Let 21: Neural Network (Deep Learning)

CS3780/5780

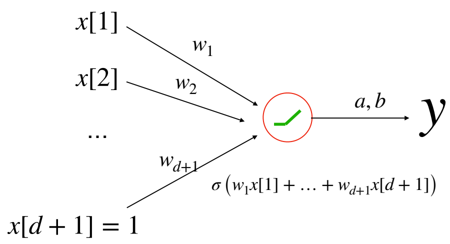
Nonlinear Method: Kernel Method

- Kernel method: In an elegant way, lift linear methods to provide non-linear predictors
- Predictor given by $a^\top \phi(x)$ where $\phi(x)$ is an implicit feature space (in much higher dimension than x)
- Implicit via kernel function $k(x, x') = \phi(x)^\top \phi(x')$
- Kernel method: Fixed feature mapping
  - Training time computation $> n^2$ (kernel matrix size)
  - Test time computation: $O(n)$

Nonlinear Method: Neural Network

- Can be viewed as explicitly learning the feature mapping based on data
- Can capture explicit structure, can put together pieces (layers of ``neurons”) into multiple layers to get richer predictors
- Multilayer Perceptron (aka neural networks) Invented here at Cornell by Frank Rosenblatt in 1963

A single neuron network



$$y = a\ \sigma(w^\top x) + b$$

Eg.

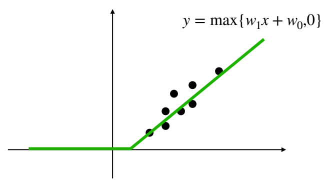
$\sigma(a) = \max\{a, 0\}$ ReLU: Rectified Linear Unit

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

$$\sigma(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

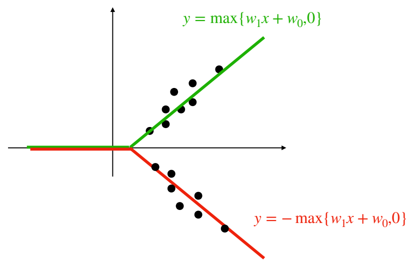
Single Neuron: ReLU Activation

$$y = \max\{w_1 x + w_0,\ 0\}$$



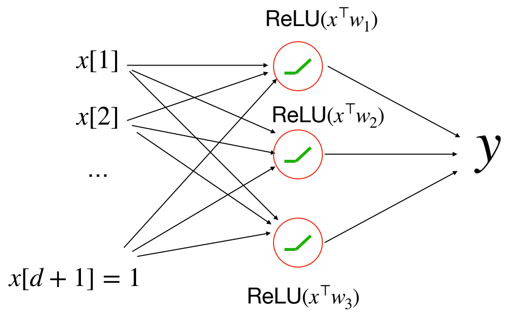
rectified linear unit (ReLU)

A single neuron network



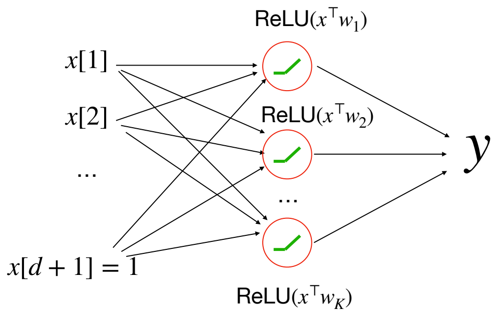
$$y = a \max\{w_1 x + w_0, 0\} + b$$

Let us stack multiple neurons together


$\text{ReLU}(x^\top w_1)$
$x[1]$
$x[2]$
$\text{ReLU}(x^\top w_2)$
$\dots$
y
$x[d+1] = 1$
$\text{ReLU}(x^\top w_3)$

$$y = \sum_{i=1}^{3} a_i \text{ReLU}(x^\top w_i) + b$$

Let us stack multiple neurons together


$\text{ReLU}(x^\top w_1)$
$x[1]$
$x[2]$
$\text{ReLU}(x^\top w_2)$
$\dots$
$\dots$
y
$x[d+1] = 1$
$\text{ReLU}(x^\top w_K)$

Vectorized form:

Define $W = \begin{bmatrix} (w_1)^\top \\ \cdots \\ (w_K)^\top \end{bmatrix} \in \mathbb{R}^{K \times d}$

$$\alpha = [a_1, \dots, a_K]^\top$$

$$y = \alpha^\top \underbrace{\left(\text{ReLU}(Wx)\right)}_{\text{Learnable feature } \phi(x)} + b$$

What does a neural network approximate

$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$

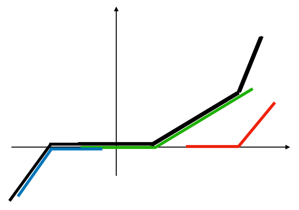
It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$

$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$
$+ a_2 \max\{w_2 x + c_2, 0\}$

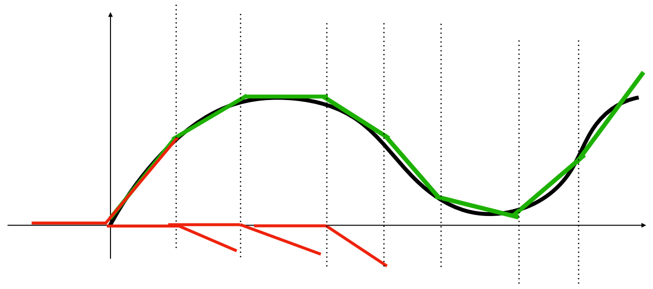
$K = 3 : y = a_1 \max\{w_1 x + c_1, 0\}$
$+ a_2 \max\{w_2 x + c_2, 0\}$
$+ a_3 \max\{w_3 x + c_3, 0\}$



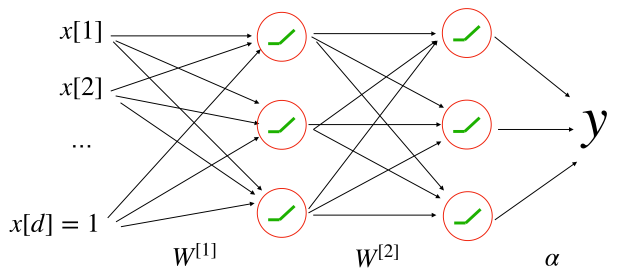
What does a neural network approximate

$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$

Claim: a wide enough one layer NN can approximate any smooth functions

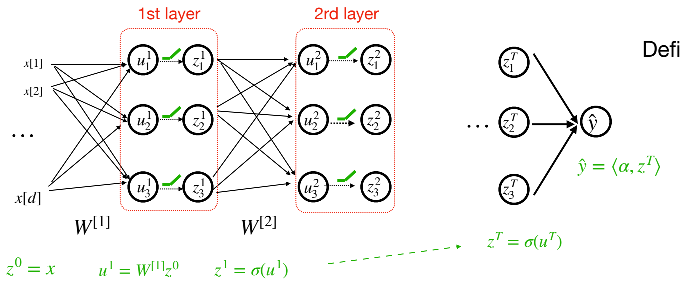


A multi-layer fully connected neural network



$y = \alpha^\top \sigma\left(W^{[2]} \sigma\left(W^{[1]} x\right)\right) + b$

A multi-layer fully connected neural network



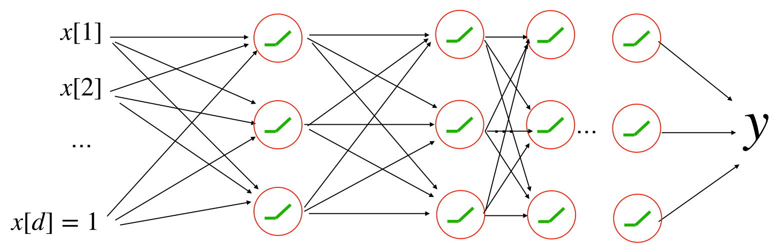
Define it by a forward pass:

```
$z^0 = x$
For t = 1 to T:
    $u^t = W^{[t]} z^{t-1}$
    $z^t = \sigma\left(u^t\right)$
End For
$y = \alpha^\top z^{[T]} + b$
```

Q: what is the computation complexity of the forward pass?

A: linear in # of Edges + # of nodes

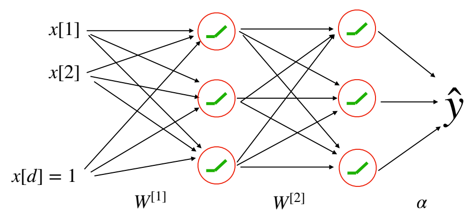
The benefits of going deep



Allows us to represent complicated functions without making NN too wide

Training neural network via SGD

$$h(x) := \alpha^\top \sigma\left(W^{[2]}\sigma\left(W^{[1]}x\right)\right) + b$$



Let $\ell(h(x), y)$ be any differentiable loss function

Compute gradients:

$$\frac{\partial \ell(h(x), y)}{\partial W^{[1]}} \quad \frac{\partial \ell(h(x), y)}{\partial W^{[2]}}$$

$$\frac{\partial \ell(h(x), y)}{\alpha} \quad \frac{\partial \ell(h(x), y)}{b}$$

(Next lecture: backpropagation for computing gradients)

Training neural network via SGD

Mini-batch Stochastic gradient descent

```
θ = [W^{[1]}, W^{[2]}, α, b]                          go through dataset multiple times
For epoc $e = 1$ to $E$:
    Randomly shuffle the dataset                     important (unbiased estimate of
                                                     the true gradient)
    Split the data into $n/B$ many batches $\mathscr{D}_i$, each with size B
    For  j = 1 to $n/B$
        Mini-batch gradient $g = \sum_{x,y \in \mathscr{D}_i} \nabla_\theta \ell(h_\theta(x), y)/B$
        $\theta = \theta - \eta g$
```

Training neural network via SGD

- We can use momentum to add stability and escape saddle points
- We can also use Adagrad (with minibatches)
- SGD/Adagrad perform well in practice and provide good generalization even for deep NN with large number of parameters

How do we compute the gradients w.r.t the weights of different layers?

We will see next class!