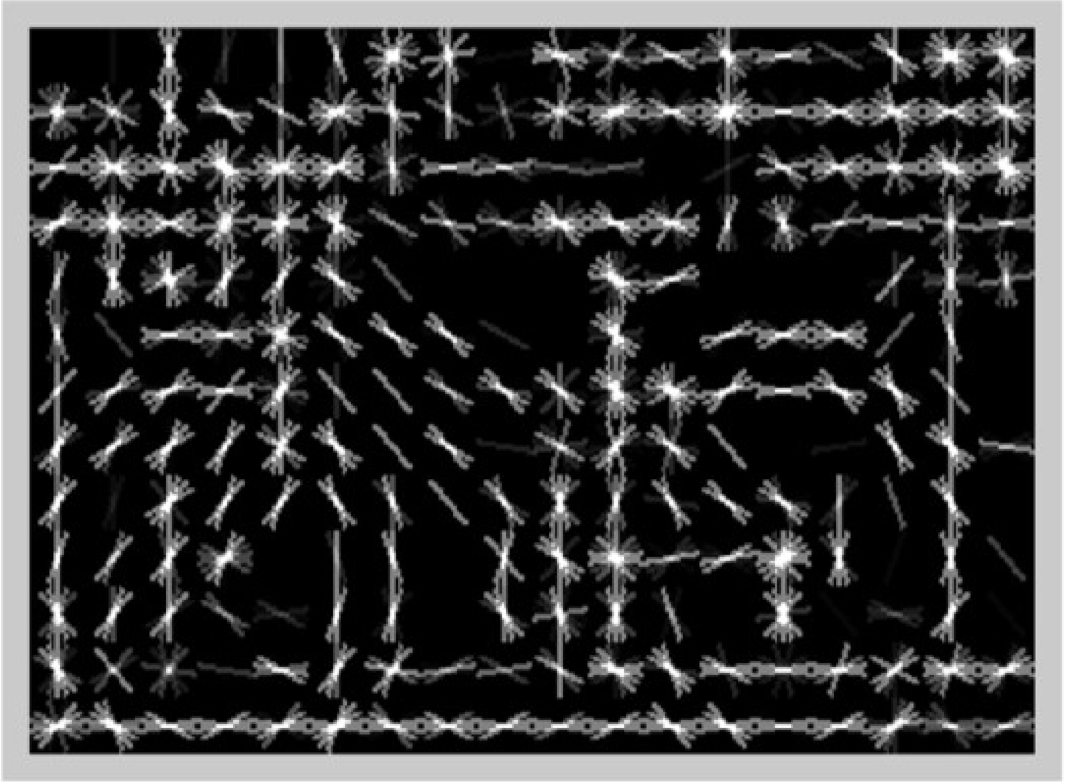
Manual Feature Design







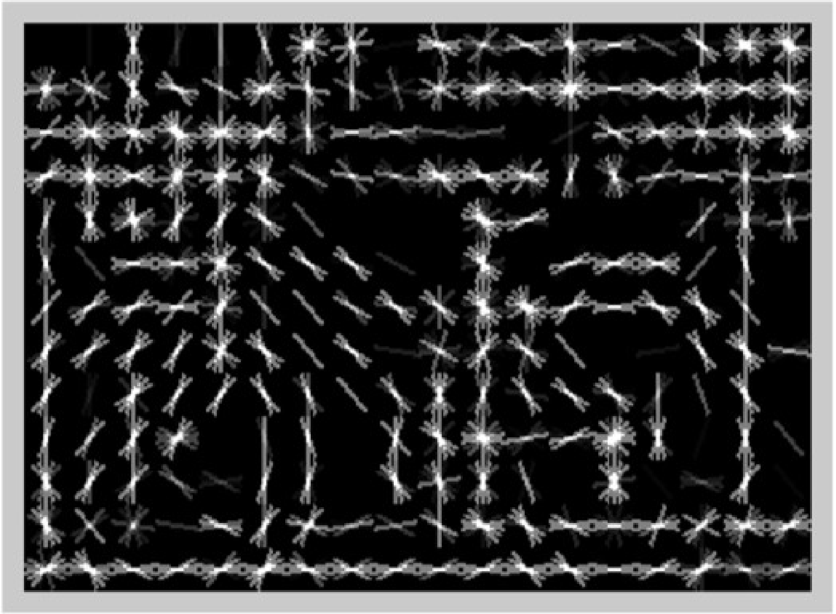
Features and Generalization



[Dalal and Triggs, 2005]

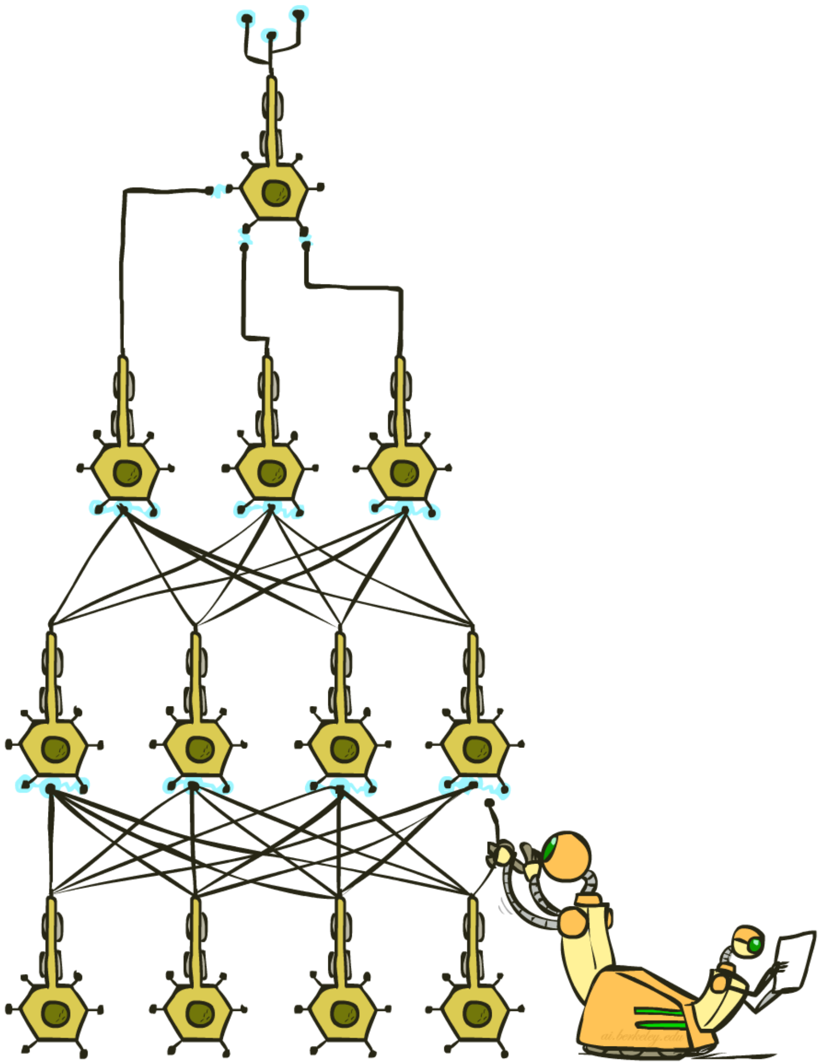
Features and Generalization

Image



HoG

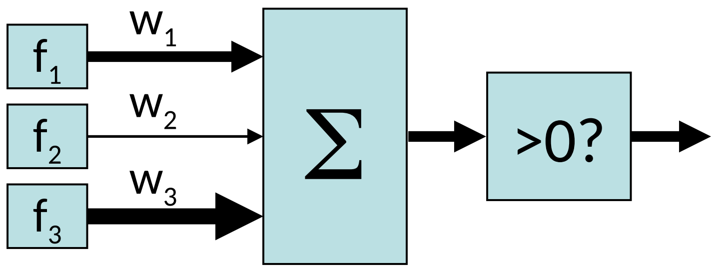
Manual Feature Design → Deep Learning



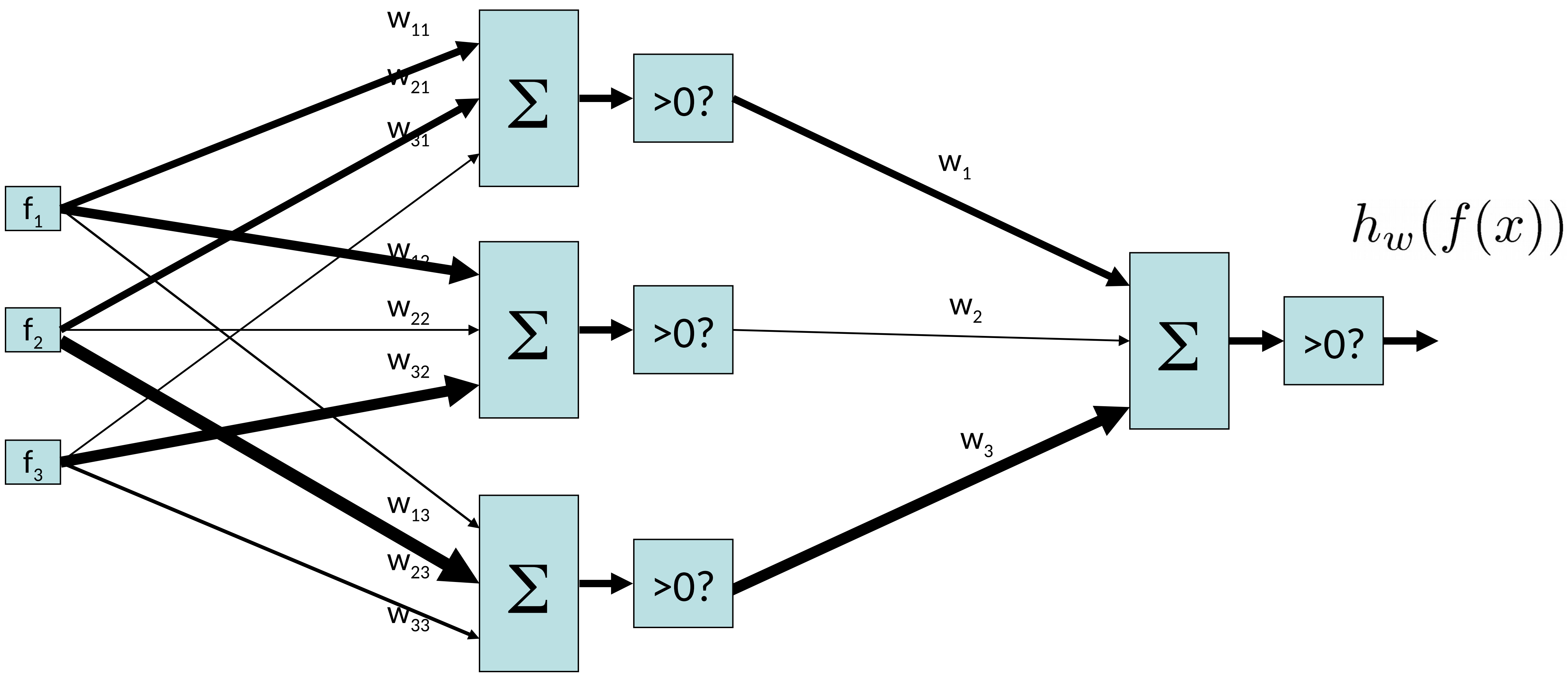
- Manual feature design requires:
  - Domain-specific expertise
  - Domain-specific effort

- What if we could learn the features, too?

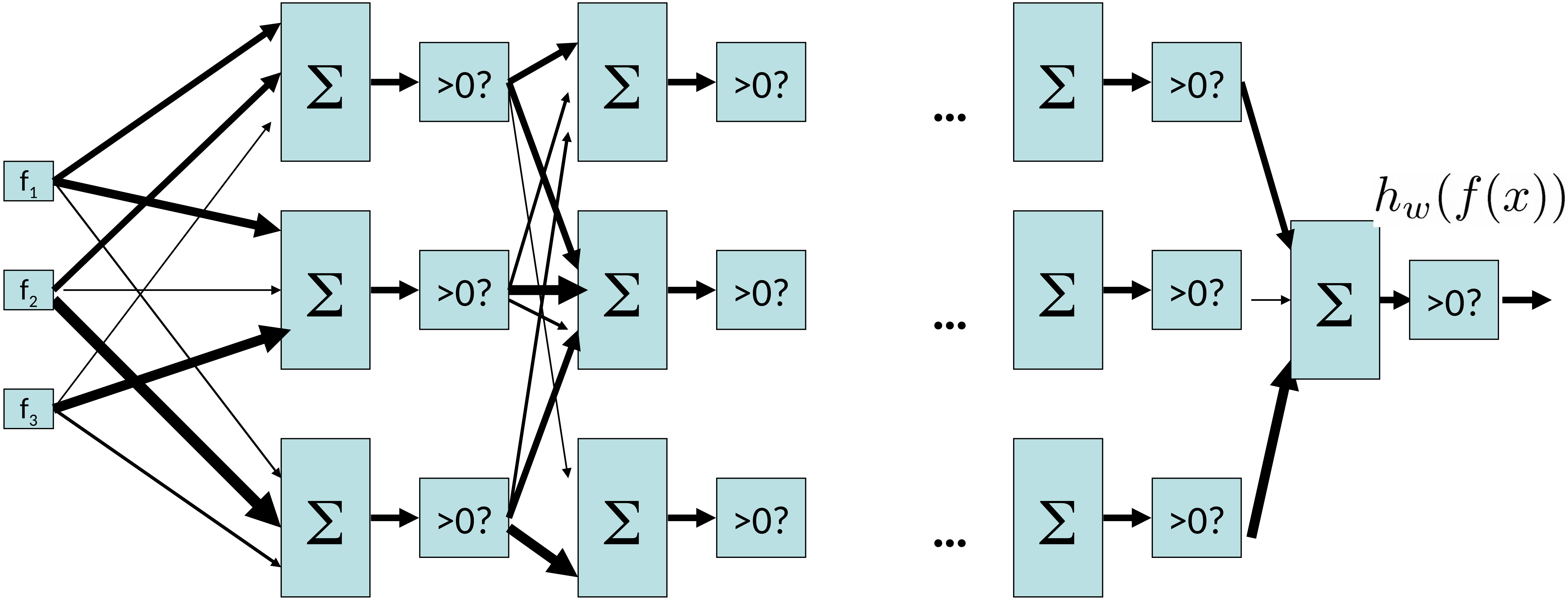
Perceptron



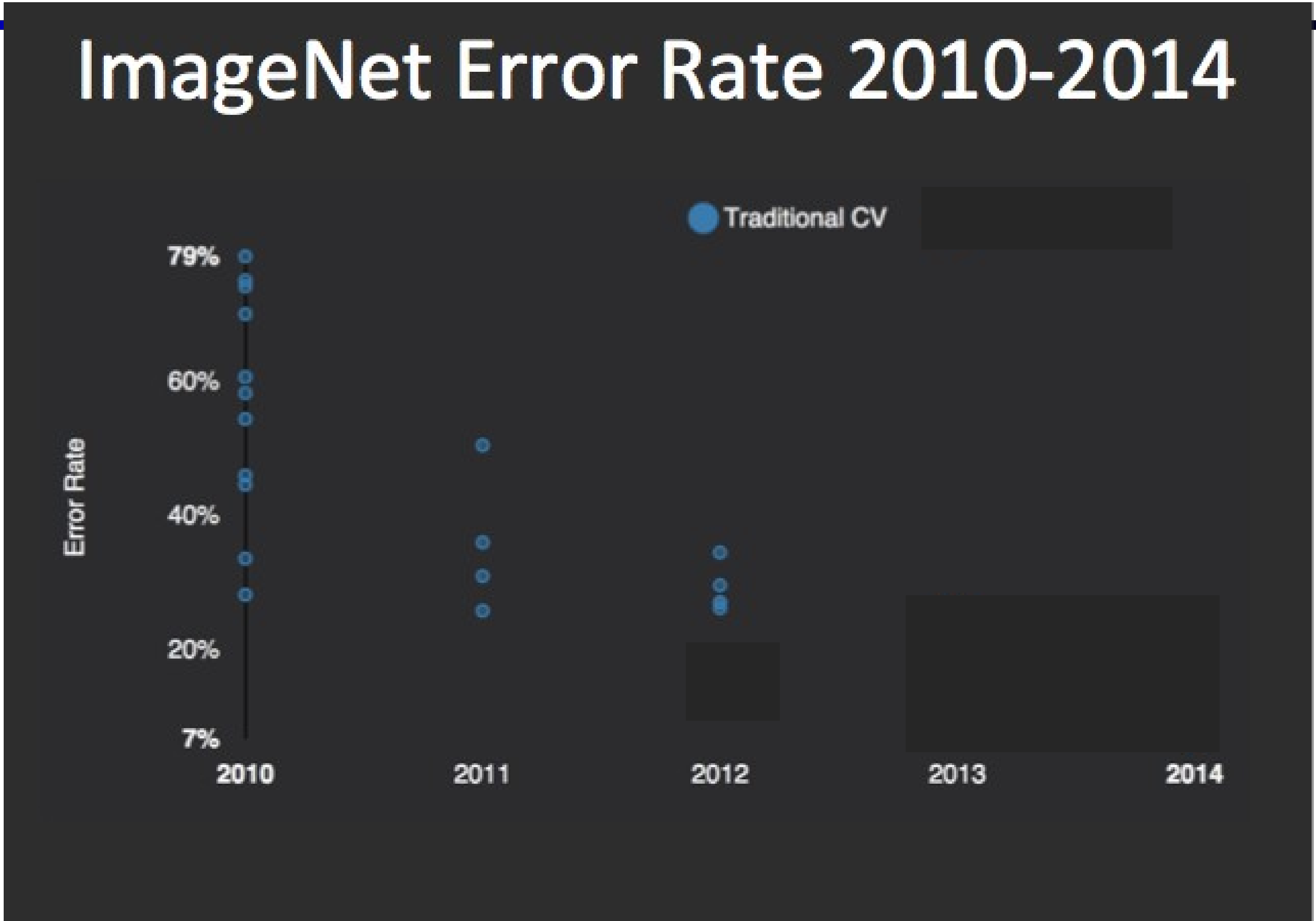
Two-Layer Perceptron Network



N-Layer Perceptron Network

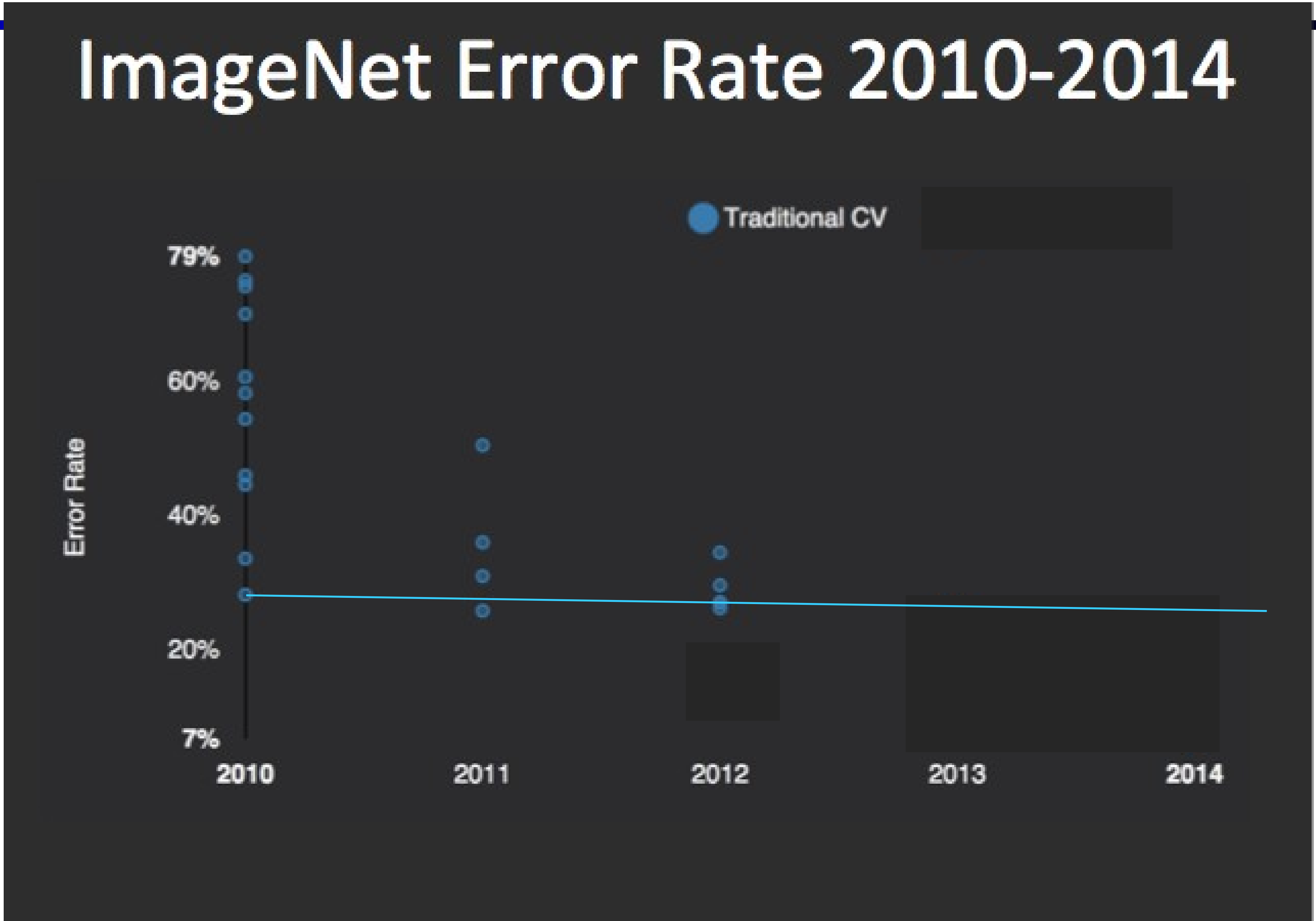


Performance



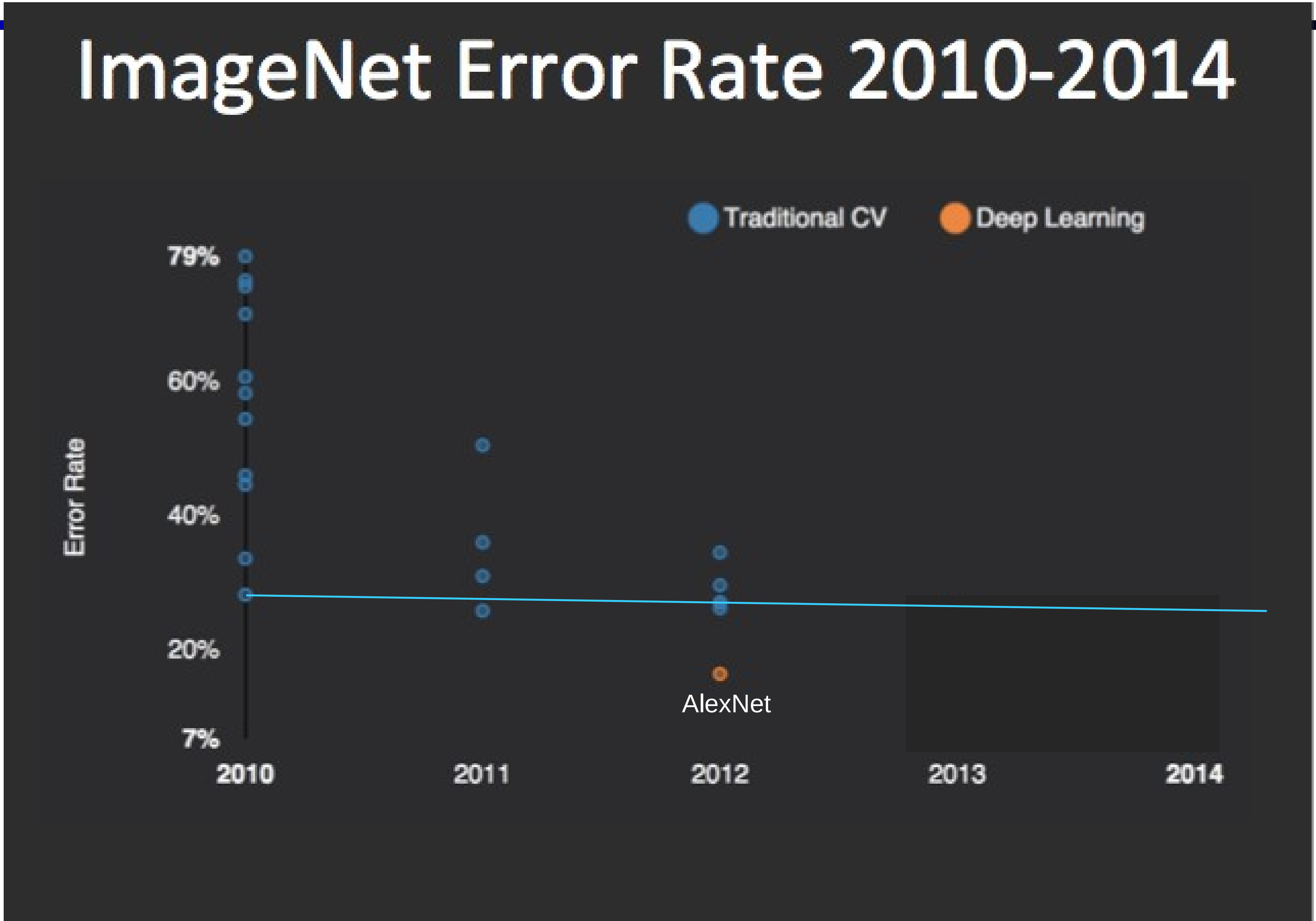
graph credit Matt Zeiler, Clarifai

Performance



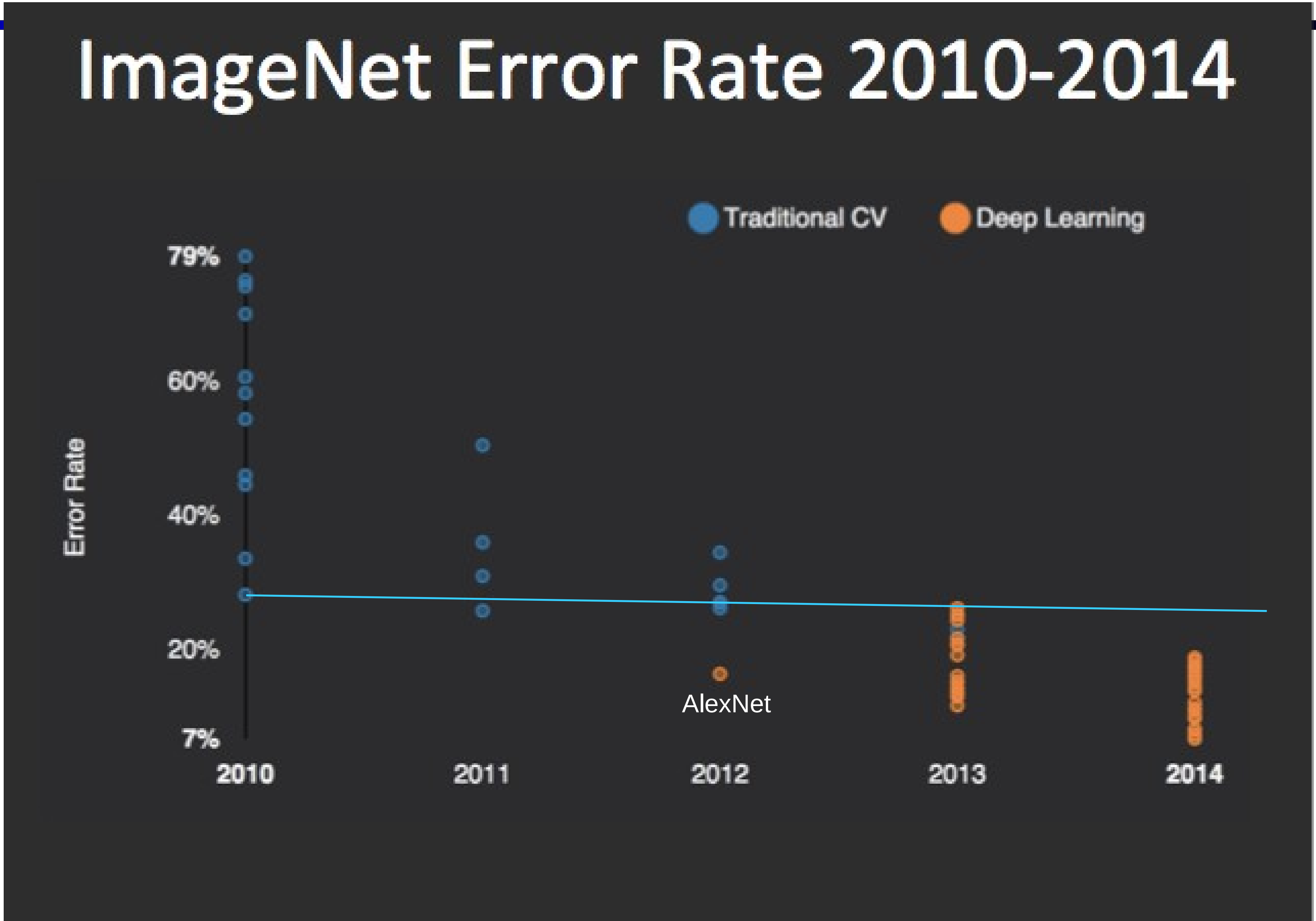
graph credit Matt Zeiler, Clarifai

Performance



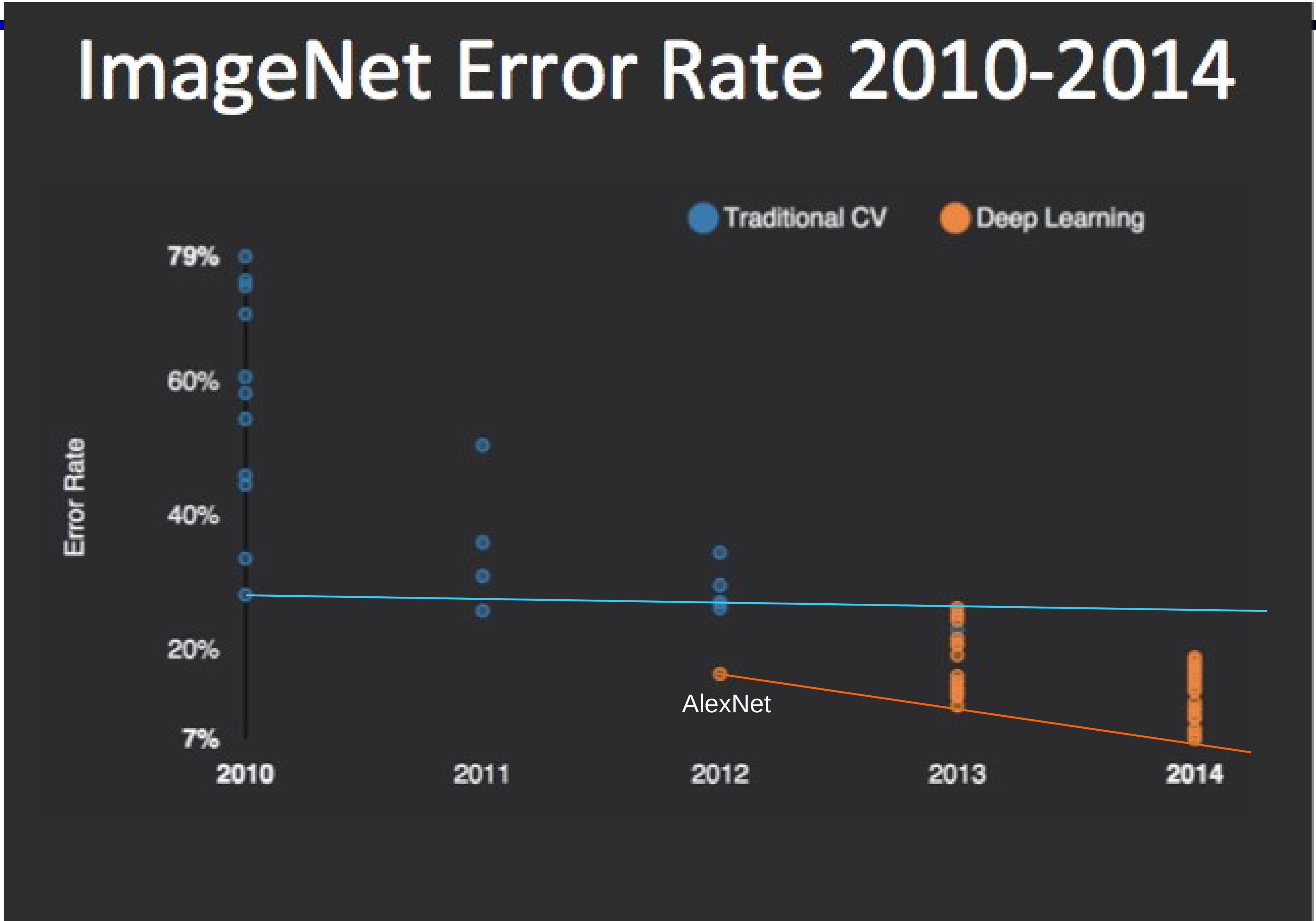
graph credit Matt Zeiler, Clarifai

Performance



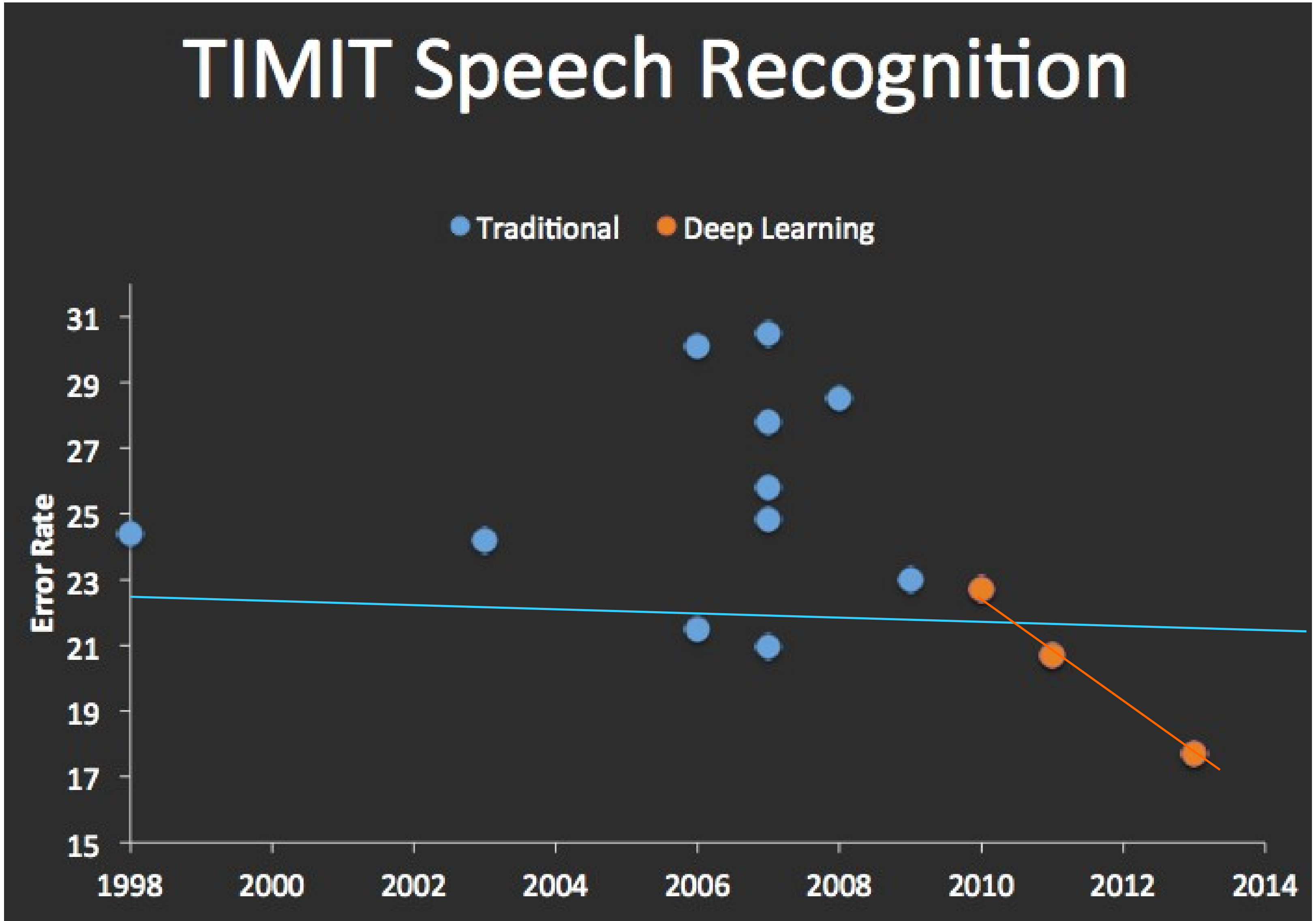
graph credit Matt Zeiler, Clarifai

Performance



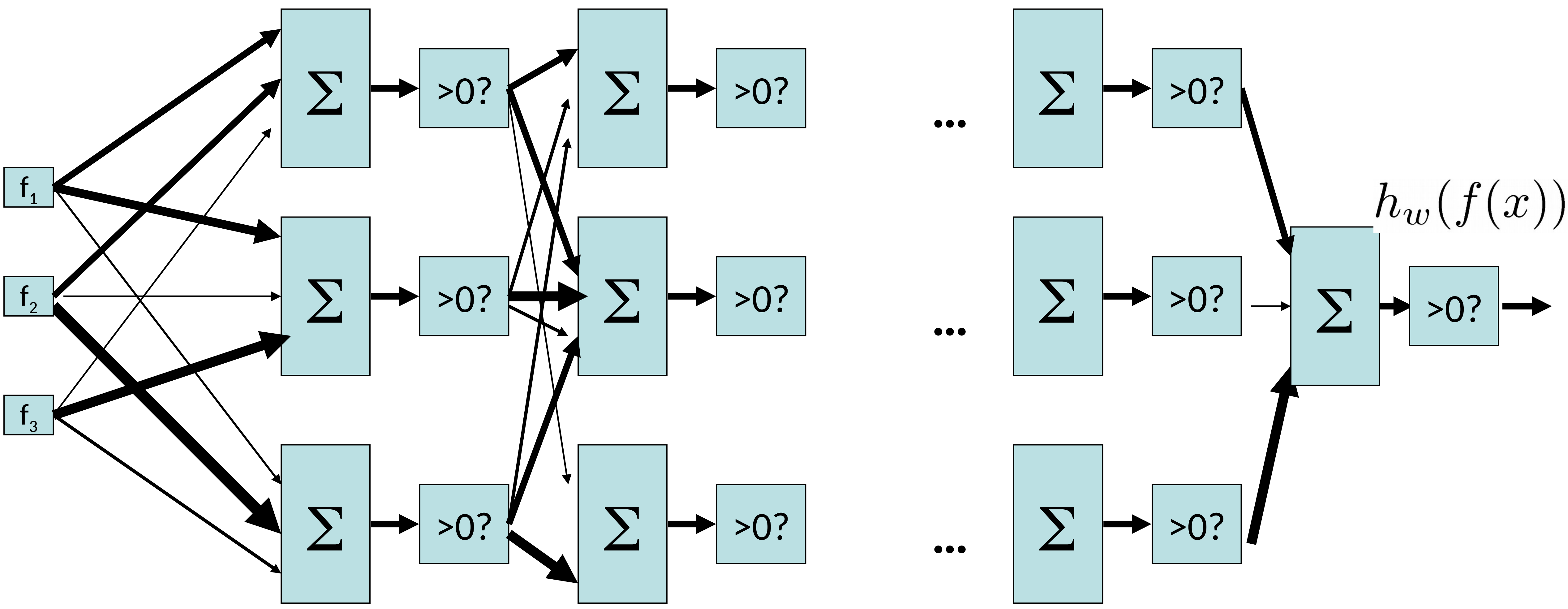
graph credit Matt Zeiler, Clarifai

Speech Recognition



graph credit Matt Zeiler, Clarifai

N-Layer Perceptron Network



Local Search

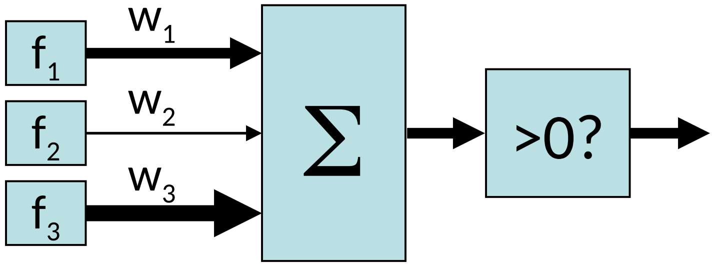
- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
  - Neighbors = small perturbations of w

- Properties
  - Plateaus and local optima

How to escape plateaus and find a good local optimum?

How to deal with very large parameter vectors? E.g., $w \in \mathbb{R}^{1billion}$

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 $\Rightarrow$ how to measure incremental progress toward a correct label?

Soft-Max

- Score for y=1: $w^\top f(x)$ Score for y=-1: $-w^\top f(x)$

- Probability of label:

$$p(y=1|f(x);w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$p(y=-1|f(x);w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

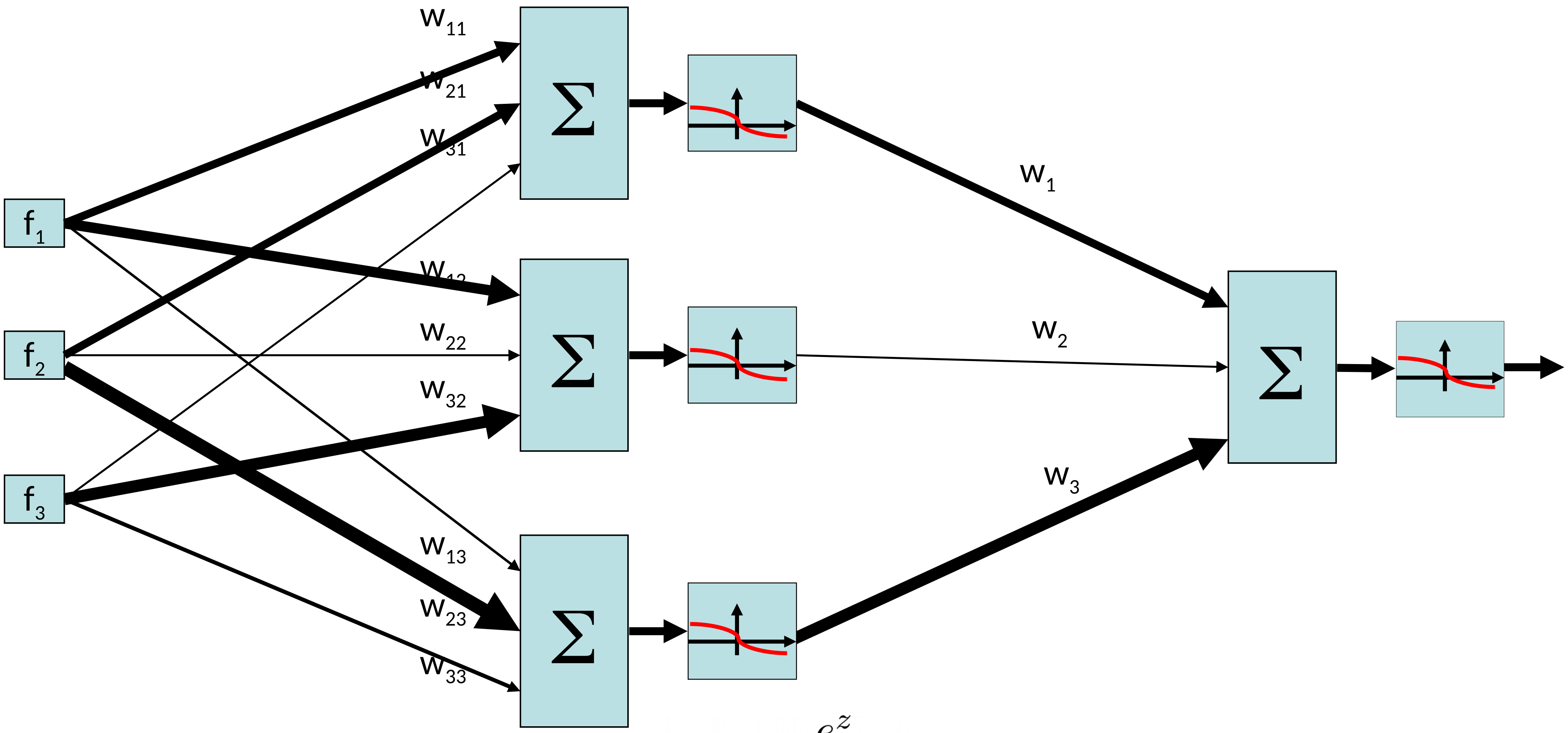
- Objective:

$$l(w) = \prod_{i=1}^{m} p(y = y^{(i)}|f(x^{(i)});w)$$

- Log:

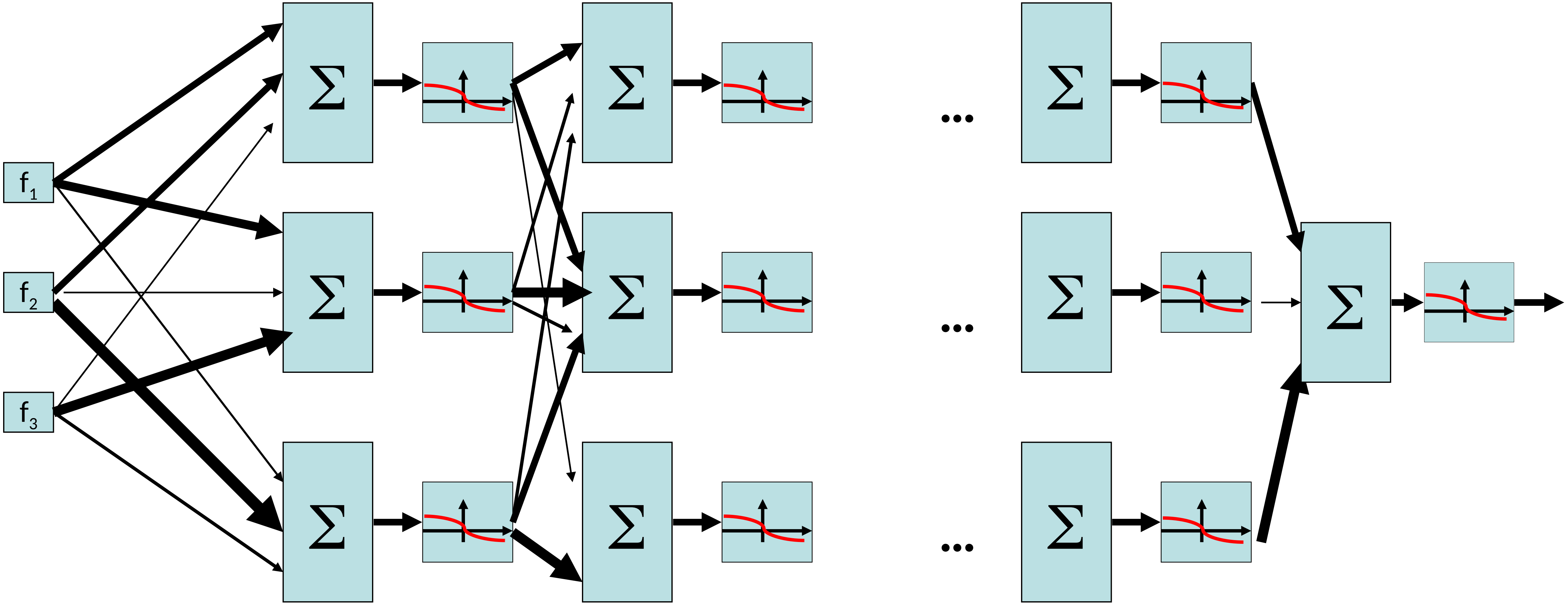
$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)}|f(x^{(i)});w)$$

Two-Layer Neural Network



$$z \rightarrow \frac{e^z}{e^z + e^{-z}}$$

N-Layer Neural Network



Our Status

- Our objective $ll(w)$
  - Changes smoothly with changes in w
  - Doesn't suffer from the same plateaus as the perceptron network

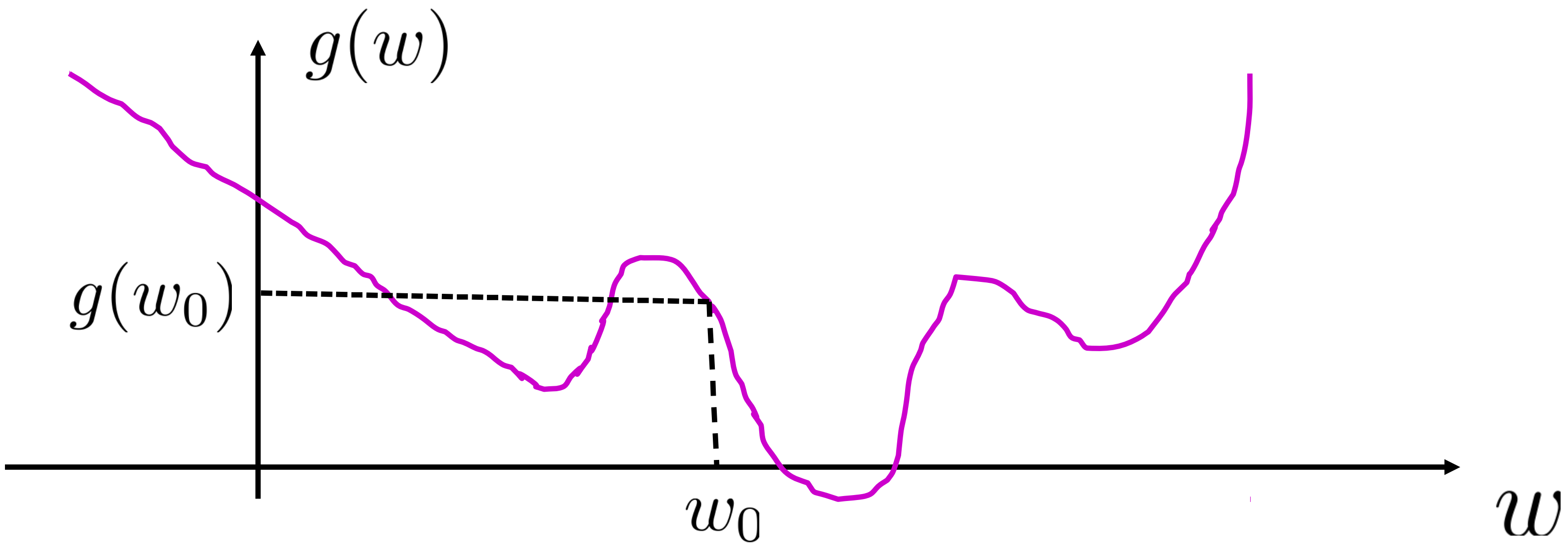
- Challenge: how to find a good w ?

$$\max_w ll(w)$$

- Equivalently:

$$\min_w -ll(w)$$

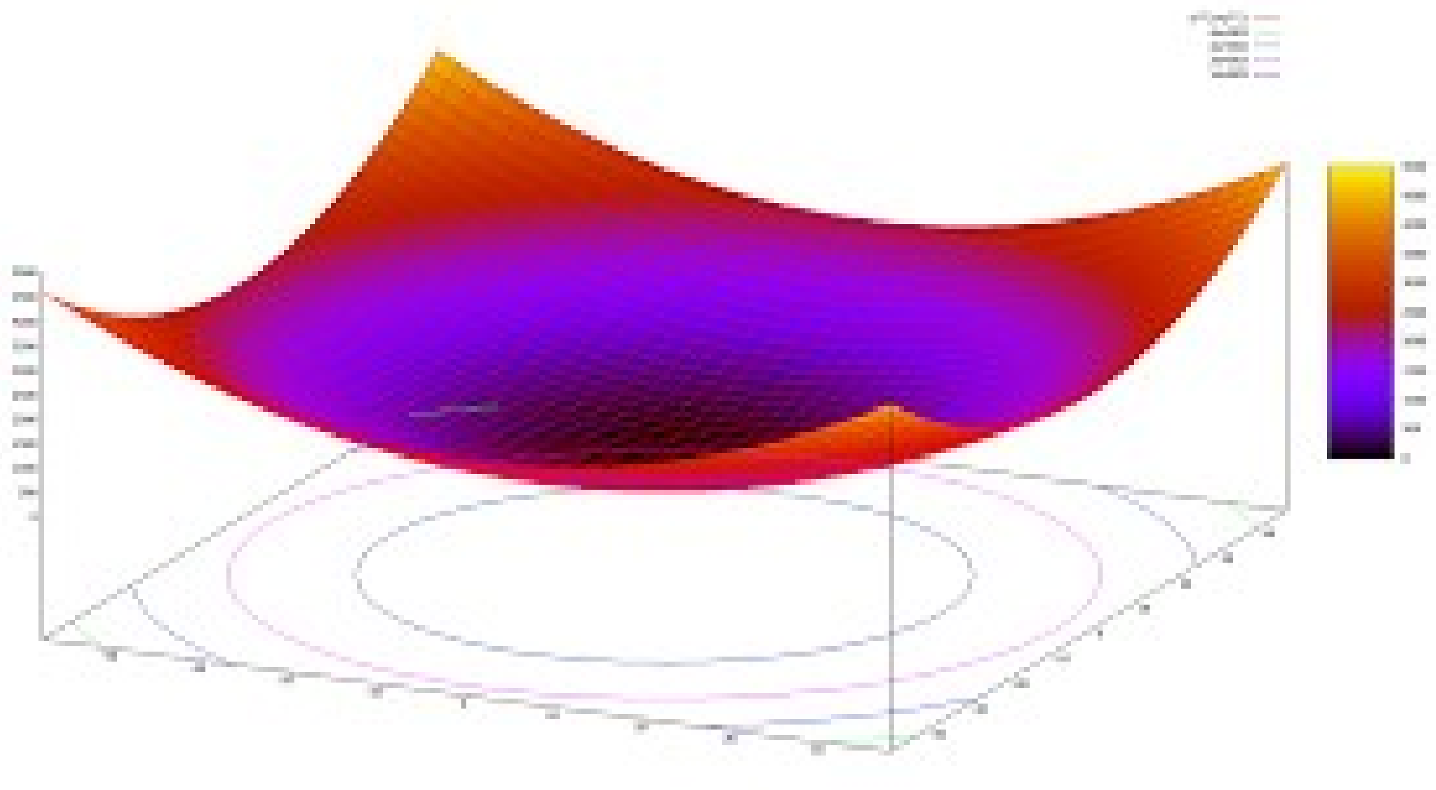
1-d optimization



- Could evaluate $g(w_0+h)$ and $g(w_0-h)$
  - Then step in best direction

- Or, evaluate derivative: $\dfrac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \dfrac{g(w_0+h) - g(w_0-h)}{2h}$
  - Tells which direction to step in

2-D Optimization



Source: Thomas Jungblut’s Blog

Steepest Descent

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

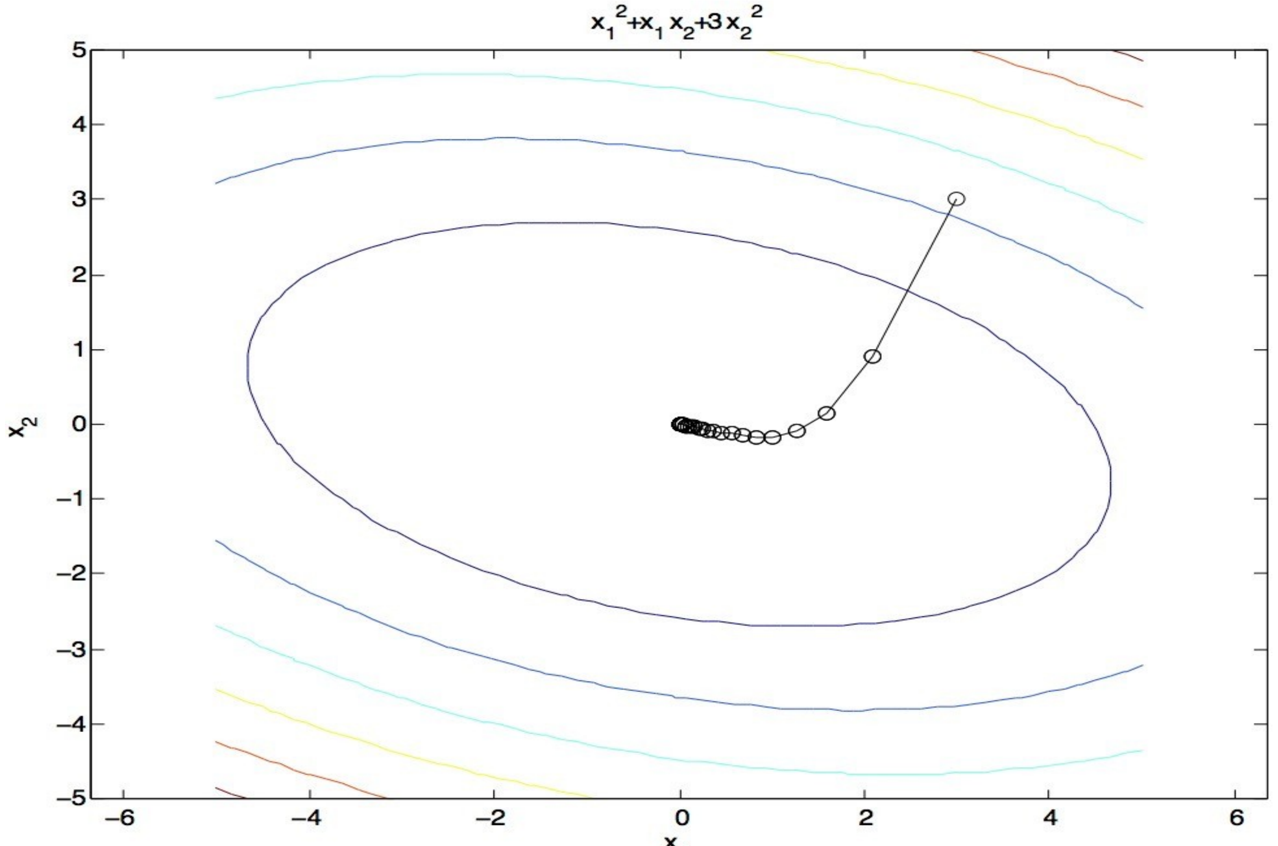


Figure source: Mathworks

What is the Steepest Descent Direction?

What is the Steepest Descent Direction?

- Steepest Direction = direction of the gradient

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

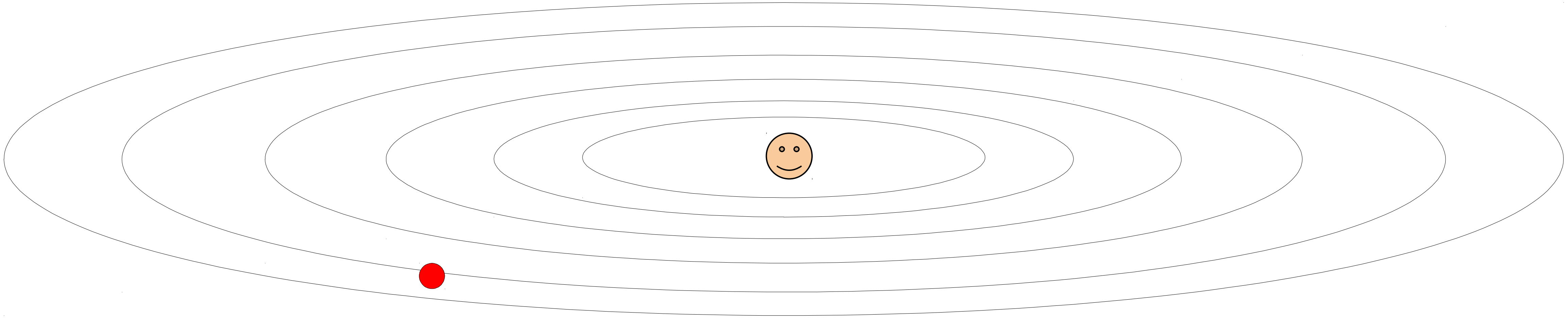
Optimization Procedure 1: Gradient Descent

- Init: w
- For i = 1, 2, …

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

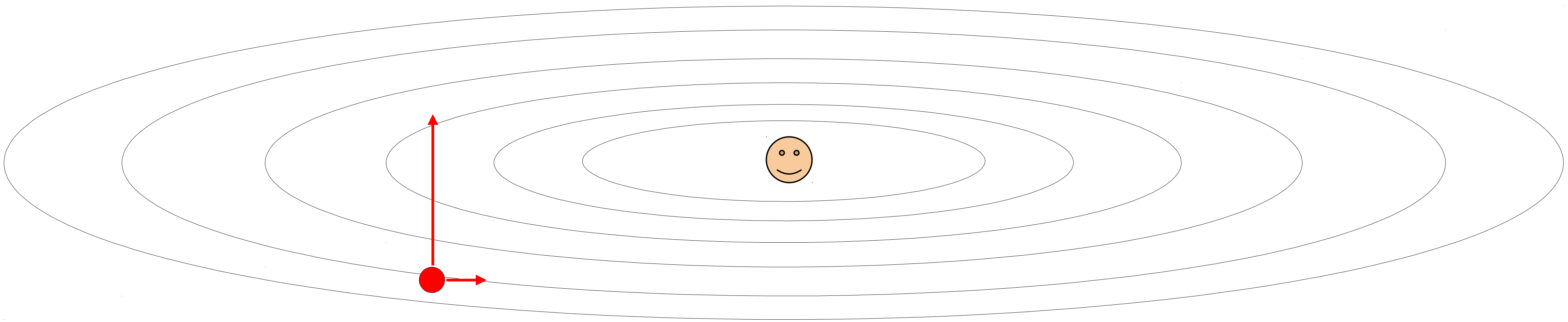
- α: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

Suppose loss function is steep vertically but shallow horizontally:



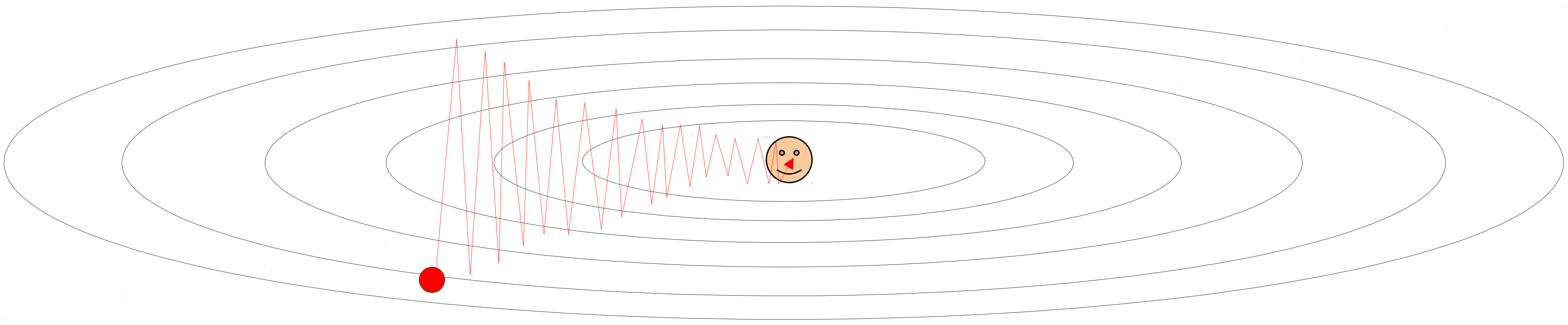
Q: What is the trajectory along which we converge towards the minimum with Gradient Descent?

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with Gradient Descent?

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with Gradient Descent? very slow progress along flat direction, jitter along steep one

Optimization Procedure 2: Momentum

- Gradient Descent
  - Init: w
  - For i = 1, 2, …

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

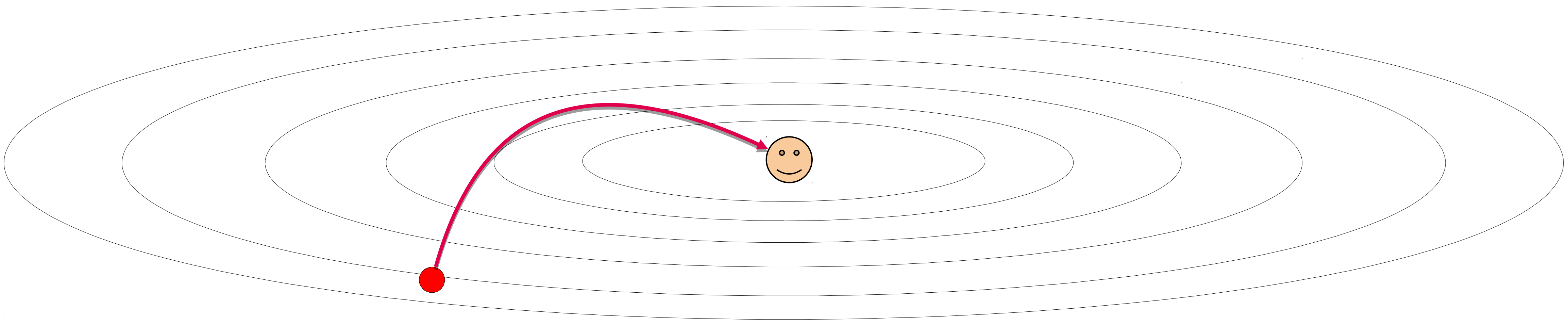
- Momentum
  - Init: w
  - For i = 1, 2, …

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

$$w \leftarrow w + v$$

- 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)

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with Momentum?

How do we actually compute gradient w.r.t. weights?

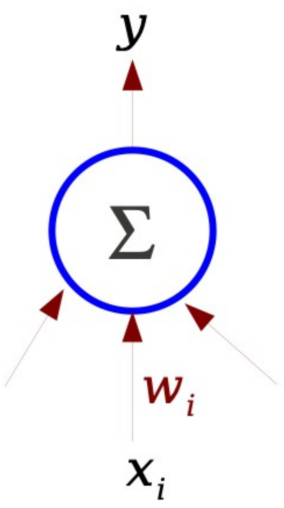
Backpropagation!

Backpropagation Learning

15-486/782: Artificial Neural Networks
David S. Touretzky

Fall 2006

LMS / Widrow-Hoff Rule

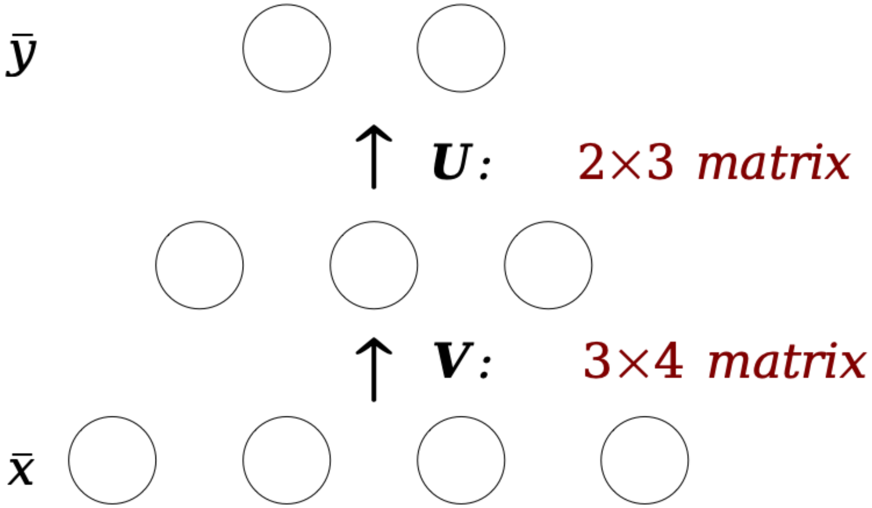


$$\Delta w_i = -\eta(y-d)x_i$$

Works fine for a single layer of trainable weights.

What about multi-layer networks?

With Linear Units, Multiple Layers Don't Add Anything



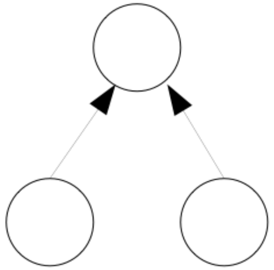
$$\bar{y} = \boldsymbol{U} \times (\boldsymbol{V}\bar{x}) = \underbrace{(\boldsymbol{U} \times \boldsymbol{V})}_{2\times 4}\bar{x}$$

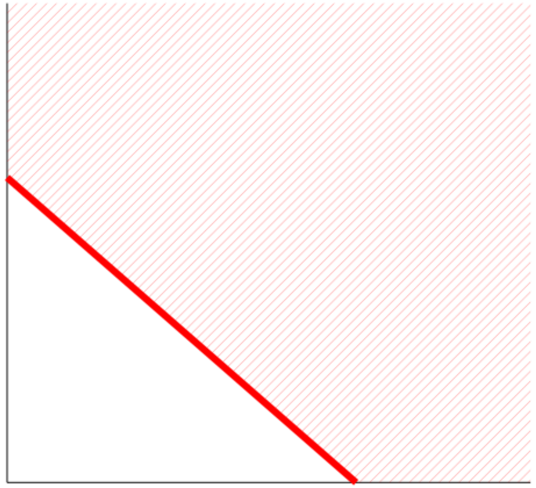
Linear operators are closed under composition.
Equivalent to a single layer of weights $\boldsymbol{W} = \boldsymbol{U} \times \boldsymbol{V}$

But with non-linear units, extra layers add computational power.

What Can be Done with Non-Linear (e.g., Threshold) Units?

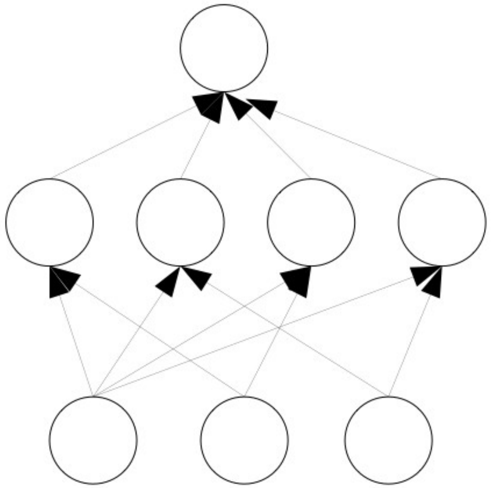
1 layer of trainable weights

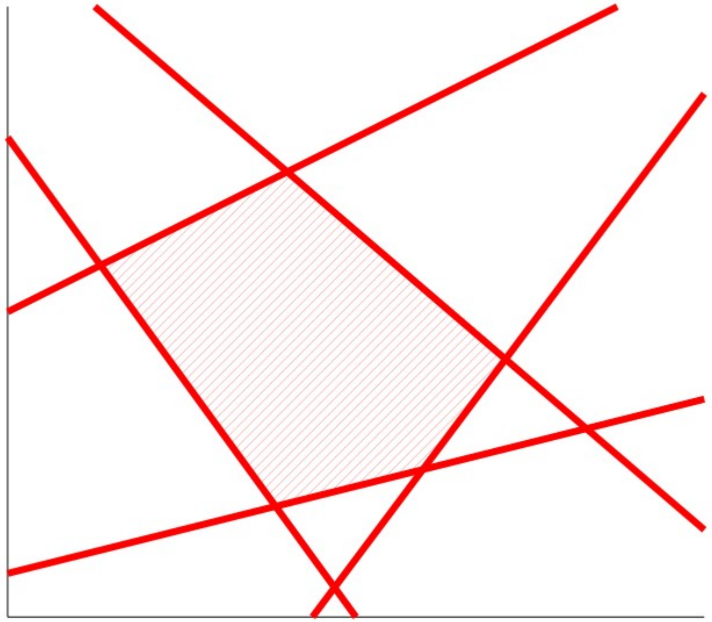




separating hyperplane

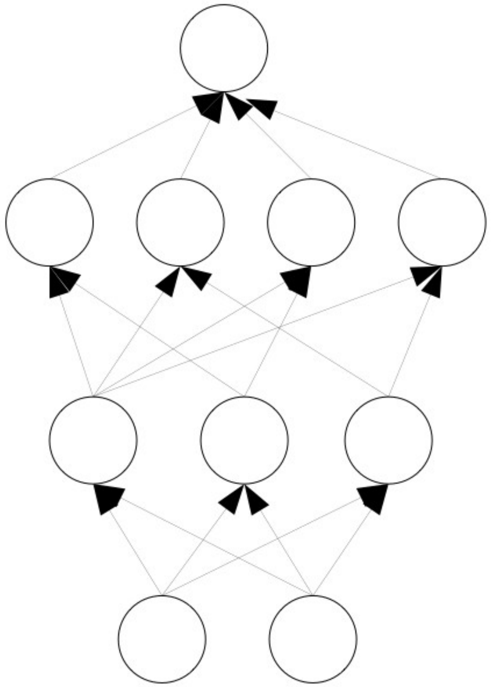
2 layers of
trainable
weights

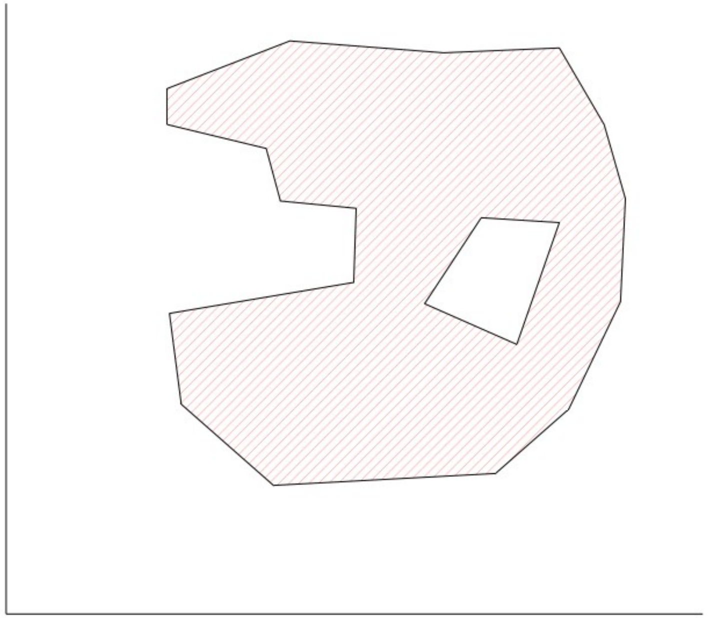




convex polygon region

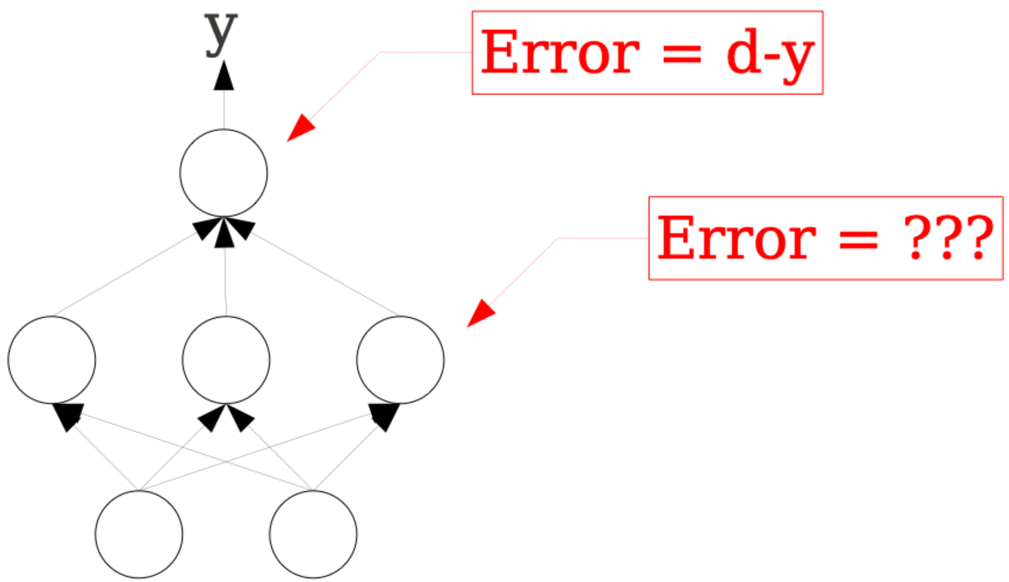
3 layers of trainable weights





composition of polygons:
non convex regions

How Do We Train A Multi-Layer Network?



Can't use perceptron training algorithm because we don't know the 'correct' outputs for hidden units.

How Do We Train A Multi-Layer Network?

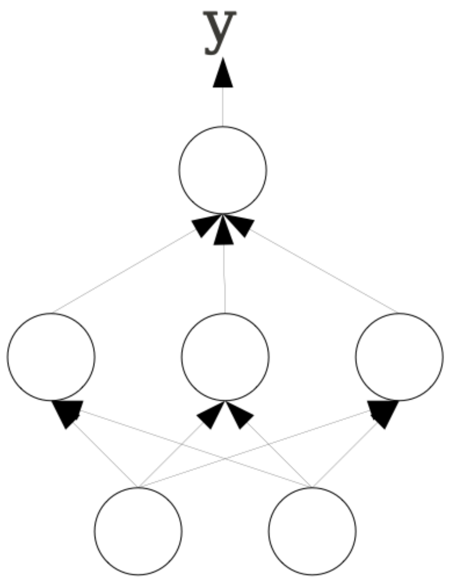
Define sum-squared error:

$$E = \frac{1}{2}\sum_{p}(d^p - y^p)^2$$

Use gradient descent error minimization:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Works if the nonlinear transfer function is differentiable.

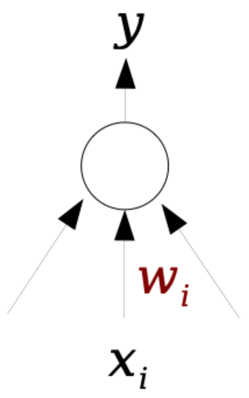


Deriving the LMS or "Delta" Rule As Gradient Descent Learning

$$y = \sum_i w_i x_i$$

$$E = \frac{1}{2}\sum_p (d^p - y^p)^2$$

$$\frac{dE}{dy} = y - d$$



$$\frac{\partial E}{\partial w_i} = \frac{dE}{dy}\cdot\frac{\partial y}{\partial w_i} = (y-d)x_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta(y-d)x_i$$

How do we extend this to two layers?

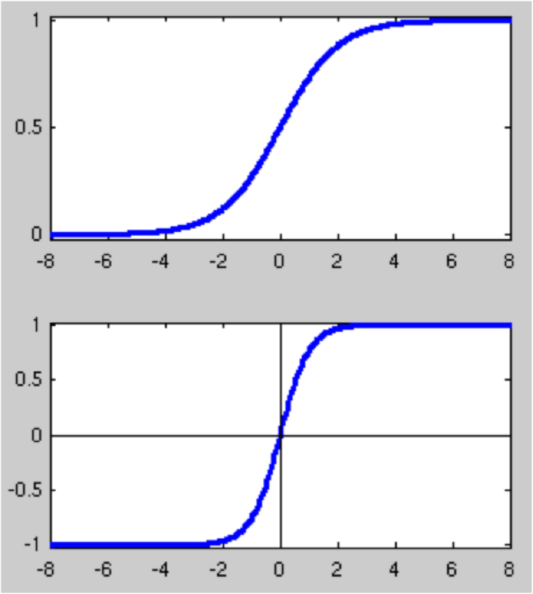
Switch to Smooth Nonlinear Units

$$\text{net}_j = \sum_i w_{ij} y_i$$

$$y_j = \text{g}(\text{net}_j)$$ *g must be differentiable*

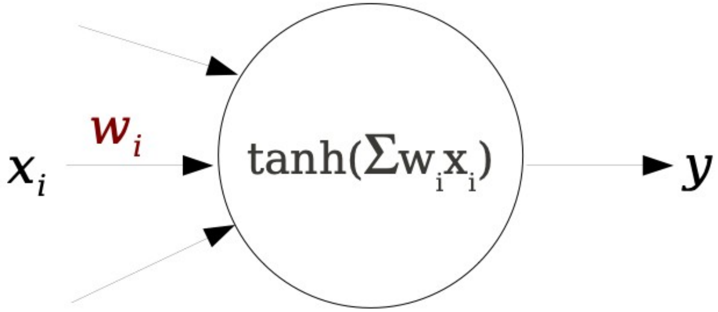
Common choices for g:

$$g(x) = \frac{1}{1+e^{-x}}$$
$$g'(x) = g(x) \cdot (1 - g(x))$$



$$g(x) = \tanh(x)$$
$$g'(x) = 1/\cosh^2(x)$$

Gradient Descent with Nonlinear Units

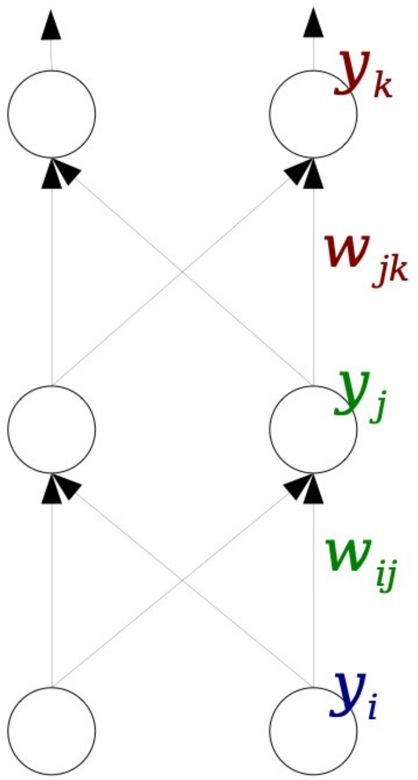


$$y = g(net) = \tanh\left(\sum_i w_i x_i\right)$$

$$\frac{dE}{dy} = (y-d), \qquad \frac{dy}{d\,net} = 1/\cosh^2(net), \qquad \frac{\partial\, net}{\partial w_i} = x_i$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{dE}{dy} \cdot \frac{dy}{d\,net} \cdot \frac{\partial\, net}{\partial w_i} \\ &= (y-d)/\cosh^2\left(\sum_i w_i x_i\right) \cdot x_i \end{aligned}$$

Now We Can Use The Chain Rule



$$\frac{\partial E}{\partial y_k} = (y_k - d_k)$$

$$\delta_k = \frac{\partial E}{\partial net_k} = (y_k - d_k) \cdot g'(net_k)$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = \delta_k \cdot y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_k \left(\frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \right)$$

$$\delta_j = \frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial y_j} \cdot g'(net_j)$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j \cdot y_i$$

Weight Updates

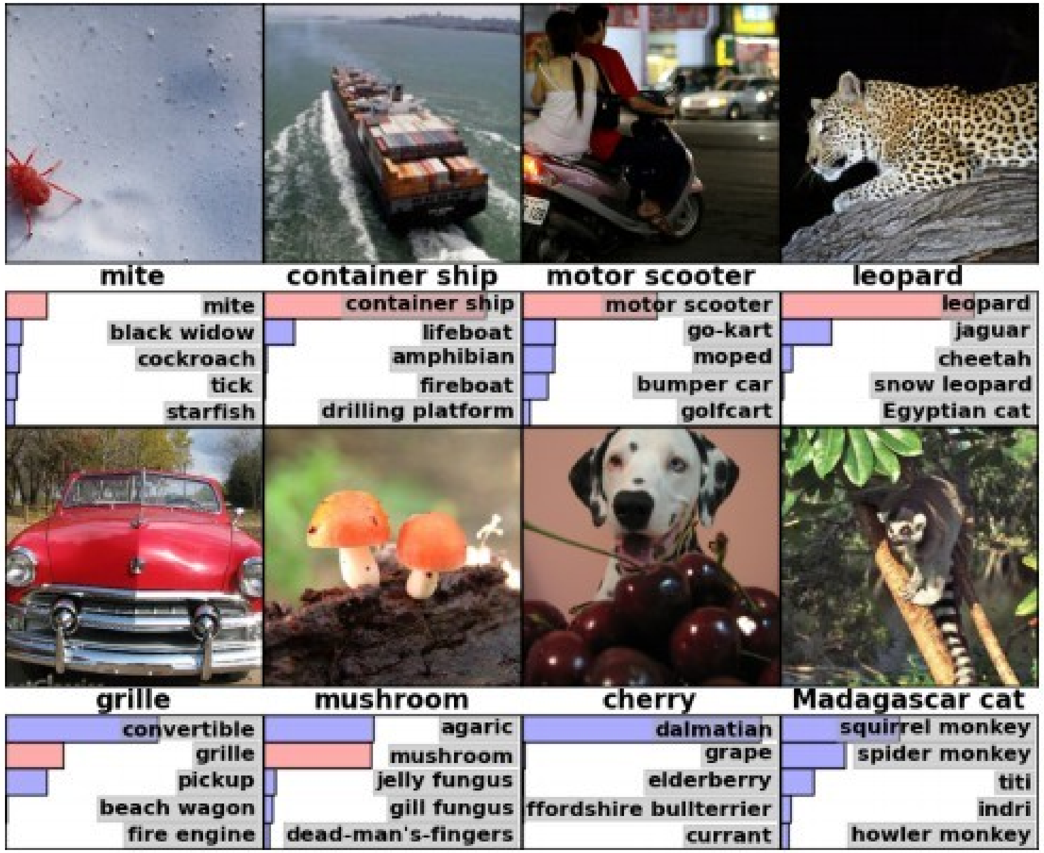
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = \delta_k \cdot y_j$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} = \delta_j \cdot y_i$$

$$\Delta w_{jk} = -\eta \cdot \frac{\partial E}{\partial w_{jk}} \qquad \Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

Deep learning is everywhere

Classification

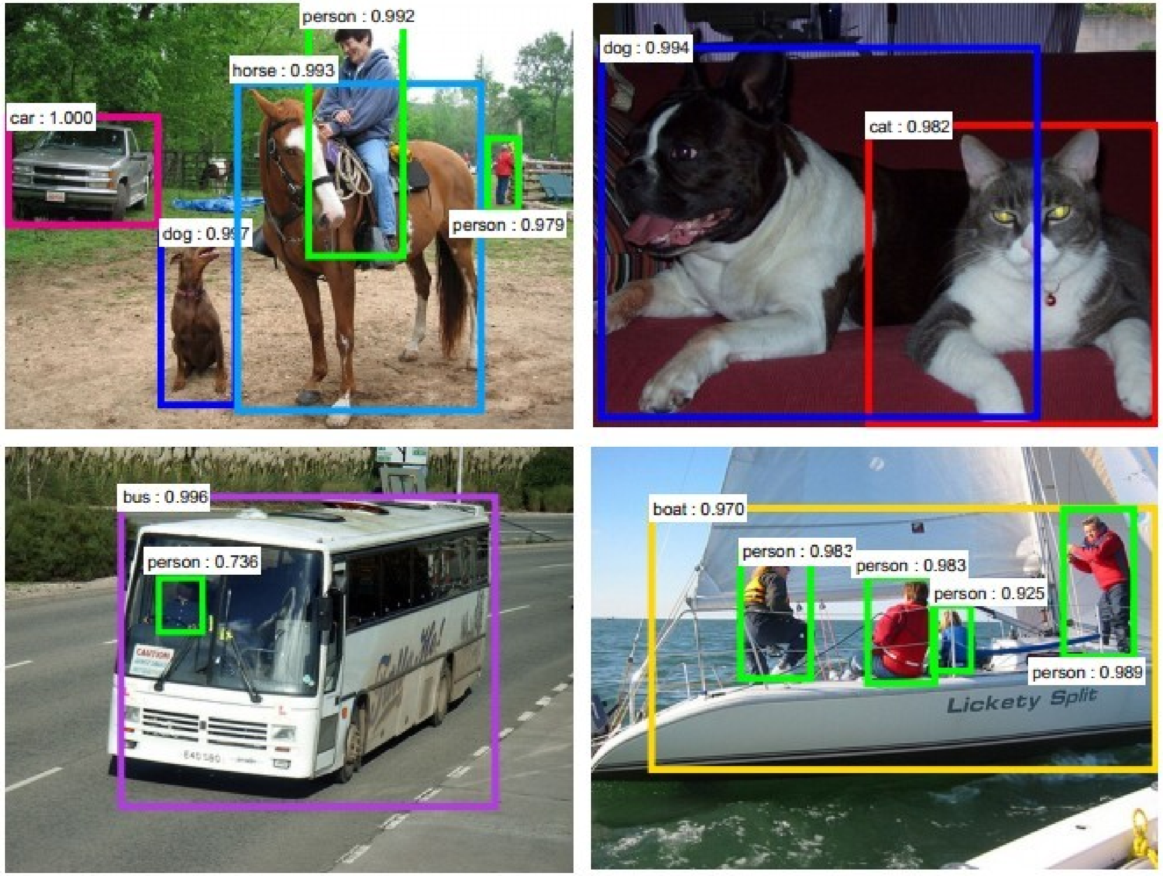


Retrieval

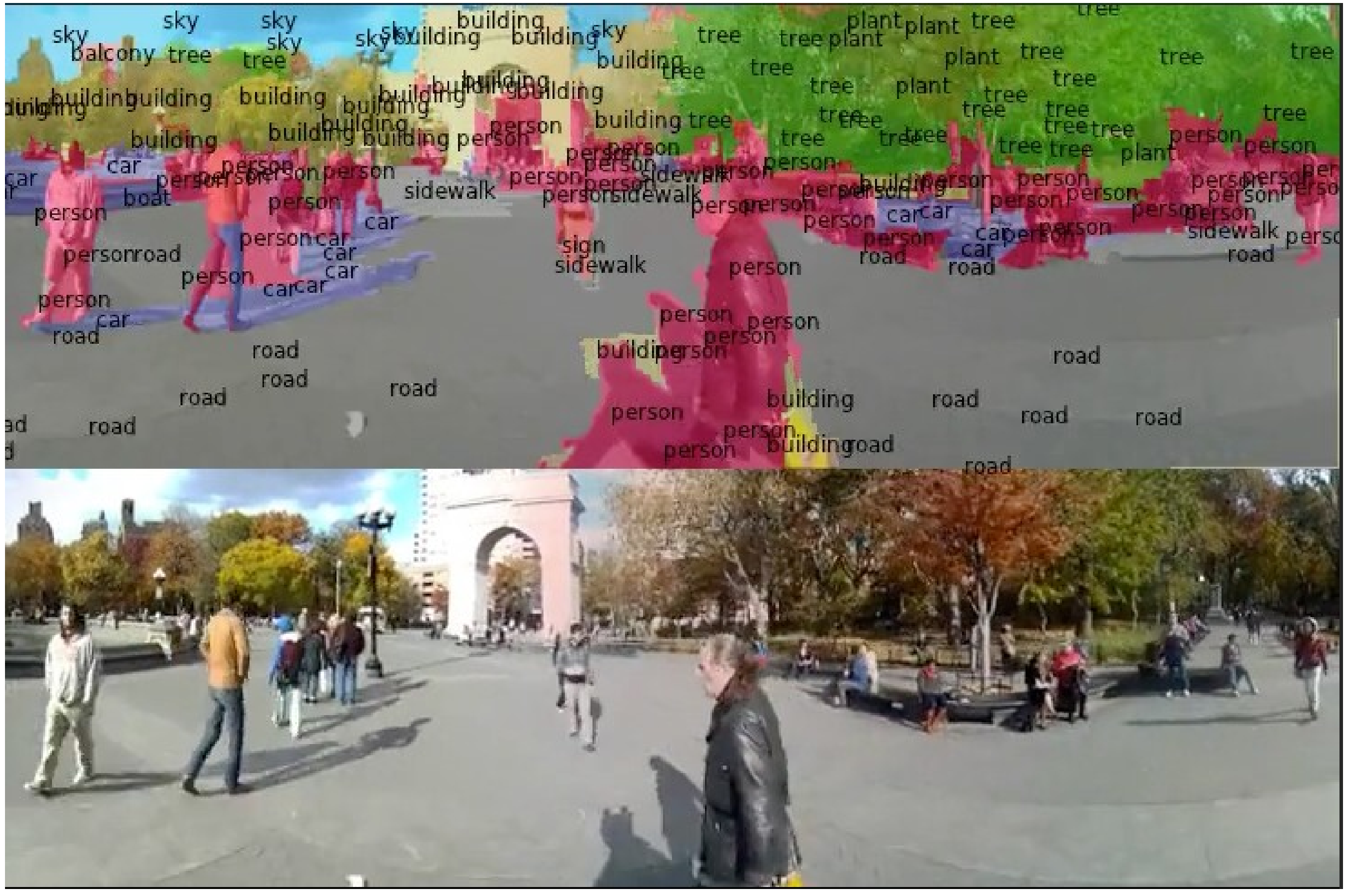
[Krizhevsky 2012]

Deep learning is everywhere

Detection



Segmentation



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

[Farabet et al., 2012]

Deep learning is everywhere

self-driving cars

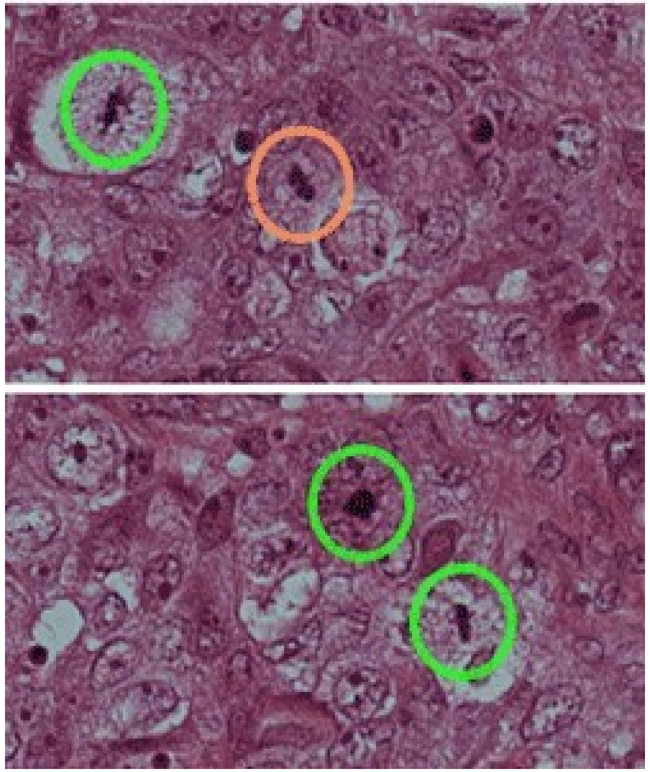
NVIDIA Tegra X1

Deep learning is everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]

Deep learning is everywhere



[Ciresan et al. 2013]

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

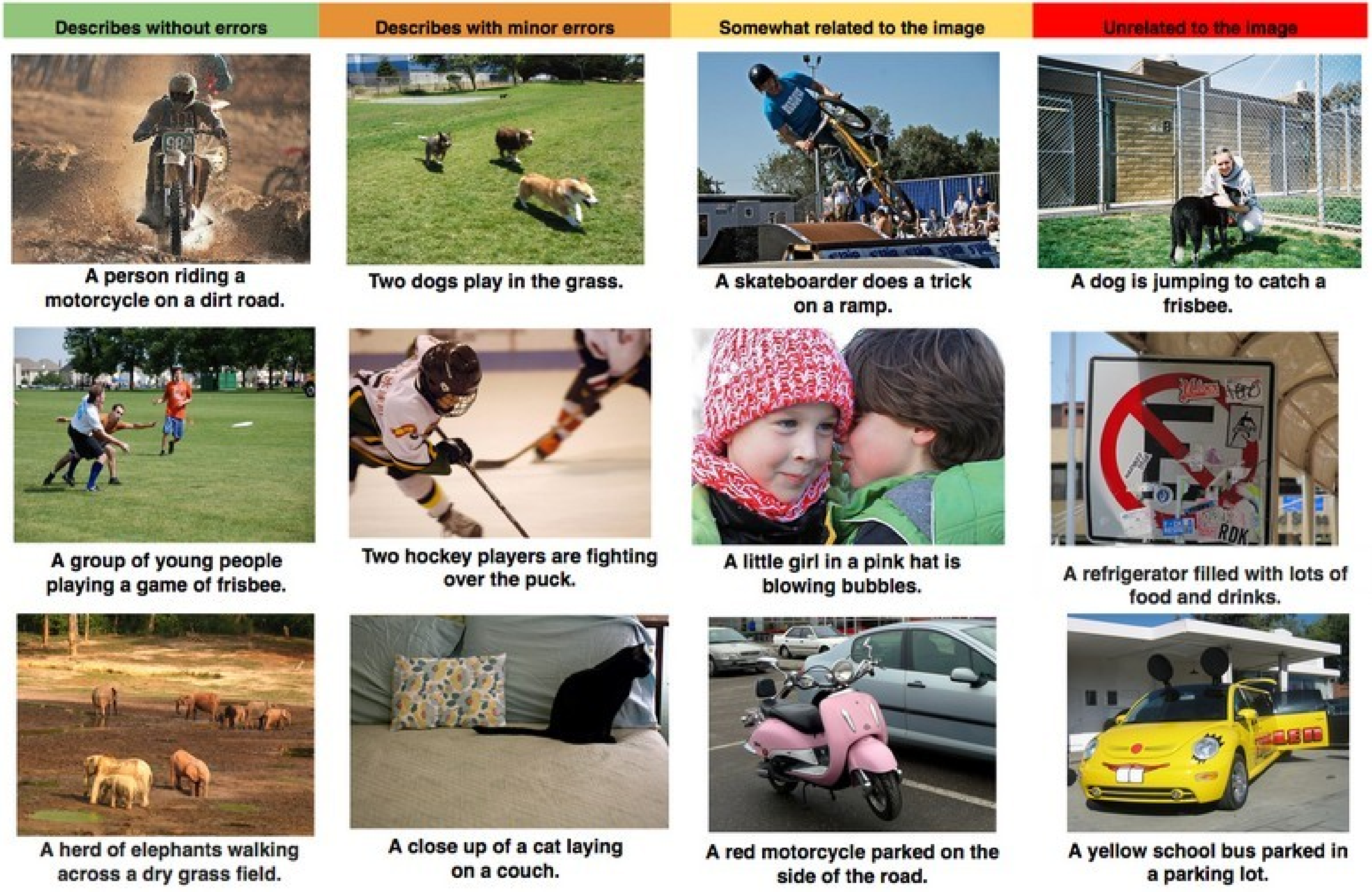
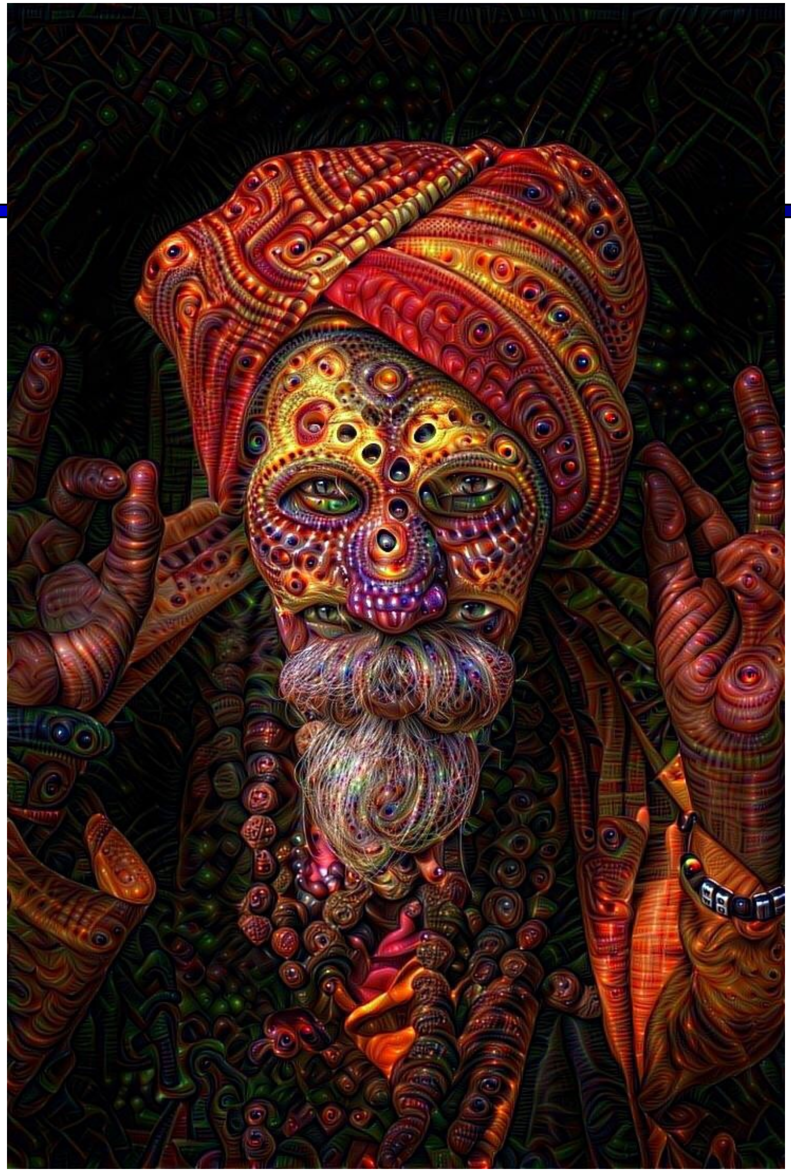
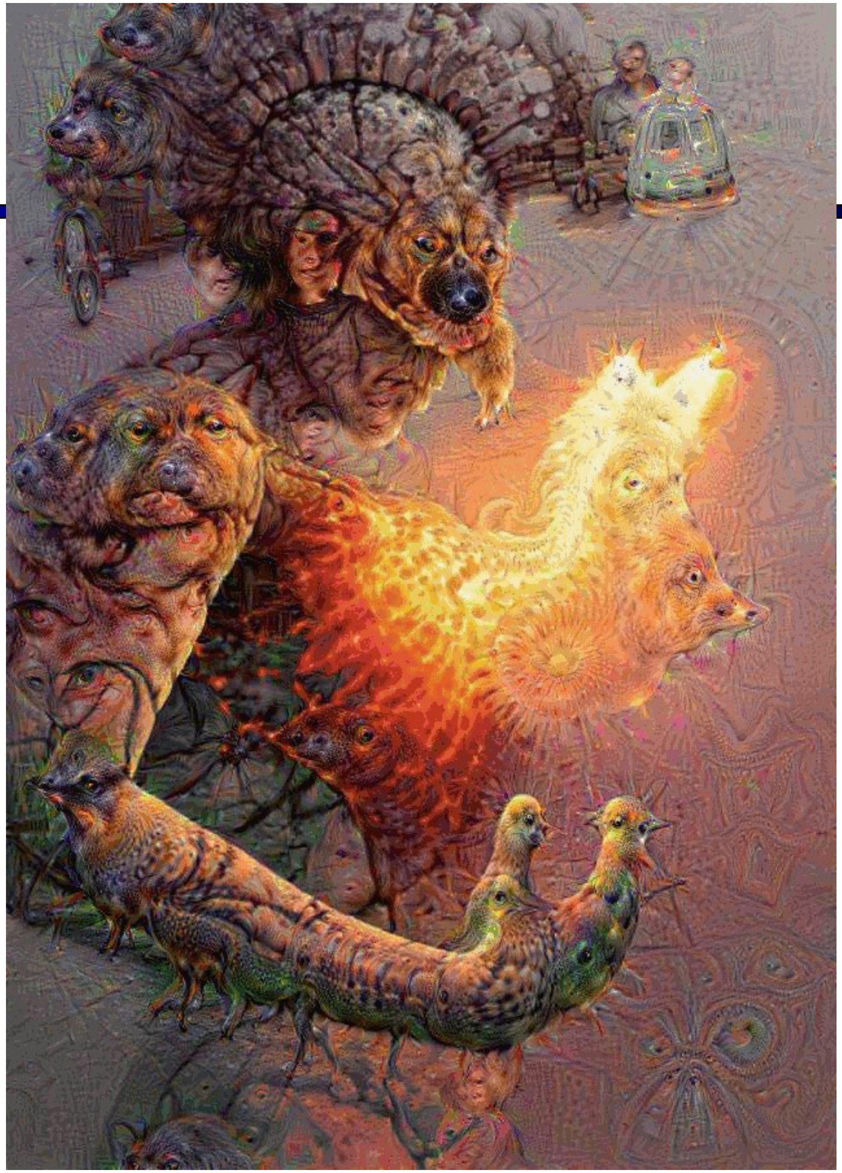


Image Captioning

[Vinyals et al., 2015]

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

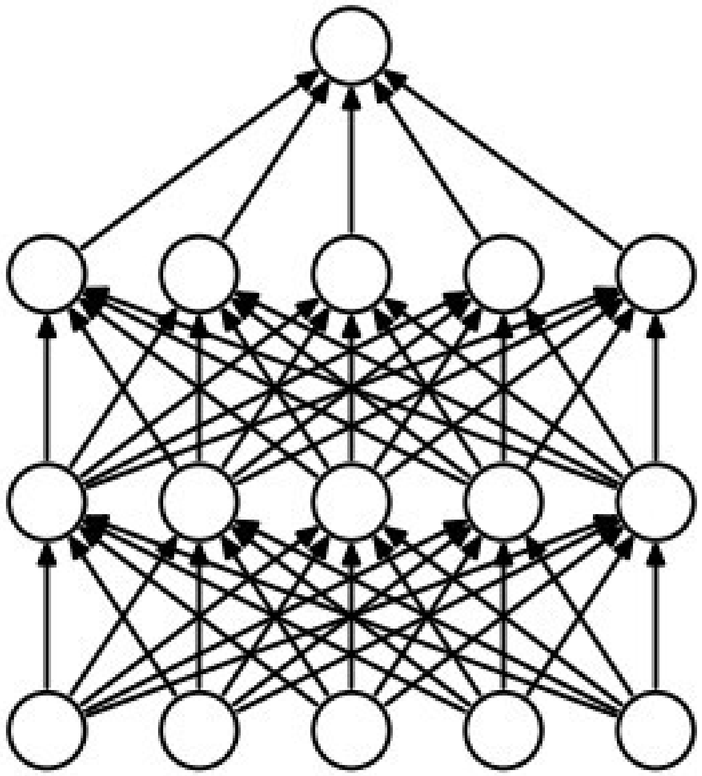
- Mini-batch gradient descent: compute gradient on mini-batch (+ cycle over mini-batches: 1..k, k+1…2k, … ; make sure to randomize permutation of data!)
- Stochastic gradient descent: k = 1

Remaining Pieces

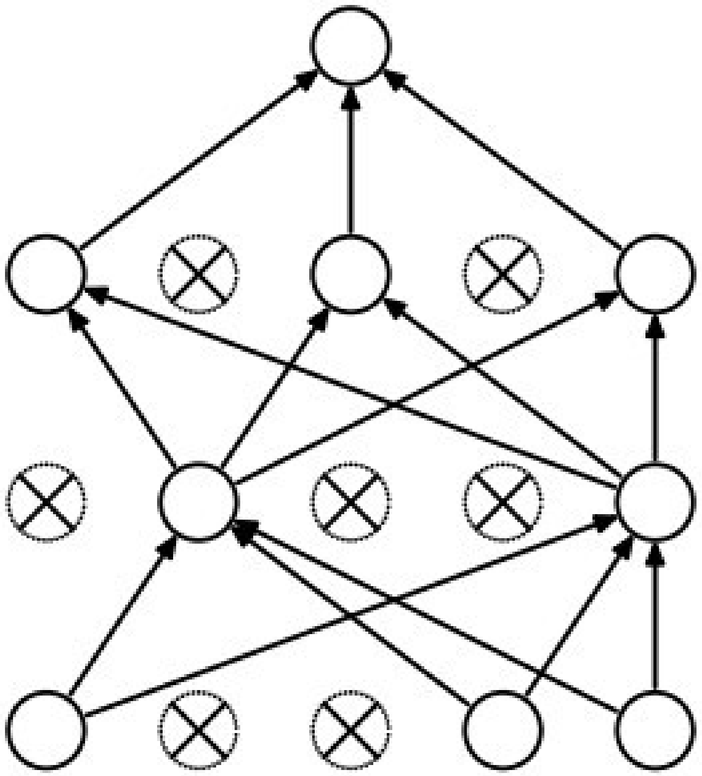
- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient $\nabla g(w)$
  - Gradient checking
  - Backprop

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net

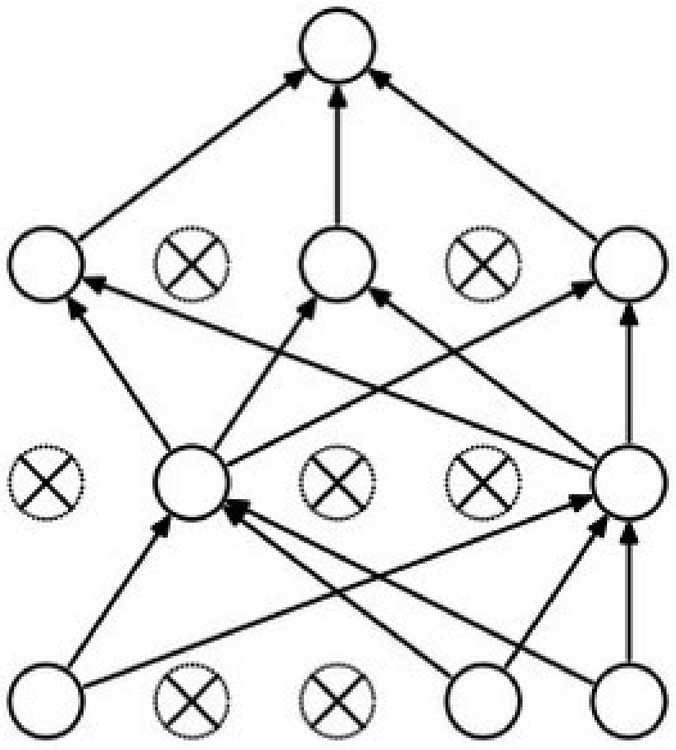


(b) After applying dropout.

[Srivastava et al., 2014]

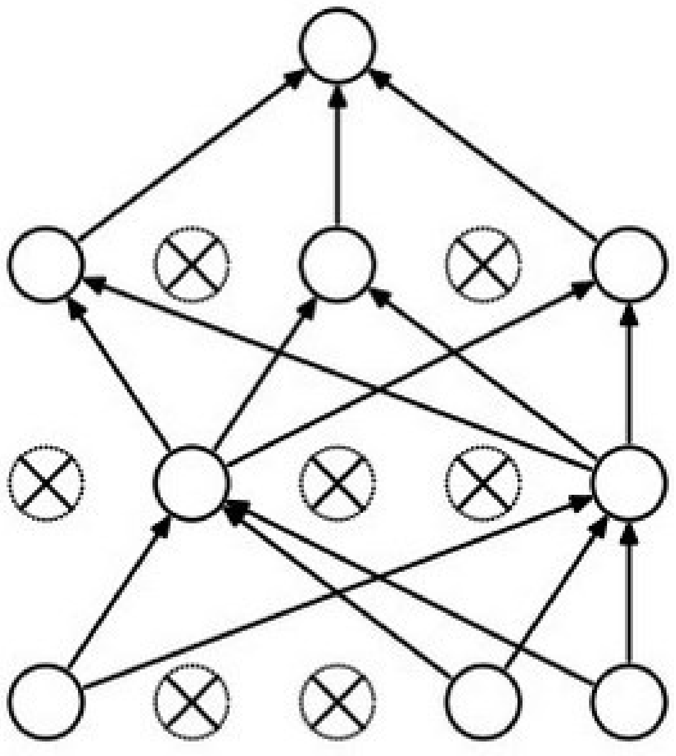
Waaaait a second…

How could this possibly be a good idea?



Waaaait a second…

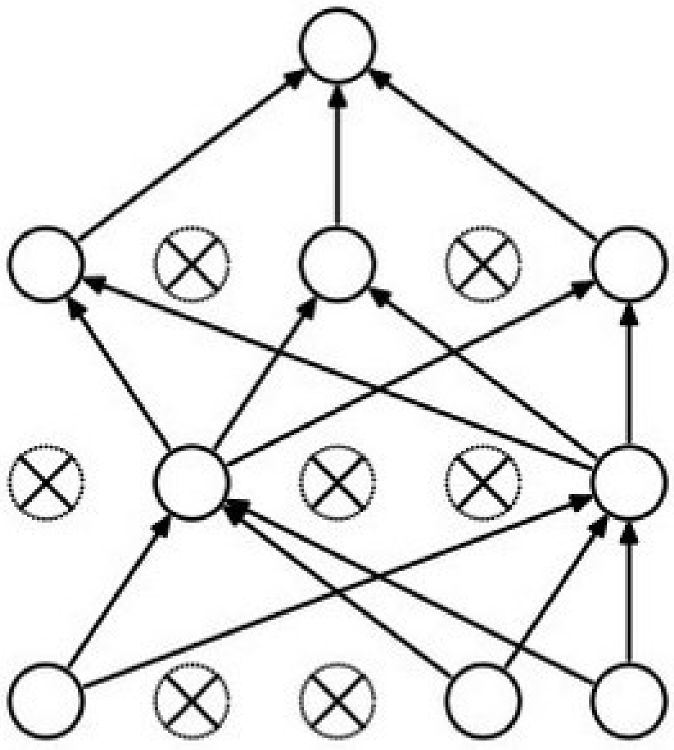
How could this possibly be a good idea?



Forces the network to have a redundant representation.

Waaaait a second…

How could this possibly be a good idea?

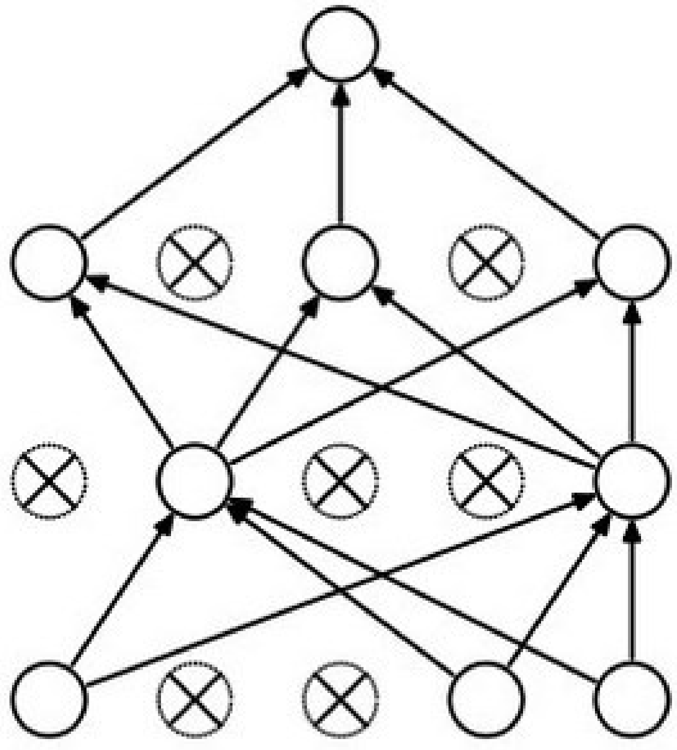


Another interpretation:

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….



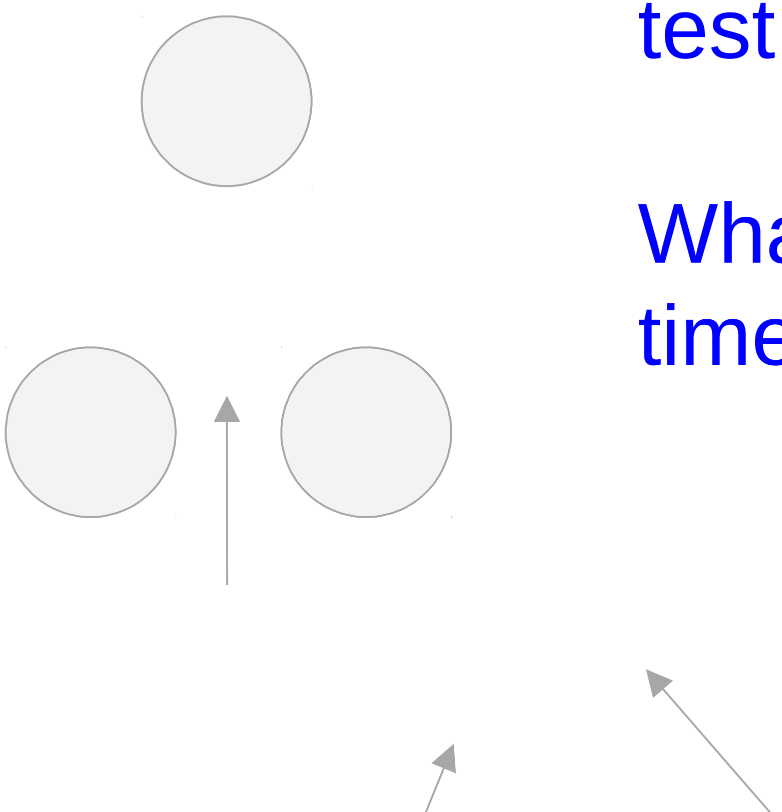
Ideally:
want to integrate out all the noise

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).



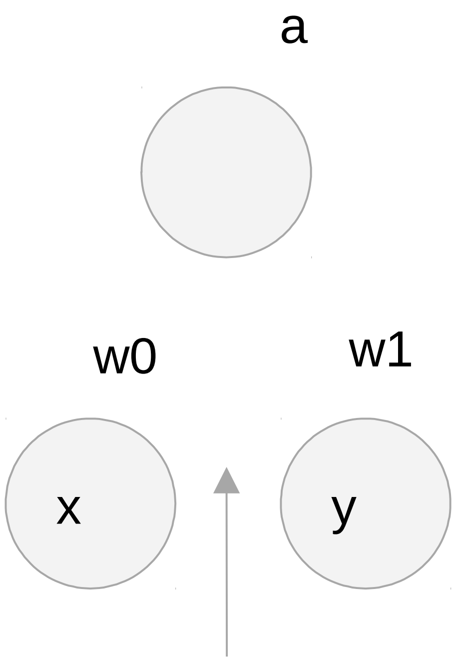
Q: Suppose that with all inputs present at test time the output of this neuron is x.

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: **a = w0*x + w1*y**

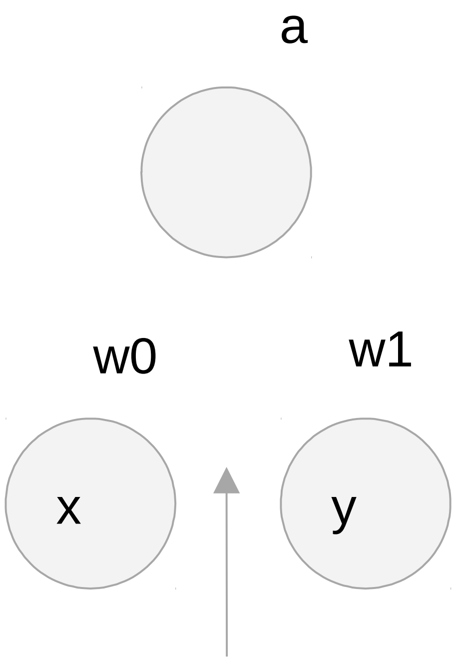
during train:

E[a] = ¼ * (w0*0 + w1*0
w0*0 + w1*y
w0*x + w1*0
w0*x + w1*y)
= ¼ * (2 w0*x + 2 w1*y)
= ½ * (w0*x + w1*y)

At test time….

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

Leave all input neurons turned on (no dropout).



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

during train:

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

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 ½

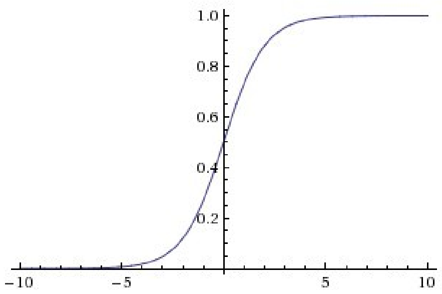
Remaining Pieces

- 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

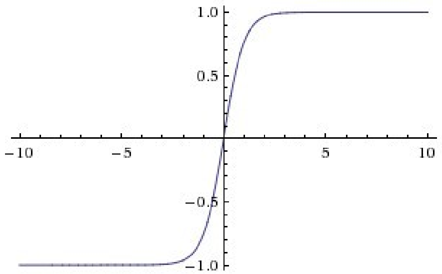
Activation Functions

Sigmoid

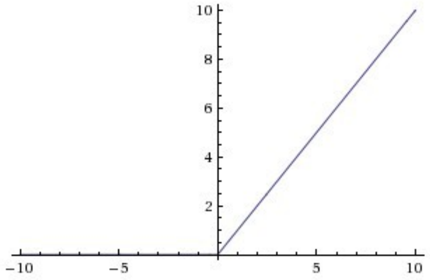
$$\sigma(x) = 1/(1+e^{-x})$$



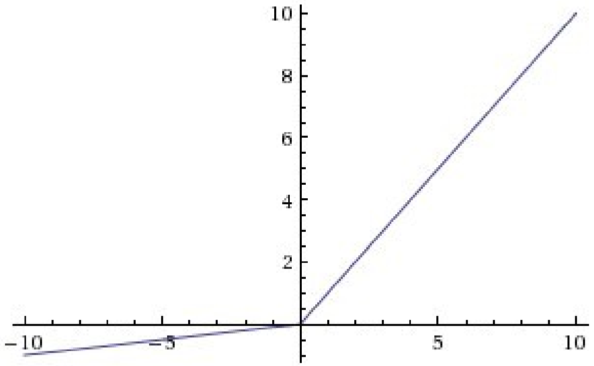
tanh tanh(x)



ReLU max(0,x)



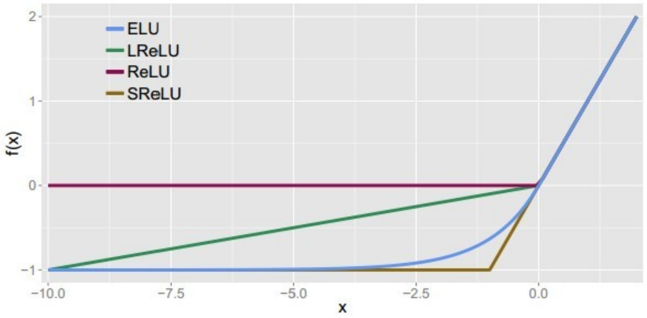
Leaky ReLU
max(0.1x, x)



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

ELU

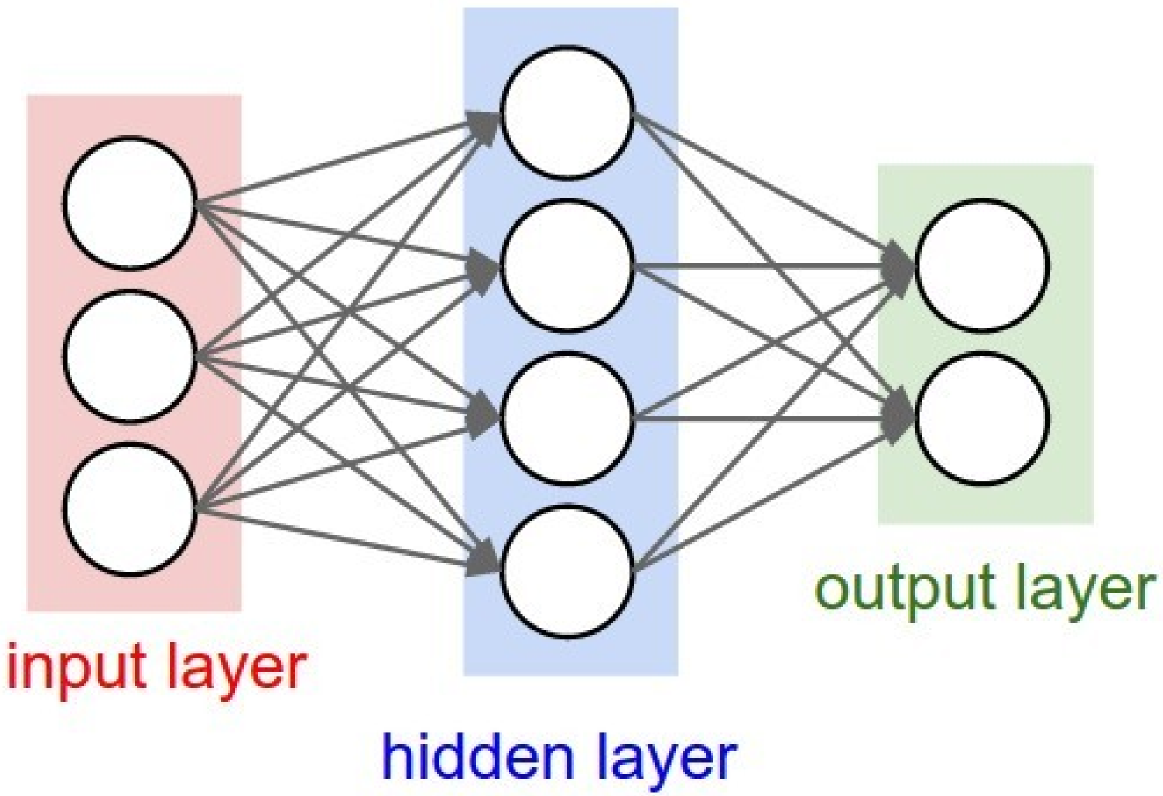
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



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

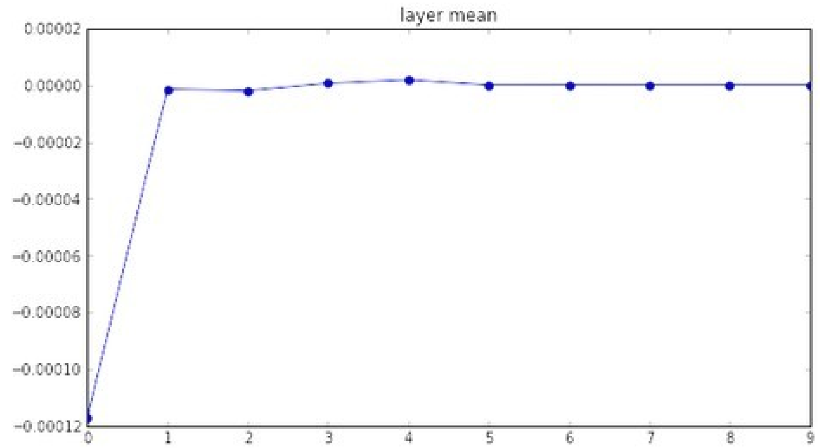
```python
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

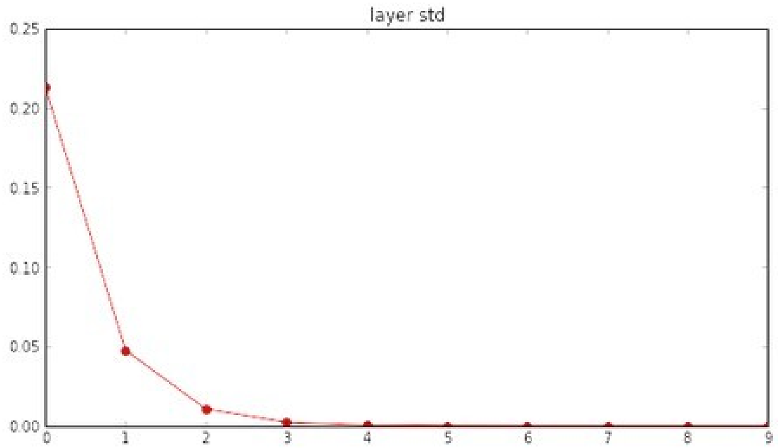
```python
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

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

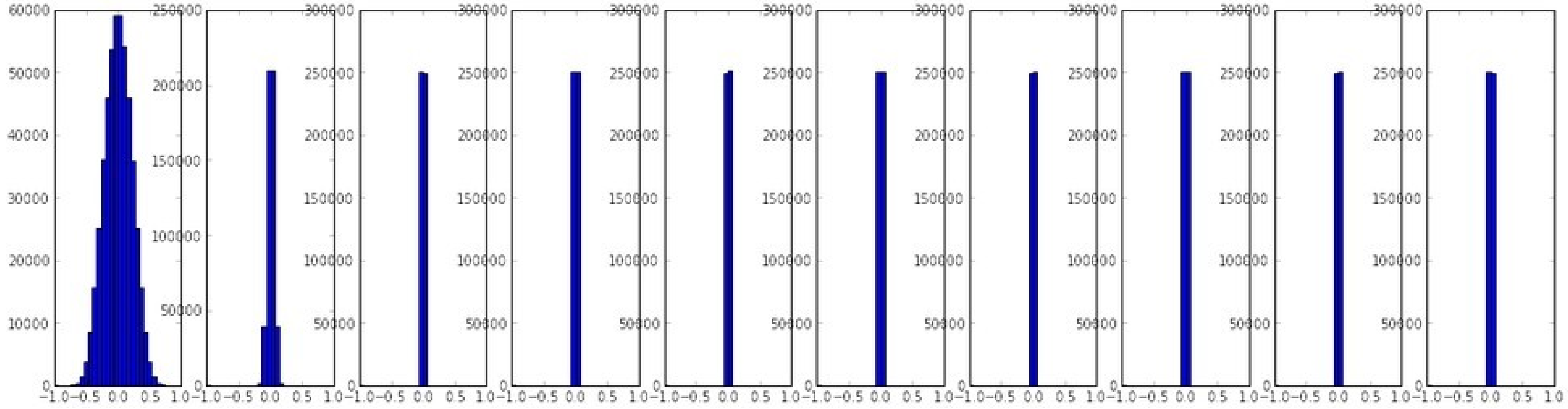
```python
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' %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

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

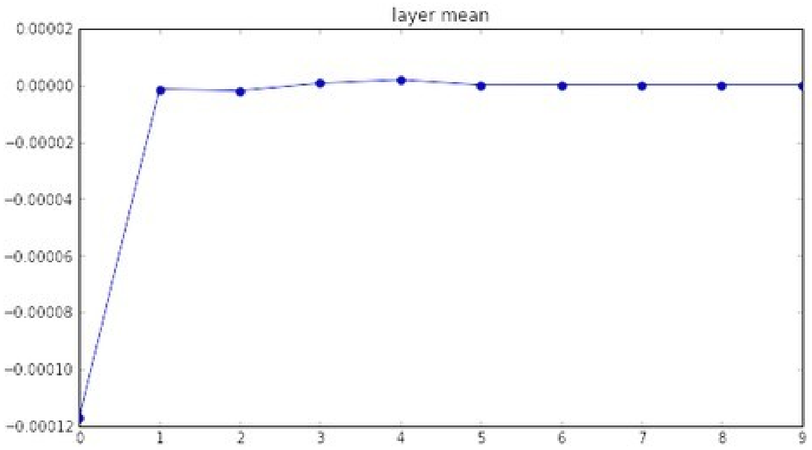
```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
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
```

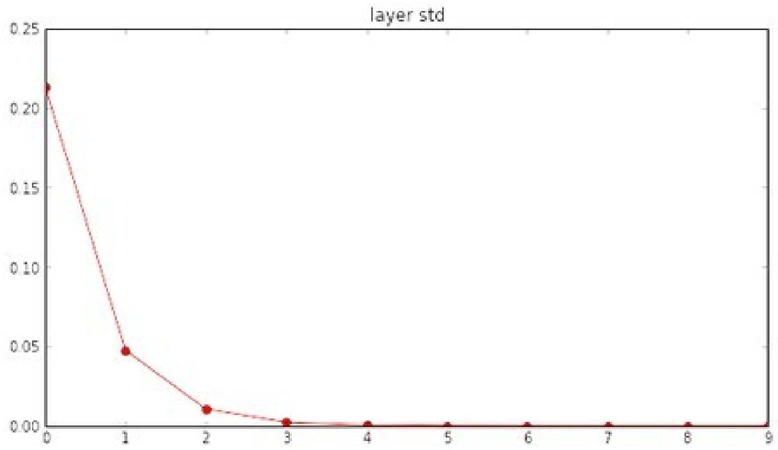


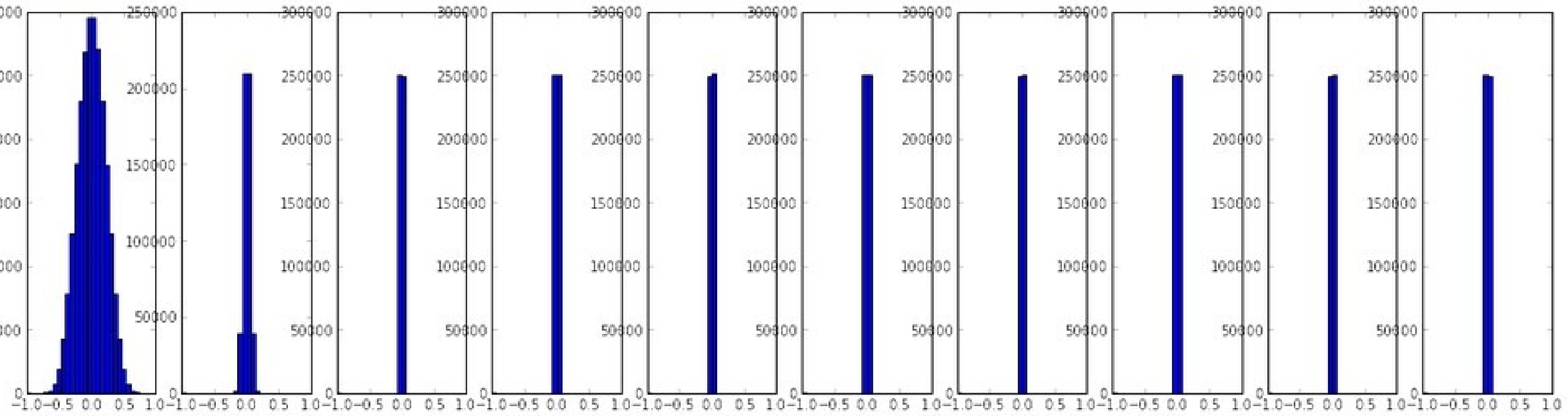




```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
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
```







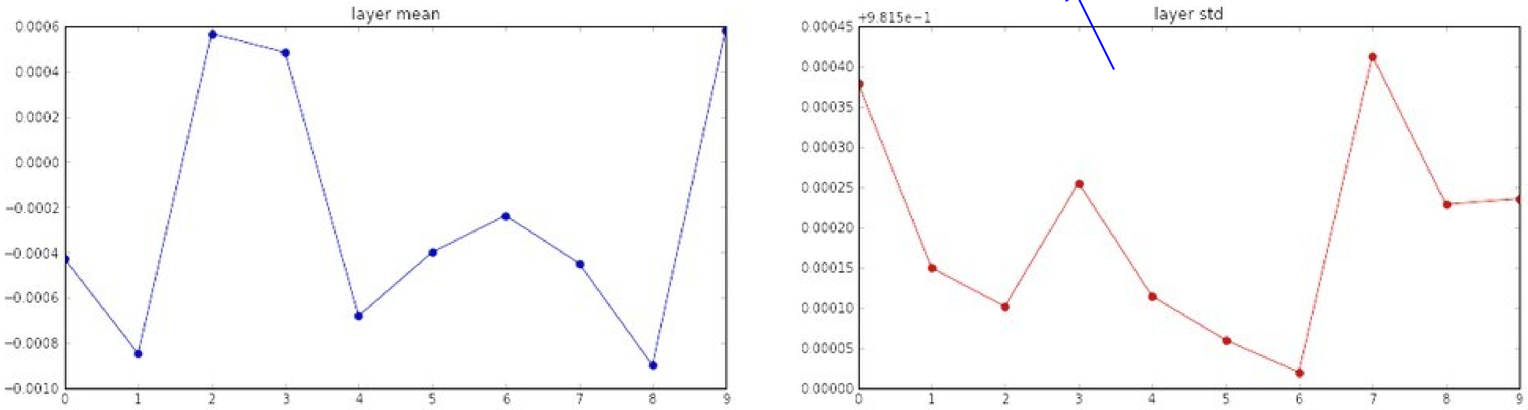
All activations become zero!

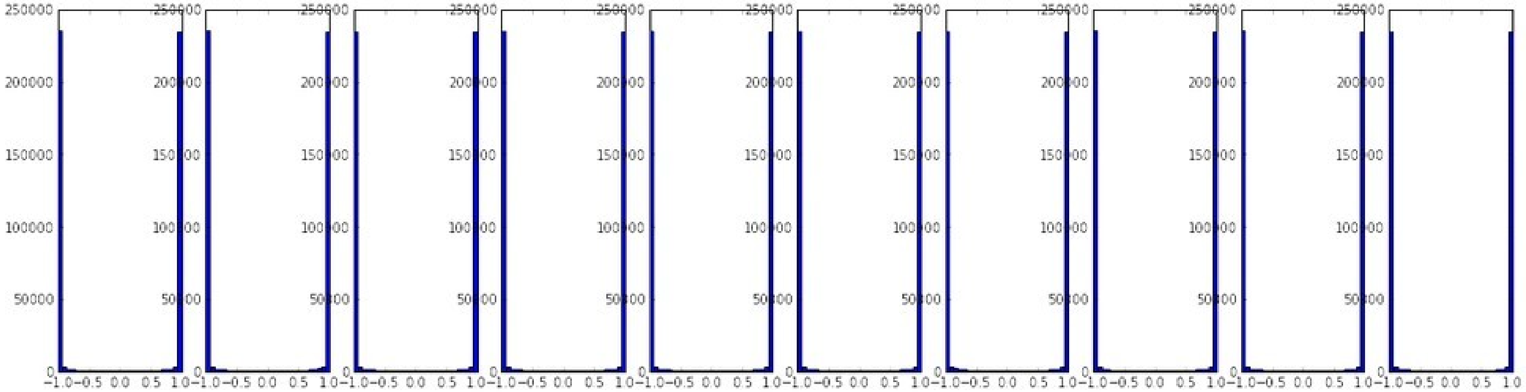
Q: What do the gradients look like?

```
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



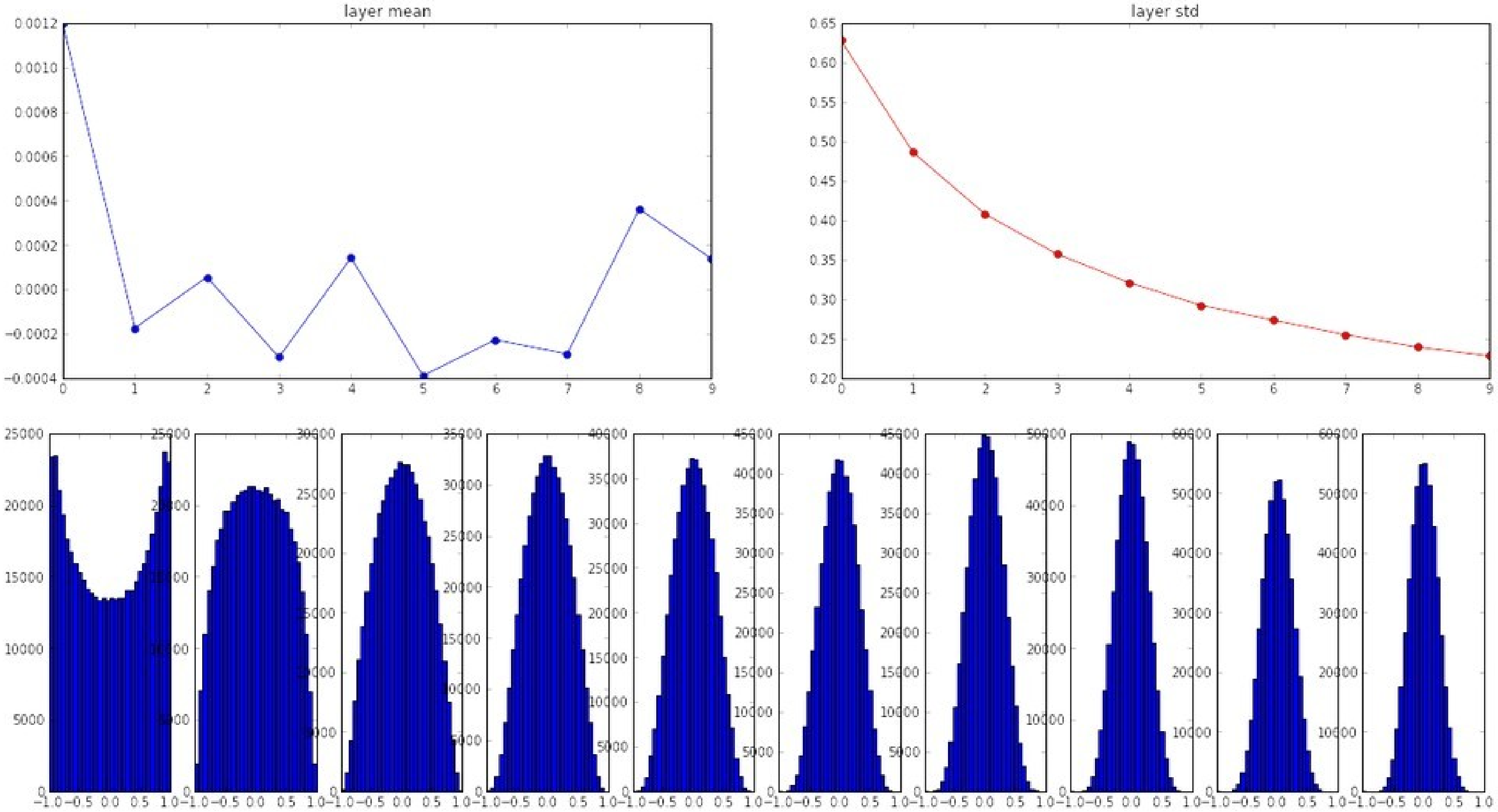


Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

```
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
```

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

"Xavier initialization"
[Glorot et al., 2010]

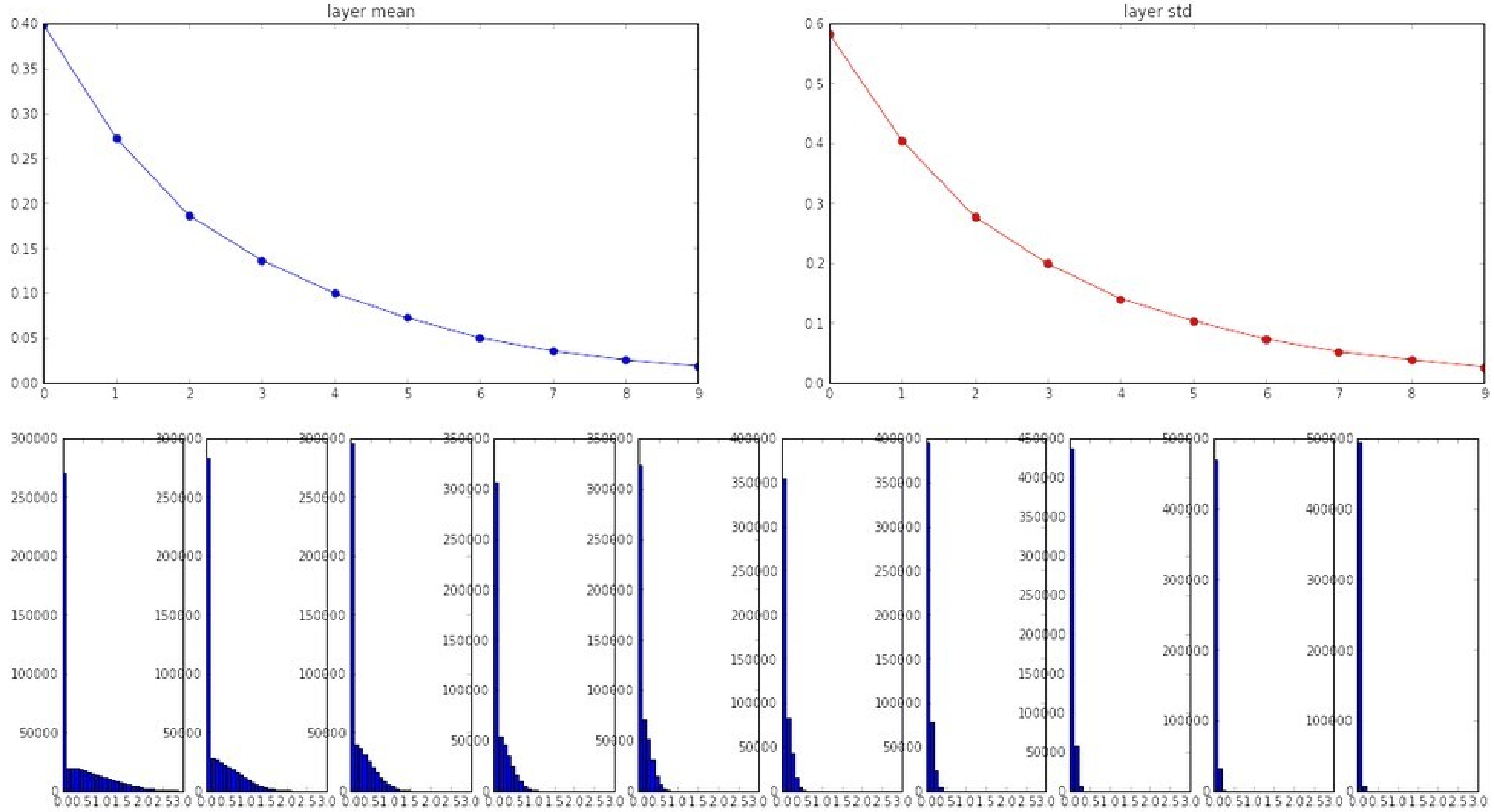


Reasonable initialization.
(Mathematical derivation assumes linear activations)

```
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
```

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

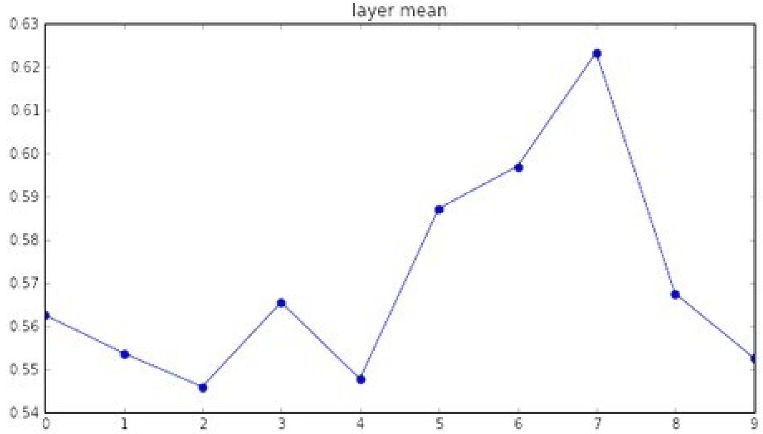
but when using the ReLU nonlinearity it breaks.

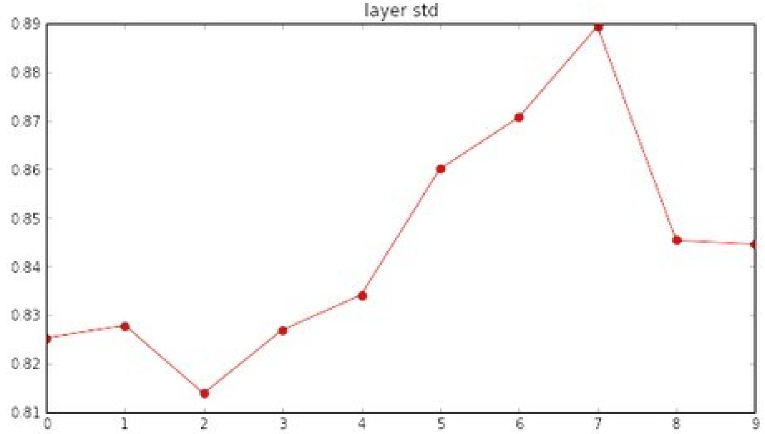


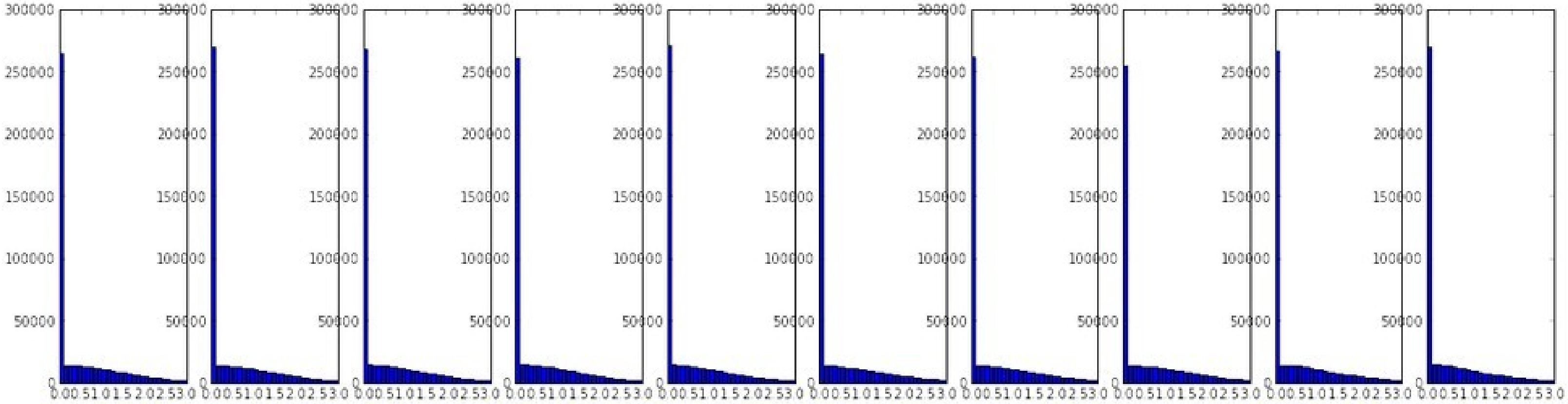
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
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)



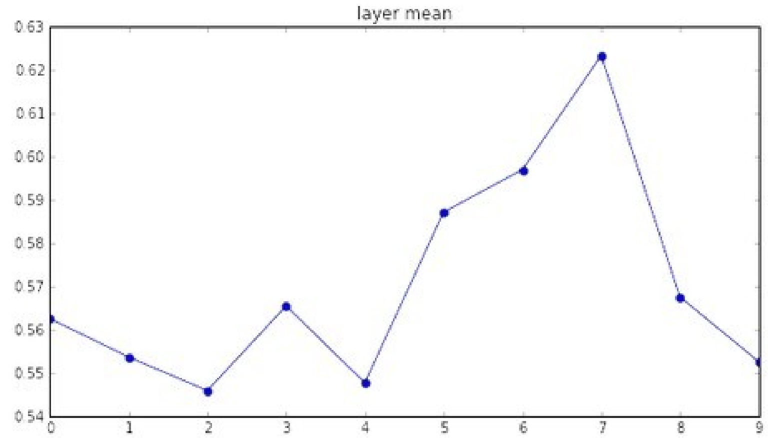


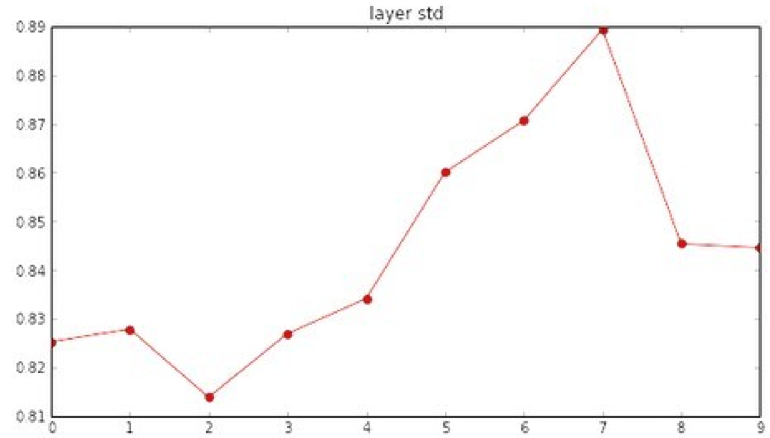


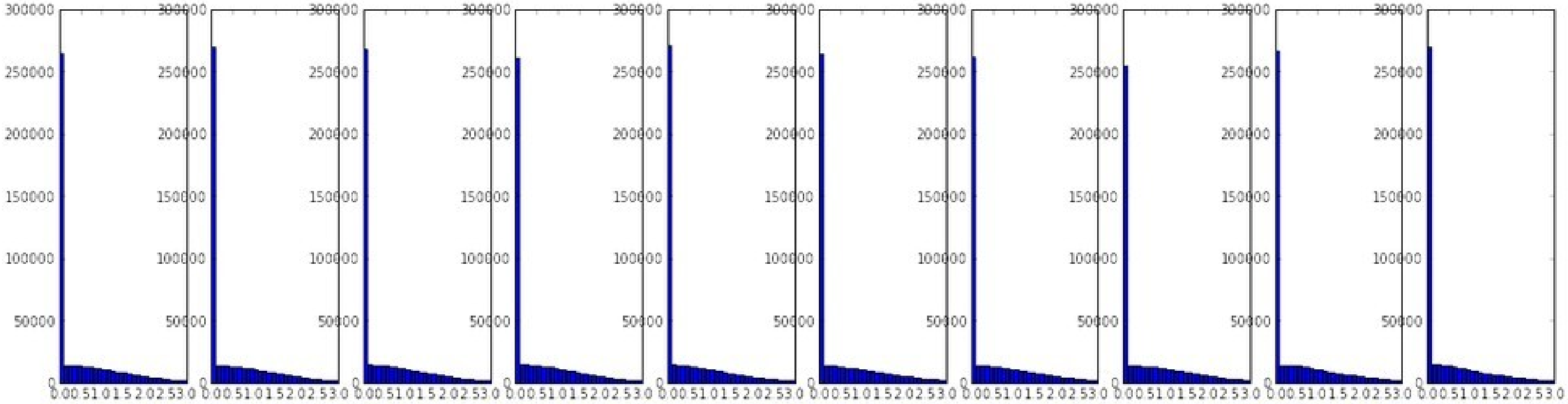
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
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
```

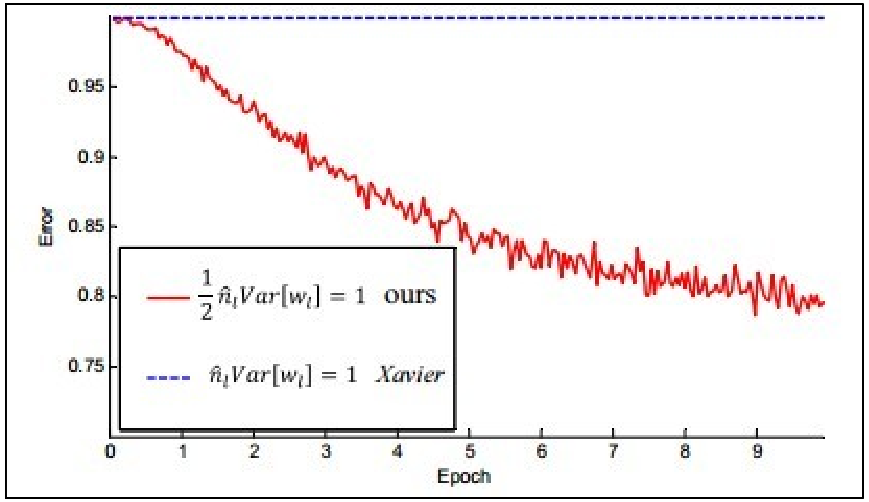
```python
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

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:

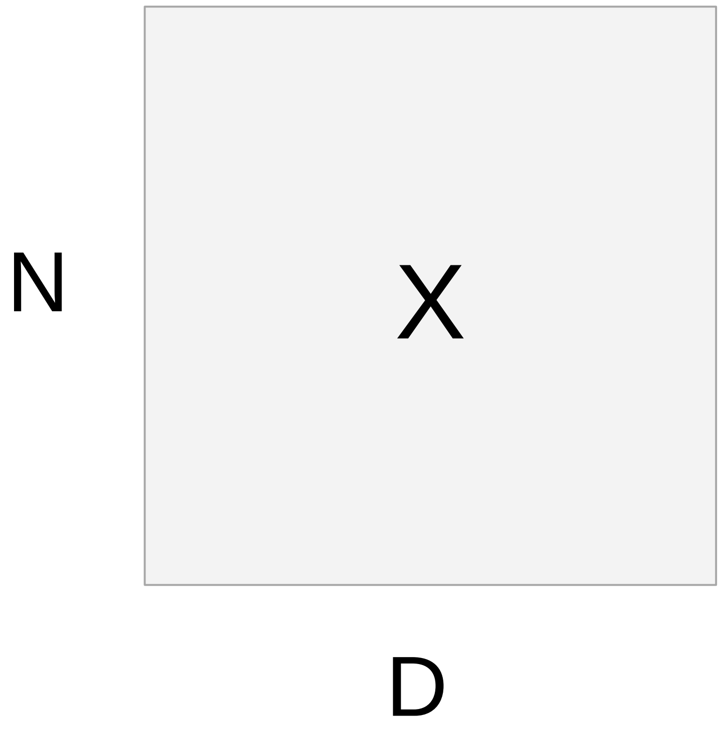
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla
differentiable function...

Batch Normalization

[Ioffe and Szegedy, 2015]

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



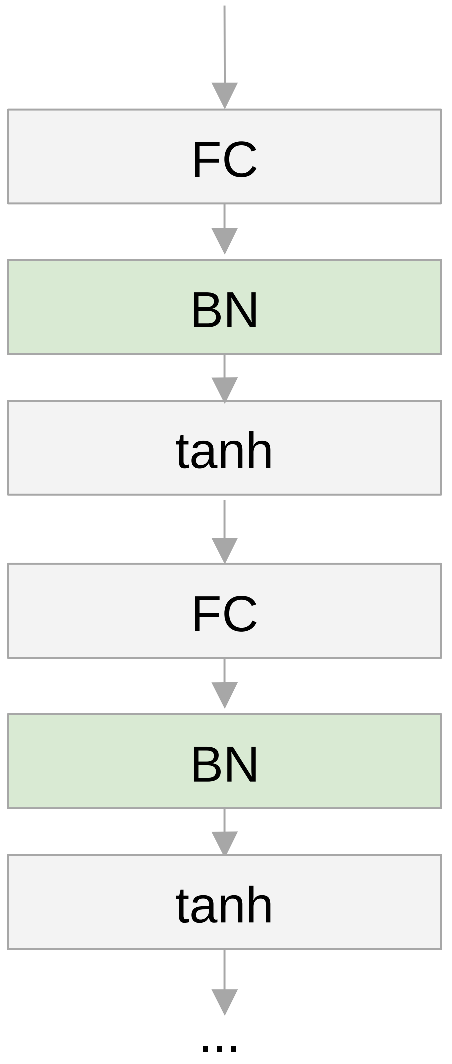
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

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

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?

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

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

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

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

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

Note, the network can learn:

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

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

to recover the identity mapping.

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

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

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

- 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: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

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

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

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

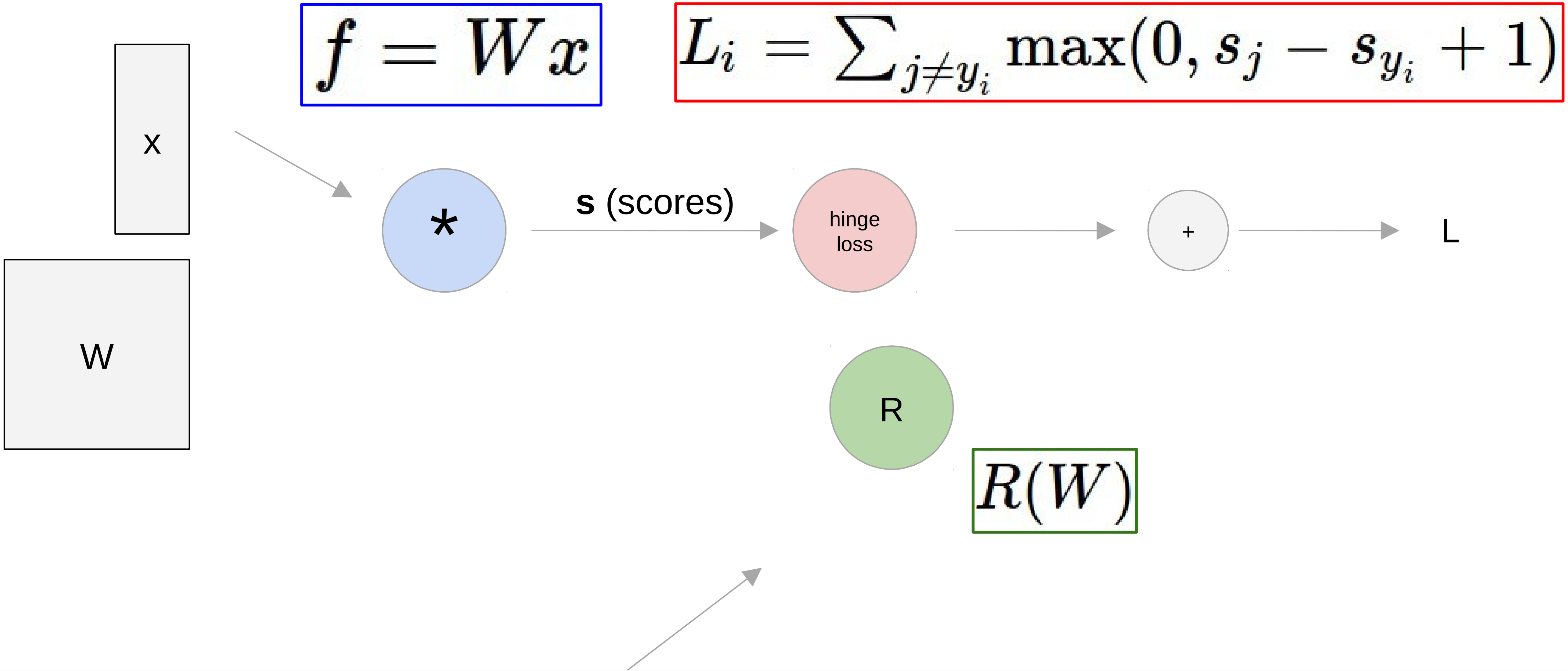
Gradient Descent

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

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

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

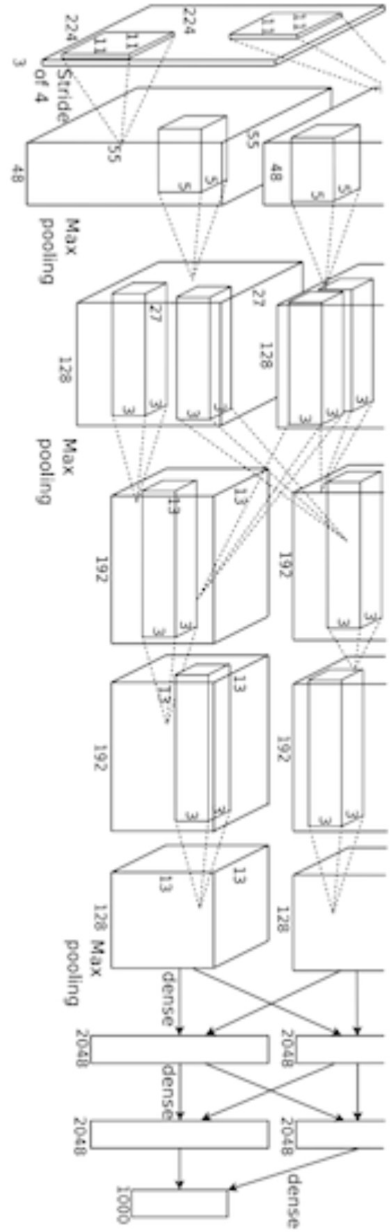
Computational Graph



Convolutional Network (AlexNet)

input image
weights

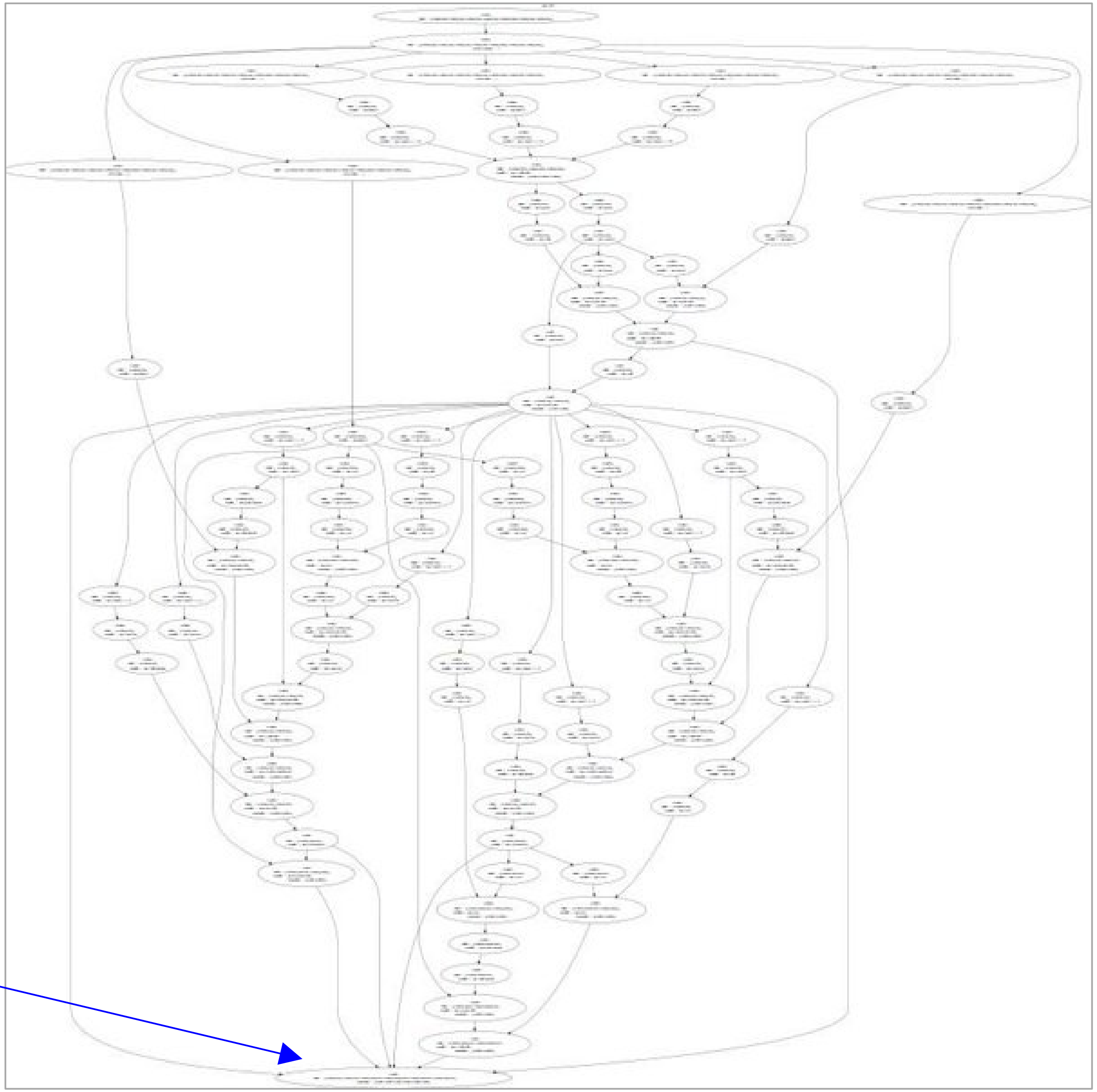
loss



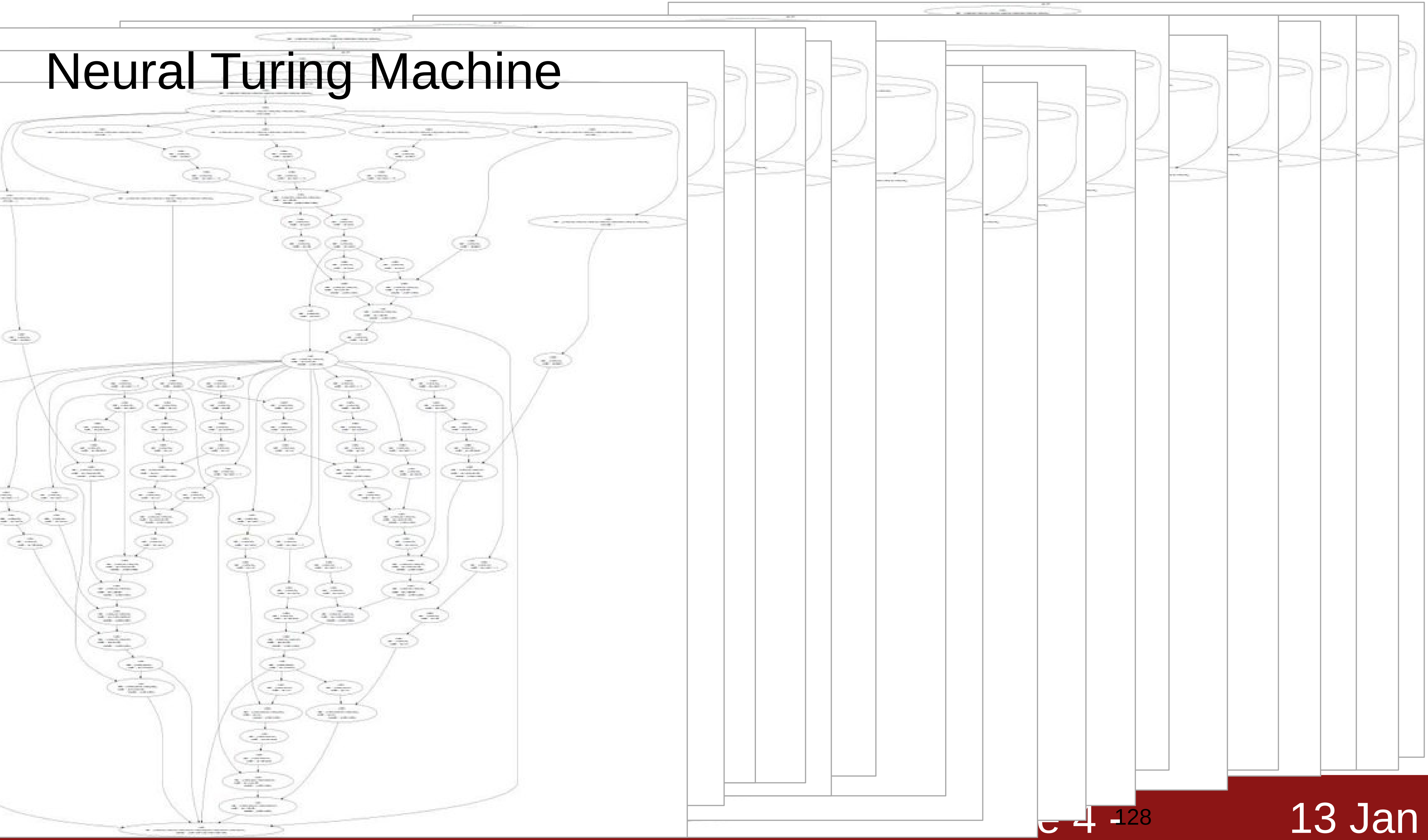
Neural Turing Machine

input tape

loss

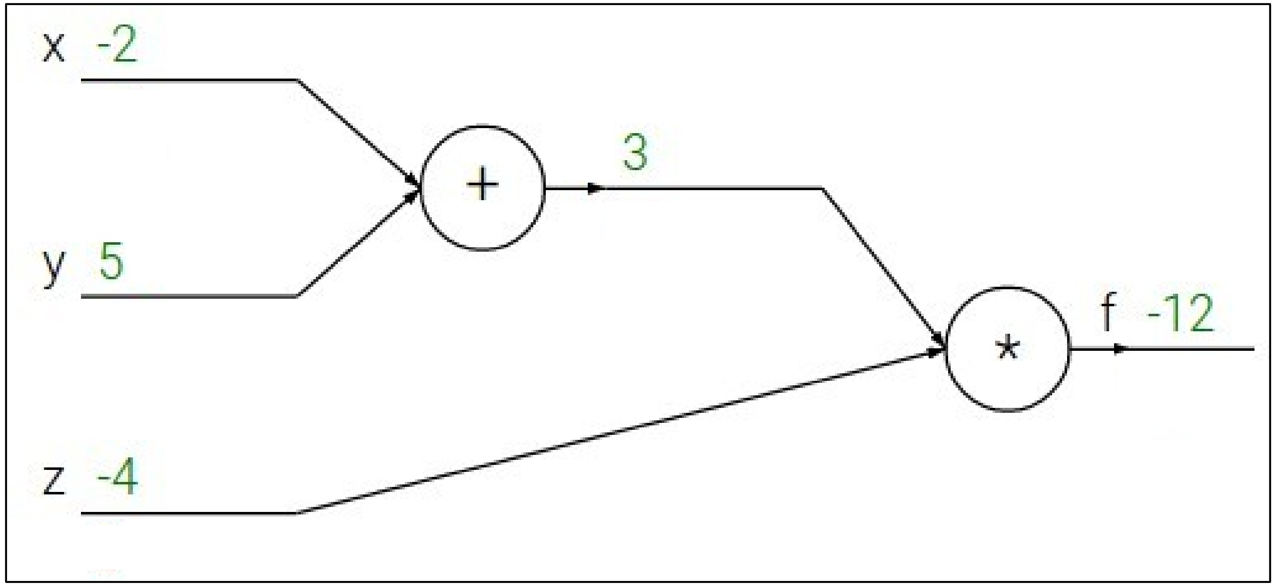


Neural Turing Machine



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

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



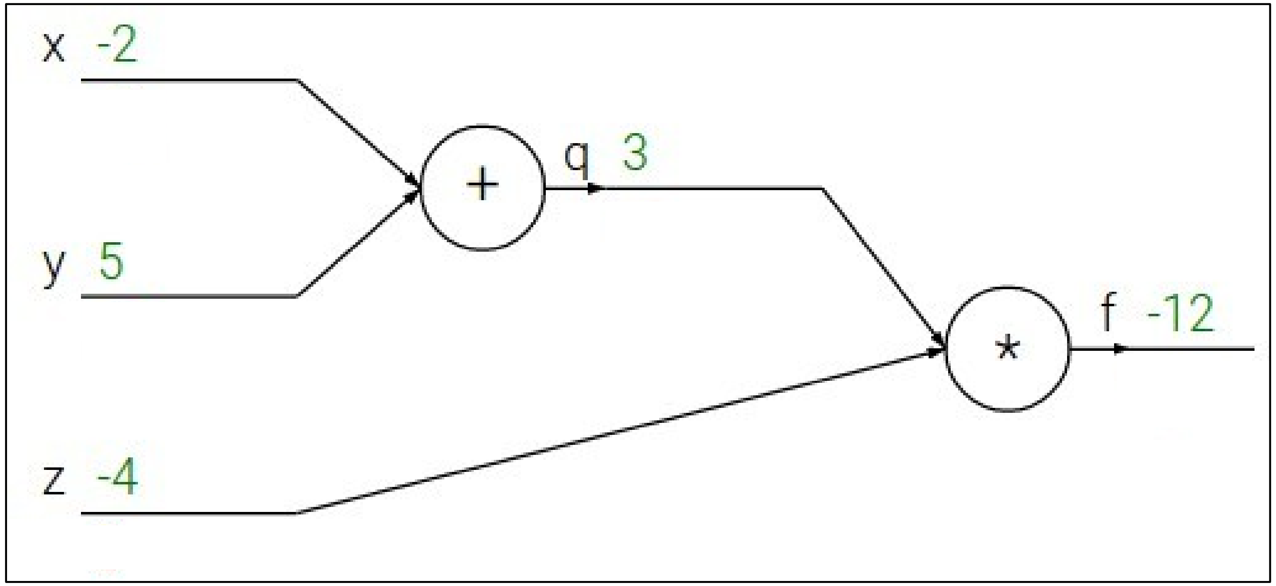
$$f(x,y,z) = (x+y)z$$

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

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

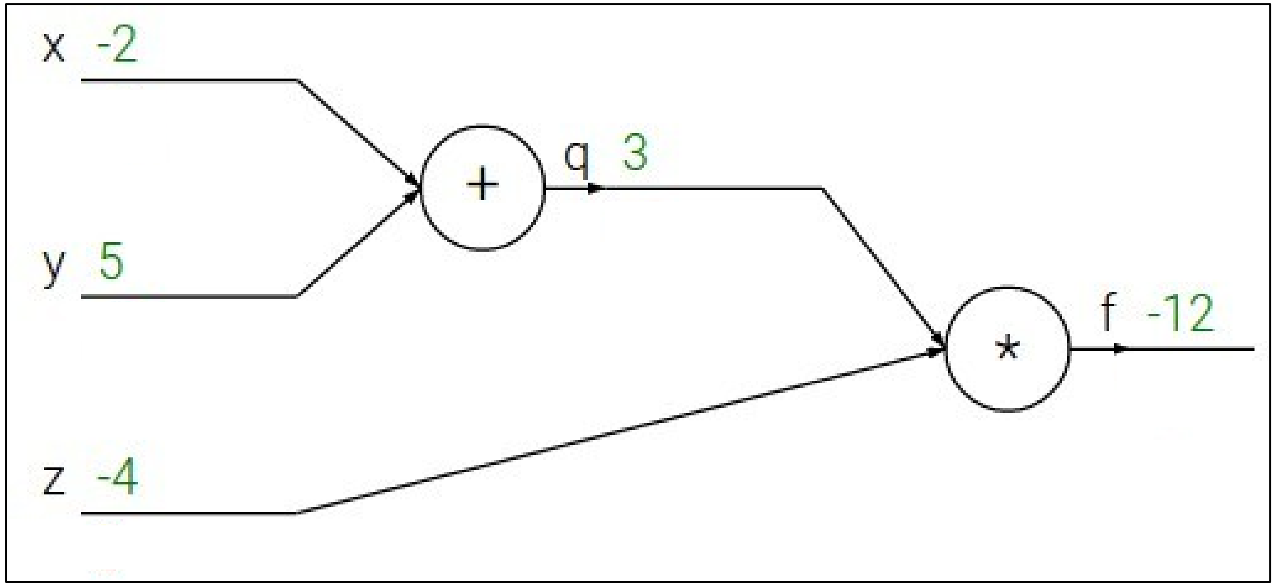
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

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



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

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

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

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

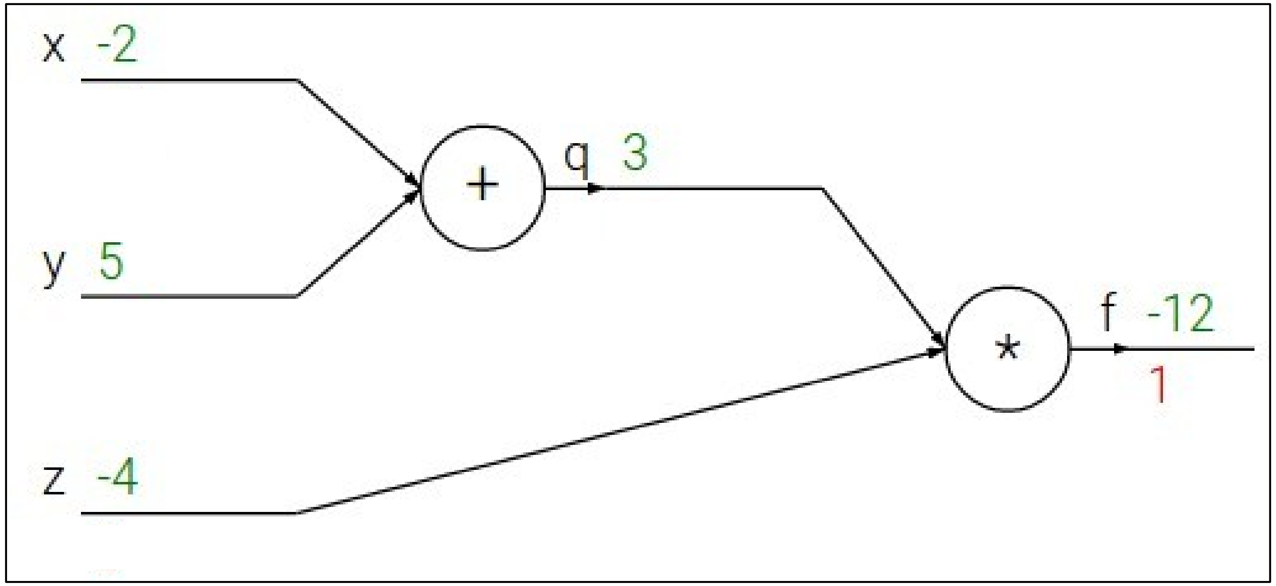
$$f(x,y,z) = (x+y)z$$

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

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

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

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



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

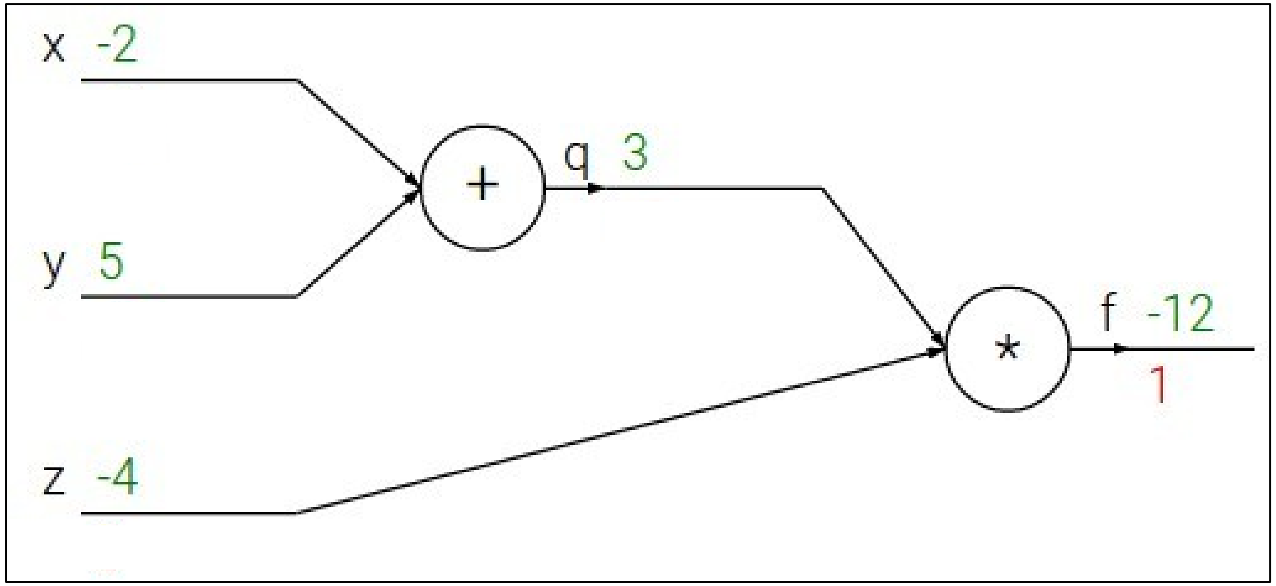
$$f(x,y,z) = (x+y)z$$

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



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

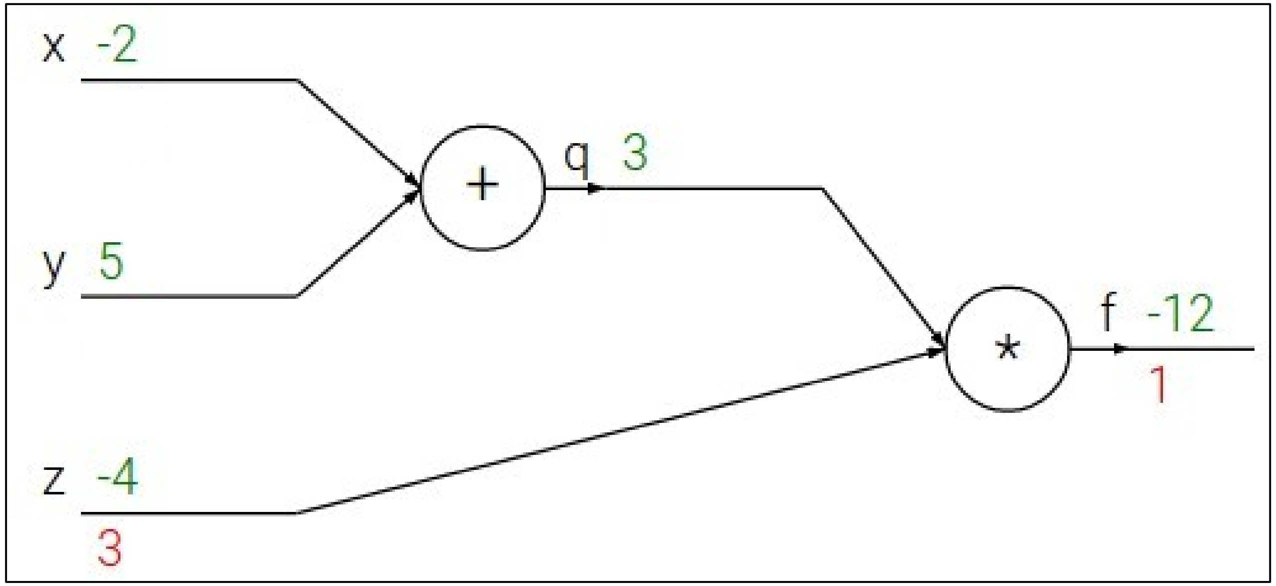
$$f(x,y,z) = (x+y)z$$

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

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

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

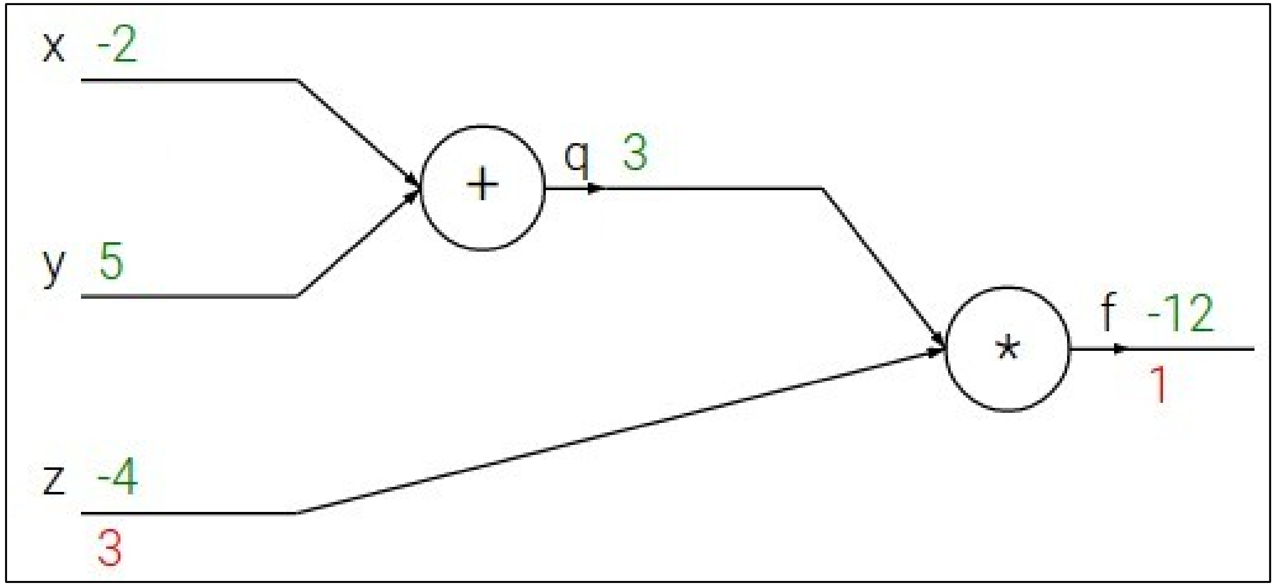
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



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

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

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



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

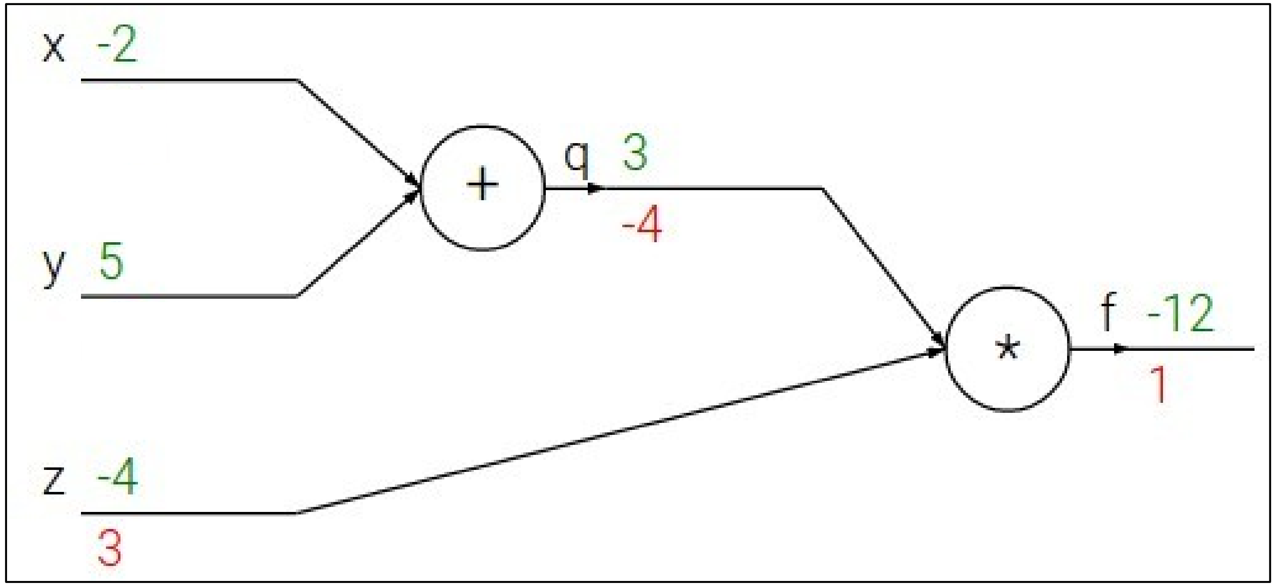
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

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



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

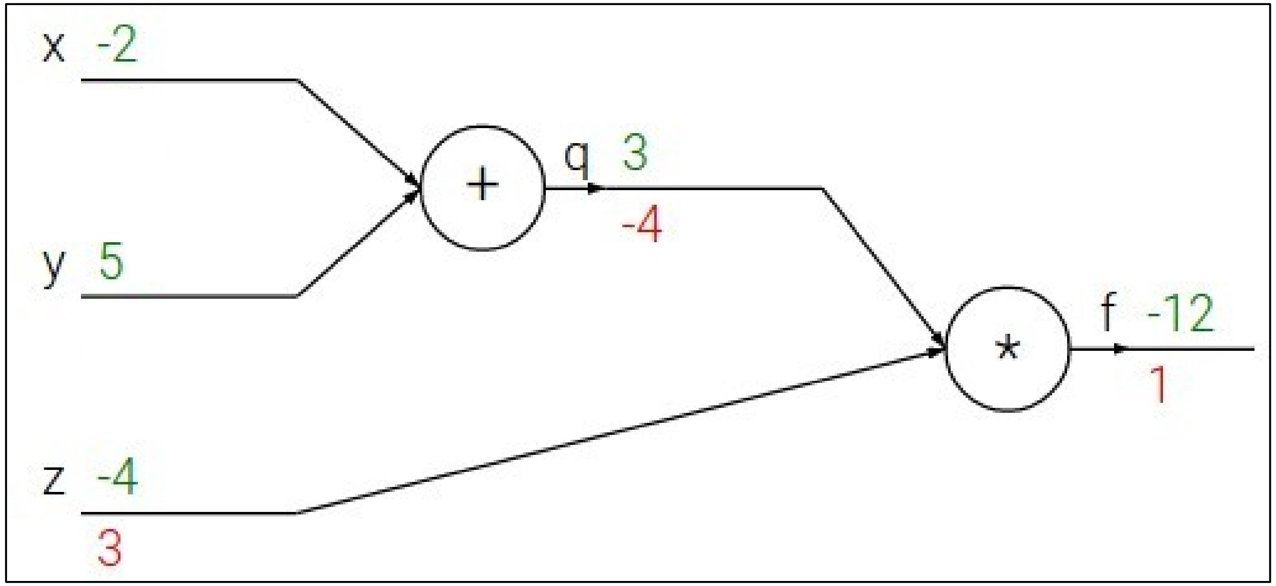
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

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



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

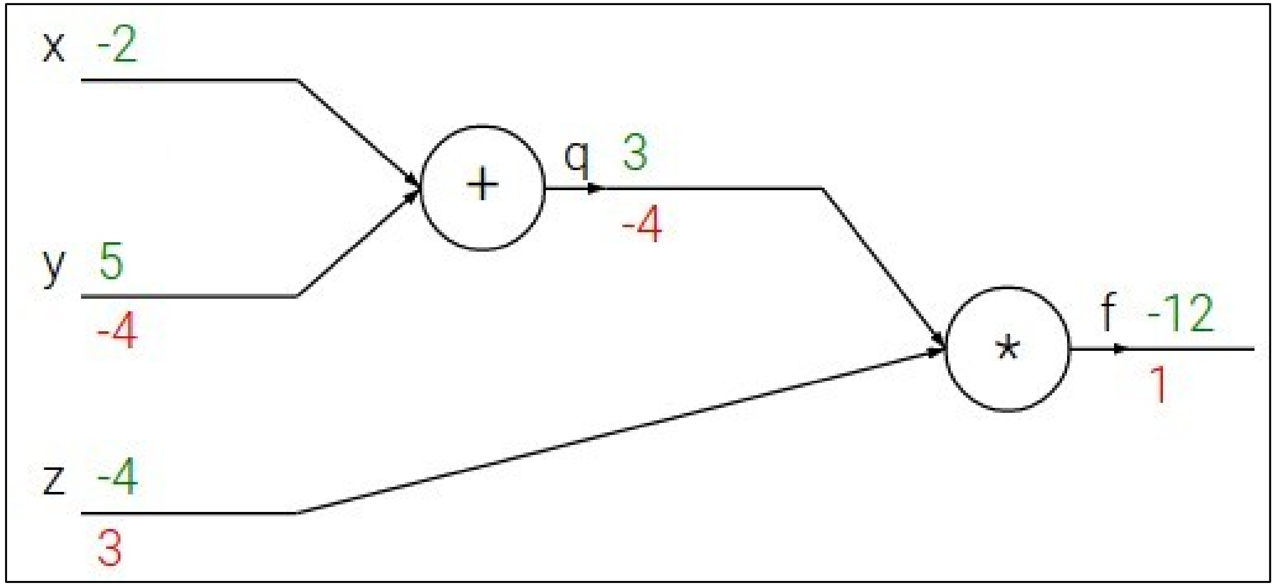
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

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



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

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

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

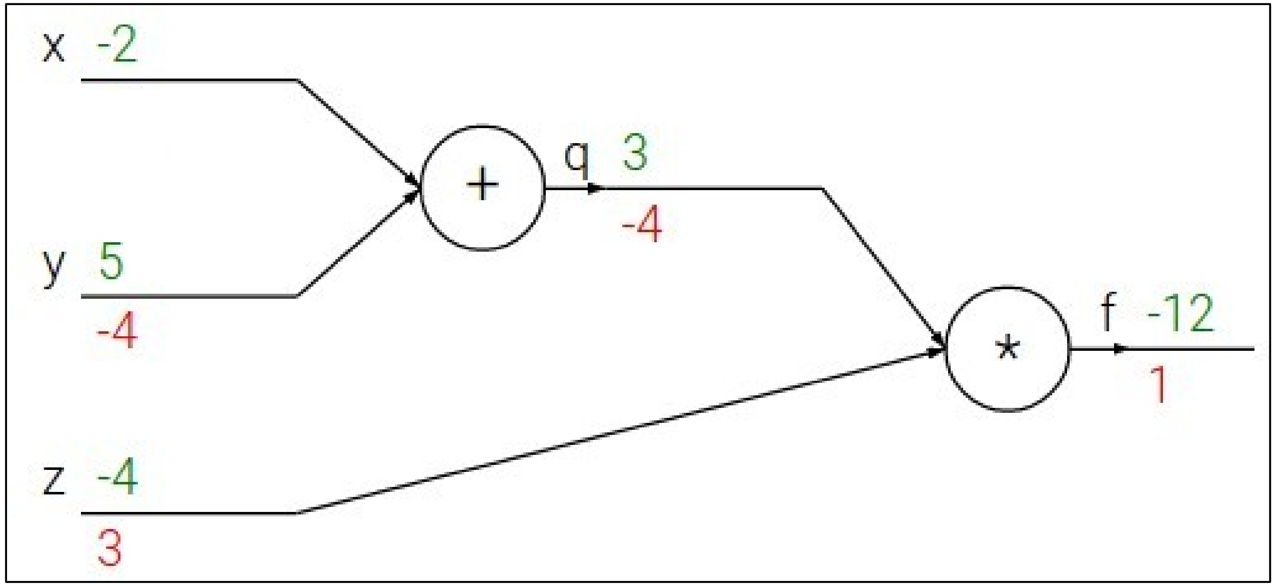
Chain rule:

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

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



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

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

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

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

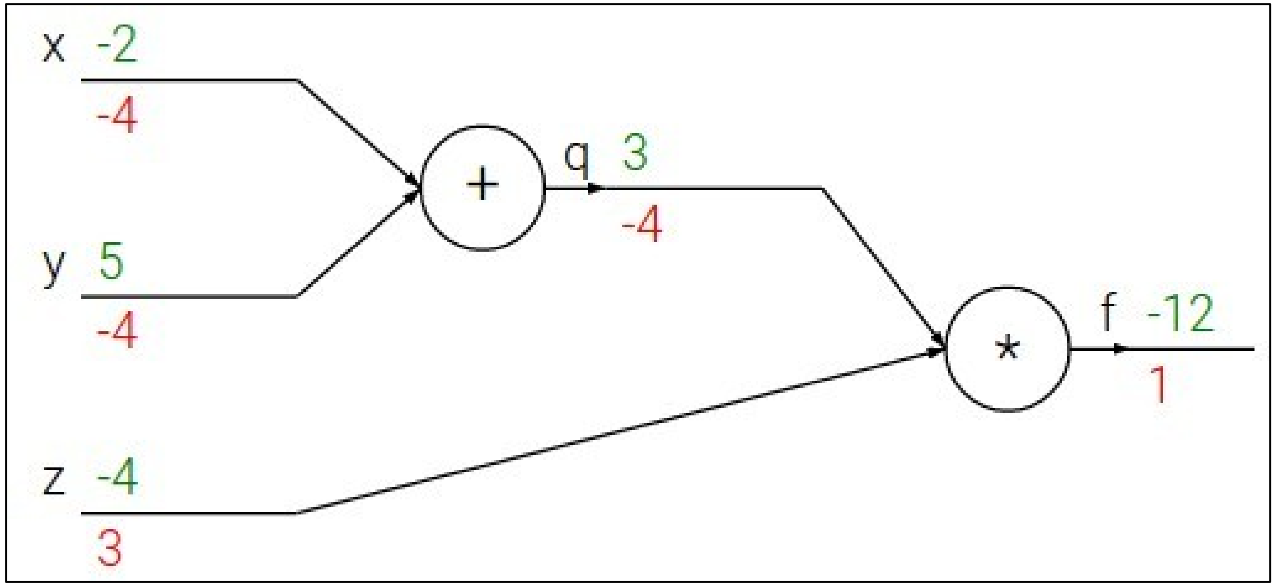
$$f(x,y,z) = (x+y)z$$

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

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

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

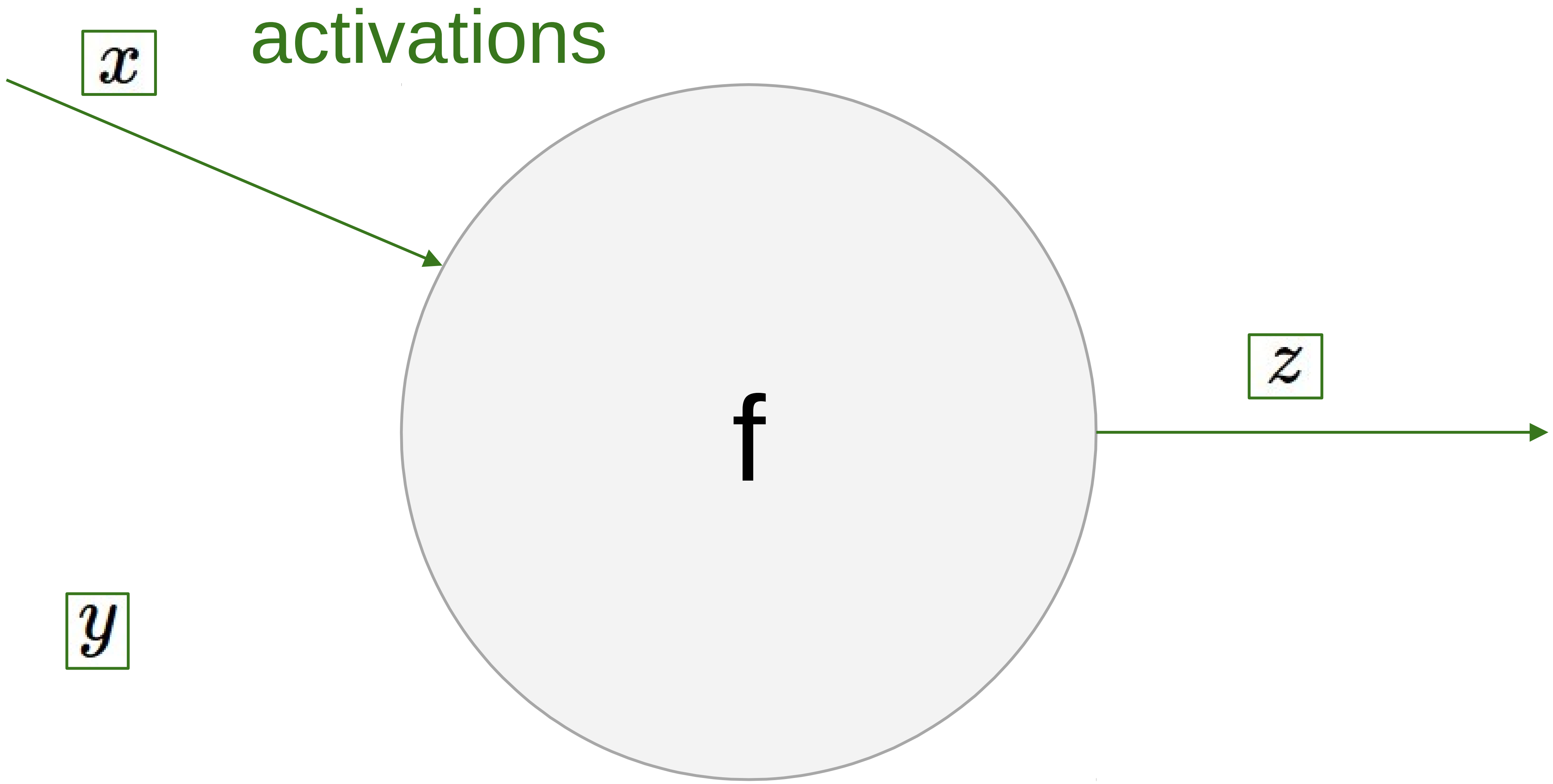
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

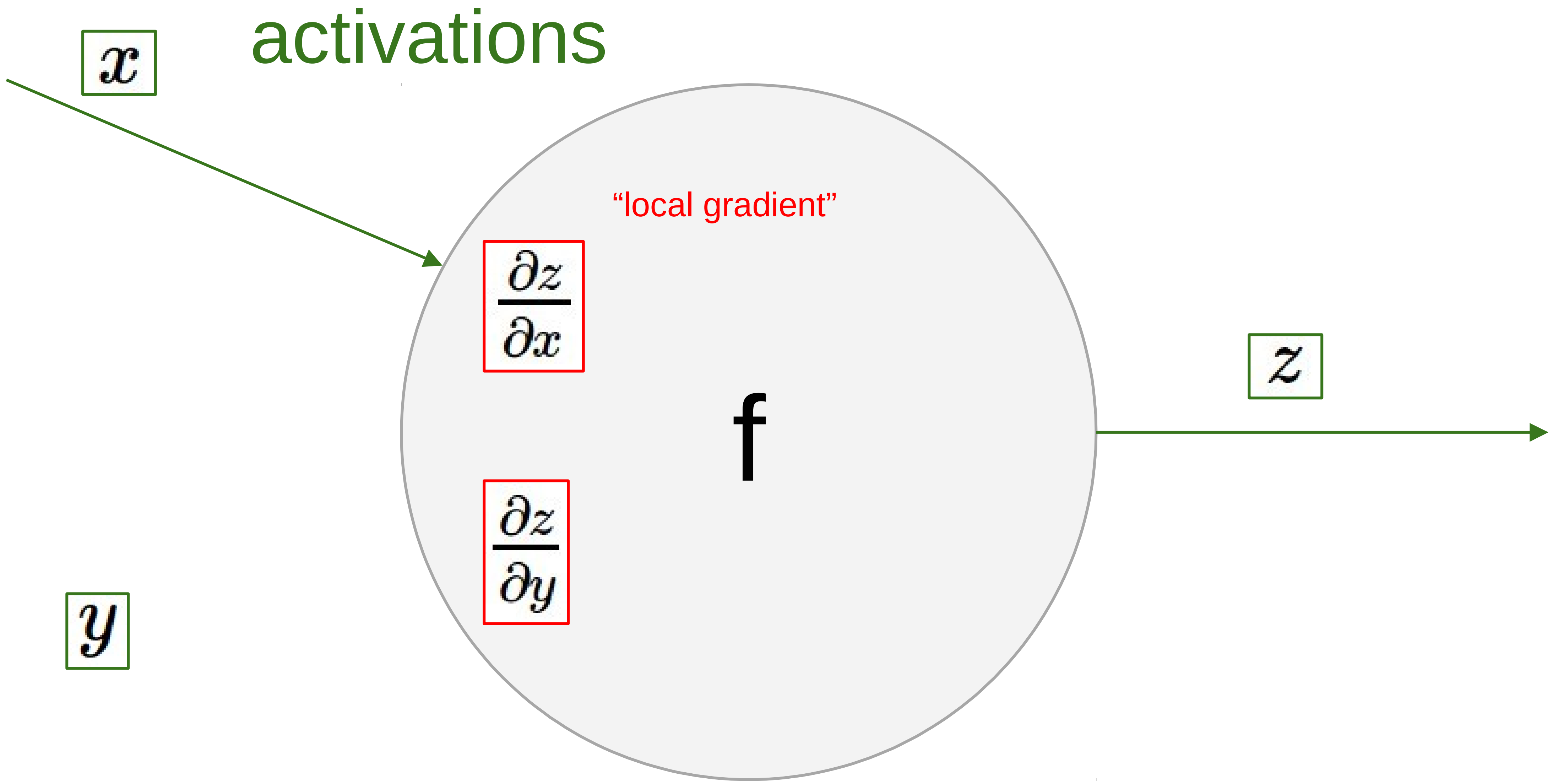


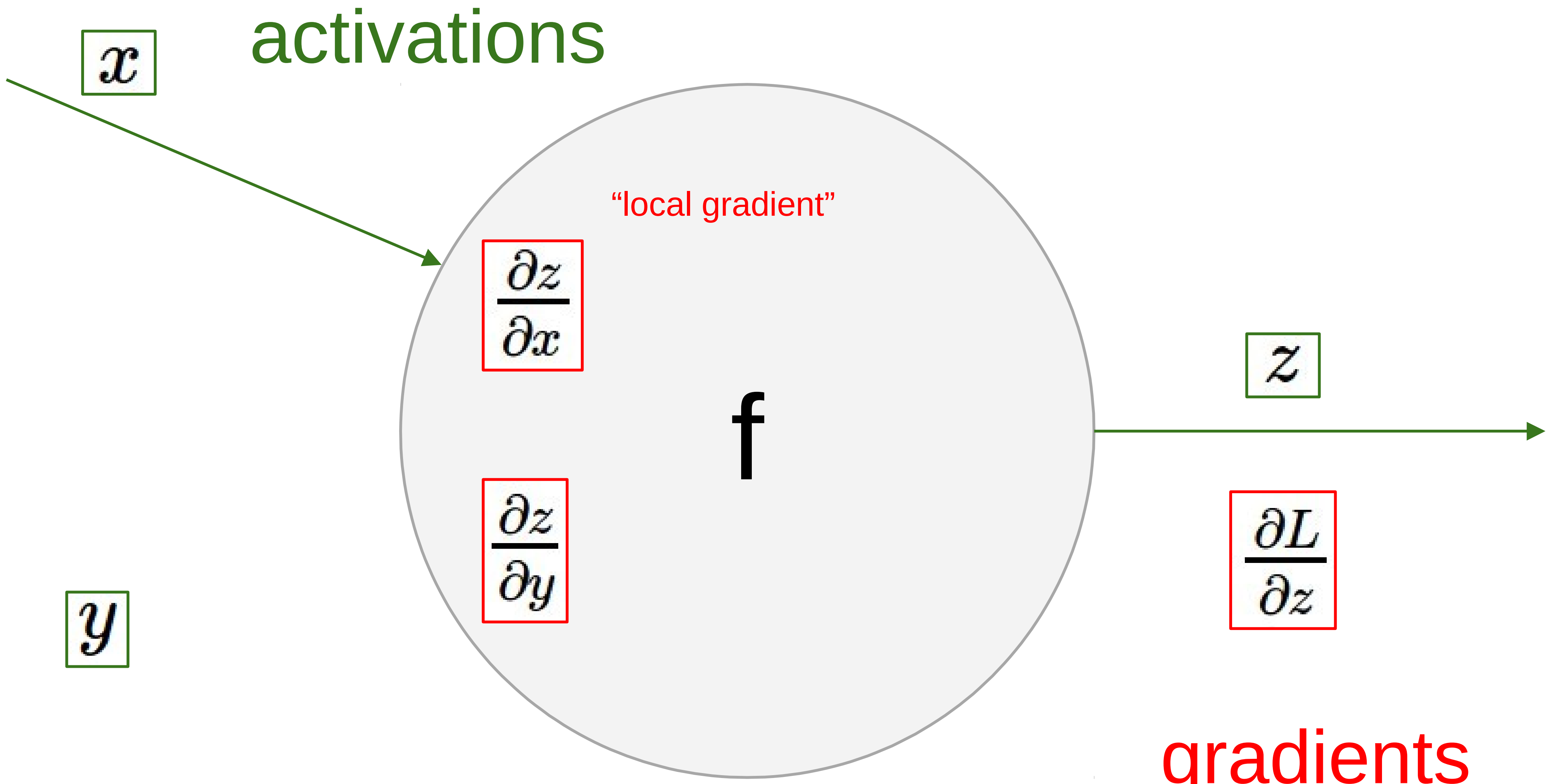
$$\frac{\partial f}{\partial x}$$

Chain rule:

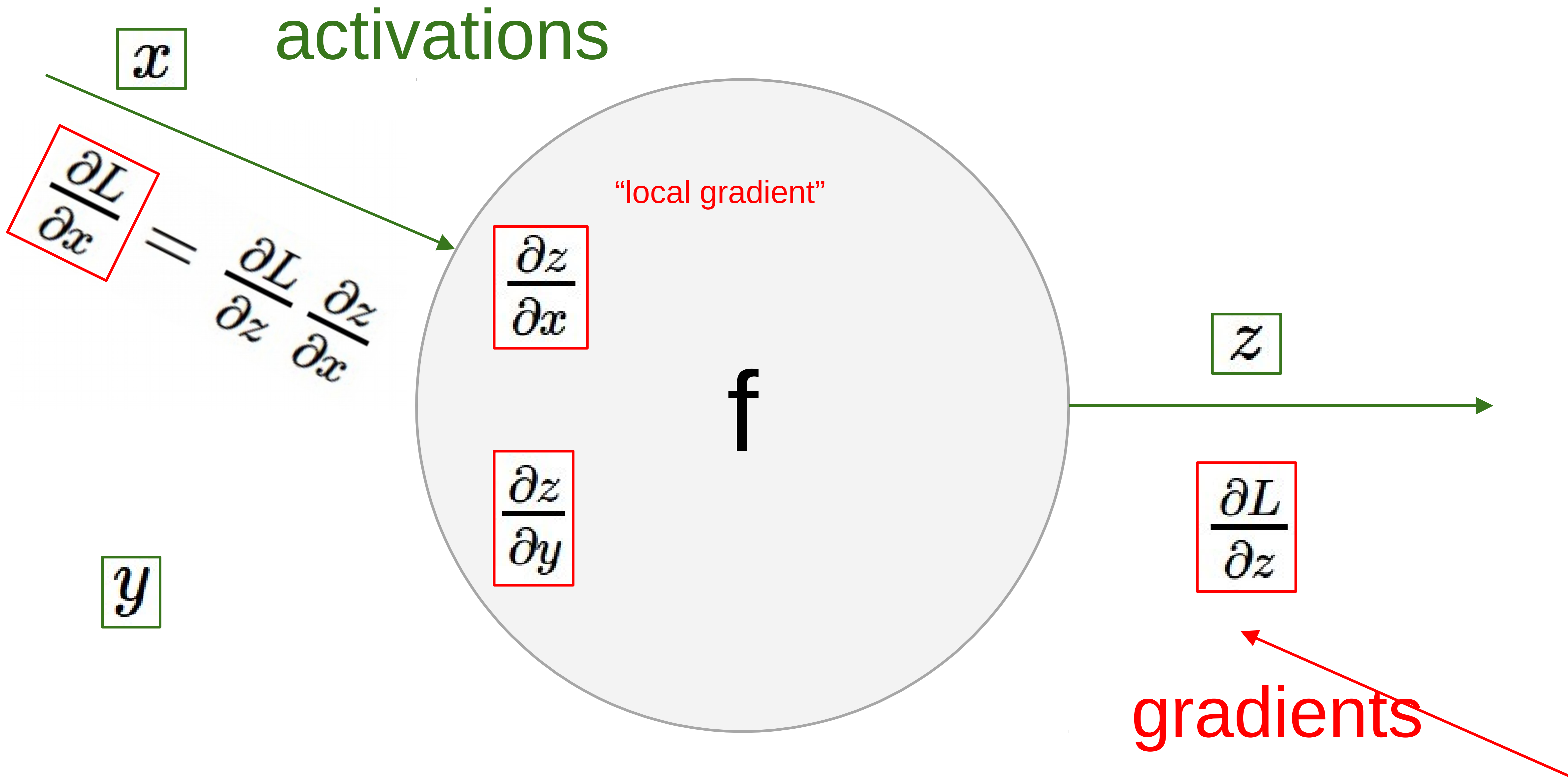
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$


activations
x
f
z
y

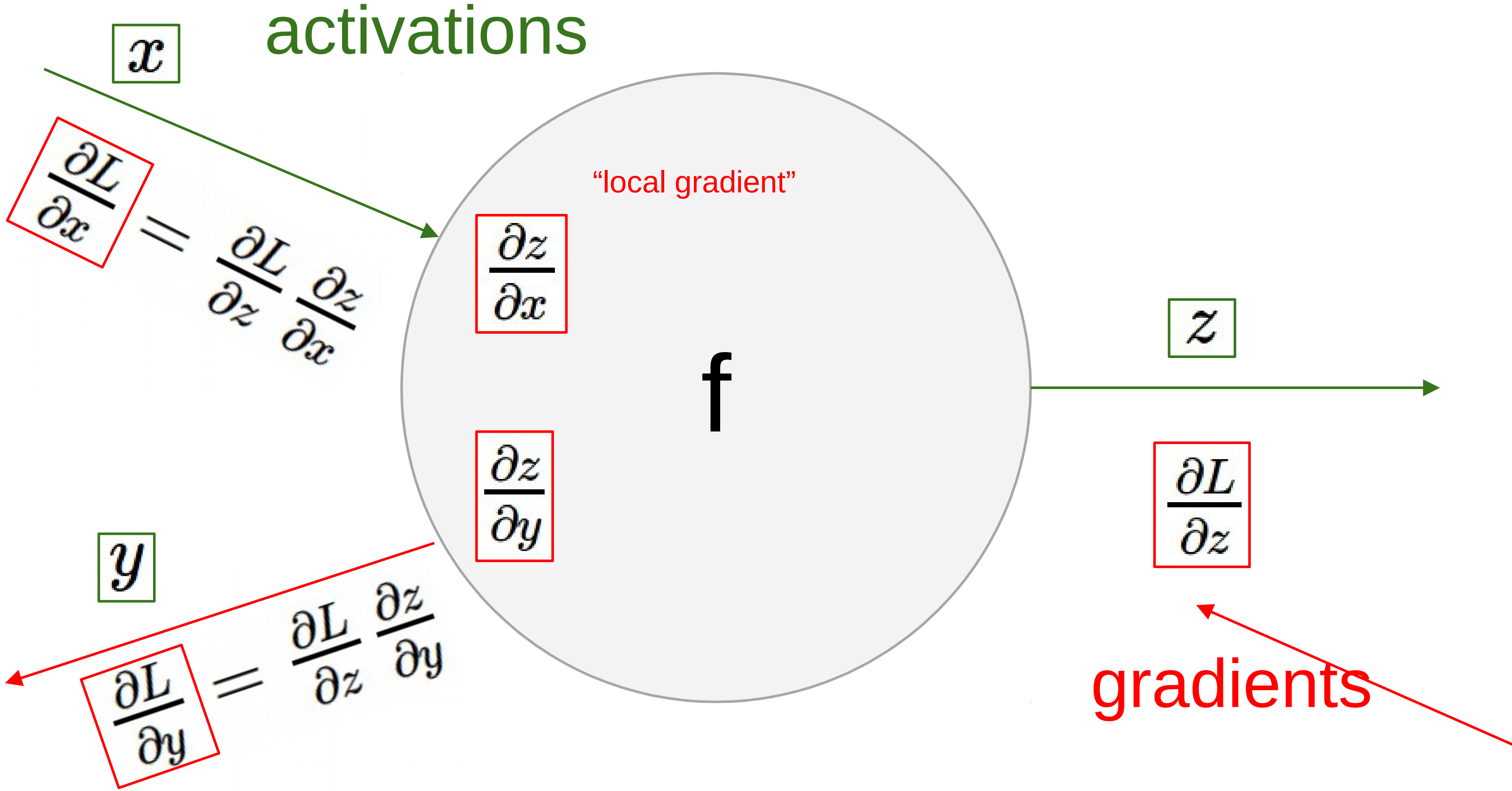

activations
x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y

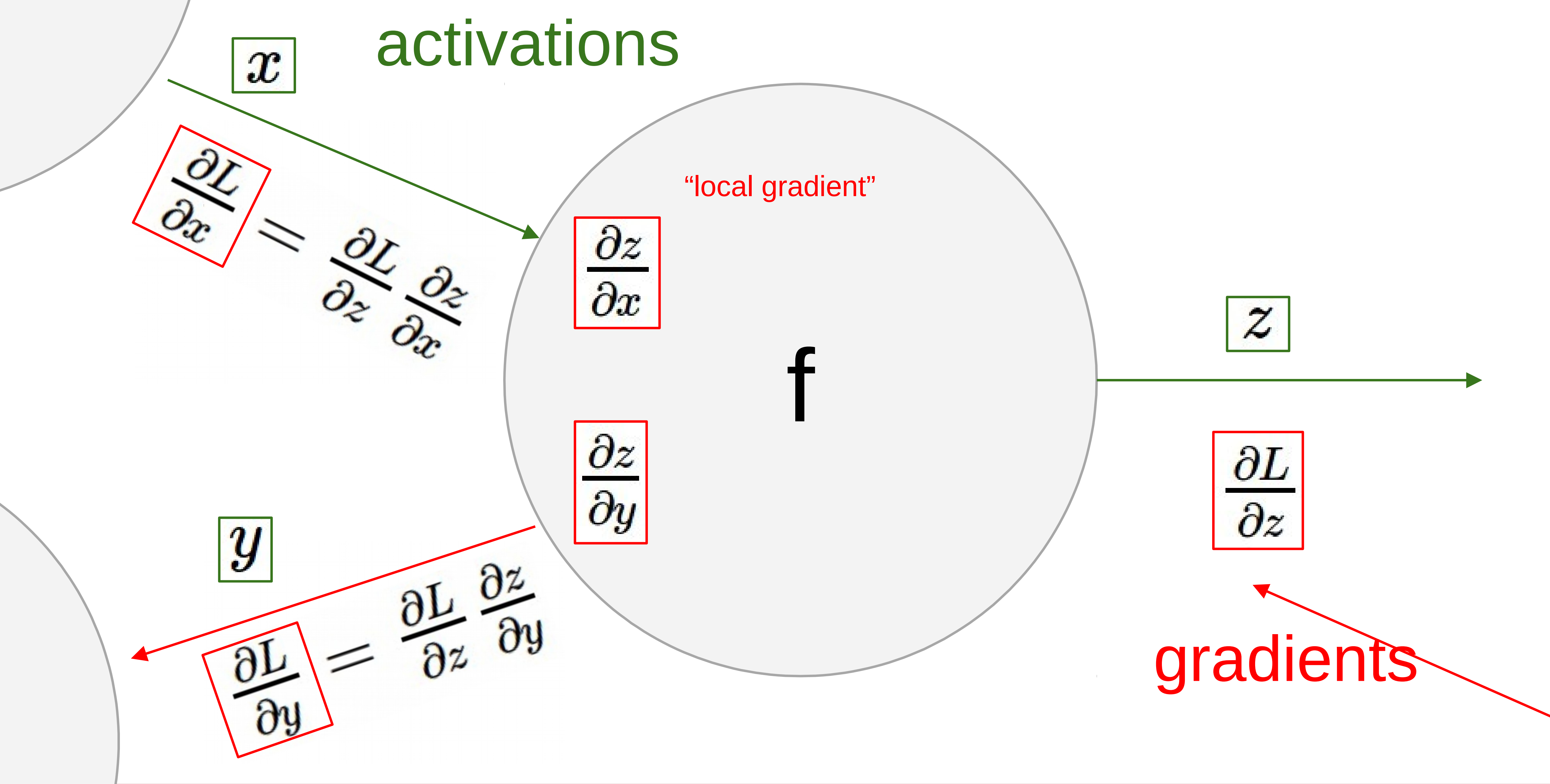


activations
x
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
y
gradients



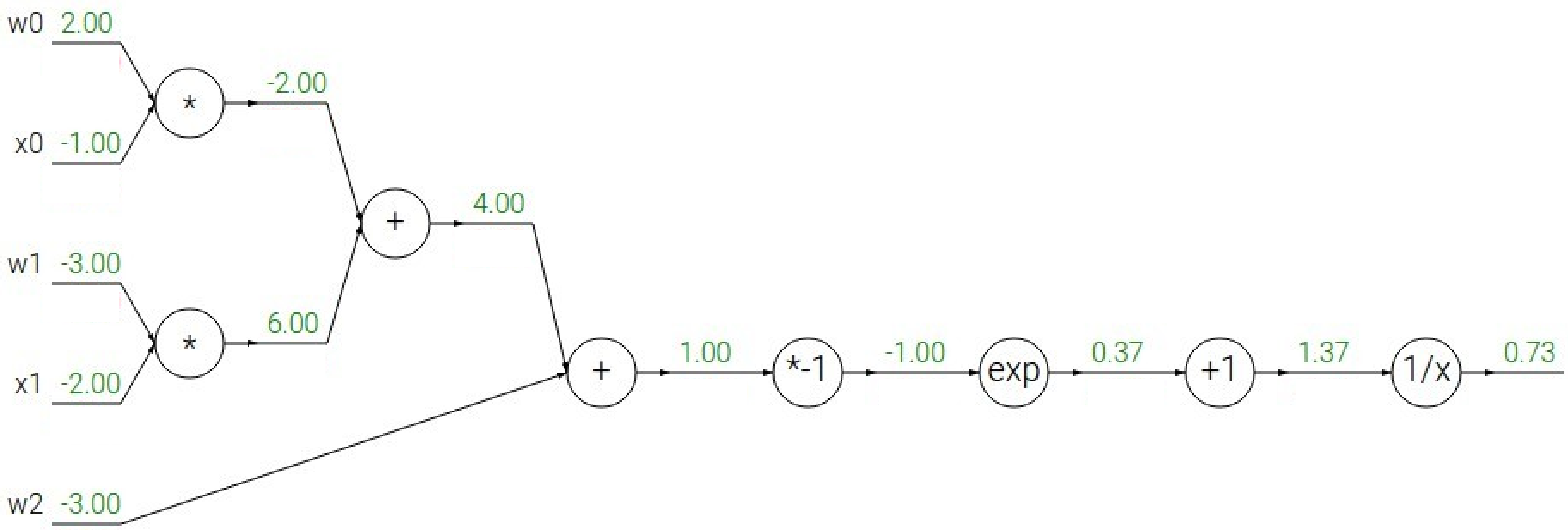
activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
gradients


activations
x
y
z
"local gradient"
f
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

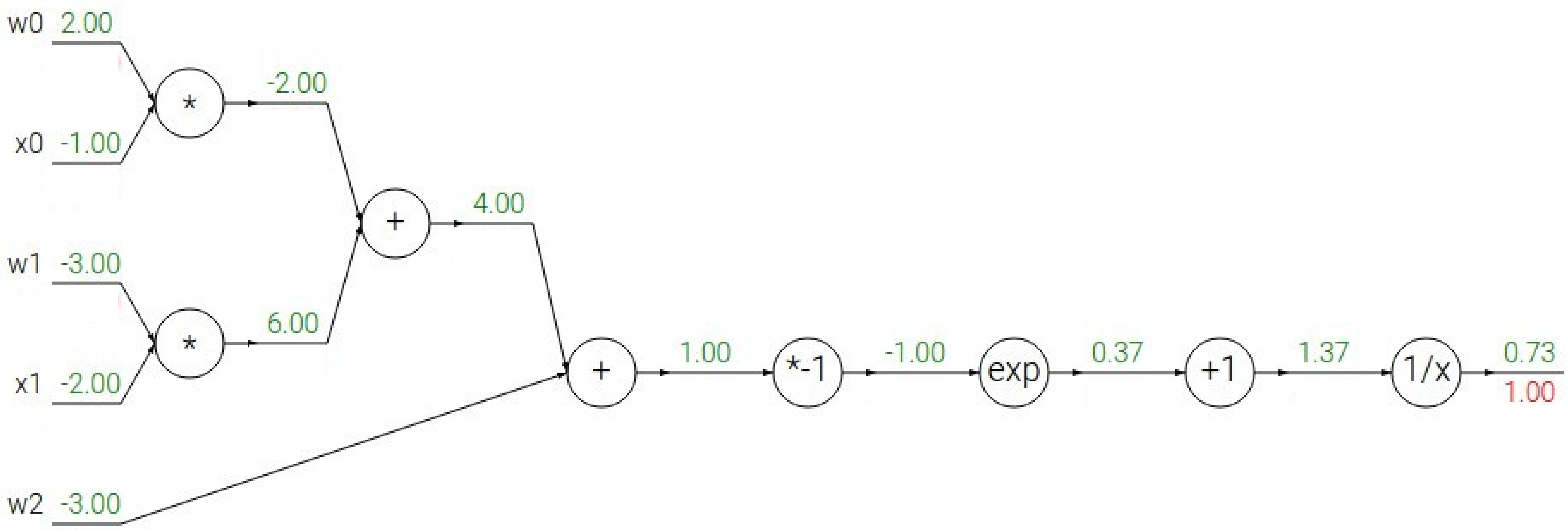

activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Another example:

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



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



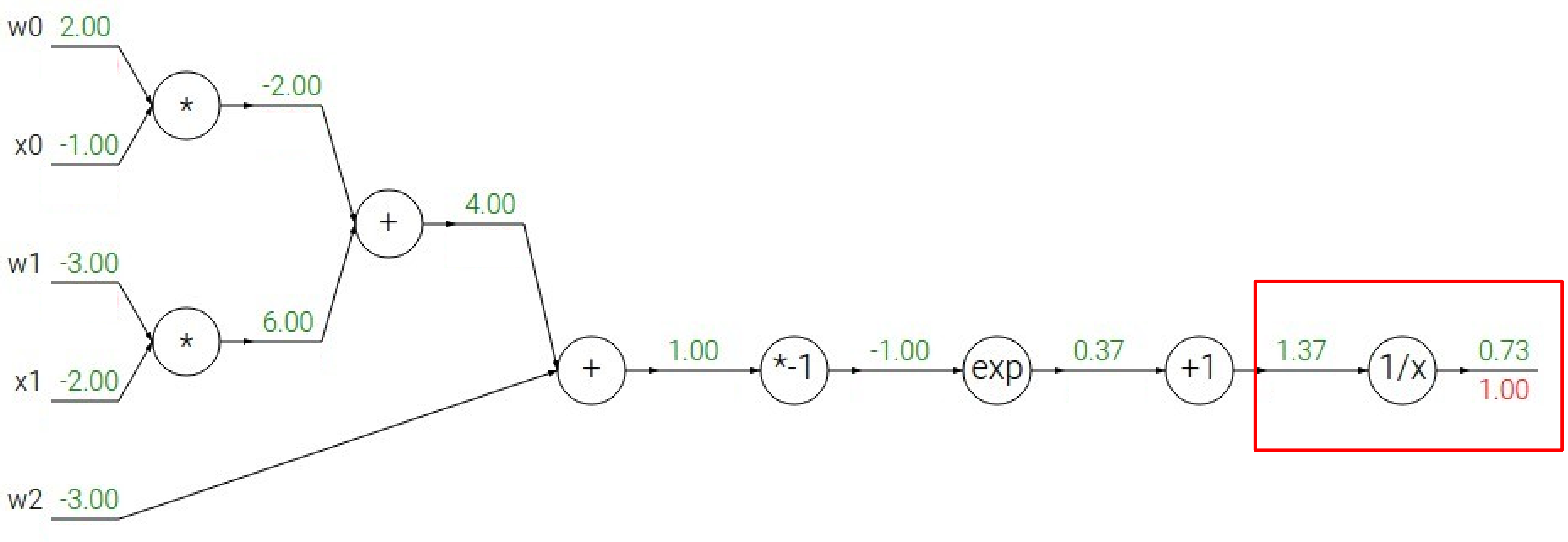
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



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

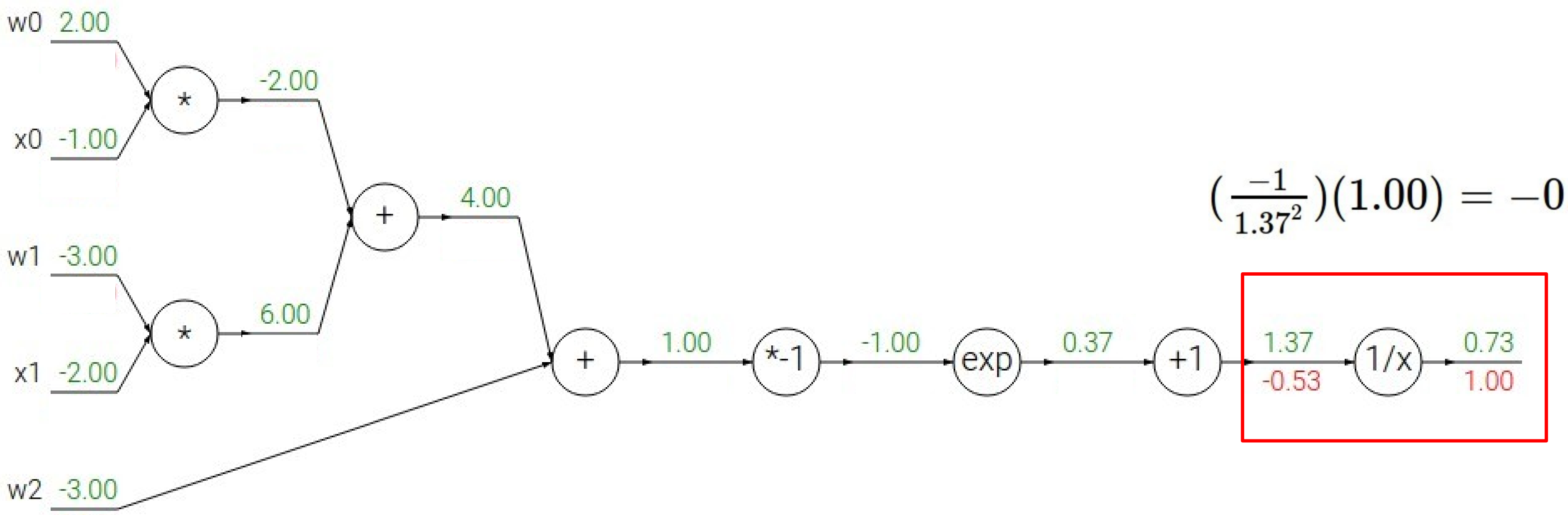
$$f_a(x)=ax \quad \rightarrow \quad \frac{df}{dx}=a$$

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

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

Another example:

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



$$\left(\frac{-1}{1.37^2}\right)(1.00) = -0.53$$

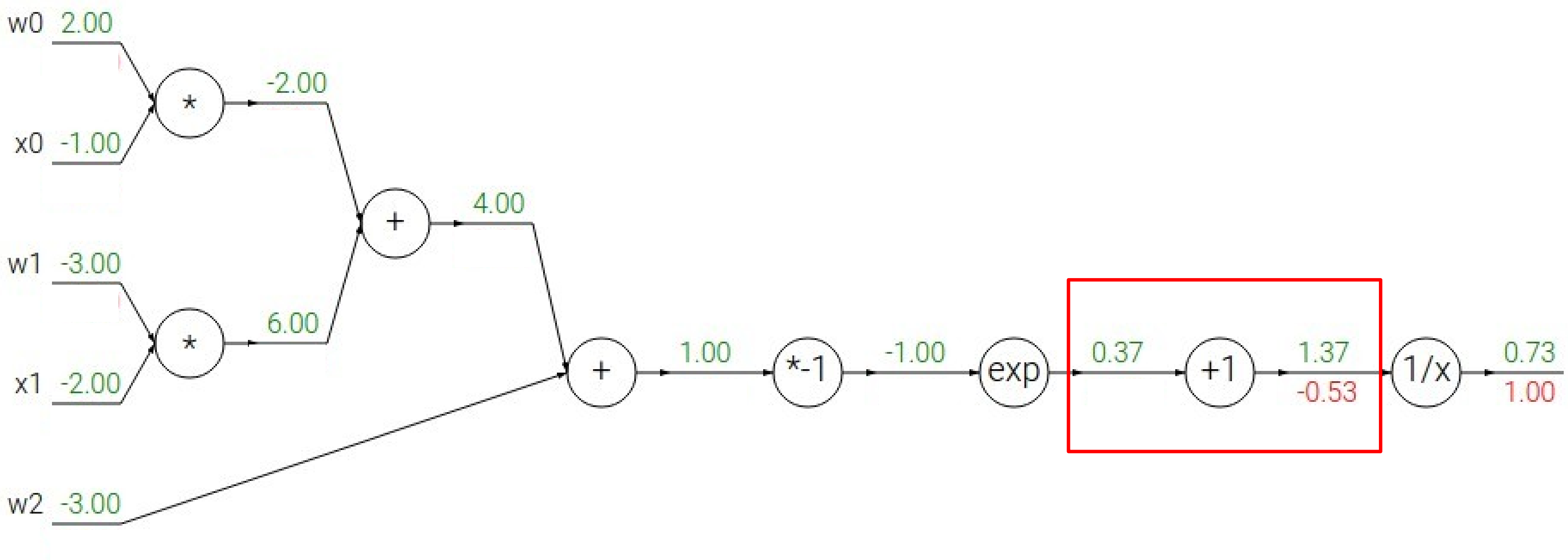
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



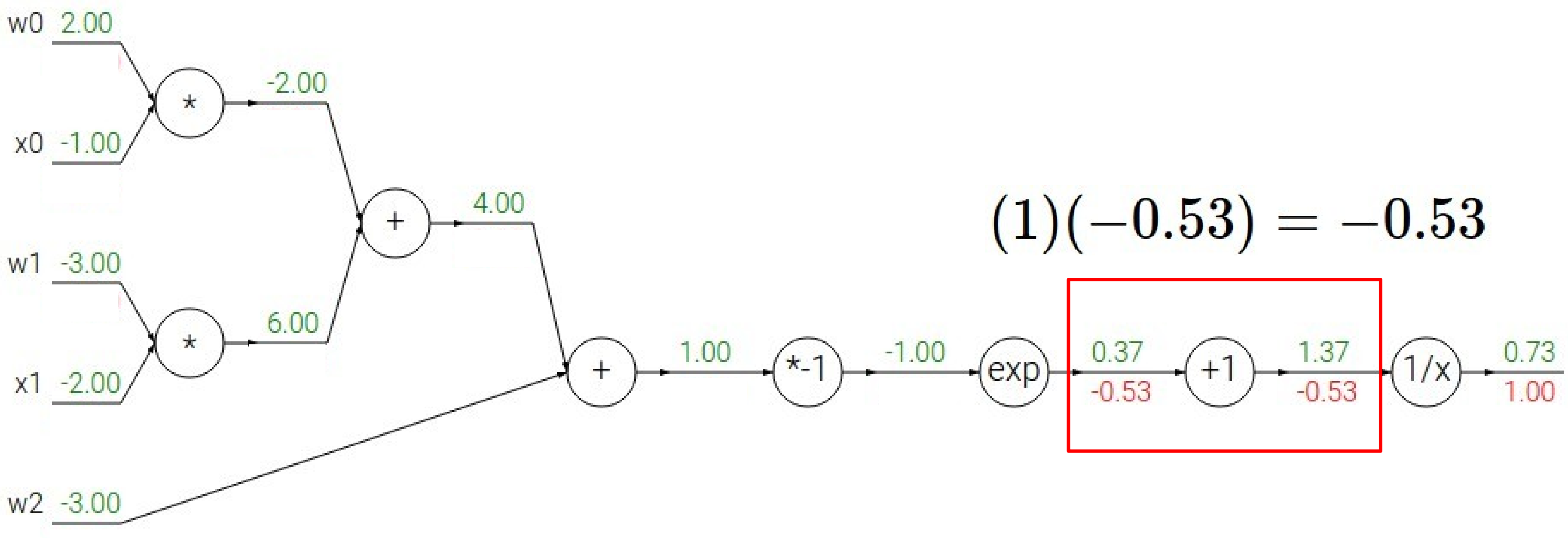
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



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

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

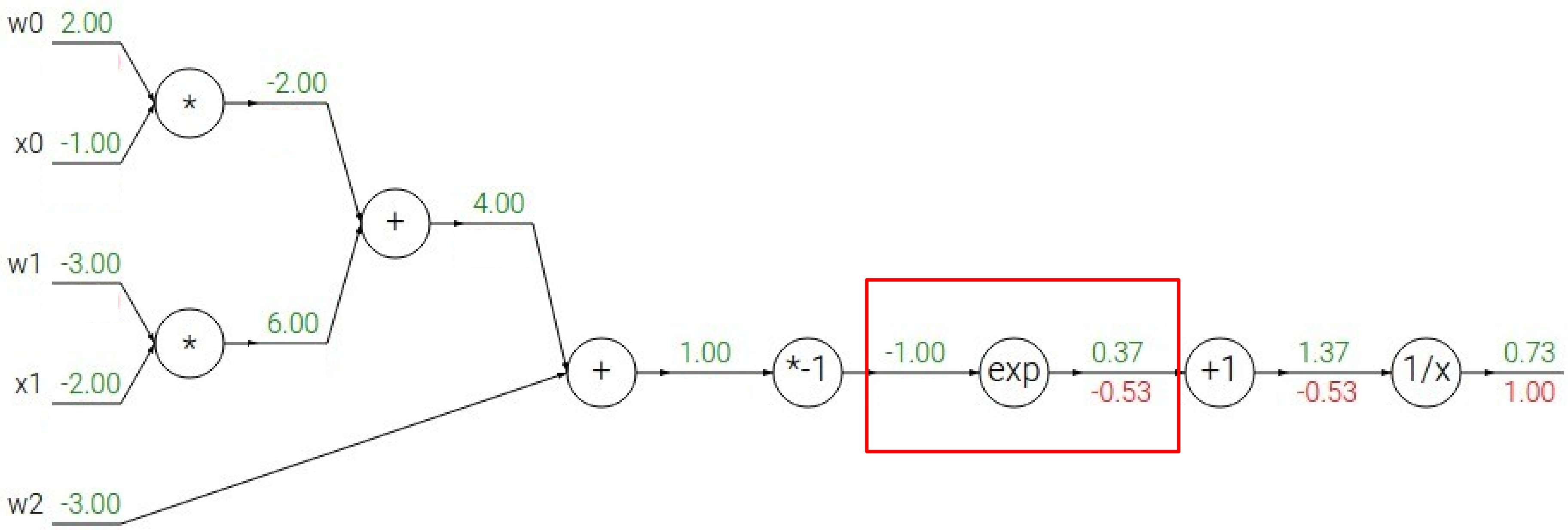
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

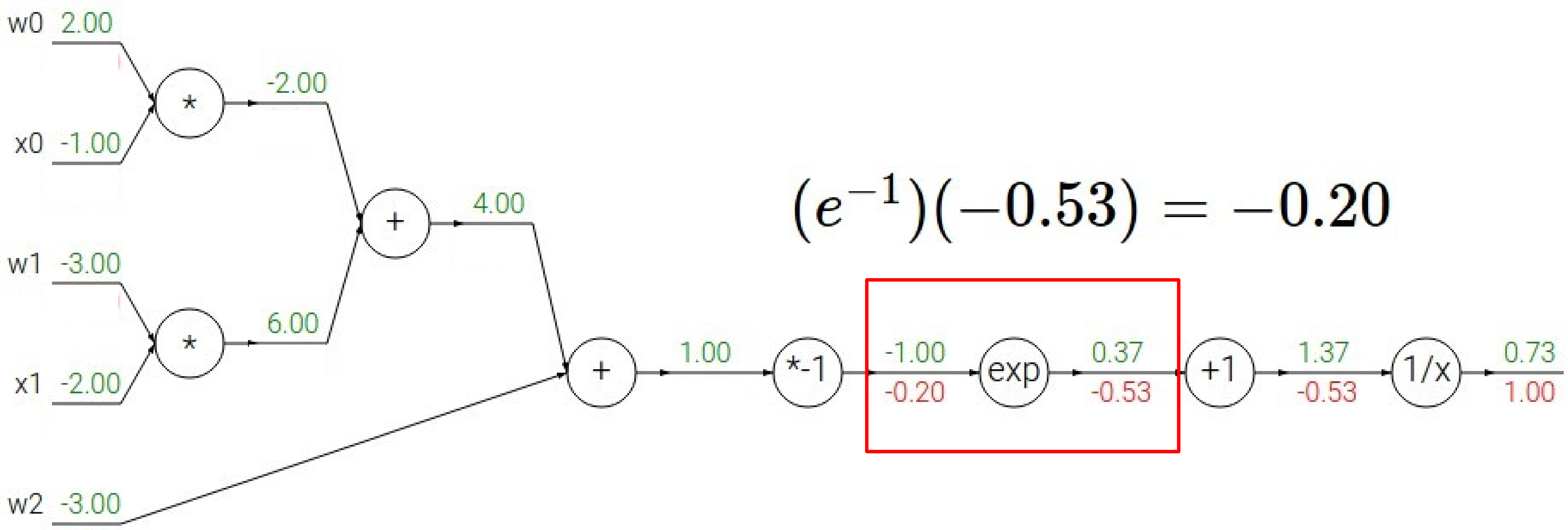
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



$$(e^{-1})(-0.53)=-0.20$$

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

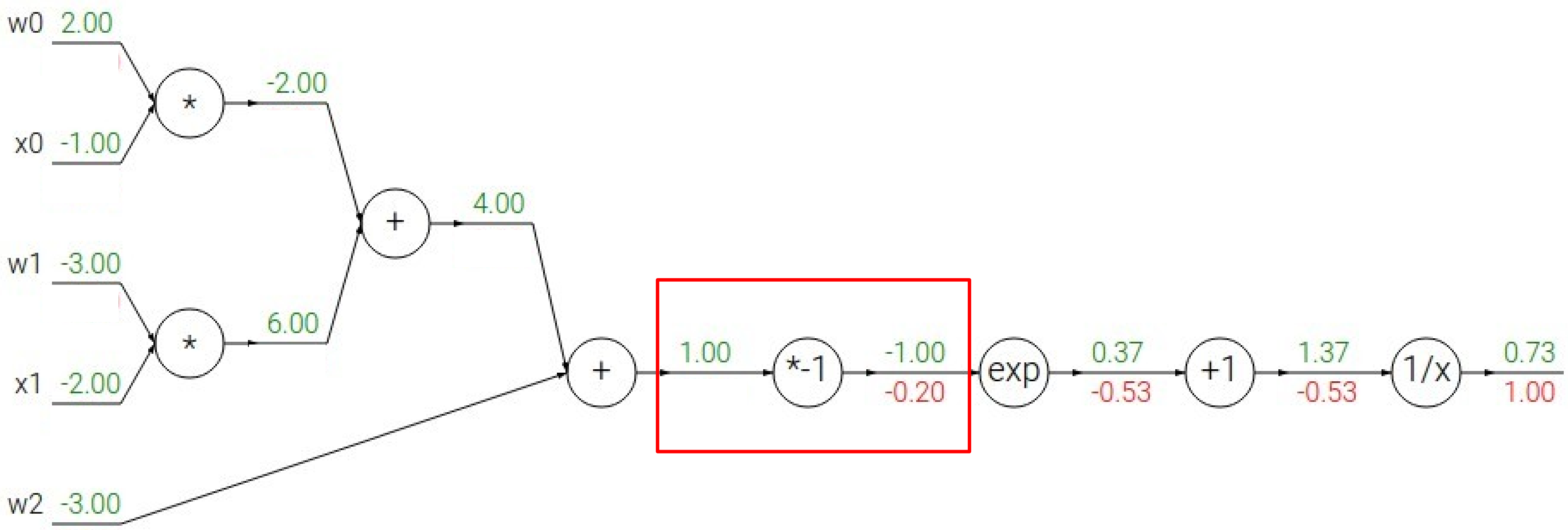
$$f_a(x)=ax \quad\rightarrow\quad \frac{df}{dx}=a$$

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

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

Another example:

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



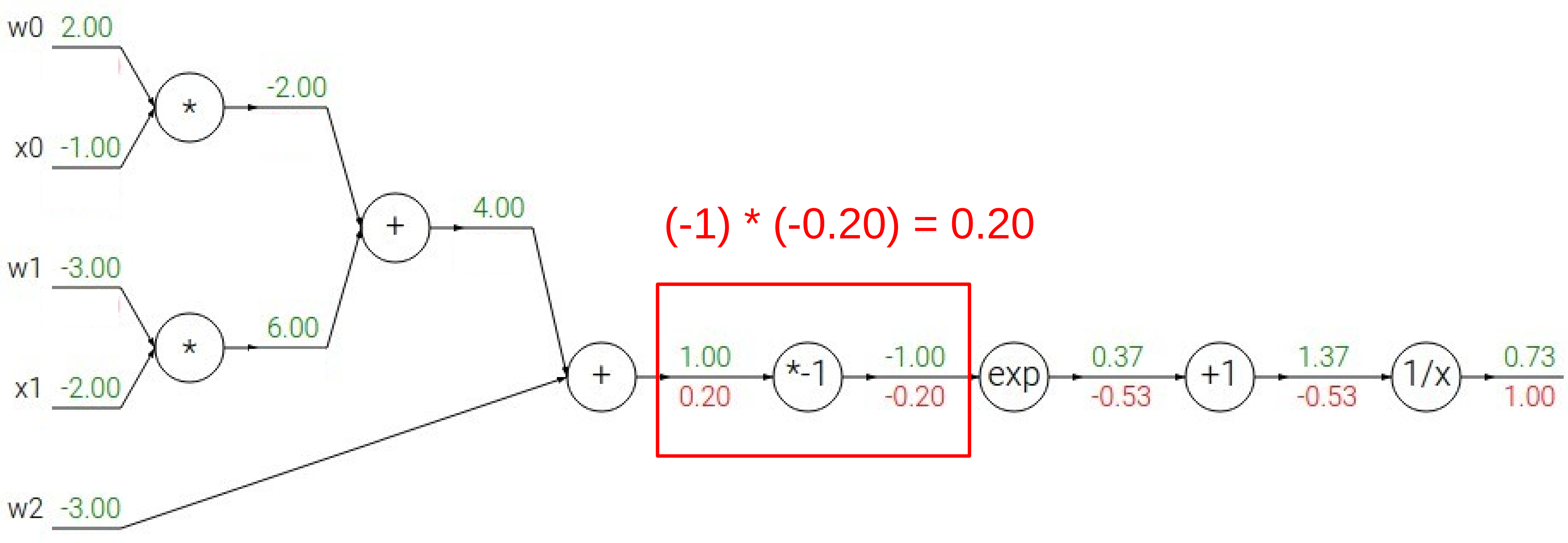
$$f(x)=e^x \quad\rightarrow\quad \frac{df}{dx}=e^x$$

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

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

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

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



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

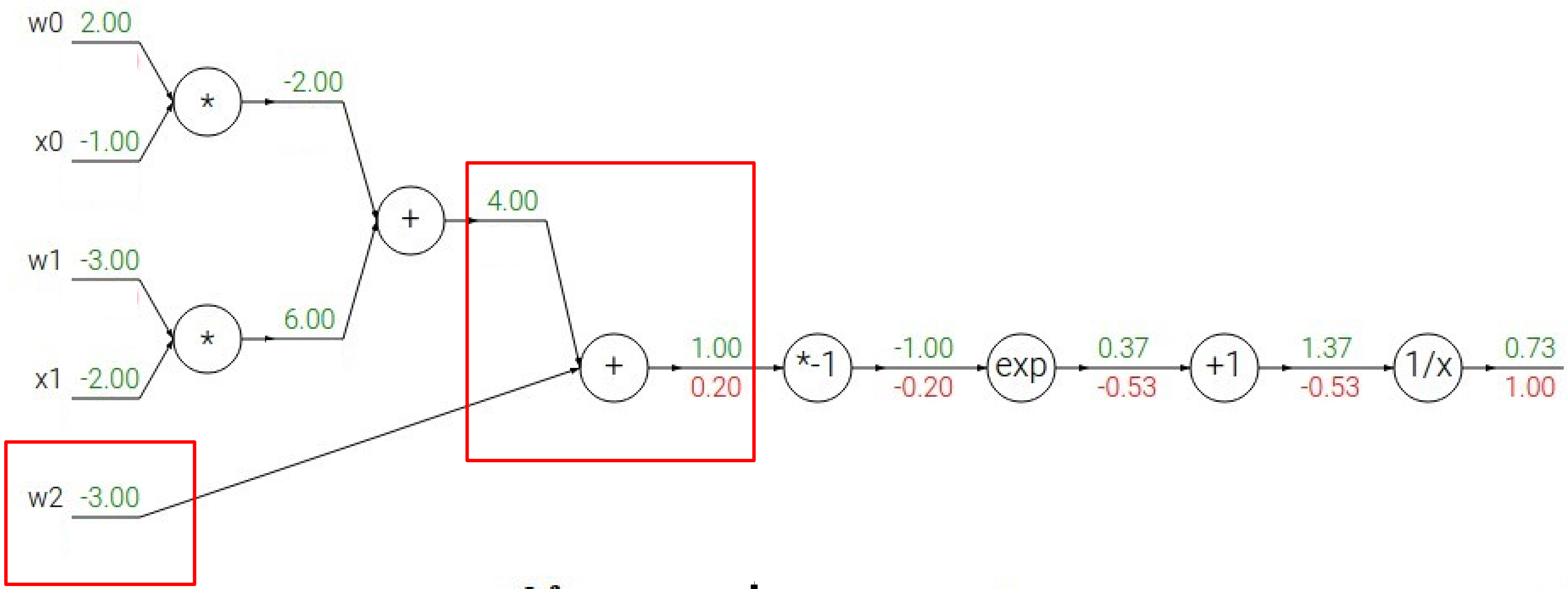
$$\boxed{f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a}$$

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

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

Another example:

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

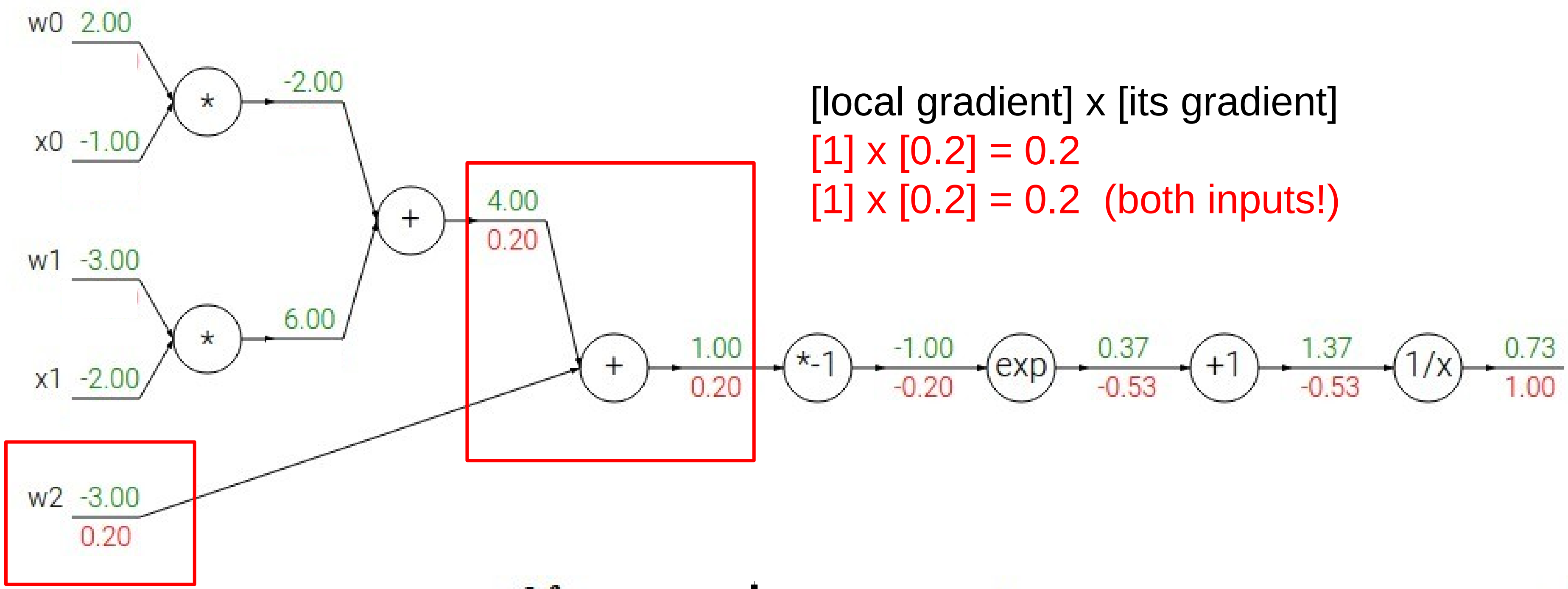


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \qquad\qquad f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

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



[local gradient] x [its gradient]
[1] x [0.2] = 0.2
[1] x [0.2] = 0.2 (both inputs!)

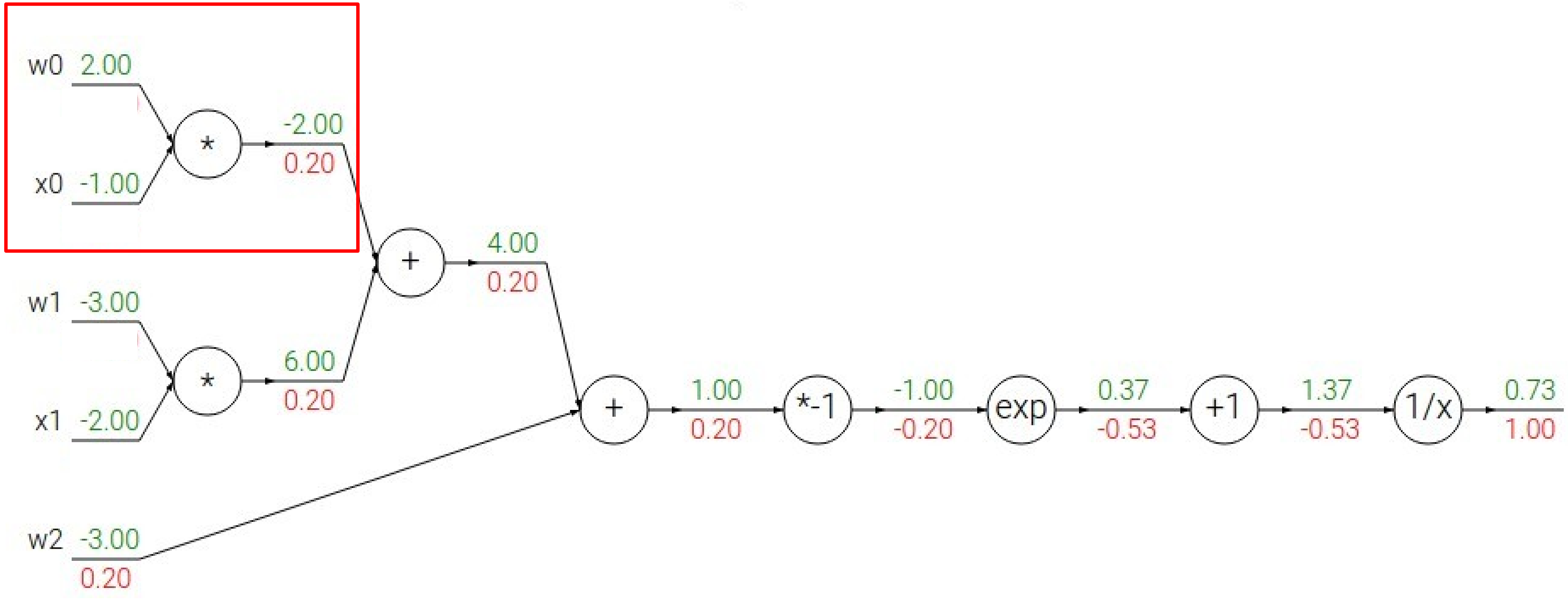
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



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

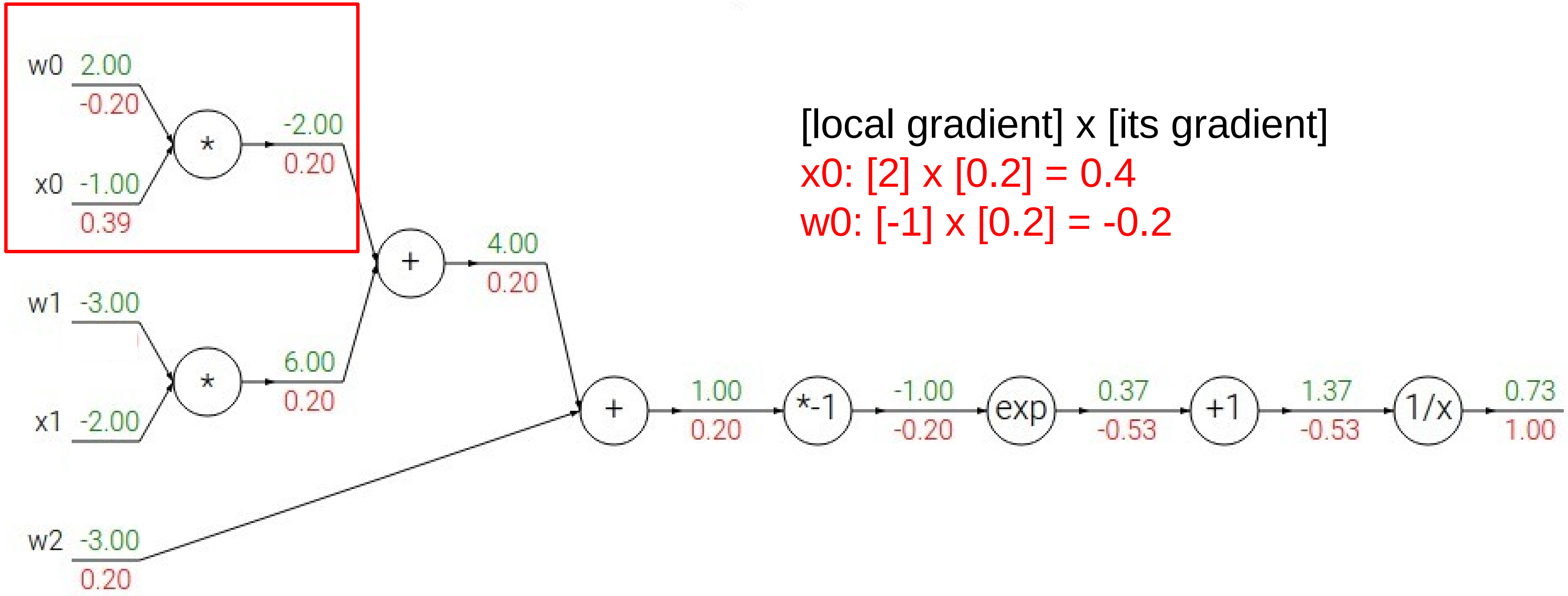
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



[local gradient] x [its gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

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

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

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

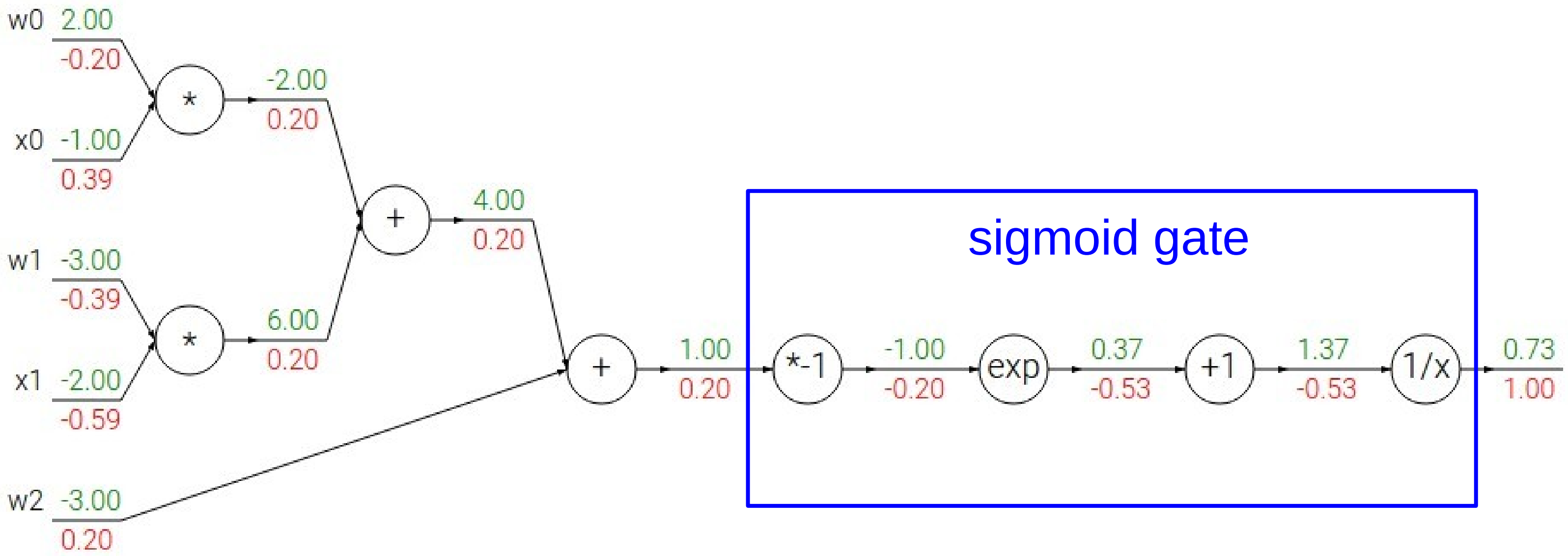
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

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

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

sigmoid function

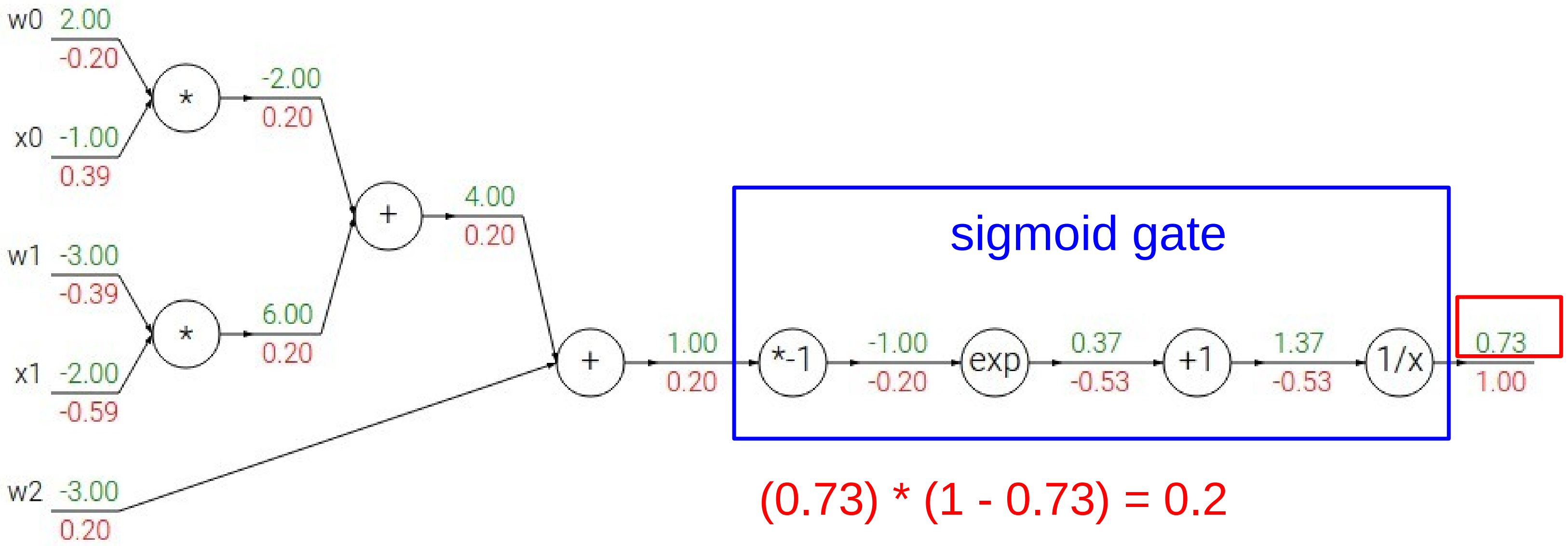
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$



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

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

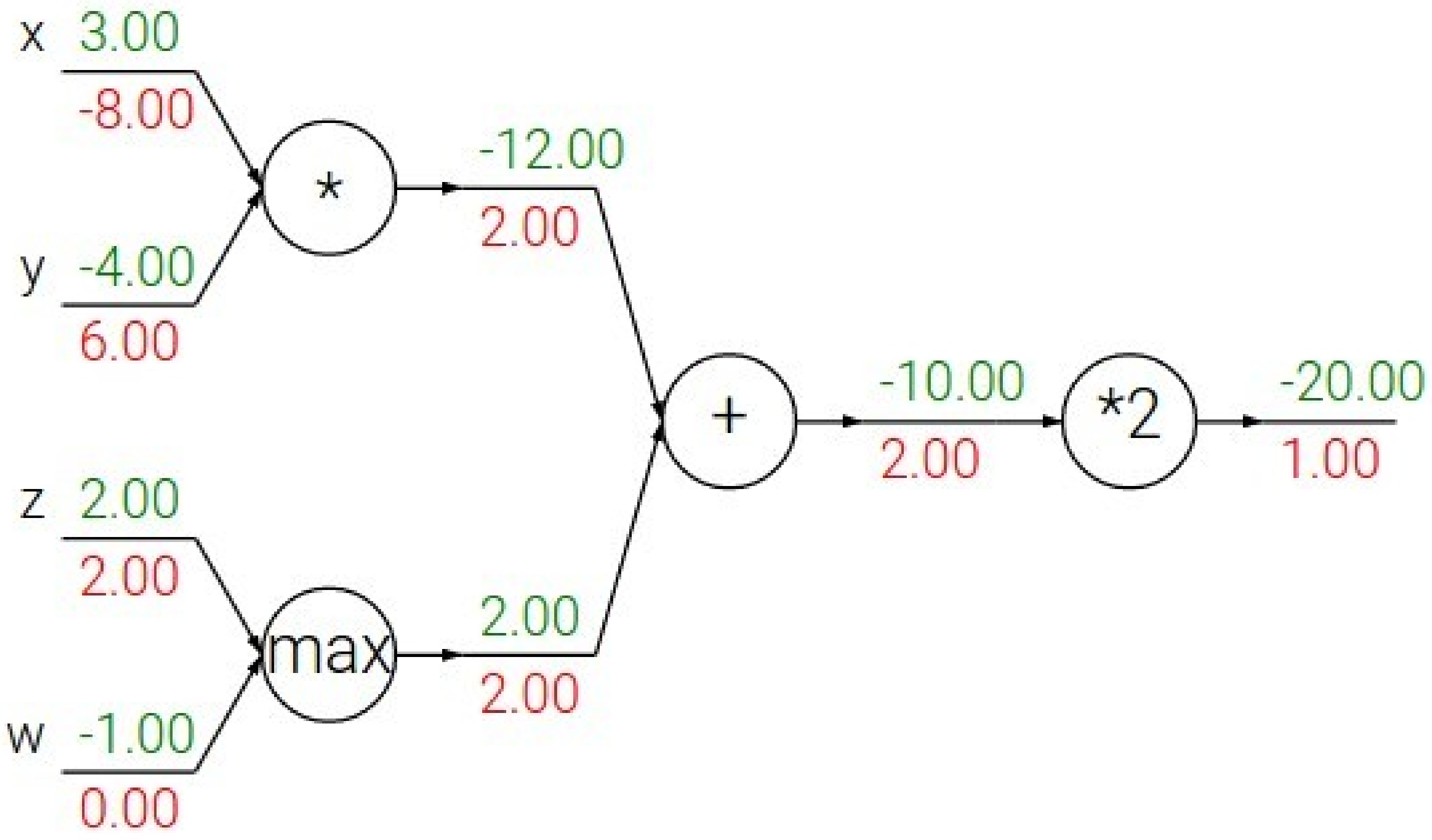
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



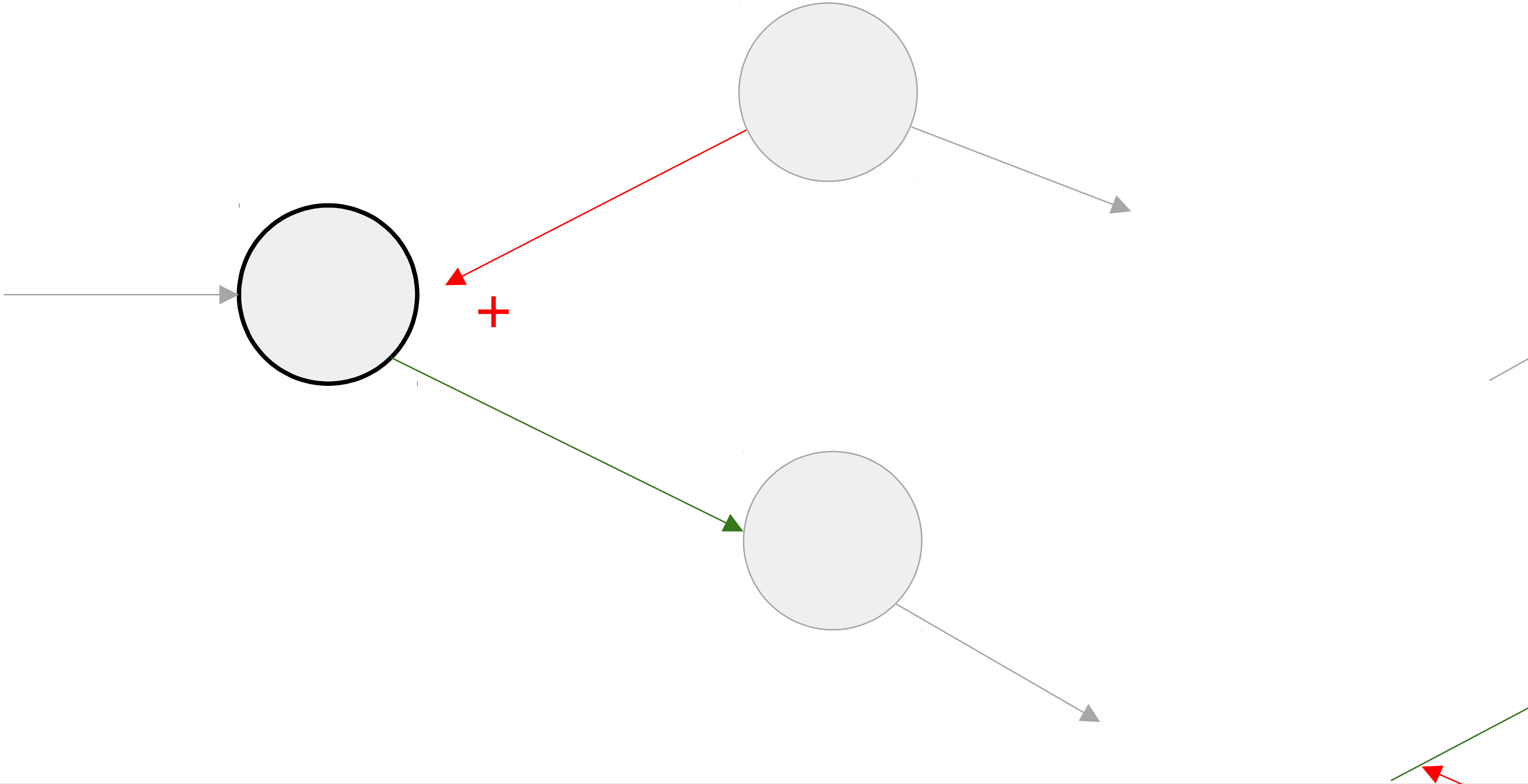
(0.73) * (1 - 0.73) = 0.2

Patterns in backward flow

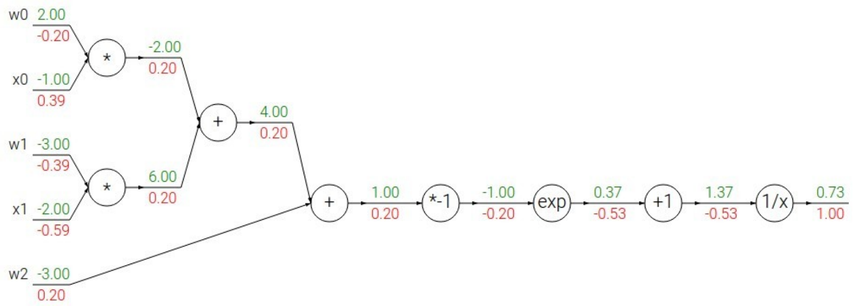
add gate: gradient distributor
max gate: gradient router
mul gate: gradient…?



Gradients add at branches



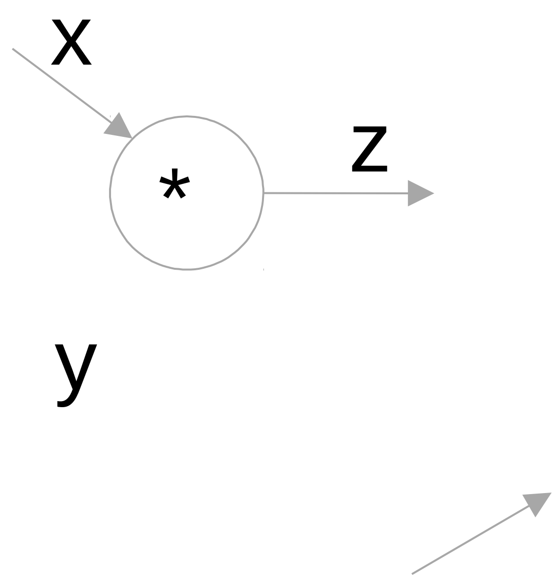
Implementation: forward/backward API



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



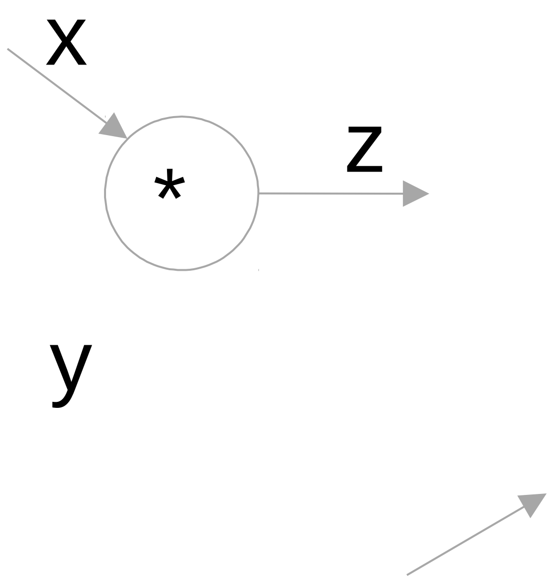
(x,y,z are scalars)

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

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

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

Implementation: forward/backward API




```python
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)