

Recall

$$s = f(x; W) = Wx$$

score function

$$L_i = \sum_j \max(0, s_j - y_j + 1)$$

SVM loss

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

data loss + regularization

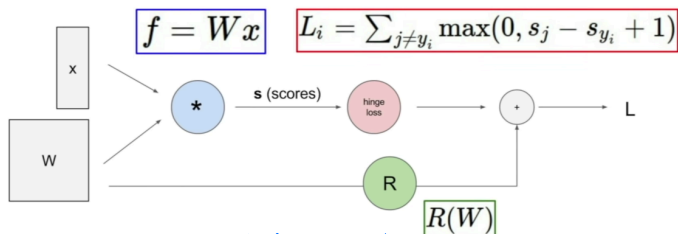
want $\nabla W L$

solve it with optimization

analytic gradient
numerical gradient

analytic gradient (thru computational graphs)

Computational graphs



important to calculate gradient for neural network (complex function)

e.g. find gradient

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

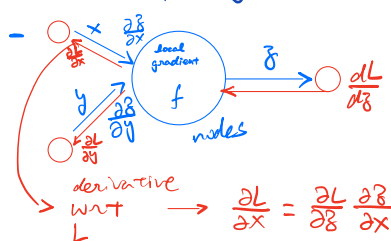
$$@ x=-2, y=5, z=-4$$

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

$$f = z \times (x+y) \quad \frac{\partial f}{\partial z} = z \quad \frac{\partial f}{\partial x} = z \quad \frac{\partial f}{\partial y} = z$$

$$\text{find } \frac{df}{dx} \quad \frac{df}{dy} \quad \frac{df}{dz}$$

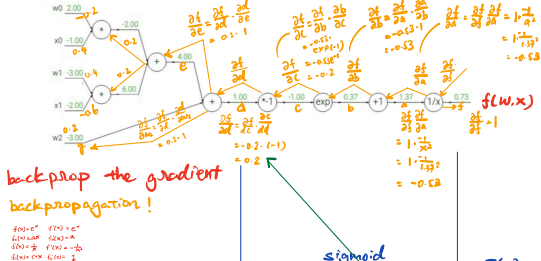
back-propagation



Another example:

forward the values

$$f(w, x) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



backprop the gradient backpropagation!

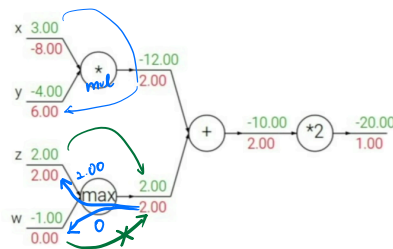
sigmoid

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

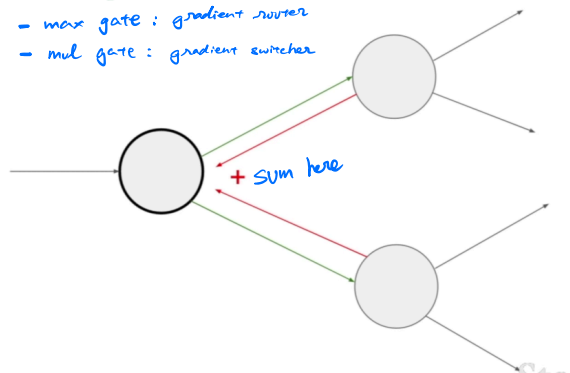
$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x))$$

→ to combine into a more complex node

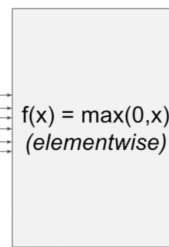
other nodes (more complex ones)



- max gate: gradient router
- mul gate: gradient switcher



4096-d input vector



4096-d output vector

Q: what is the size of the Jacobian matrix?
[4096 x 4096!]

in practice we process an entire minibatch (e.g. 100) of examples at one time:

i.e. Jacobian would technically be a [409,600 x 409,600] matrix

get it will be diagonal (very sparse)

$$\text{e.g. } f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^n (W_i \cdot x_i)^2$$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix} = \begin{bmatrix} 0.116 \\ 1.00 \end{bmatrix}$$

$$z = W \cdot (x) = \begin{bmatrix} W_{11}x_1 + \dots + W_{1n}x_n \\ W_{21}x_1 + \dots + W_{2n}x_n \\ \vdots \\ W_{m1}x_1 + \dots + W_{mn}x_n \end{bmatrix}$$

$$f(z) = \|z\|^2 = z_1^2 + \dots + z_n^2$$

$$\nabla_z f = 2z = \begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix} \in \mathbb{R}^2$$

$$\frac{\partial f}{\partial W_{ij}} = \sum_k \frac{\partial f}{\partial z_k} \frac{\partial z_k}{\partial W_{ij}} = \sum_k (2z_k) (1_{k=i} x_j) = 2z_i x_j$$

$$\nabla_W f = 2z \cdot x^T$$

$$\frac{\partial f}{\partial x_i} = W_{ki} x_j$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial z_k} \frac{\partial z_k}{\partial x_i} = \sum_k 2z_k W_{ki}$$

$$\nabla_x f = 2W^T z$$

Note that: the gradient with respect to a variable should be the same shape as the variable

Modularized implementation: forward / backward API

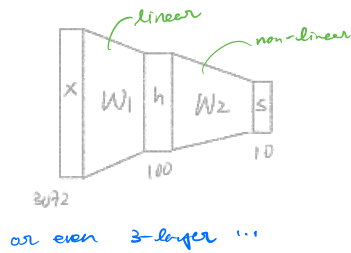


Graph (or Net) object (rough psuedo code)

```
class ComputationalGraphObject:
    # ...
    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
```

• Neural Network

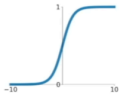
- Before | Linear score function : $f = Wx$
- after | 2-layer Neural Network: $f = W_2 \max(0, W_1 x)$



Activation functions

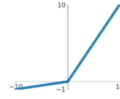
Sigmoid

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



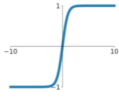
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

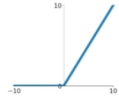


Maxout

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

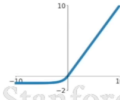
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural networks: Architectures

