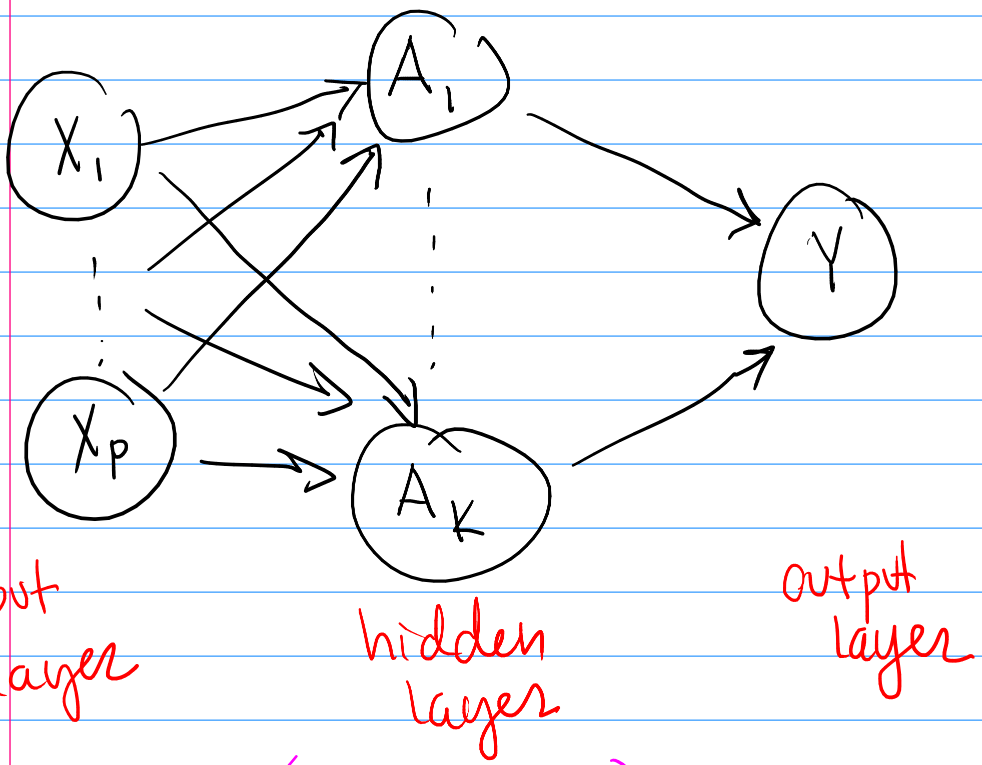
Neural Networks

Basically: build a prediction function $\hat{f}$ as a composition of functions

$$\hat{f} = \hat{f}_K(\hat{f}_{K-1}(\hat{f}_{K-2}(\hat{f}_{K-3}(\cdots \hat{f}_1(X)))) \cdots)$$

Single hidden layer (feed-forward network)



① $A_k = \sigma(w_k^T X + b_k)$

$b_k \in \mathbb{R}$

$(X_1, \ldots, X_p)$

$w_k \in \mathbb{R}^p$

"activation function", non-linear

(2) $Y = u^T A + c$

$A \in \mathbb{R}^k$

$c \in \mathbb{R}$

$u \in \mathbb{R}^k$

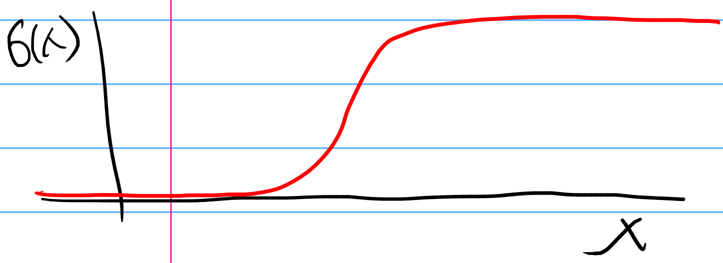
The activation function σ can be of many different forms,

popular choices:

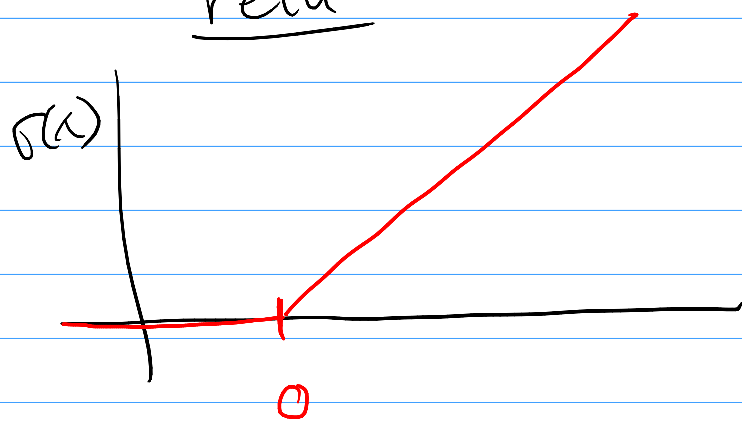
- $\sigma(x) = \dfrac{1}{1+e^{-x}}$ (sigmoid)

- $\sigma(x) = \max(0, x)$ (ReLU)

sigmoid



relu

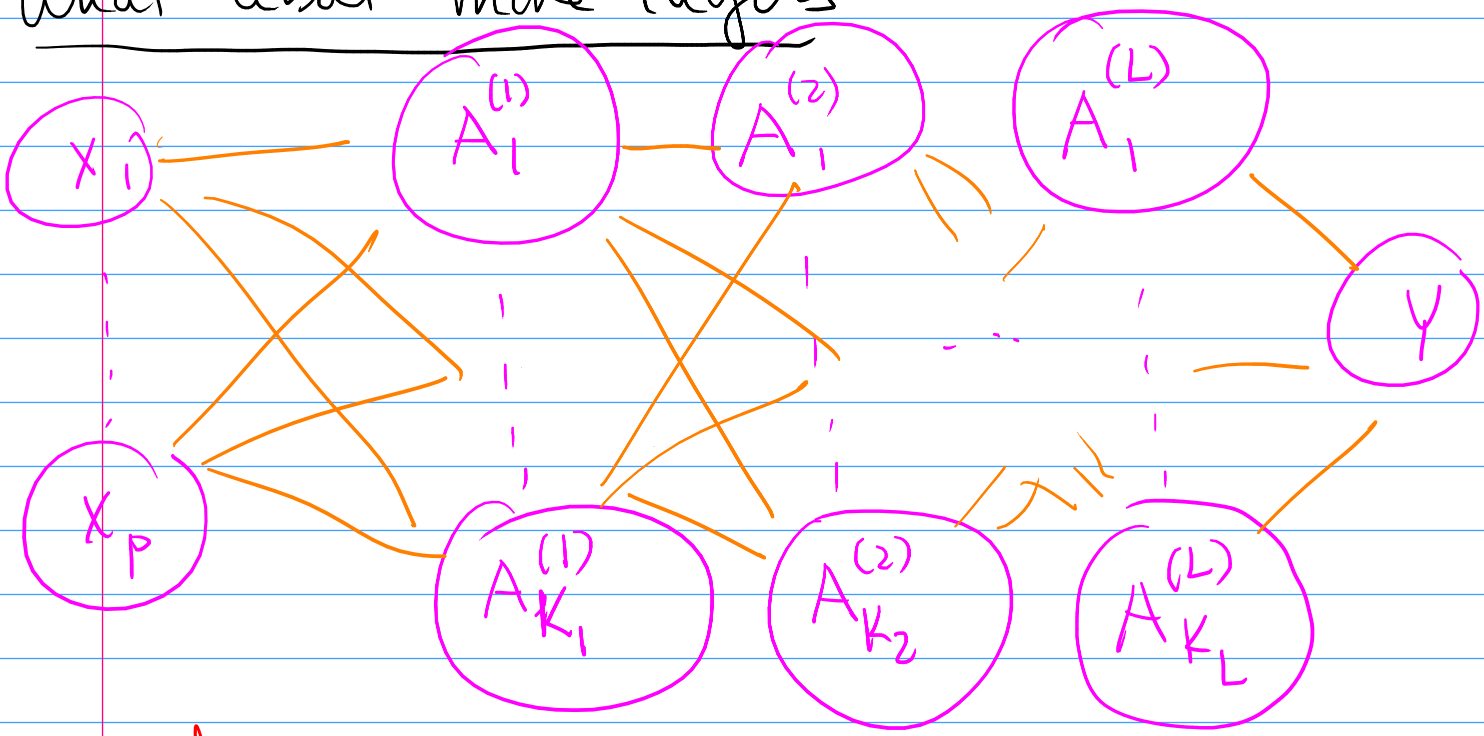


applying element-wise

$$\hat{f}(x) = u^T \sigma(W^T x + b) + c$$

mtx of weights

What about more layers



First hidden

$$A_k^{(1)} = \sigma\left(w_k^{(1)T} x + b_k^{(1)}\right)$$

or

$$A^{(1)} = \sigma_1\left(W^{(1)T} x + b^{(1)}\right)$$

$(A_1^{(1)}, \dots, A_{K_1}^{(1)})$

mtx $W^{(1)} = \begin{bmatrix} - w_1^{(1)} - \\ - w_{K_1}^{(1)} - \end{bmatrix}$

$b = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_{K_1}^{(1)} \end{bmatrix}$

Subsequent hidden layers

$$A^{(\ell)} = \sigma_\ell\left(W^{(\ell)T} A^{(\ell-1)} + b^{(\ell)}\right)$$

Output layer

$$Y = u^T A^{(L)} + c$$

$$\hat{f}(x) = \cdots \sigma_3\left(W^{(3)T}\sigma_2\left(W^{(2)T}\sigma_1\left(W^{(1)T}x + b^{(1)}\right) + b^{(2)}\right) + b^{(3)}\right)$$

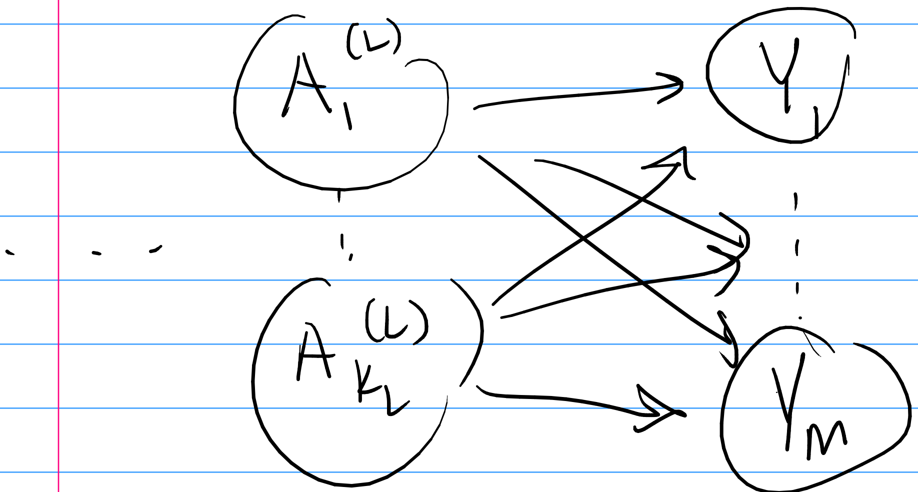
To fit, I need to specify all my params:

$$b^{(1)}, \ldots, b^{(L)}, W^{(1)}, \ldots, W^{(L)}$$

<u>For classification:</u>

All the same, except last layer typically

Y has M values (for each of M classes)



the prob. of class m

$$Y = \text{SoftMax}(U^T A^{(L)} + c)$$

btwn $[0,1]$ sum to 1

$$\text{Softmax}(z)_i = e^{z_i} / \sum_i e^{z_i}$$

How to learn params?

Let θ = vec. of all params

then we "learn" θ as $\hat{\theta}$ where

$$\hat{\theta} = \arg\min_{\theta} \sum_n L(y_n, f_\theta(x_n))$$

f_θ = NN

Problem: - super-high-dim'l opt. problem
- likely to over-fit

Soln:
1. learn "slowly" via gradient descent
2. regularize

Gradient descent:

0. initial guess $\theta^{(0)}$
1. For $t = 1, \ldots,$
$$\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla_\theta L \Big|_{\theta^{(t-1)}}$$
2. stop when L isn't decreasing w/ further steps

Calculate $\nabla_\theta L$ via back-propagation

(applying chain rule from calc I)

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))\, g'(x)$$

GD can be slow!

Speed up! Stochastic Grad. Descent.

$$\nabla_\theta \sum_n L(y_n, f_\theta(x_n)) = \sum_n \nabla_\theta L(y_n, f_\theta(x_n))$$

don't sum over all training data - use a subset.

Also regularize!

1) penalize loss e.g. $L + \|\theta\|_2^2$

$L + \|\theta\|_1$

2) early stopping - stop SGD when val. perf. stop getting better

3) drop out: randomly set some weights to zero during training