CS1138

Machine Learning

Lecture : Training Neural Networks: Loss Function and Backpropagation

Arpan Gupta

Overview

- Training NNs
  - Loss Function
  - Forward Propagation
  - Backward Propagation
  - Computational Graphs
  - Regularization

Softmax Activation Function

- Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say ***v***) with probabilities of each possible outcome. The probabilities in vector ***v*** sums to one for all possible outcomes or classes.
- It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.

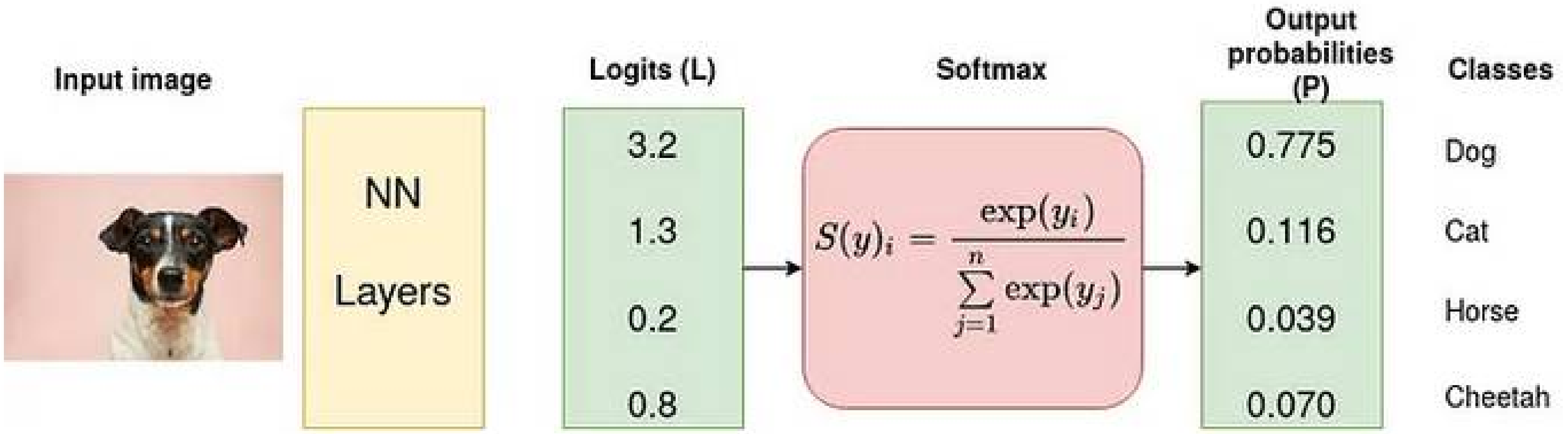
$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

y_i – i^{th} element of the input vector to softmax

$\exp(y_i)$ - standard exponential function.

n – vector size

Softmax Activation Function



Input image source: Photo by Victor Grabarczyk on Unsplash . Diagram by author.

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural network:

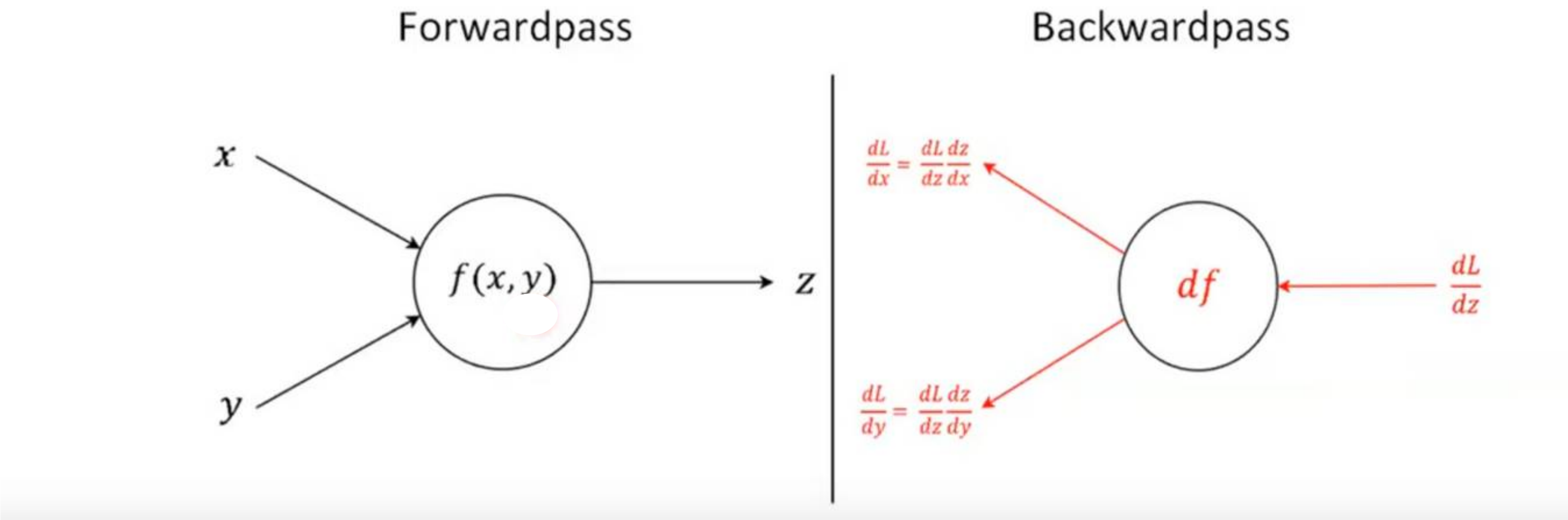
$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Backpropagation – Chain Rule

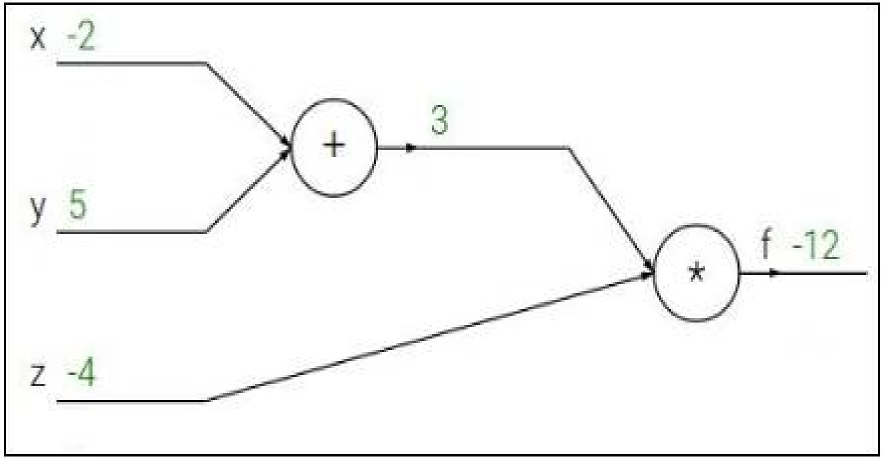
- Chain rule $$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$



Backpropagation – Chain Rule

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

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



Backpropagation – Chain Rule (Computational Graph)

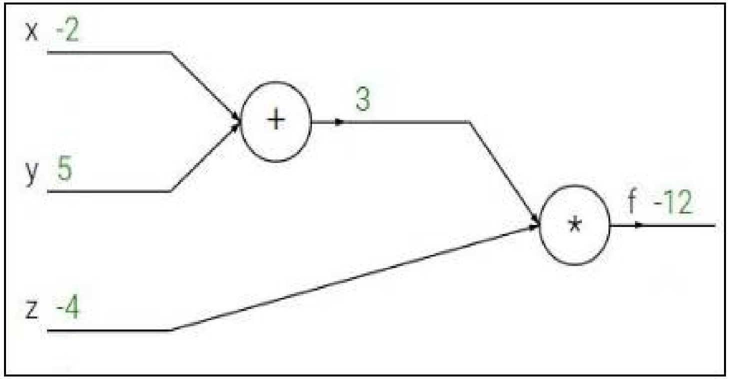
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation – Chain Rule (Computational Graph)

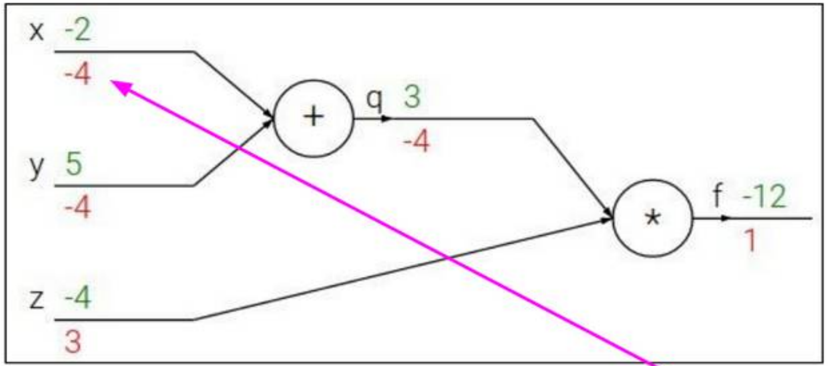
$$f(x, y, z) = (x + y)z$$

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



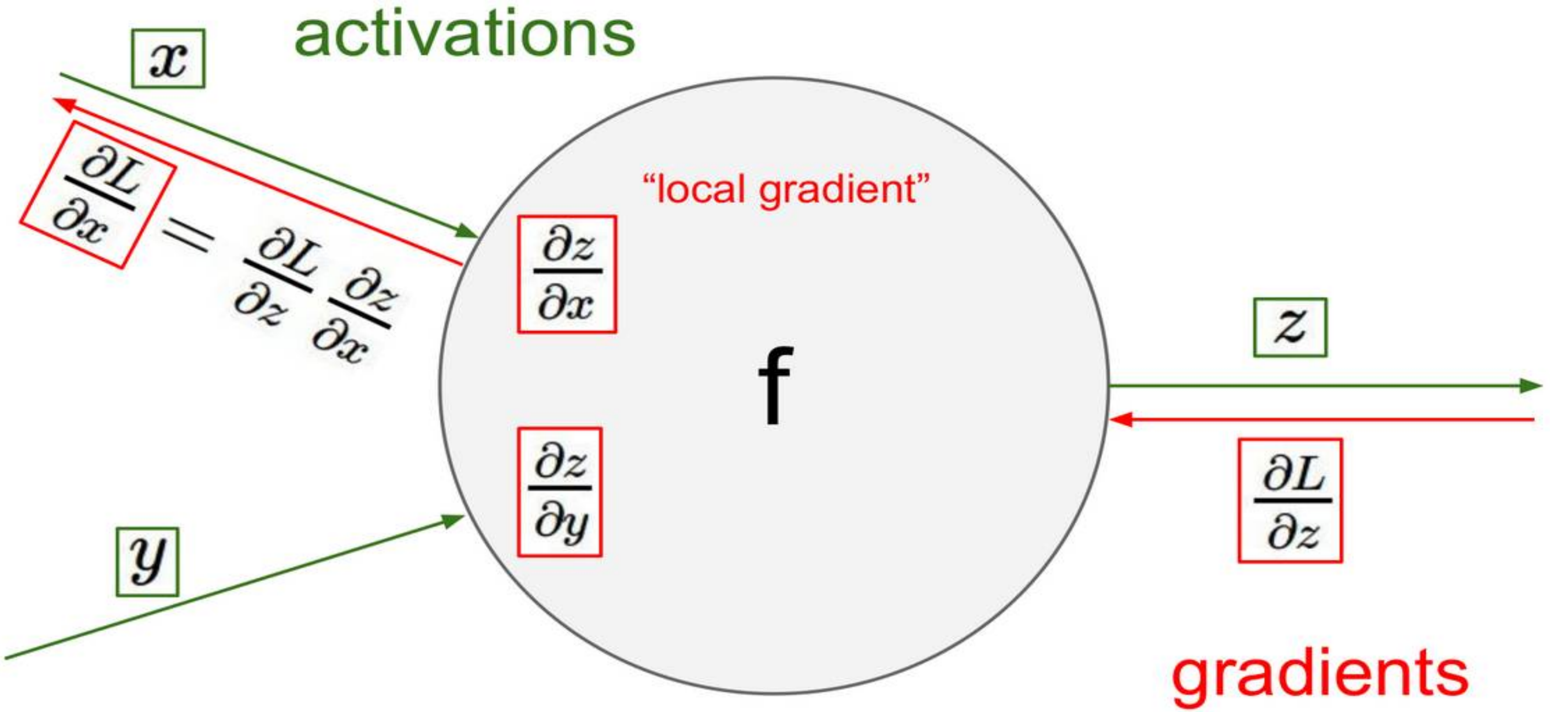
$$\frac{\partial f}{\partial x}$$

Chain rule:

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

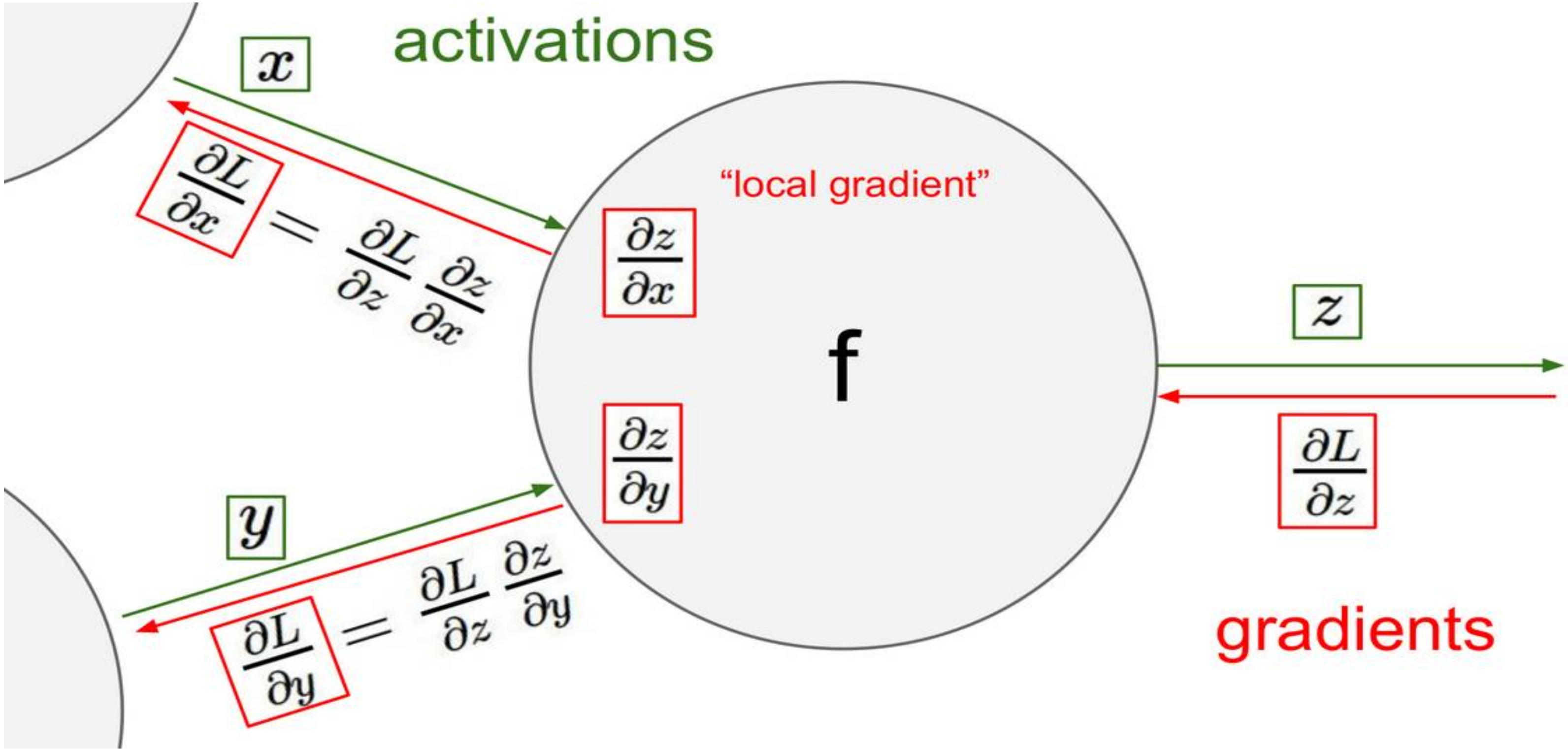
Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Backpropagation – Chain Rule (Computational Graph)



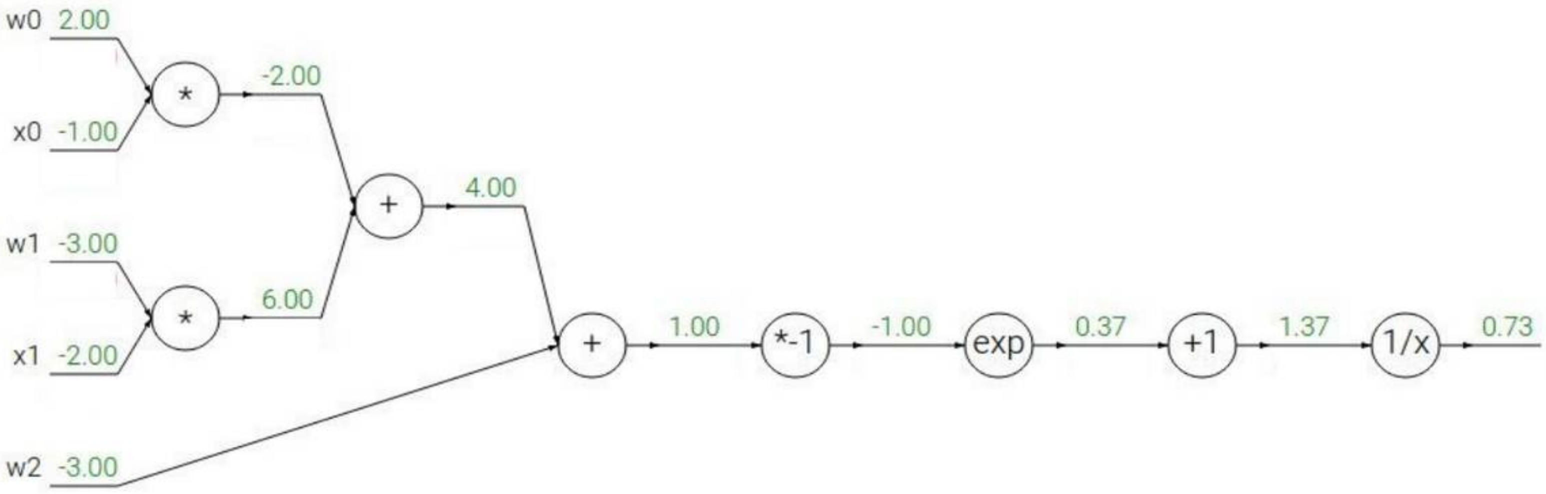
Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Backpropagation – Chain Rule (Computational Graph)



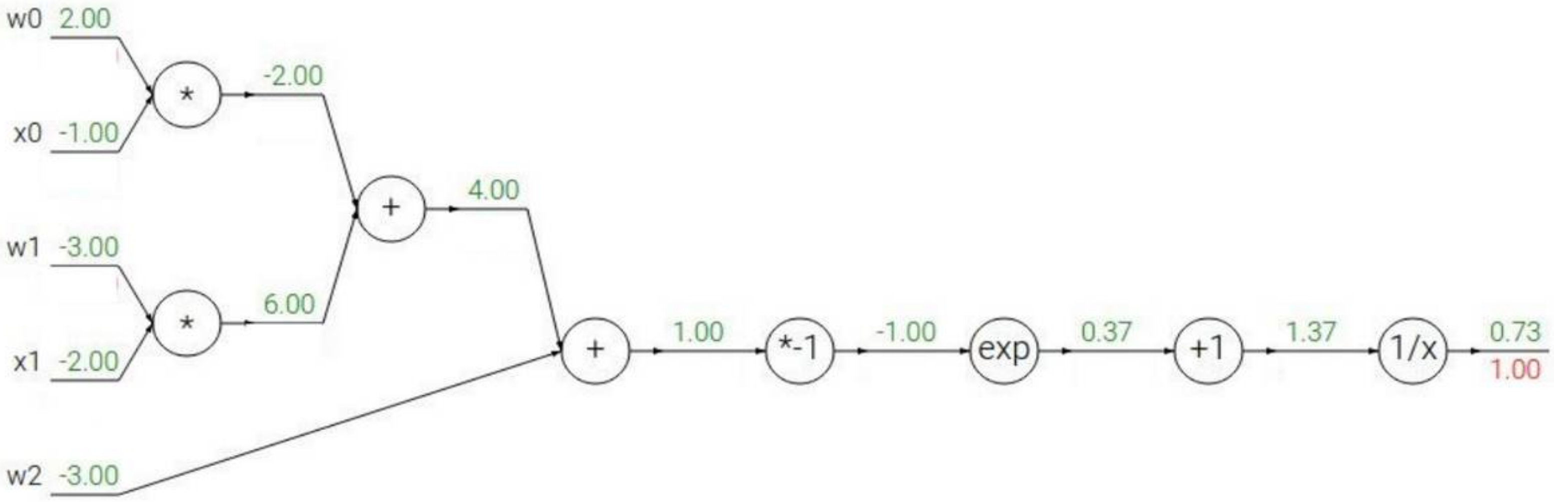
Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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



Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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



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

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

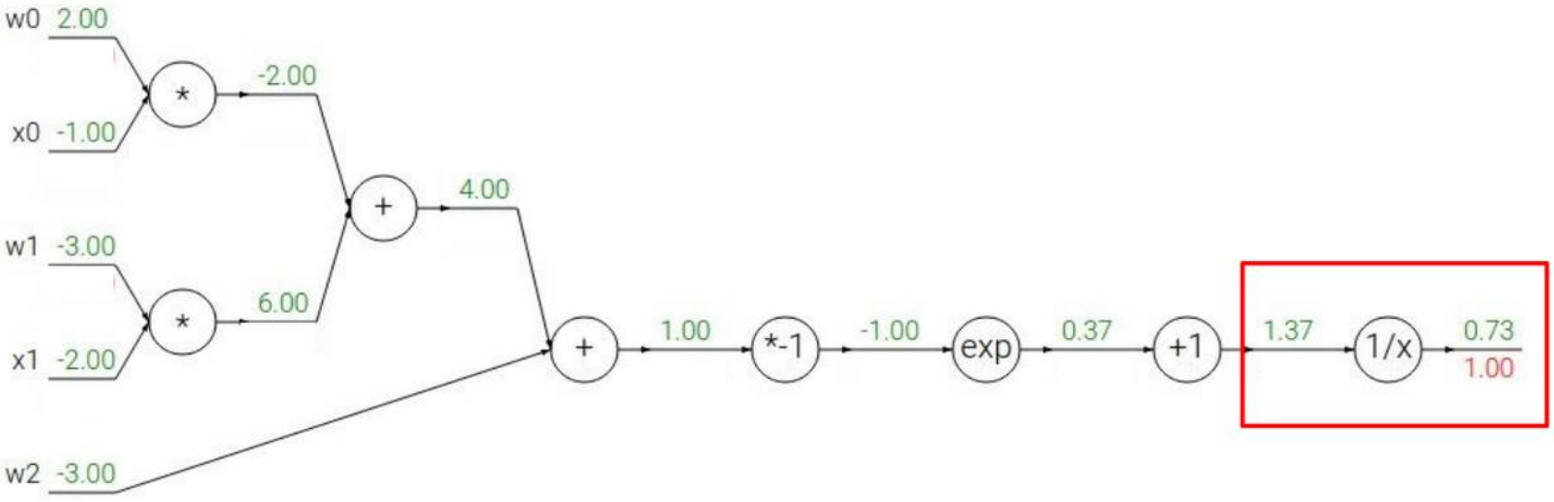
$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Another example:

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



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

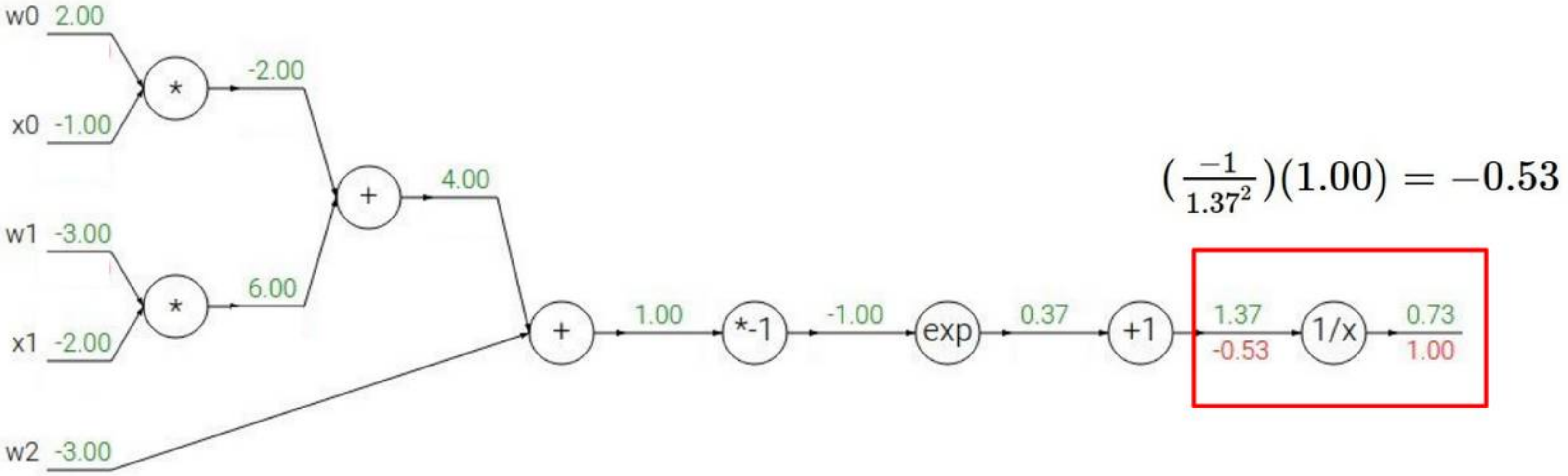
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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



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

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

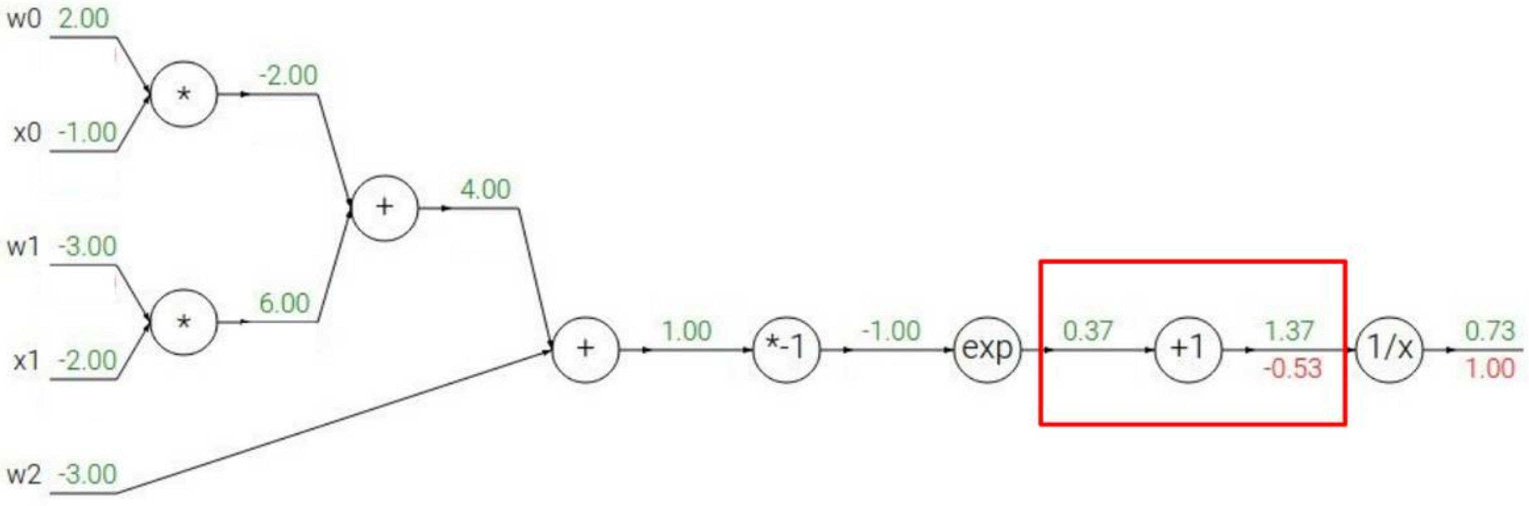
$$\boxed{f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2}$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Another example:

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



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

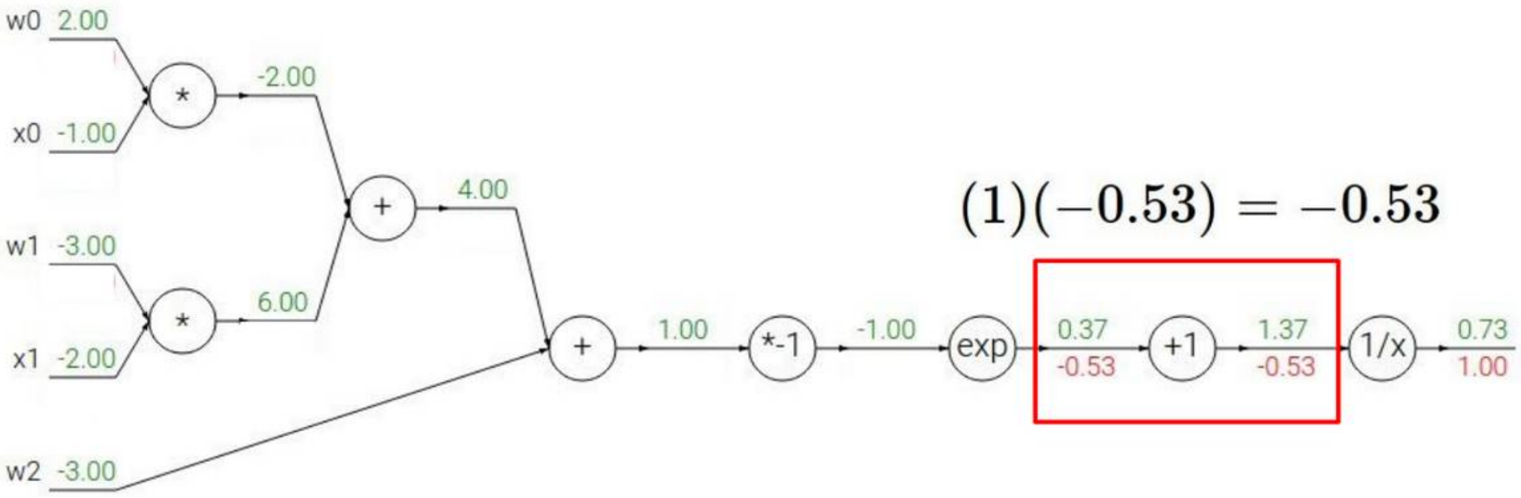
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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

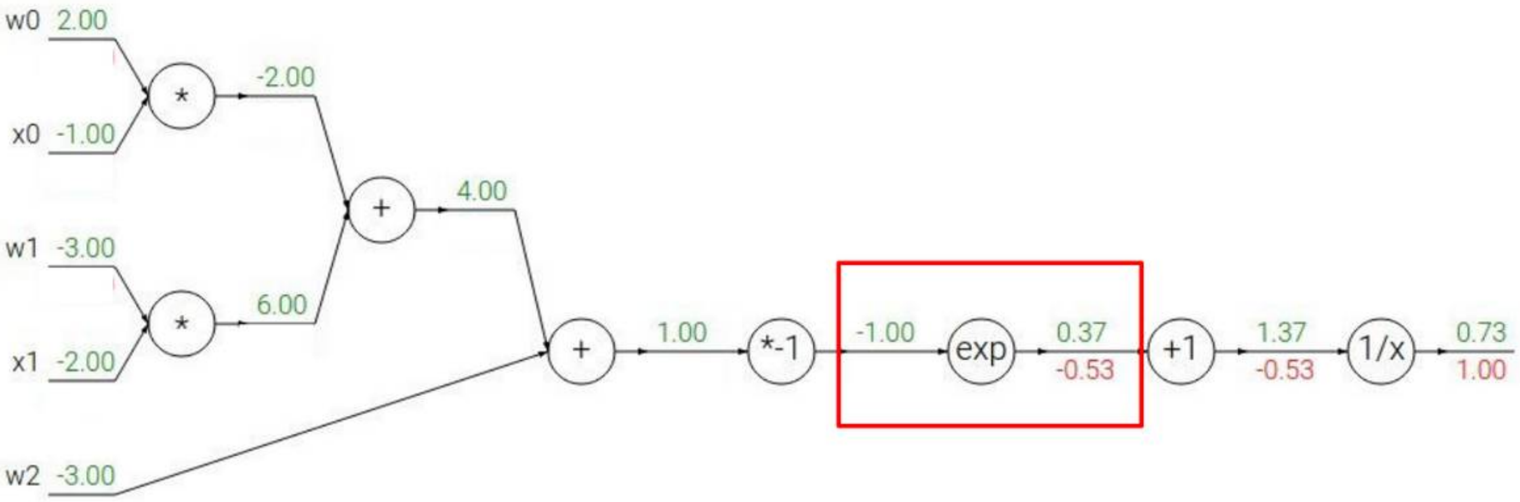


$$(1)(-0.53) = -0.53$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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



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

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

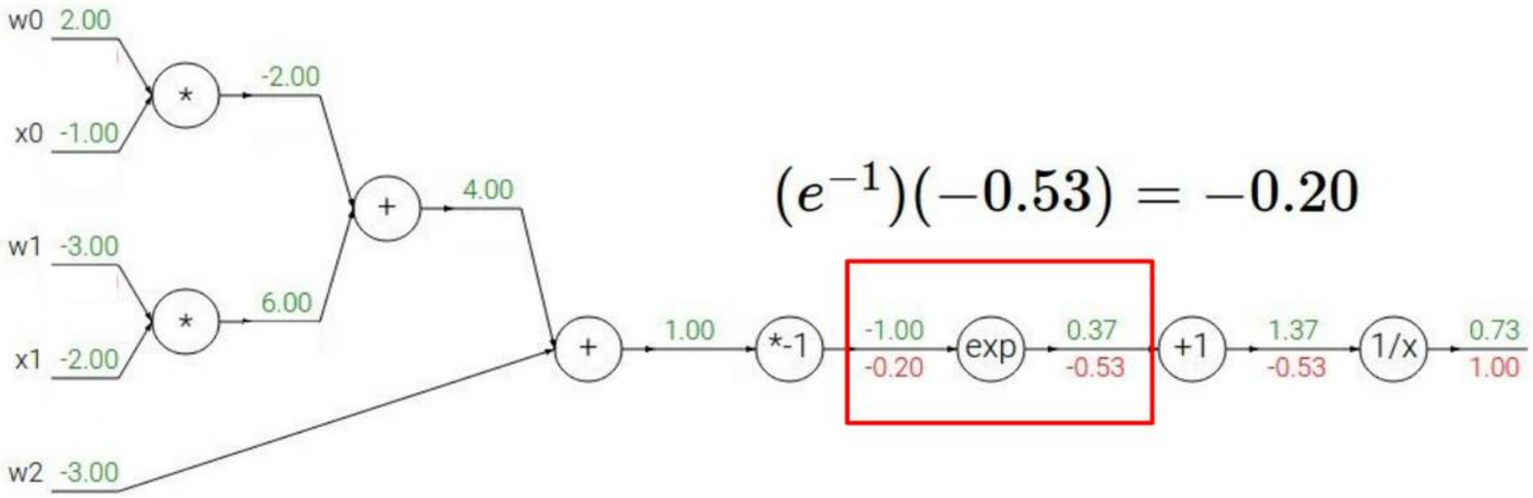
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Another example:

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



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

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

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

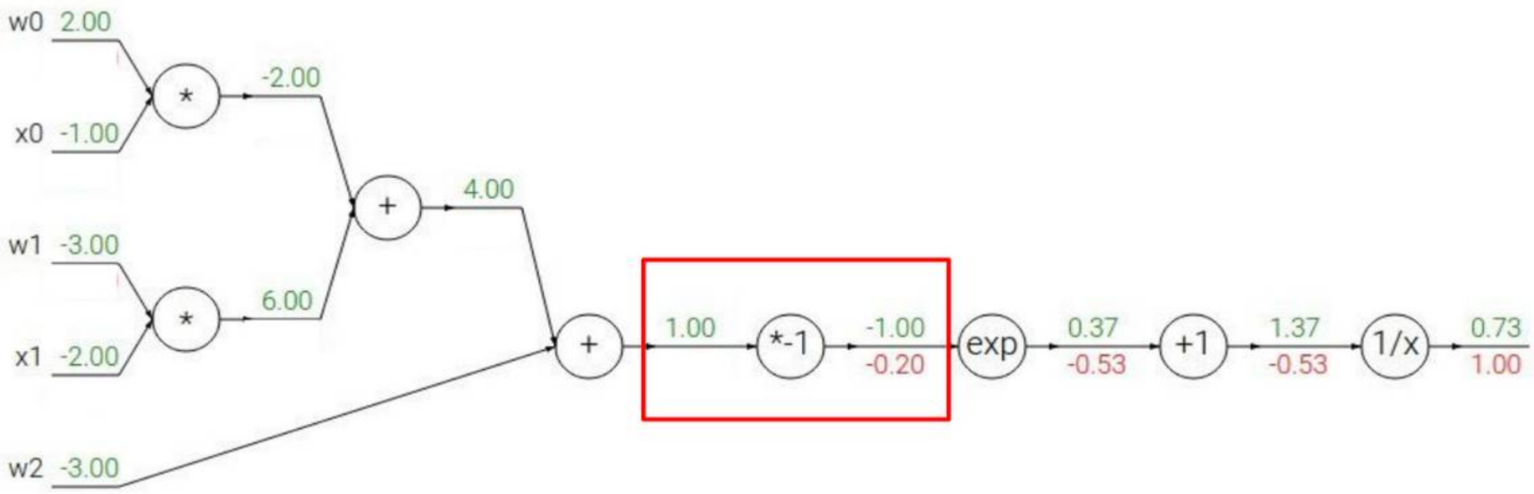
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Another example:

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



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

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

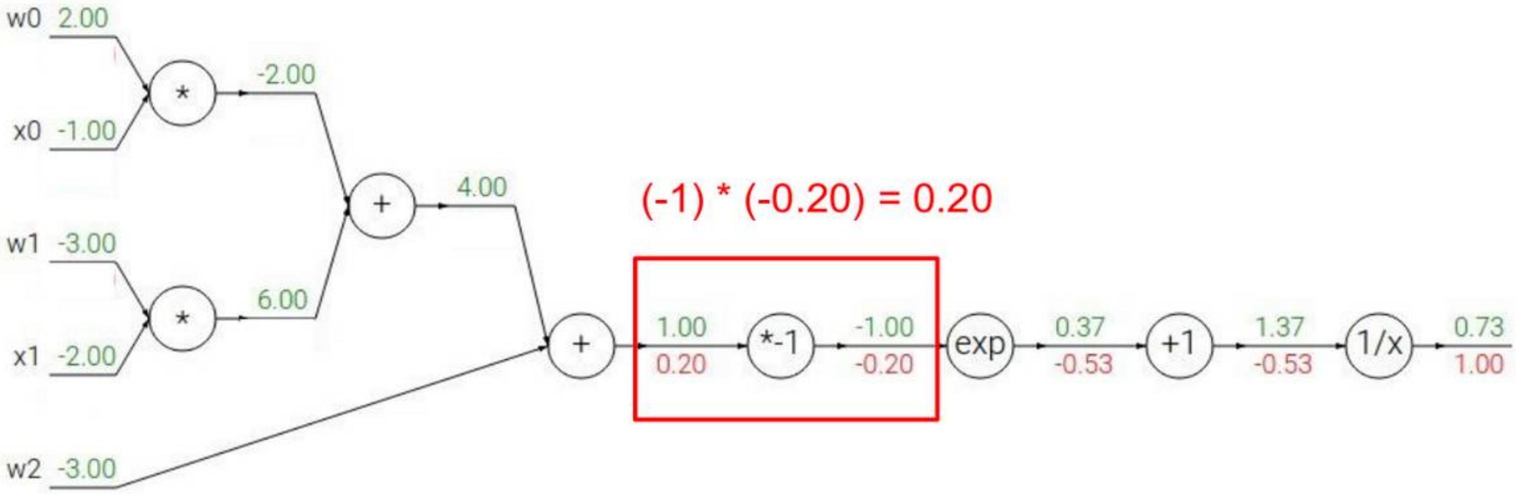
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Another example:

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



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

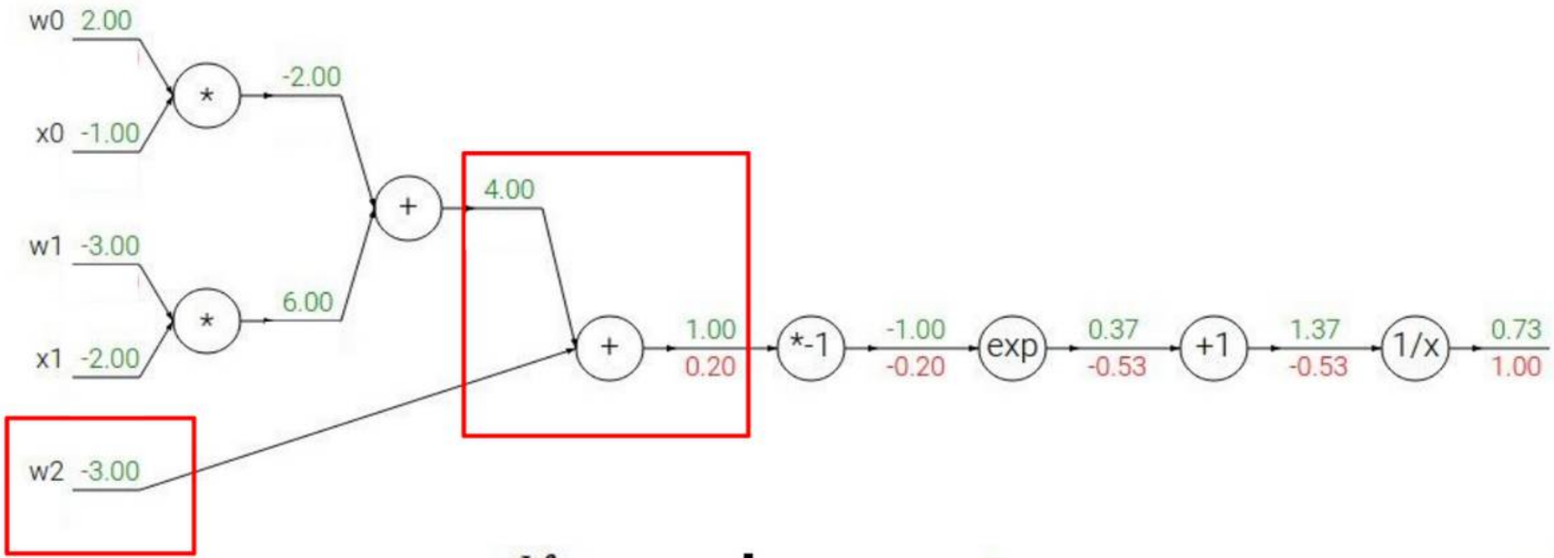
$$\boxed{f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a}$$

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

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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

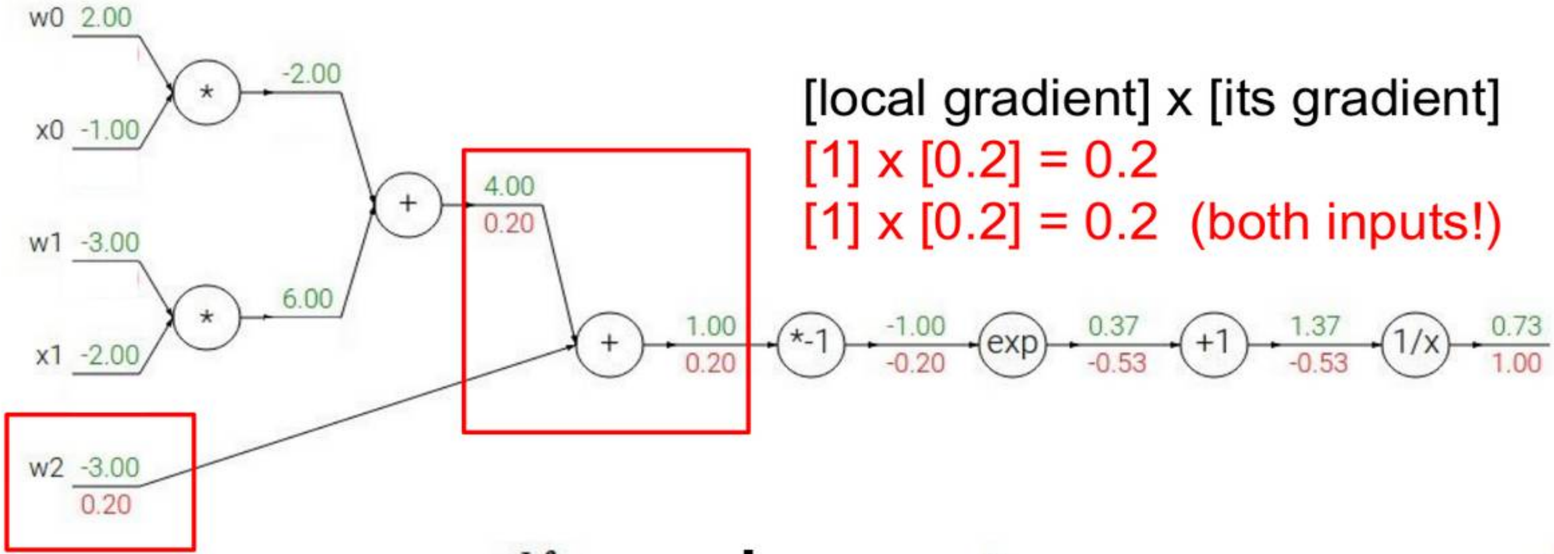


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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!)

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

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

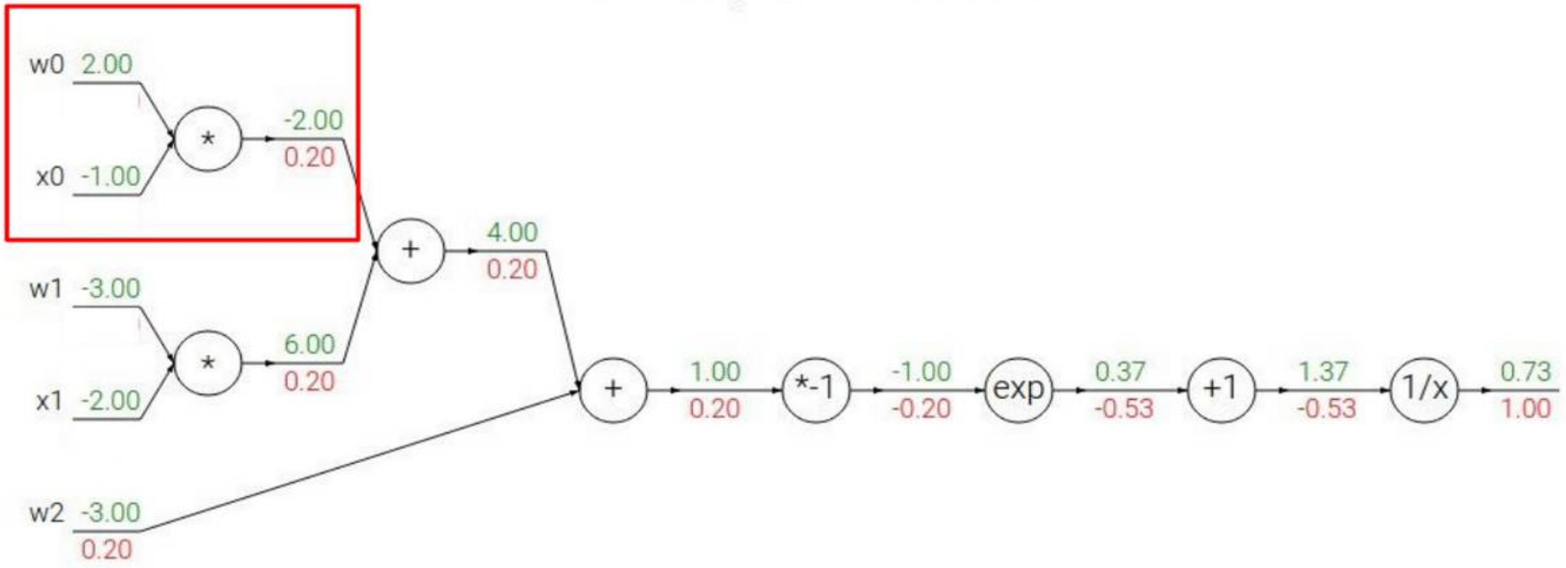
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Another example:

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



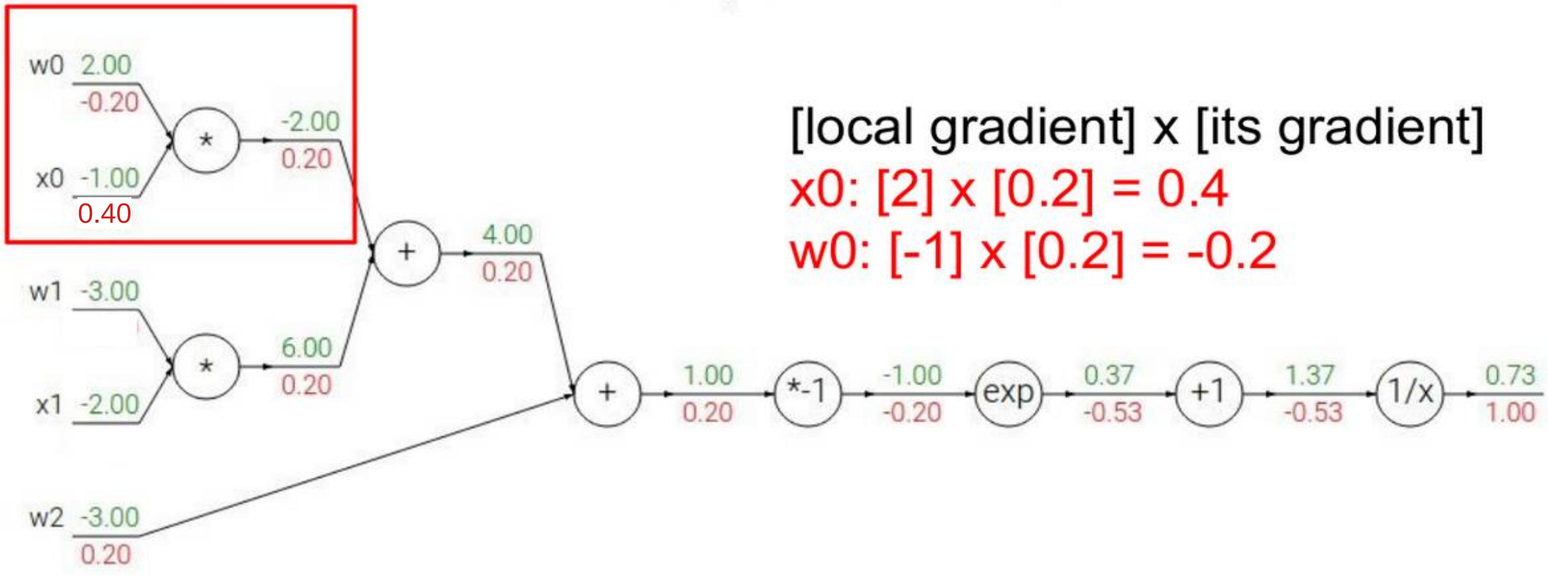
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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

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

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

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

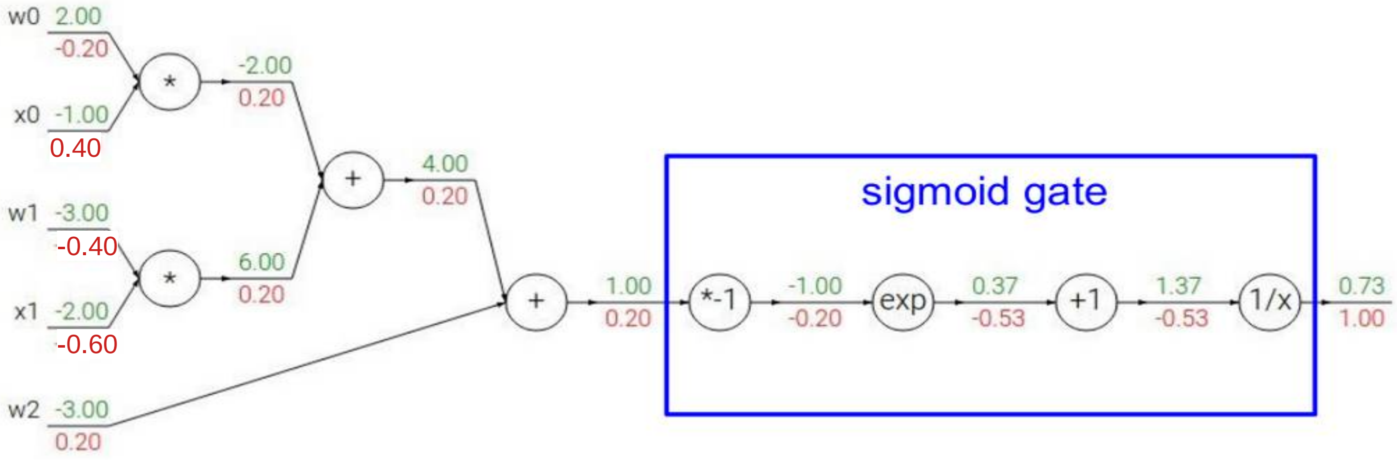
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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

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

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

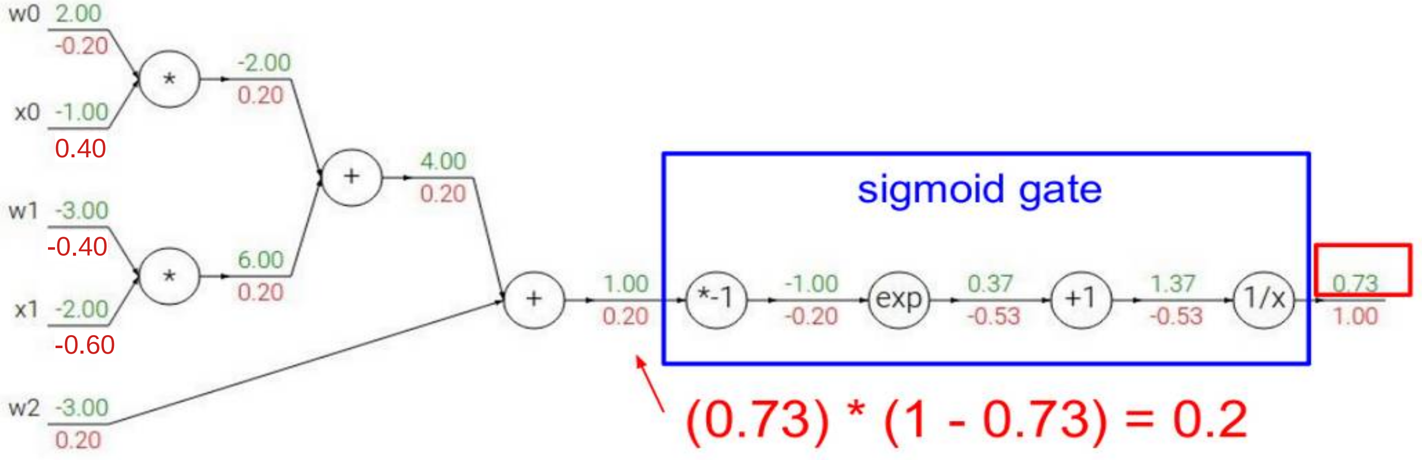


Source: CS231n 2016: Lecture 4 by Andrej Karpathy

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

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

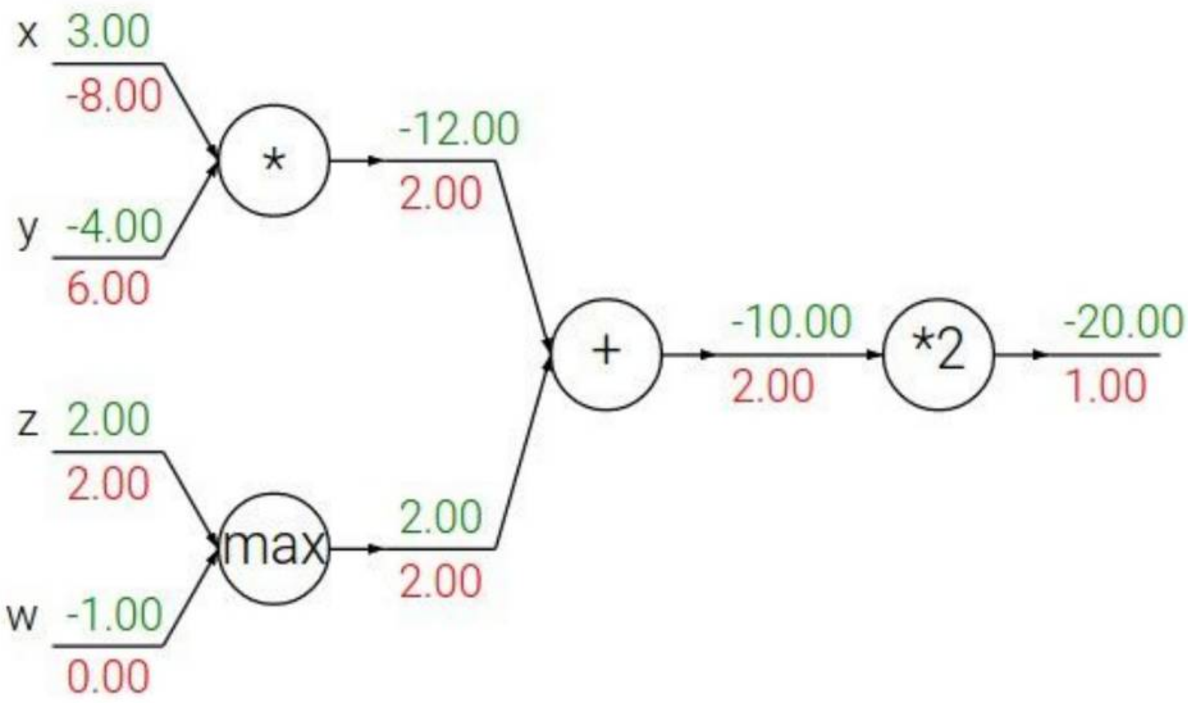
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Source: CS231n 2016: Lecture 4 by Andrej Karpathy

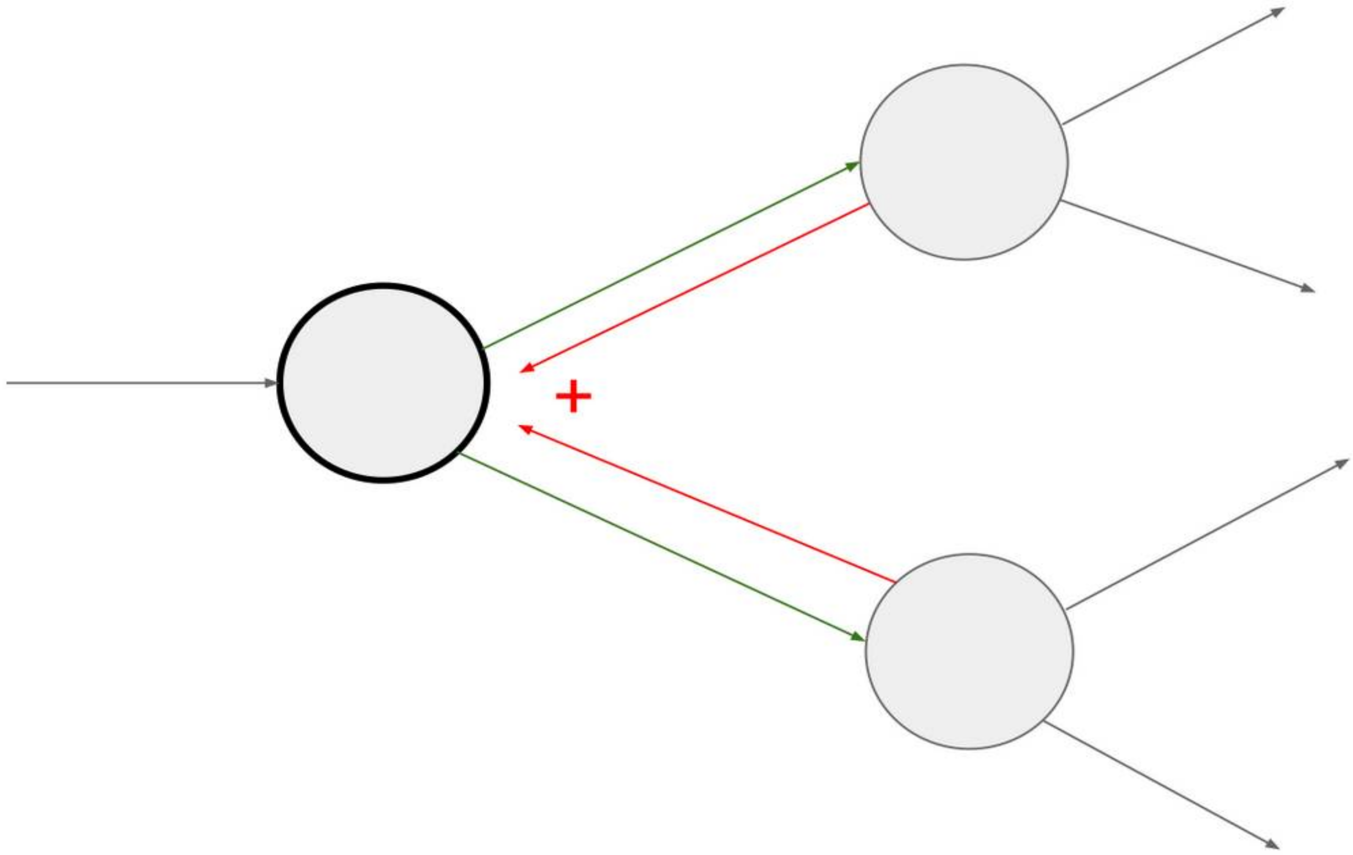
Patterns in backward flow

add gate: gradient distributor
max gate: gradient router
mul gate: gradient… “switcher”?



Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Gradients add at branches

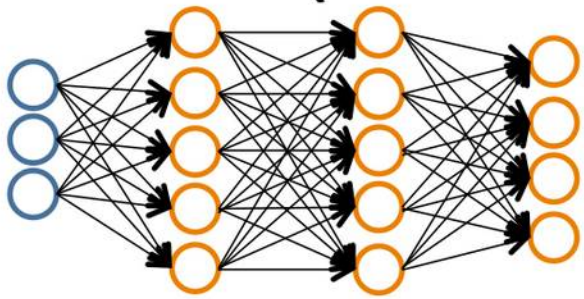


Source: CS231n 2016: Lecture 4 by Andrej Karpathy

Computational Graph using PyTorch Tensors

```
>>> import torch
>>> x = torch.tensor([-2], requires_grad=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: Only Tensors of floating point and complex dtype can require gradients
>>> x = torch.tensor([-2.], requires_grad=True)
>>> y = torch.tensor([-5.], requires_grad=True)
>>> z = torch.tensor([-4.], requires_grad=True)
>>> y = torch.tensor([5.], requires_grad=True)
>>> x
tensor([-2.], requires_grad=True)
>>> y
tensor([5.], requires_grad=True)
>>> z
tensor([-4.], requires_grad=True)
>>> q = x+y
>>> q
tensor([3.], grad_fn=<AddBackward0>)
>>> f = q*z
>>> f
tensor([-12.], grad_fn=<MulBackward0>)
>>> x.grad
>>> y.grad
>>> z.grad
>>> f.backward()
/home/arpan/anaconda3/envs/opencv-py3/lib/python3.8/site-packages/torch/autograd/__init__.py:1
 this may be due to an incorrectly set up environment, e.g. changing env variable CUDA_VISIBLE_
s to be zero. (Triggered internally at  /opt/conda/conda-bld/pytorch_1640811806235/work/c10/cu
  Variable._execution_engine.run_backward(
>>> z.grad
tensor([3.])
>>> y.grad
tensor([-4.])
>>> x.grad
tensor([-4.])
```

Neural Network (Classification)



$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

Binary classification

$y = 0 \text{ or } 1$

1 output unit

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

Given one training example (x, y):

Forward propagation:

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

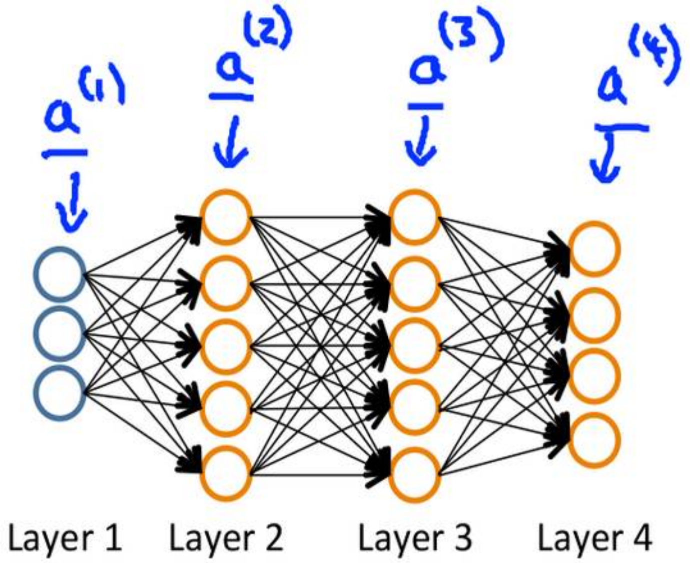
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$

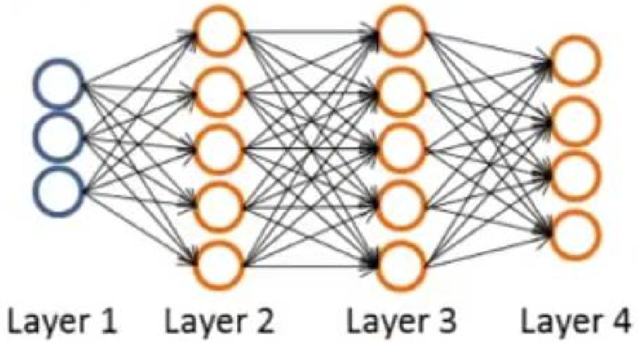
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$\delta^{(4)} = a^{(4)} - y$

Vectorized

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$a^{(3)} .* (1 - a^{(3)})$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

$a^{(2)} .* (1 - a^{(2)})$

(No $\delta^{(1)}$)

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ (ignoring λ; if $\lambda = 0$).

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

No $\delta^{(1)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)}$ if $j = 0$

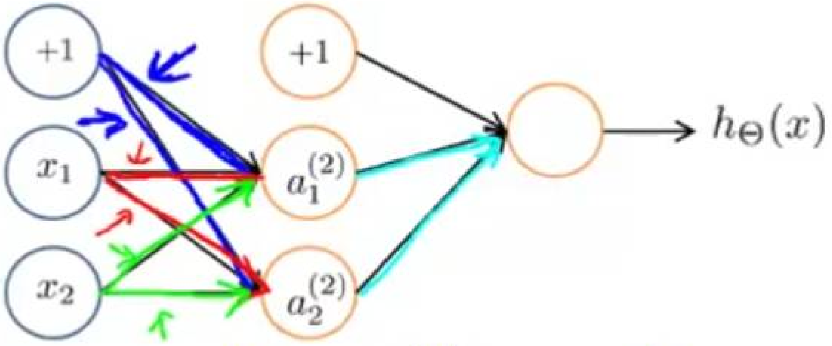
For bias term

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Homework

- Differentiate the loss function (BCE without taking regularization term) with respect to the parameters. (Partial Differentiation). - refer CS229 notes + lecture.

Zero initialization



$\rightarrow \Theta_{ij}^{(l)} = 0$ for all i, j, l.

$a_1^{(2)} = a_2^{(2)}$. Also $\delta_1^{(2)} = \delta_2^{(2)}$.

$$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta) \qquad \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$$

After each update, parameters corresponding to inputs going into each of two hidden units are identical.

$a_1^{(2)} = a_2^{(2)}$

Random initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

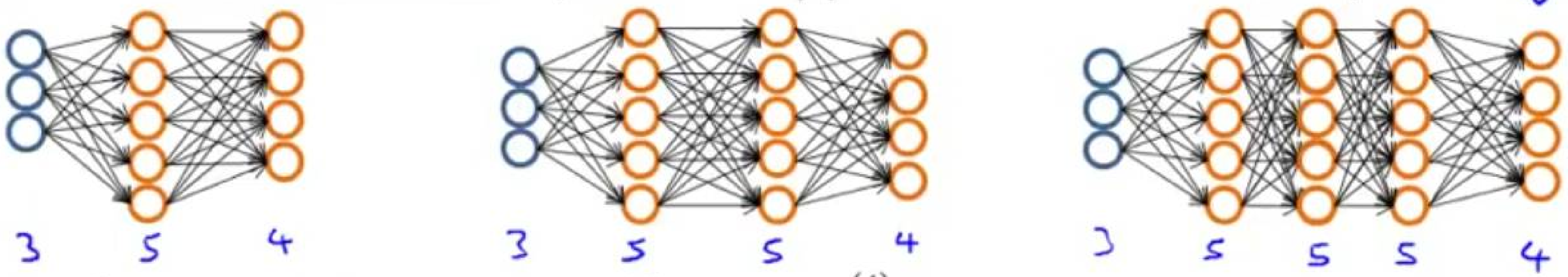
Uniform random

```
Theta1 =  rand(10,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;

Theta2 =  rand(1,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;
```

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



- No. of input units: Dimension of features $x^{(i)}$
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$

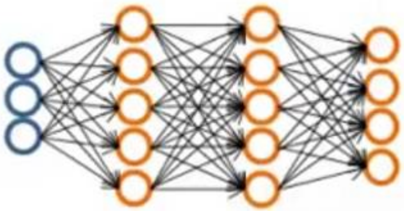
```
for i = 1:m
```

Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

compute $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$



Training a Neural Network

(optional step): Use gradient checking to compare the calculated $dJ/d\Theta$ using backprop vs using numerical estimate of gradient of $J(\Theta)$. After checking we must disable this step as it is computationally expensive.

5. Use gradient descent or advanced optimization method (e.g., LBFGS or conjugate gradient) with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ.

Note that $J(\Theta)$ is non-convex for neural networks. It is susceptible to local minima. But in practice that is not a big problem. The local minima values perform reasonably well in practice.

Backpropagation Example

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Stochastic Gradient Descent (SGD)

- Dataset can be too large
  - Can not apply gradient descent wrt. all data points.
- Randomly sample a data point
  - Perform gradient descent per sample and iterate.
  - Picking a subset of points: “*mini-batch*”

“Batchsize“

Randomly initialize starting W and pick learning rate γ

While not at minimum:

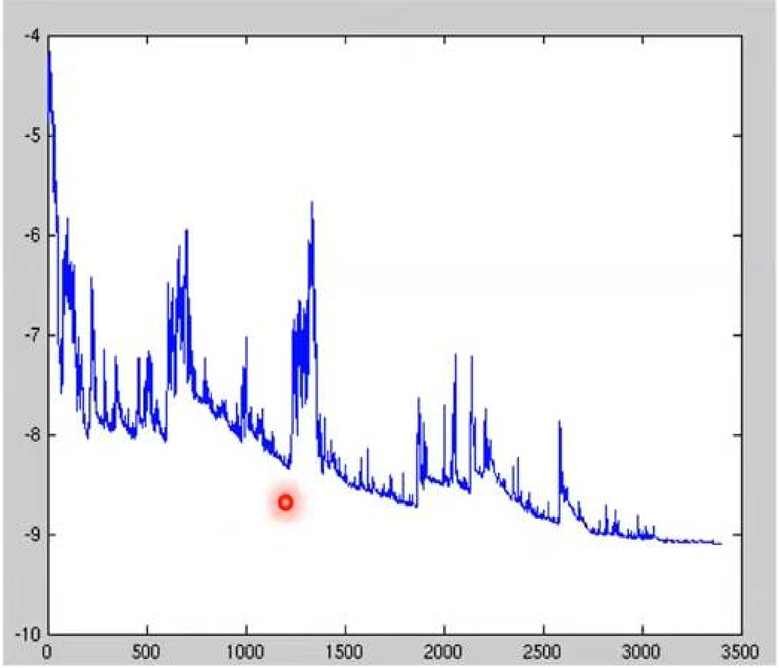
- Shuffle training set
- For each data point *i=1…n (maybe as mini-batch)*
  - *Gradient descent*

“Epoch“

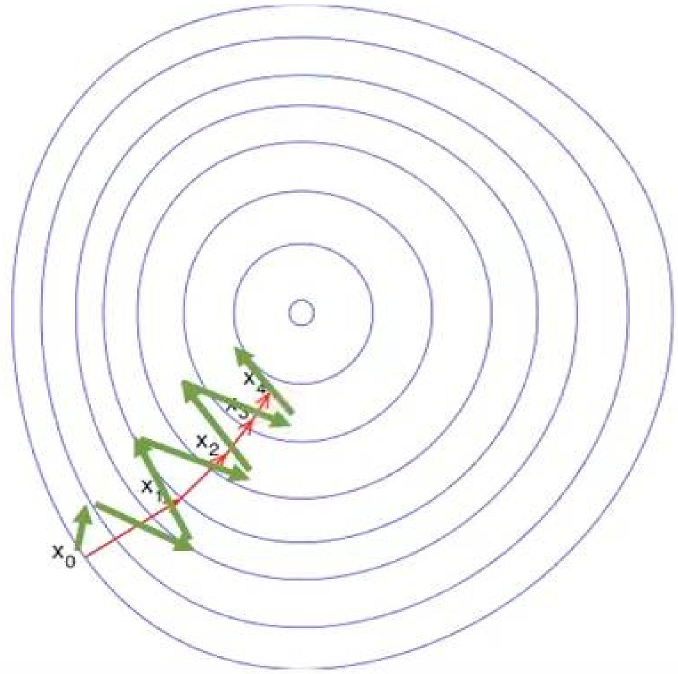
Stochastic Gradient Descent (SGD)

- Loss will not always decrease (locally)
  - As training data point is random.

- Still converges over time.



Gradient Descent Oscillations



Slow to converge to the (local) optimum

Momentum

- Adjust the gradient by a weighted sum of the previous amount plus the current amount.

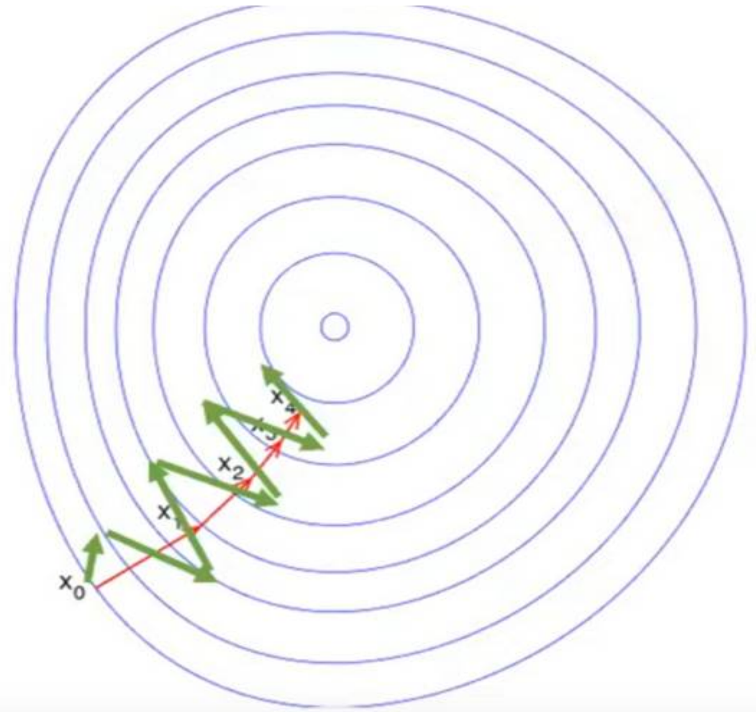
  Zig-zag is less, more smooth loss curve.

- Without momentum: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{\partial L}{\partial \boldsymbol{\theta}}$

- With momentum (new α parameter):

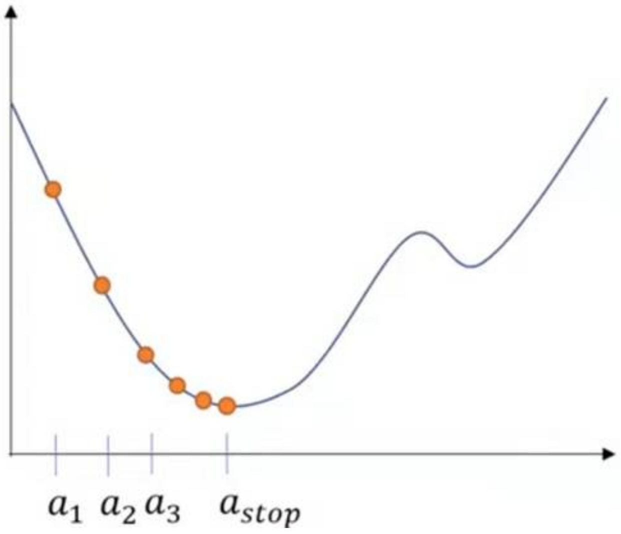
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \left(\alpha \underbrace{\left[\frac{\partial L}{\partial \boldsymbol{\theta}} \right]_{t-1}}_{\text{Prev gradient}} + \underbrace{\left[\frac{\partial L}{\partial \boldsymbol{\theta}} \right]_t}_{\text{Current gradient}} \right)$$

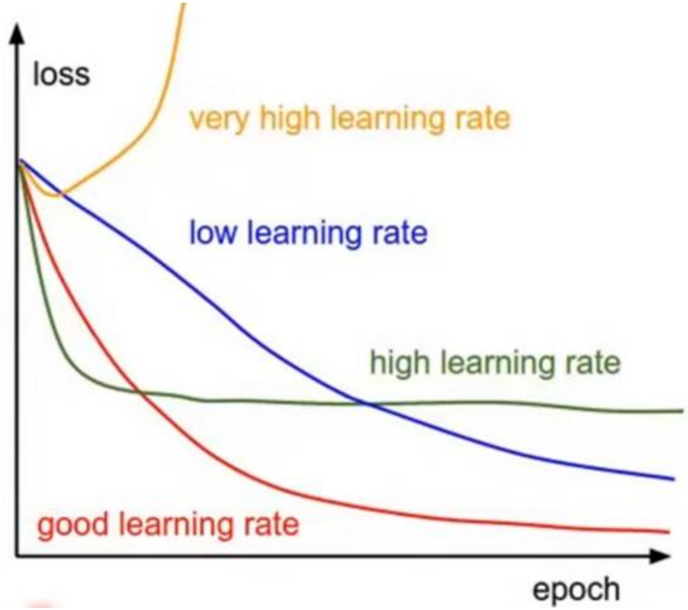
Lowering the learning rate



-Takes longer to get to the optimum

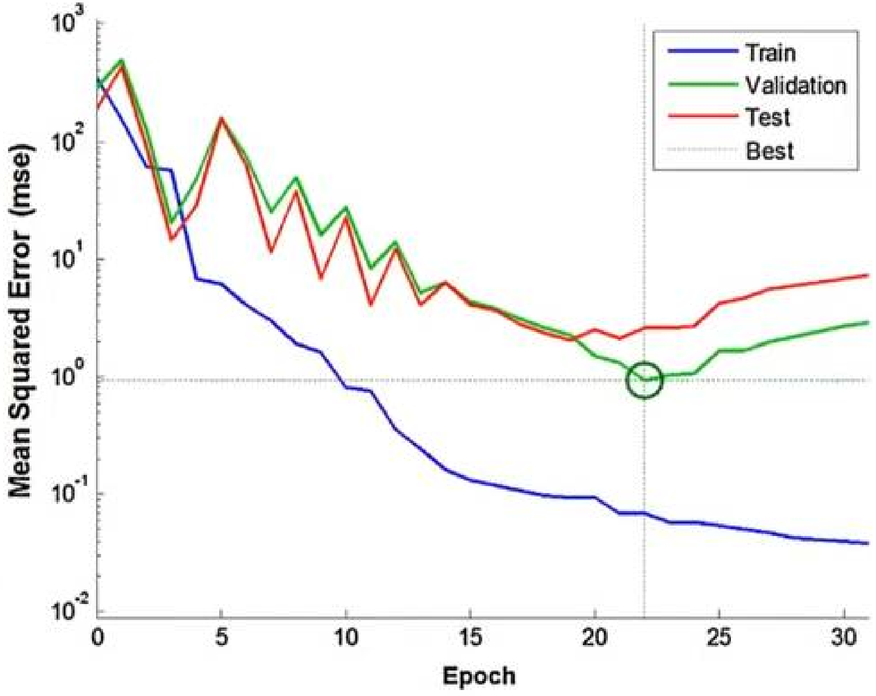
Learning rate





Overfitting

- Too many params = **overfitting**
- Not enough params = **underfitting**
- More data = less chance to overfit.



Ways to reduce overfitting

- Early stopping
- Regularization

Regularization

- Attempt to guide solution to not *overfit*.
- But still give freedom with many parameters.
- Penalize the use of parameters to prefer small weights

- Methods for regularization
  - Penalize the weight matrices
    - L1 or L2 norm added to loss terms
  - Dropout

Regularization: Penalizing the weights

- Add a cost to having high weights

Weight decay Regularization

Loss

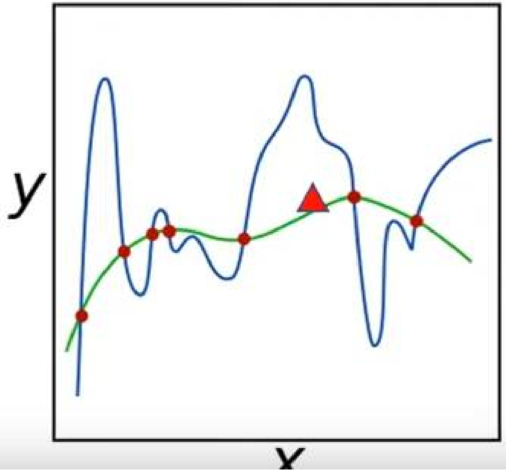
$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), yi) + \lambda R(W)$$

- In common use,

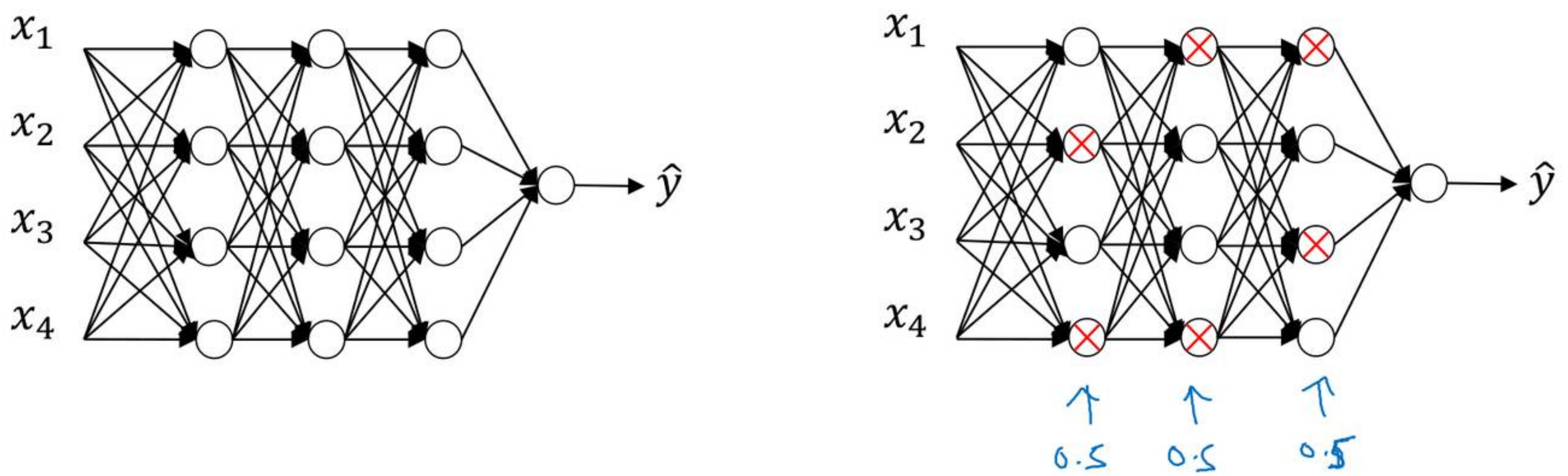
  L1 Norm – $R(W) = \sum_i \sum_j |Wij|$

  L2 Norm – $R(W) = \sum_i \sum_j W^2{}_{ij}$

  Elastic net – $R(W) = \sum_i \sum_j \beta W^2{}_{ij} + |Wij|$



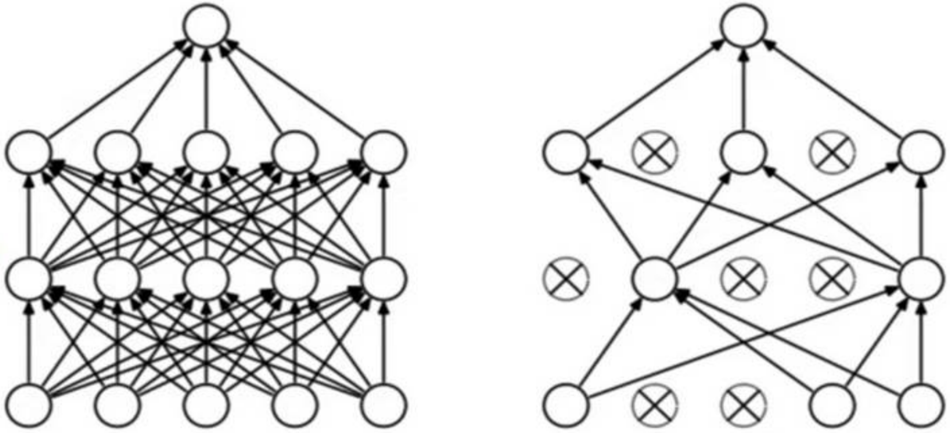
Regularization: Dropout



Andrew Ng

Dropout

- Stochastically switch neurons off
  - Each neuron is set to 0 with probability p
  - Hidden units cannot co-adapt to each other
  - Units are useful independently
- Hyperparameter
  - P is usually set to 0.5



(a) Standard Neural Net (b) After applying dropout.

Summary: Training Steps

- Define network
- Loss function
- Initialize network parameters
- Get training data
  - Prepare batches
- Feedforward one batch
  - Compute loss
  - Backpropagate gradients
  - Update network parameters

References

- CS229 Stanford Course by Andrew Ng. Link
- Blog by Christopher Olah: https://colah.github.io/posts/2015-08-Backprop/
- Lecture 4 : CS231n Convolutional Neural Networks Course by Andrej Karpathy, Fei Fei Li: Link
- Computer Vision Lectures by Dr. Yogesh Rawat - UCF CRCV.

End of Lecture