
بسم الله الرحمن الرحیم

Deep Learning

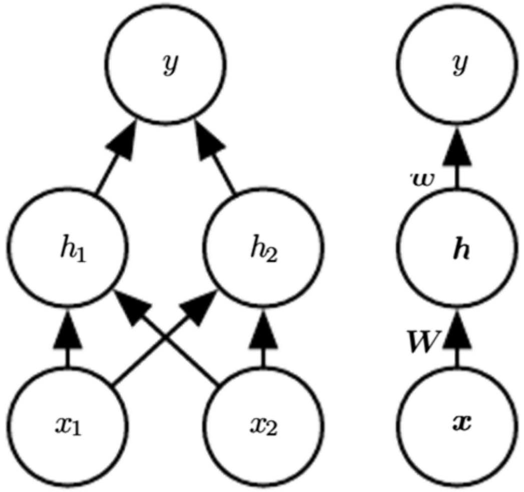
Mohammad Reza Mohammadi

2021

How to compute gradients?

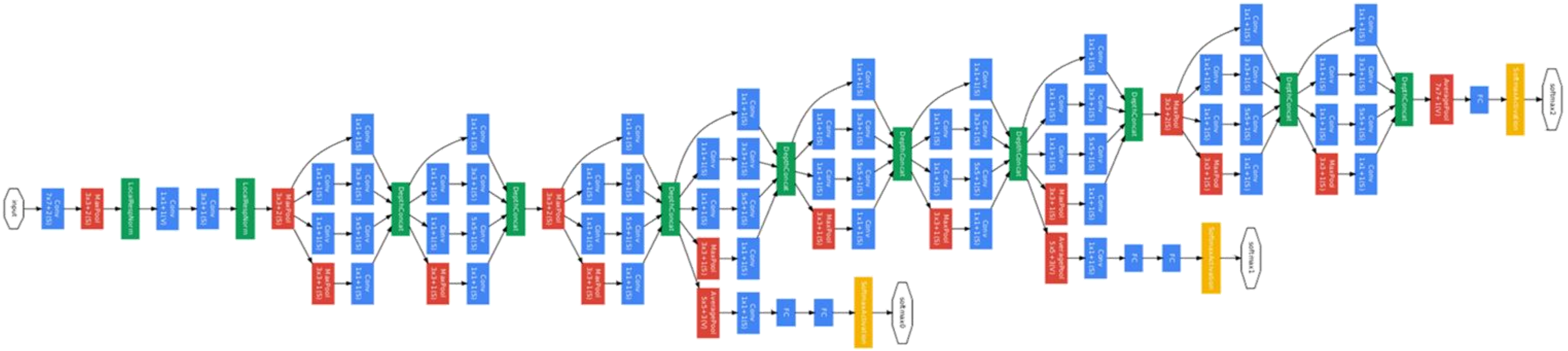
- $h_i = g\left(\boldsymbol{x}^T \boldsymbol{W}_{:,i} + c_i\right) = f^{(1)}(\boldsymbol{x})$
- $y = f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right) = f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c}\} + b$
- $J(\boldsymbol{\theta}) = \frac{1}{N} \sum \left(f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b)\right)^2$

- If we can compute $\frac{\partial J}{\partial \boldsymbol{W}}$, $\frac{\partial J}{\partial \boldsymbol{c}}$, $\frac{\partial J}{\partial \boldsymbol{w}}$ and $\frac{\partial J}{\partial b}$ then we can learn the parameters
- What if we want to change loss?
  - Need to re-derive from scratch



How to compute gradients?

- Very tedious: Lots of matrix calculus, need lots of paper



Backpropagation: a simple example

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

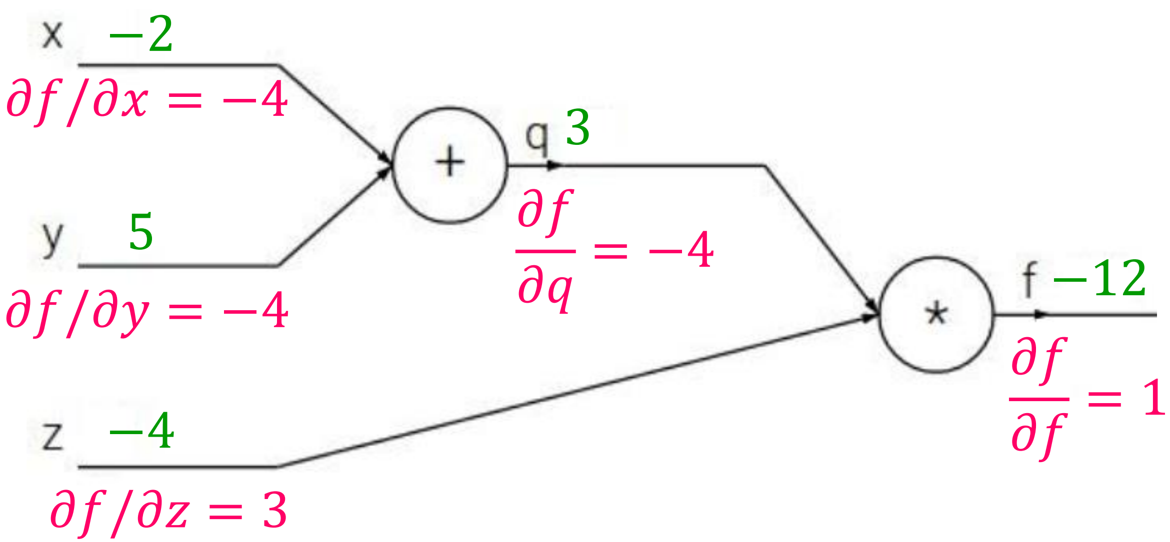
e.g., $x = -2,\ y = 5,\ z = -4$

$$q = x + y \qquad \Rightarrow f = qz$$

we want $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$

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

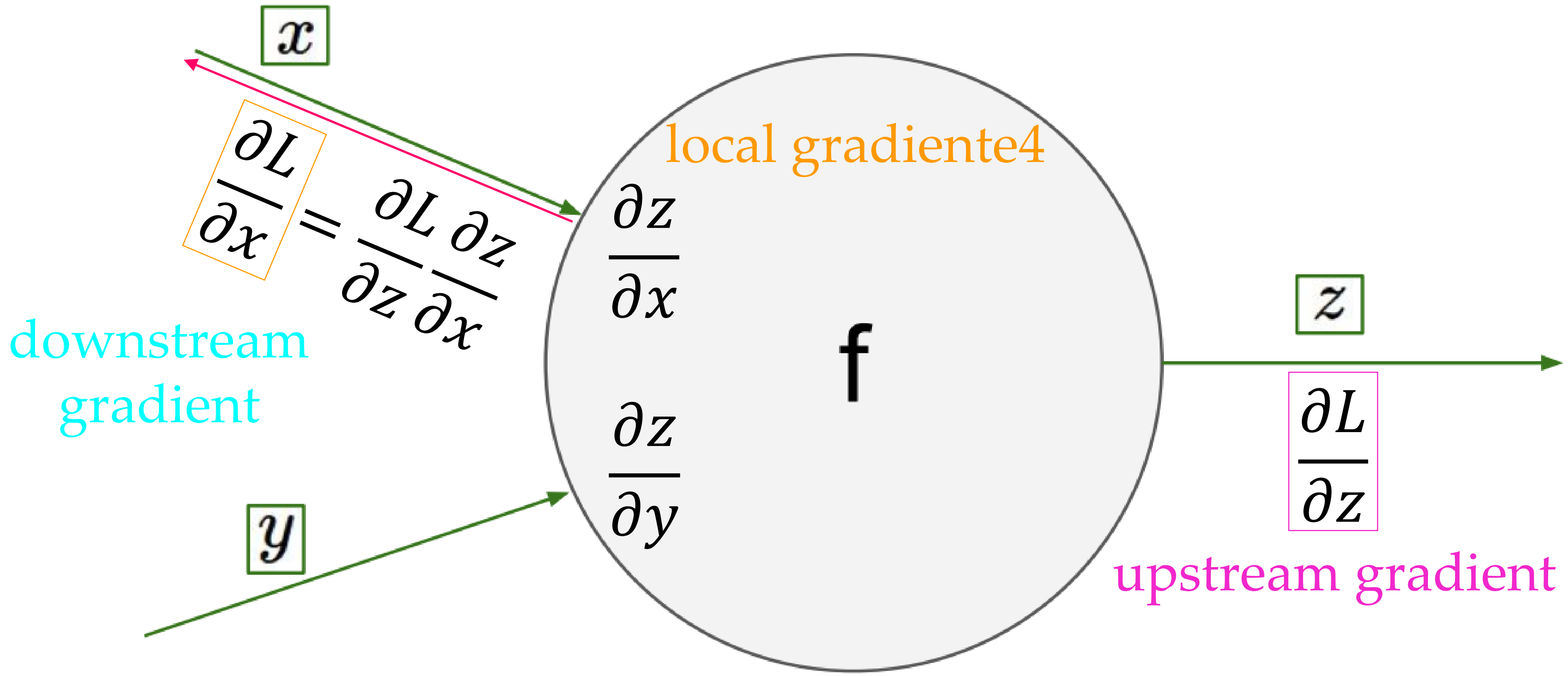
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$



Chain rule

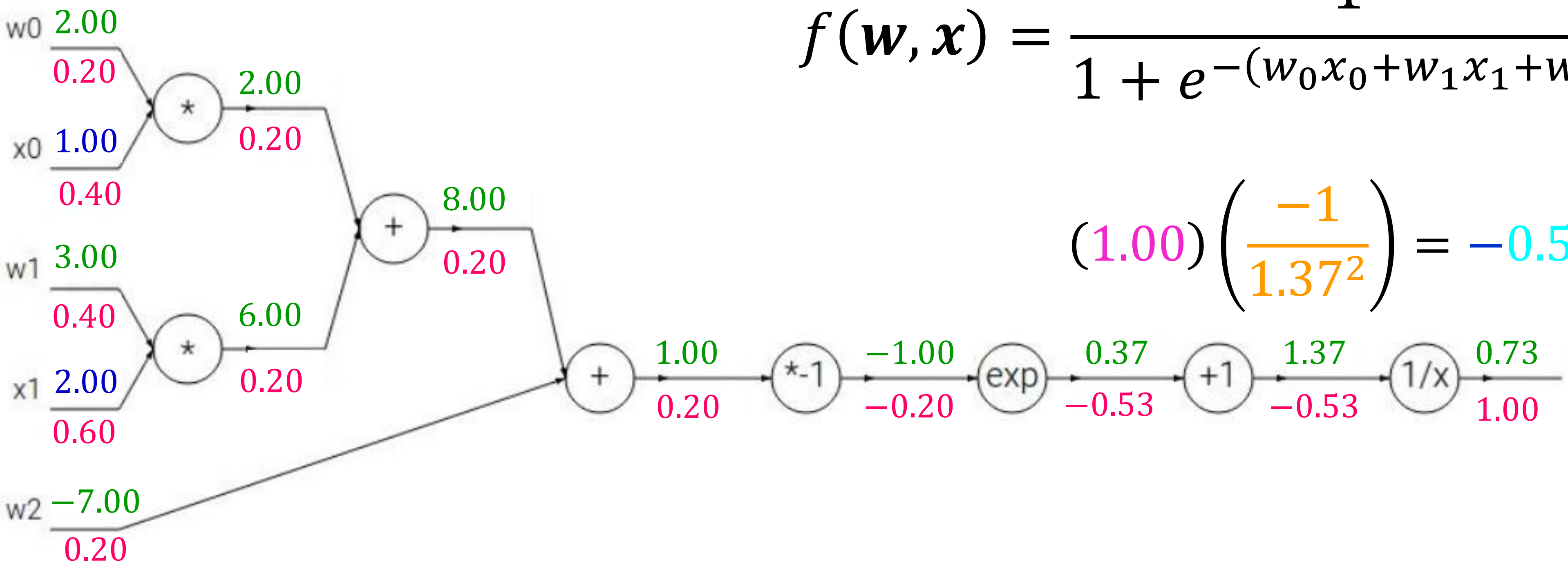
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Backpropagation



Backpropagation: Linear + Sigmoid

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

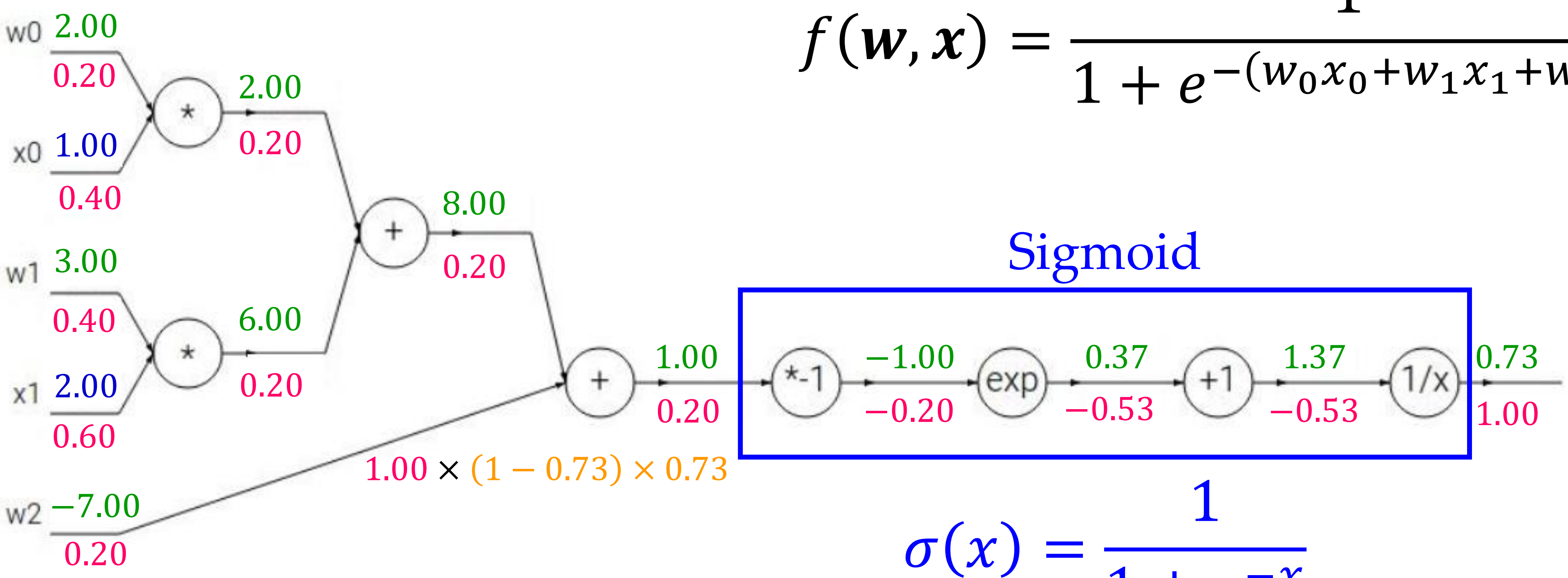


$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

$$\frac{\partial(ax)}{\partial x} = a \qquad \frac{\partial(c+x)}{\partial x} = 1 \qquad \frac{\partial(e^x)}{\partial x} = e^x \qquad \frac{\partial(1/x)}{\partial x} = \frac{-1}{x^2}$$

Backpropagation: Linear + Sigmoid

$$f(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

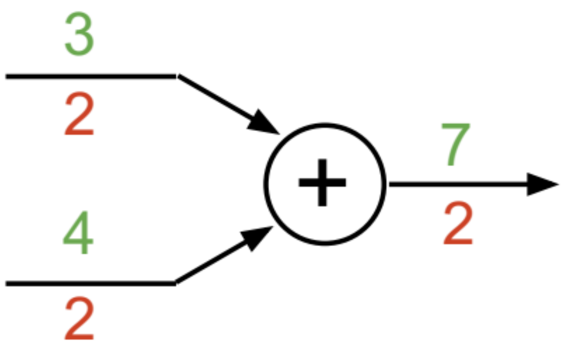


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

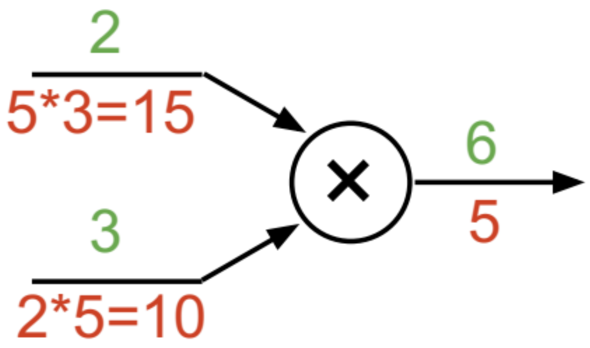
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1 - 1 + e^{-x}}{1 + e^{-x}} \frac{1}{1 + e^{-x}} = \big(1 - \sigma(x)\big)\sigma(x)$$

Patterns in gradient flow

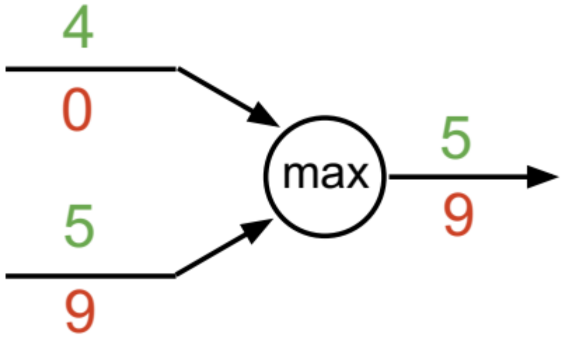
add gate: gradient distributor



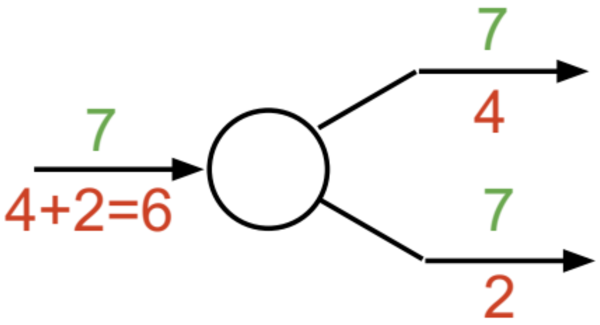
mul gate: “swap multiplier”



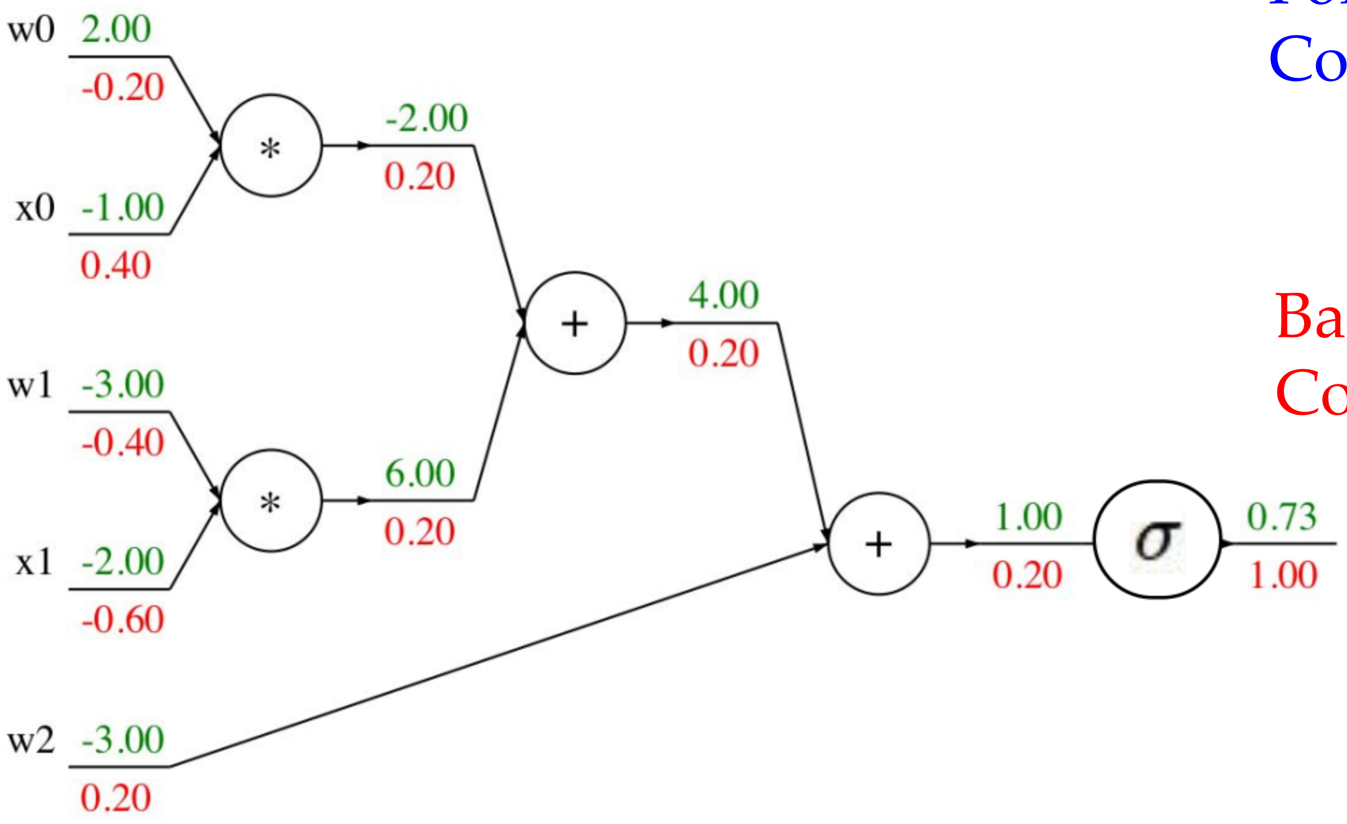
max gate: gradient router



copy gate: gradient adder



Backpropagation



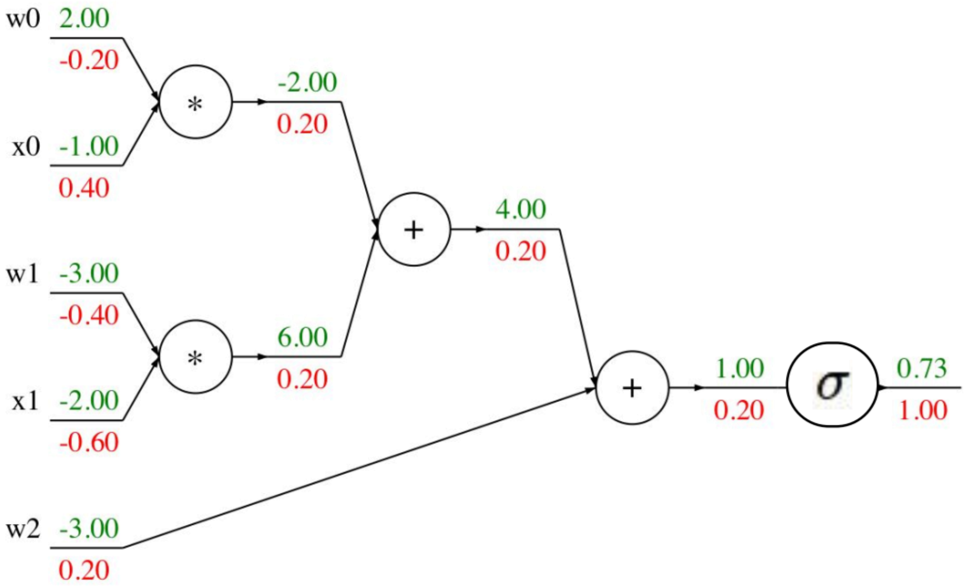
Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute grads

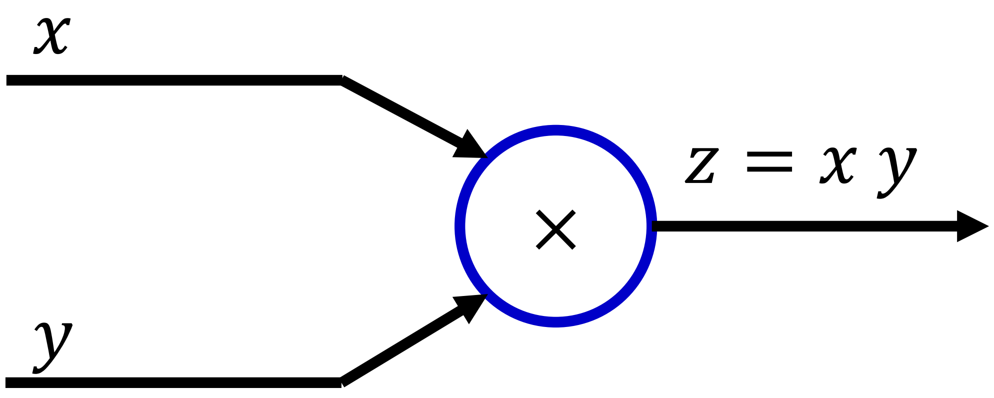
```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Modular implementation




```python
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
```

Multiply Gate




```
class MultuplyGate(object):
  """
  x,y are scalars
  """
  def forward(x,y):
    z = x*y
    self.x = x  # Cache
    self.y = y  # Cache
    # We cache x and y because we know that the derivatives contains them.
    return z
  def backward(dz):
    dx = self.y * dz         #self.y is dx
    dy = self.x * dz
    return [dx, dy]
```

Caffe layers

BVLC / caffe

Watch 2,231 | Star 26,174 | Fork 15,942

Code | Issues 620 | Pull requests 259 | Projects 0 | Wiki | Insights

Branch: master ▾ caffe / src / caffe / layers /

Create new file | Find file | History

Noiredd Merge branch 'master' into patch_1 — Latest commit 828dd10 on Aug 21

absval_layer.cpp	dismantle layer headers	3 years ago
absval_layer.cu	dismantle layer headers	3 years ago
accuracy_layer.cpp	Added count==0 safeguard to CPU accuracy calculation	a year ago
accuracy_layer.cu	explain use of scratch diffs in comments	10 months ago
argmax_layer.cpp	dismantle layer headers	3 years ago
base_conv_layer.cpp	Shape mismatch CHECK logging improvements	2 years ago
base_data_layer.cpp	Using default from proto for prefetch	2 years ago
base_data_layer.cu	Switched multi-GPU to NCCL	2 years ago
batch_norm_layer.cpp	CPU BatchNormLayer: replace powx with sqr and sqrt	2 years ago

Sigmoid layer

```
template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}
```

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

```
template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}
```

$$\frac{d\sigma(x)}{dx} = \sigma(x)\big(1 - \sigma(x)\big)$$

Custom layer in Keras

Keras

Search Keras documentation...

- About Keras
- Getting started
- Developer guides
- Keras API reference
- Code examples
- Why choose Keras?
- Community & governance
- Contributing to Keras

» Developer guides / Making new layers and models via subclassing

Making new layers and models via subclassing

Author: fchollet
Date created: 2019/03/01
Last modified: 2020/04/13
Description: Complete guide to writing `Layer` and `Model` objects from scratch.

View in Colab • **GitHub source**

https://colab.research.google.com/drive/1R7b8nj6xxKqj2ELeUFFpXx582zDM5pnQ?usp=sharing