

## Recall

$$s = f(x; W) = Wx \quad \text{score function}$$

$$L_i = \sum_j y_i \max(0, s_j - s_{y_i} + 1) \quad \text{SVM loss}$$

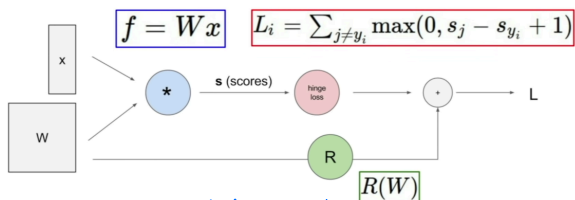
$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2 \quad \text{data loss + regularization}$$

want  $\nabla W L$

solve it with optimization  $\left\{ \begin{array}{l} \text{analytic gradient} \\ \text{numerical gradient} \end{array} \right.$

## analytic gradient (thru computational graphs)

### Computational graphs



- important to calculate gradient for neural network (complex function)

- e.g. find gradient

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

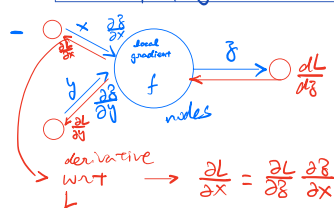
$$@ x=-2, y=5, z=-4$$

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

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

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

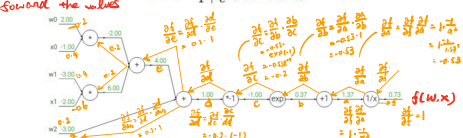
$$\text{back-propagation} \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \frac{\partial z}{\partial x} \quad \text{chain rule}$$



Another example:

forward the values

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$



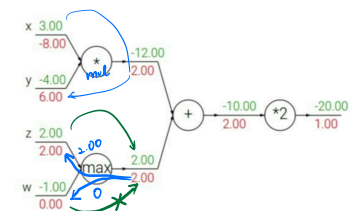
backprop the gradient backpropagation!

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

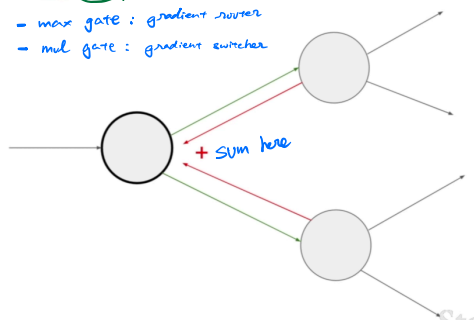
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

$$\rightarrow \text{to combine into a more complex node}$$

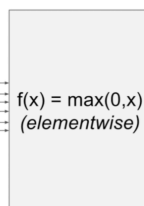
## other nodes (more complex ones)



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



4096-d input vector



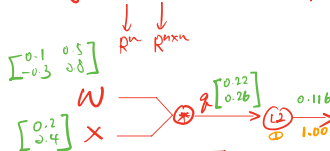
4096-d output vector

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)

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



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

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

$$\nabla_z f = 2z = \begin{bmatrix} 0.4 \\ 0.8 \end{bmatrix} \in \mathbb{R}^2$$

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

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

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

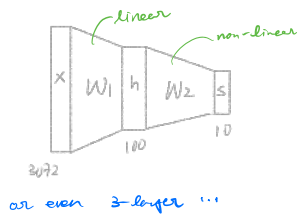
Modularized implementation: forward / backward API

Graph (or Net) object (rough pseudo 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
```

## • Neural Network

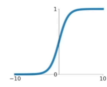
- Before | Linear score function :  $f = Wx$
- after | 2-layer Neural Network:  $f = W_2 \max(0, W_1 x)$



### Activation functions

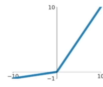
#### Sigmoid

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



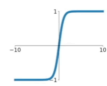
#### Leaky ReLU

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



#### tanh

$$\tanh(x)$$

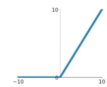


#### Maxout

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

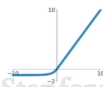
#### ReLU

$$\max(0, x)$$



#### ELU

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



### Neural networks: Architectures

