Recap: kernel machines are an approach to ML with nonlinear features

- Pick a nonlinear feature map $\phi: \mathbb{R}^d \to \mathcal{H}$ (Hilbert space ∫ (vector space))

ex: $\phi: [x_1\ x_2] \to \{x_1\ x_2\ x_1^2\ x_1x_2\ x_2^2\}$

$\phi: [x_1\ x_2] \to \left[\sqrt{x_1^2 + x_2^2}\quad \tan^{-1}\left(\frac{x_2}{x_1}\right)\right]$

$\phi: [x_1\ x_2] \to \{$all the monomials in x_1 and $x_2\}$

- We showed if we want to learn a function of the form

$$f(x) = \langle w, \phi(x) \rangle$$

this can be accomplished using just the kernel function

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

so learning can be done using K and the kernel matrix

$$K_{ij} = K(x_i, x_j)$$

In particular, the optimal f for an ERM takes the form

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$$

so we just have to learn the α weights

Last example: logistic regression w/ non-linear features

$$w^* = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y_i \langle w, \phi(x_i) \rangle}\right) + \frac{\lambda}{2} \|w\|_2^2$$

we know in fact $f(x_i) = \langle w^*, \phi(x_i) \rangle$

$$= \sum_{j=1}^{n} \alpha_j^* K(x_i, x_j)$$

$$= \{K\alpha\}_i \leftarrow \text{ith row of } K\alpha$$

so we can predict using $f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$

if we find the optimal α :

$$\alpha_* = \underset{\alpha}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y_i (K\alpha)_i}\right) + \frac{\lambda}{2} \alpha^T K \alpha$$

Kernel logistic regression (recall $K \in \mathbb{R}^{n \times n}$)

Question: how can we gain accuracy like kernel methods while having a faster runtime?

An answer:

- now the cost of kernel methods is that of

1) multiplying by K $O(n^2)$

- for example in subgradient descent, or GD

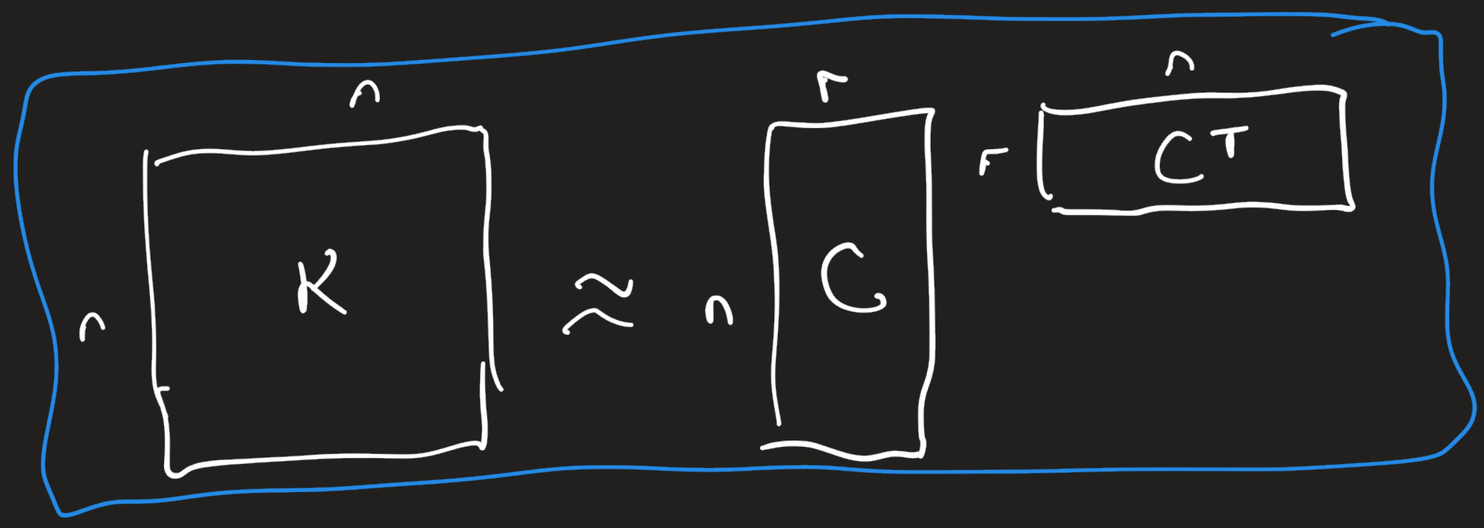
2) inverting K $O(n^3)$

- for example in ridge regression

$$\alpha = (K + n\lambda I)^{-1} y$$

- approximate K by a low-rank matrix

$$K \approx CC^T \quad \text{where} \quad C \in \mathbb{R}^{n \times r}$$



call $\tilde{K} = CC^T$ our low-rank approximation.
and replace K in all our algorithms with $\tilde{K}$.

Now

1) cost of multiplying by $\tilde{K}$ is $O(nr)$
   consider for example $n = 50K$ and $r = 1600$

2) cost of inverting $\tilde{K} + \lambda n I$ $\quad O(nr^2)$
   (use the Woodbury identity)

Two approaches to getting these low rank approximations:

1) Nyström approximations (originated in PDE literature)

2) Random feature maps (Recht & Rahimi, 2013)

Nyström approximations

Note that $K = K^T$ b/c $k(x_i, x_j) = k(x_j, x_i)$

Write

$$K = \begin{array}{|c|c|} \hline W & C^T \\ \hline C & \\ \hline \end{array}$$

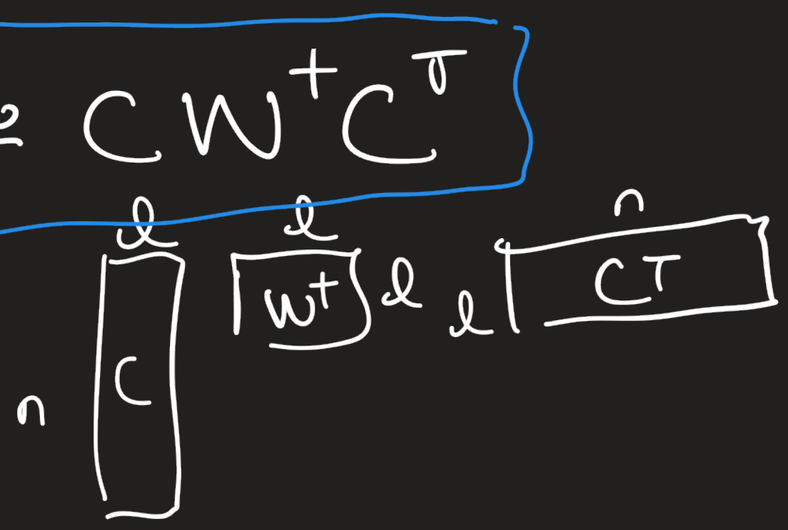
where C are l sampled columns and W is the principal-submatrix where C and C^T overlap

note that this is equivalent to selecting
l "landmark points" $p_1, \dots, p_l$ and
comparing all of our training points to these
landmark points

$$C_{i,j} = K(x_i, p_j)$$

and now we extend these comparisons to the
unobserved points (the non-landmark points) by
noting that

$$K \approx C W^{+} C^{T}$$



where W^+ is the Moore-Penrose pseudoinverse of W. We call W^+ the coupling matrix

Why is $K \approx C W^+ C^T$?

- Assume W is full rank, then $W^+ = W^{-1}$

and

$$C W^+ C^T = \underbrace{\begin{bmatrix} W \\ \hline L \end{bmatrix}}_{C} W^+ \left[\begin{array}{c|c} W & L^T \end{array}\right]$$

$$= \begin{bmatrix} W & L^T \\ L & L W^+ L^T \end{bmatrix}$$

so $$K - CW^{+}C^{T} = \begin{bmatrix} 0 & 0 \\ 0 & \underbrace{\bar{K} - LW^{+}L^{T}} \end{bmatrix}$$

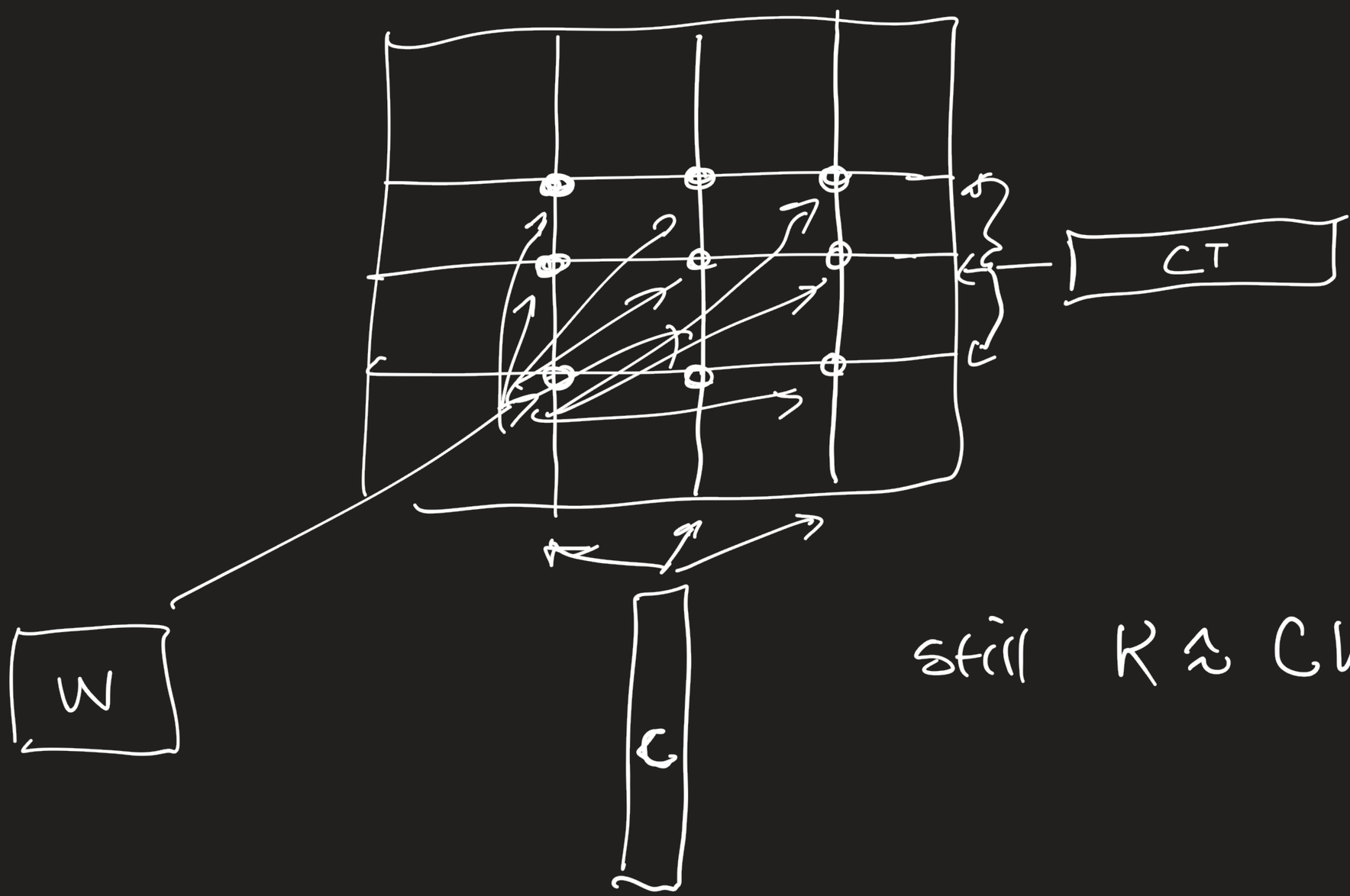
Schur complement of W in K

if $$K = \begin{bmatrix} W & L^{T} \\ L & \bar{K} \end{bmatrix}$$

Fact: $rk(W) = r(K)$ then $\bar{K} - LW^{+}L^{T} = 0$

Take-away: if $rk(W) = rk(K)$ then $K = CW^{+}C^{T}$

Note we could pick any $\underline{C}$ subset of columns of K



still $K \hat{\approx} C W^{+} C^{T}$

Nystrom alg

Input: K – kernel function
X – training data
l – # of landmark points

Output: C, W

use uniform sampling w/ or w/o replacement

Alg

1. randomly pick l landmark points $= p_1, \ldots, p_l$
2. construct C by comparing all of the datapoints to my landmark points

$$C_{i,j} = K(x_i, p_j)$$

3. construct W by comparing the landmark points to themselves

$$W_{i,j} = K(p_i, p_j)$$

return C, W

Random feature maps

Motivation:

$$K = \Phi\Phi^T \quad \text{where} \quad \Phi = \begin{matrix} n \\ \\ \end{matrix}\begin{bmatrix} \overset{D}{\phi(x_1)^T} \\ \vdots \\ \phi(x_n)^T \end{bmatrix}$$

would be a low-rank approximation if

$D \ll n$

But in general $D \gg n$ (that's why we work with the kernel K instead of Φ directly)

What we want to do is find an approximate feature map

$$\tilde{\phi} : \mathbb{R}^d \to \mathbb{R}^E$$

$E \ll D$ and $E \ll n$

where $\tilde{\phi}$ is chosen so:

- $E \ll n, D$
- $\langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle \approx \kappa(x, y)$

because then

$$K_{ij} = \kappa(x_i, x_j) \approx \langle \tilde{\phi}(x_i), \tilde{\phi}(x_j) \rangle$$

so

$$K \approx \tilde{\Phi}\,\tilde{\Phi}^T \quad \text{where} \quad \tilde{\Phi} = \begin{bmatrix} \tilde{\phi}(x_1)^T \\ \vdots \\ \tilde{\phi}(x_n)^T \end{bmatrix} \in \mathbb{R}^{n \times E}$$

and $E \ll n$ means this is a low-rank approx, so is cheaper to use than K

Q: how to find these approx feature maps

A: randomness!

For a given kernel K, choose $\tilde{\phi}$ ← a random function so that

$$\mathbb{E}\langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = K(x,y)$$

Ex: if $K(x,y) = e^{-\frac{\|x-y\|_2^2}{2\sigma^2}}$

then $\tilde{\phi}(x) = \sqrt{2}\cos(\omega^T x + b)$ where $\omega \sim \mathcal{N}(0, \sigma^2 I)$

and b unif $\in [0, 2\pi]$

satisfies

$$\mathbb{E}_{\omega,b} \langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = K(x,y)$$

Another example: polynomial kernel

$$k(x,y) = \langle x, y \rangle^r$$

Idea: $\tilde{\phi}(x) = \varepsilon^T x$ where $\varepsilon \in \mathbb{R}^d$ is a vector of uniformly random signs, iid $\{\pm 1\}$

$$\mathbb{E}\langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = \mathbb{E}\left[(\varepsilon^T x)(\varepsilon^T y)\right]$$

$$= \mathbb{E}\left(\sum_{i,j=1}^{d} x_i y_j \varepsilon_i \varepsilon_j\right)$$

$$= \sum_{i,j=1}^{d} x_i y_j \, \mathbb{E}[\varepsilon_i \varepsilon_j]$$

Since the entries of ε are independent,

$$\mathbb{E}[\varepsilon_i \varepsilon_j] = \begin{cases} \mathbb{E}[\varepsilon_i]\,\mathbb{E}[\varepsilon_j] = 0 & \text{if } i \neq j \\ \mathbb{E}\{\varepsilon_i^2\} = 1 & \text{if } i = j \end{cases}$$

$$\Rightarrow \quad \mathbb{E}\langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = \sum_{i,j=1}^{d} x_i y_j \, \mathbb{E}[\varepsilon_i \varepsilon_j]$$

$$= \sum_{i=1}^{d} x_i y_i = \langle x, y \rangle$$

Now I use this fact to approximate the polynomial feature map:

- sample $\varepsilon_1 \in \mathbb{R}^d, \dots, \varepsilon_r \in \mathbb{R}^d$ i.i.d. random sign vectors
- $\tilde{\phi}(x) = (\varepsilon_1^T x) \cdots (\varepsilon_r^T x)$

with this choice:

$$\mathbb{E}\langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = \mathbb{E}\left[\prod_{i=1}^{r} (\epsilon_i^T x)(\epsilon_i^T y) \right]$$

$$= \prod_{i=1}^{r} \mathbb{E}\{\epsilon_i^T x y^T \epsilon_i\} = \prod_{i=1}^{r} \langle x, y \rangle = \langle x, y \rangle^r$$

Alg:

Sample $W_1 \in \mathbb{R}^{D \times d}$ so all its entries are i.i.d. random signs

Sample $W_2, \dots, W_r$ i.i.d. in the same manner

$$\tilde{\phi}(x) = \frac{1}{\sqrt{D}} (W_1 x) \circ (W_2 x) \circ \cdots \circ (W_r x) \in \mathbb{R}^D$$

Note that with this choice,

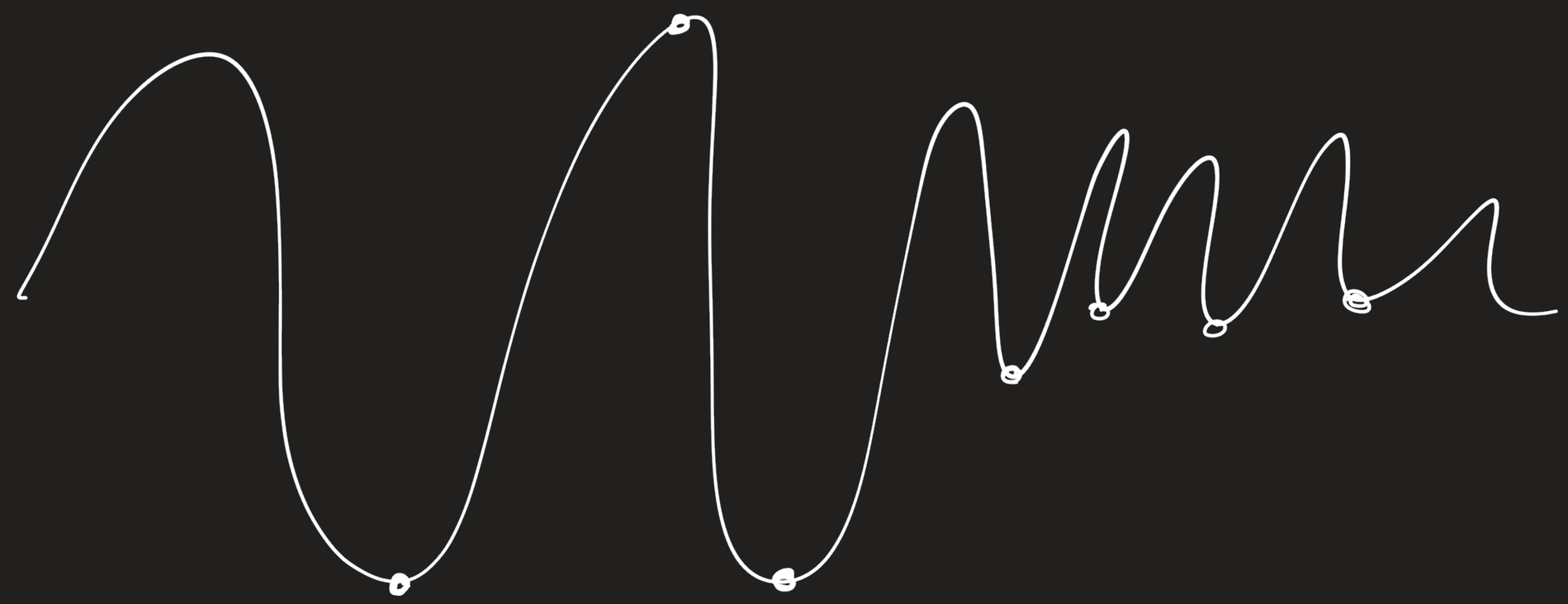
$$\mathbb{E}\langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = \mathbb{E} \sum_{i=1}^{D} \tilde{\phi}(x)_i \, \tilde{\phi}(y)_i$$

$$= \sum_{i=1}^{D} \mathbb{E}\left[\tilde{\phi}(x)_i \tilde{\phi}(y)_i \right]$$

$$= \sum_{i=1}^{D} \frac{1}{D} \mathbb{E}\left[\prod_{j=1}^{r} (\omega_j)_i^T \times (\omega_j^T)_i \, y \right]$$

$$= \frac{1}{D} \sum_{i=1}^{D} \langle x, y \rangle^r$$

$$= \langle x, y \rangle^r$$



A more general approach that kernel methods to learning nonlinear functions is that of neural networks

Recall w/ kernel methods, we

1) pick a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^D$
2) results in a kernel $\langle \phi(x), \phi(y) \rangle$
3) know the resulting ERM is <u>convex</u> and takes the form

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$$

4) solve for α^*, again is a <u>convex prob</u> w/ any of the methods we talked about before and we know

$$f(\alpha_T) \leq f(\alpha^*) + \varepsilon \quad \text{if we run long enough}$$

Drawbacks:

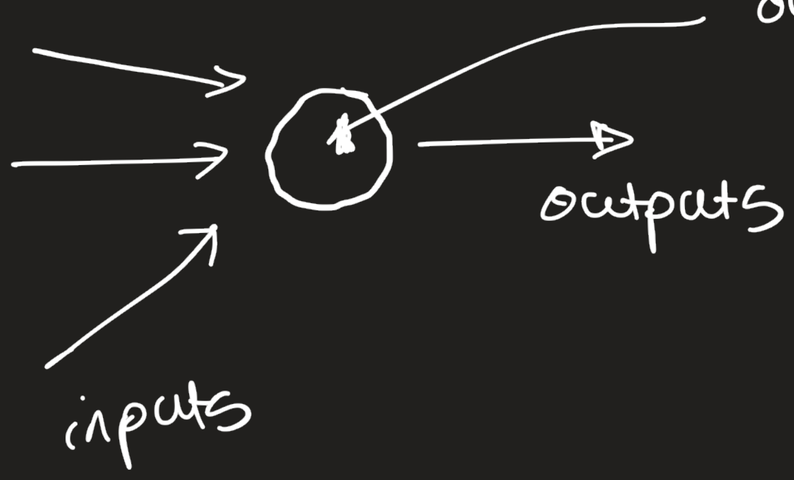
1) we have to do a good job of selecting our kernel K

2) we have to keep around our training data to predict

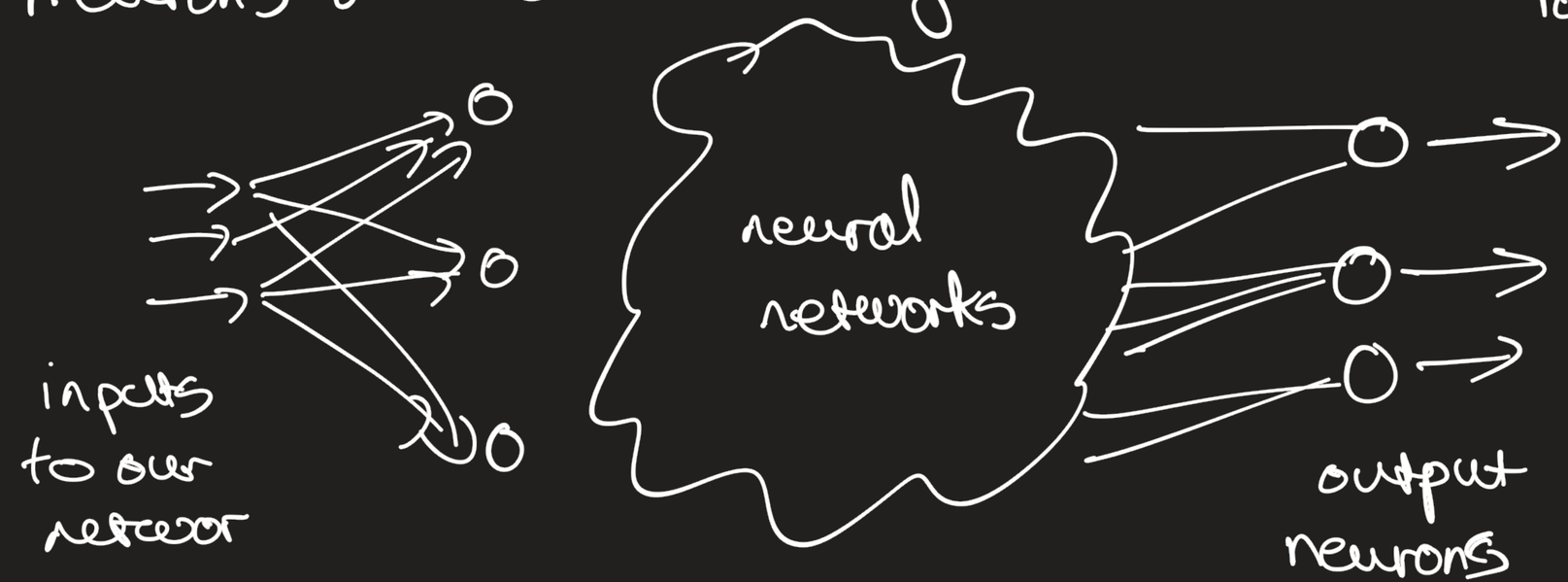
Another approach is neural nets. This is more general and learns our features for use

What is a neural network?

- motivated by biological neural networks (collections of neurons)



- neurons are connected together as DAG (directed acyclic graph: no feedback loops or self loops)

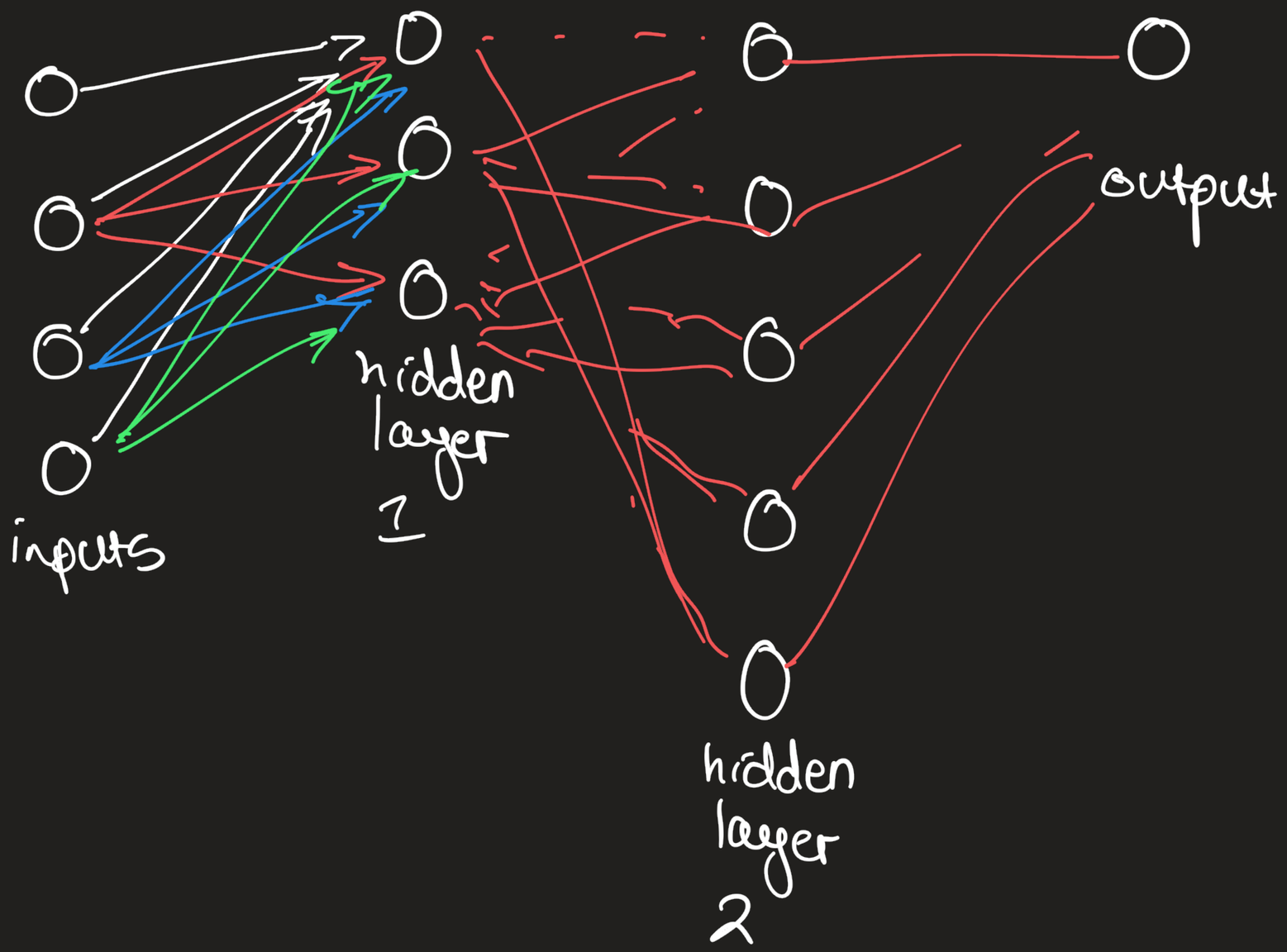


Given a particular DAG (architecture of the neural network) we learn parameters for the neurons in the neural net to minimize our ERM objective

$$w = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; w), y_i) + \cdot R(w)$$

The architecture determines how many parameters are in w; and the way in which the function f depends on these parameters

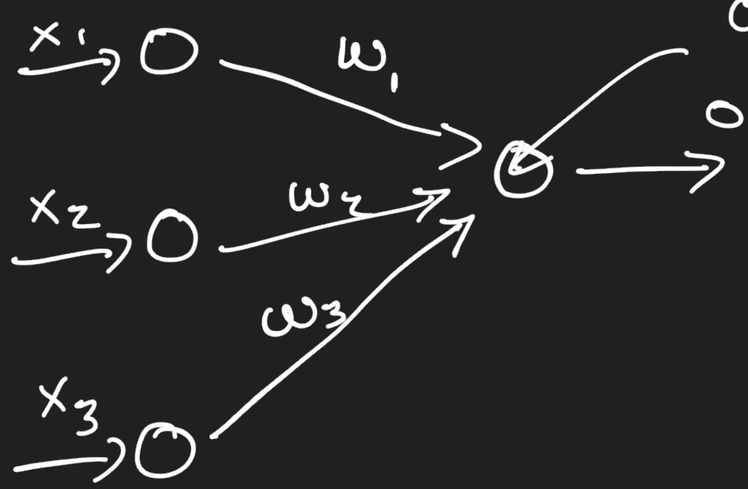
Architecture Choice I: Fully-connected neural networks



- The neurons are divided into an input layer, an output layer, and one or more hidden layers
- all the neurons in layer l are connected to all the neurons in layer $l+1$

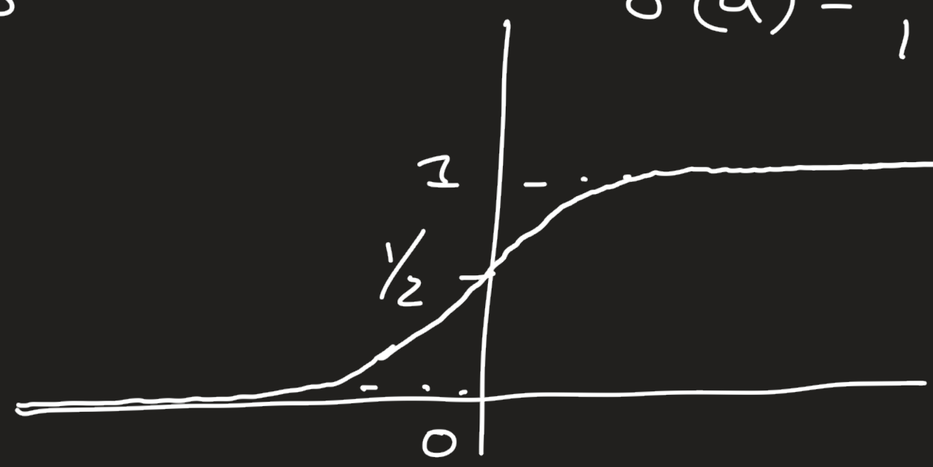
Ex Binary Classification (logistic regression)

Inputs are in $\mathbb{R}^3$

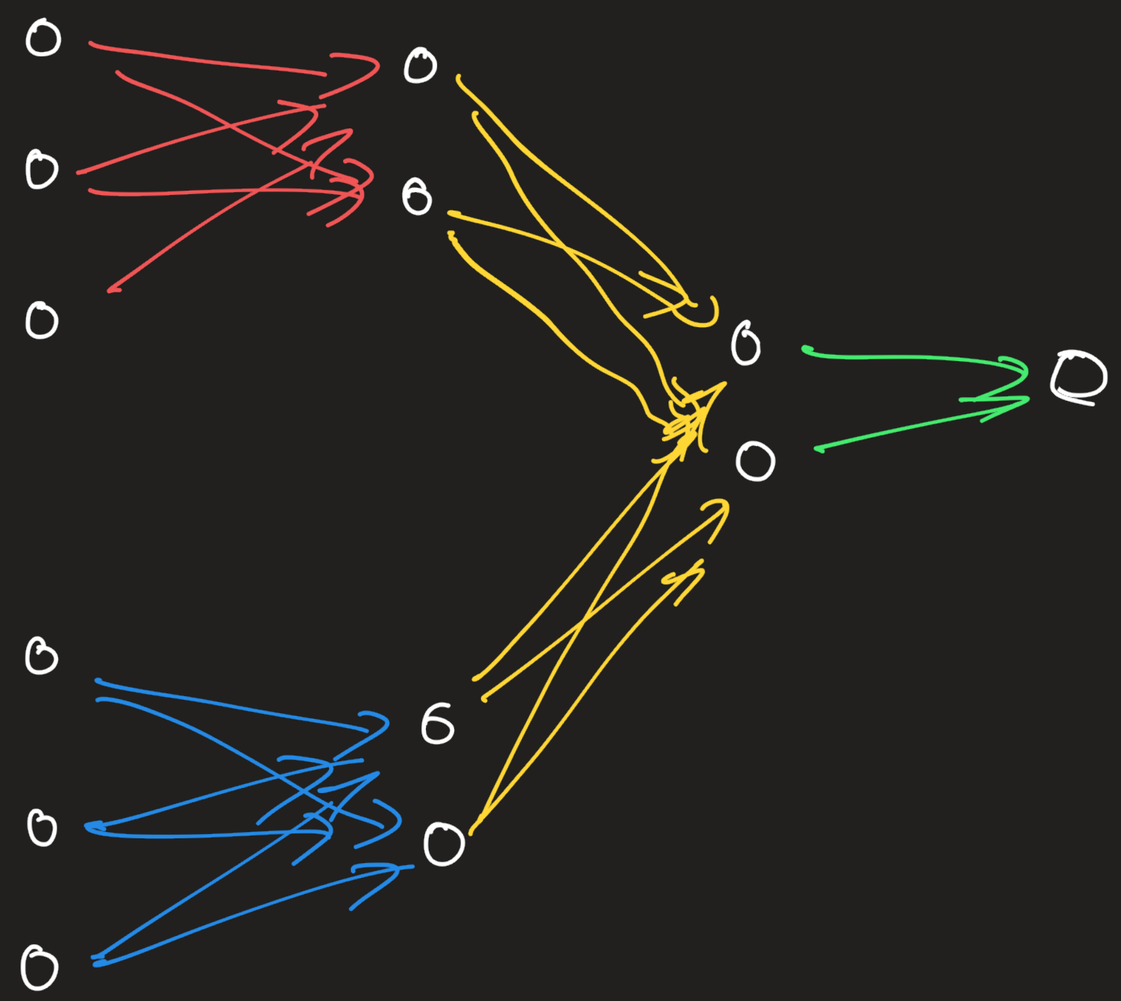


$$o = f\left(\sum_{i=1}^{3} w_i x_i + b\right) = \sigma\left(\sum_{i=1}^{3} w x_i + b\right)$$

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



$x, y \in \mathbb{R}^3, \mathbb{R}^3$ inputs



$K \in \mathbb{R}^{n \times n}$ costs $\mathcal{O}(n^2)$ mult
$\mathcal{O}(n^3)$ invert

$K \approx \tilde{\Phi}\tilde{\Phi}^T$ where $\tilde{\Phi} \in \mathbb{R}^{n \times E}$

costs $\mathcal{O}(nE)$ mult
costs $\mathcal{O}(nE^2)$ invert

randomized, low-dim (E) approx feature map

$\Phi\Phi^T$ where $\Phi \in \mathbb{R}^{n \times D}$ where $D > n$

costs $\mathcal{O}(nD) = \mathcal{O}(n^2)$

cost $\mathcal{O}(nD^2) = \mathcal{O}(n^3)$

deterministic, high-dim (D) exact feature map