RBE474X/595-B01-ST: Deep Learning For Perception

Class 2: Multi Layer Perceptrons And Backpropagation

Prof. Wei Xiao

Shreyas Devdatta Khobragade
he/him/his
Office Hour Location: UH243 Curtain Space
Office Hours: Tuesday 2:00pm-3:00pm

https://piazza.com/wpi/fall2025/rbe474x595

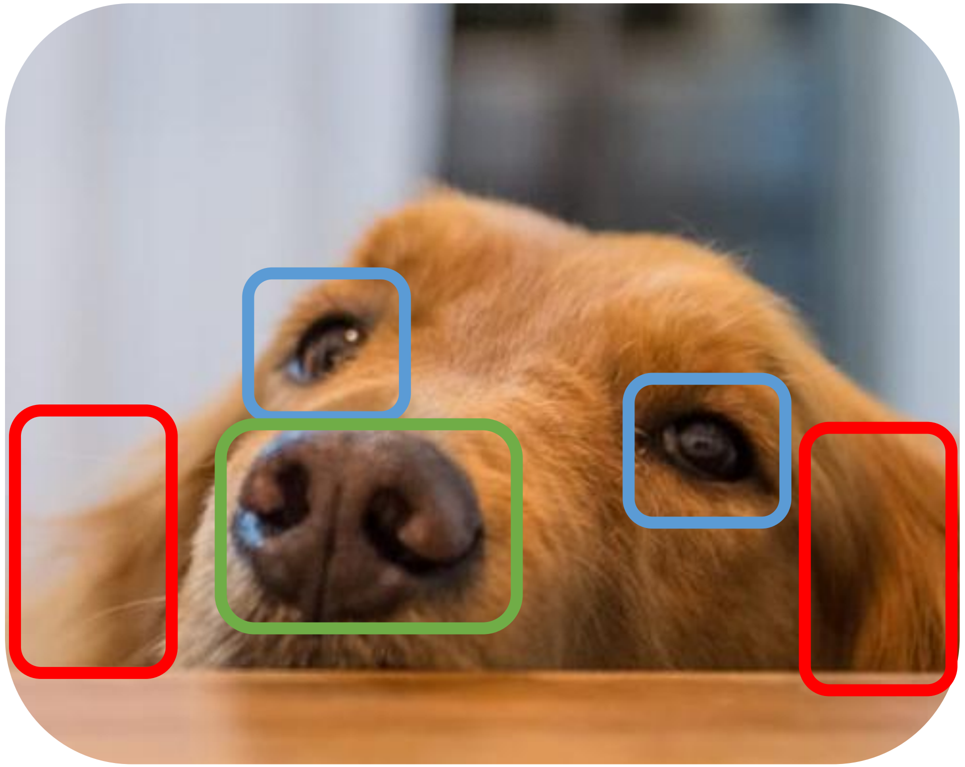
Please Ask Questions!

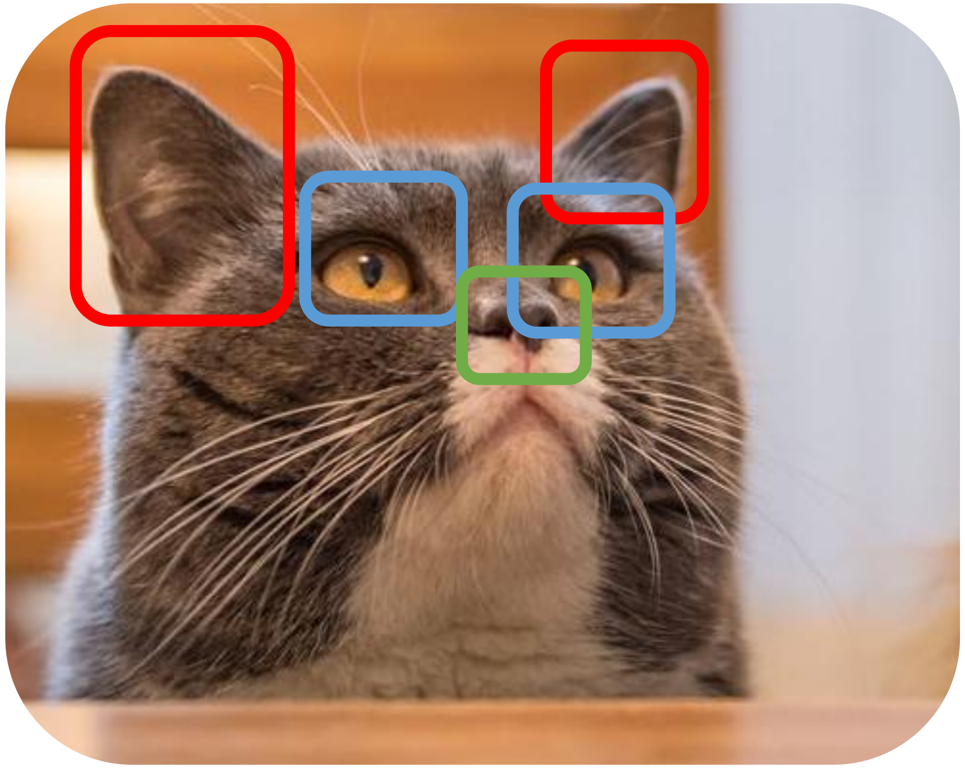
Today's lecture is Math heavy! Slow me down if needed!

Dog or Cat

Image Classification

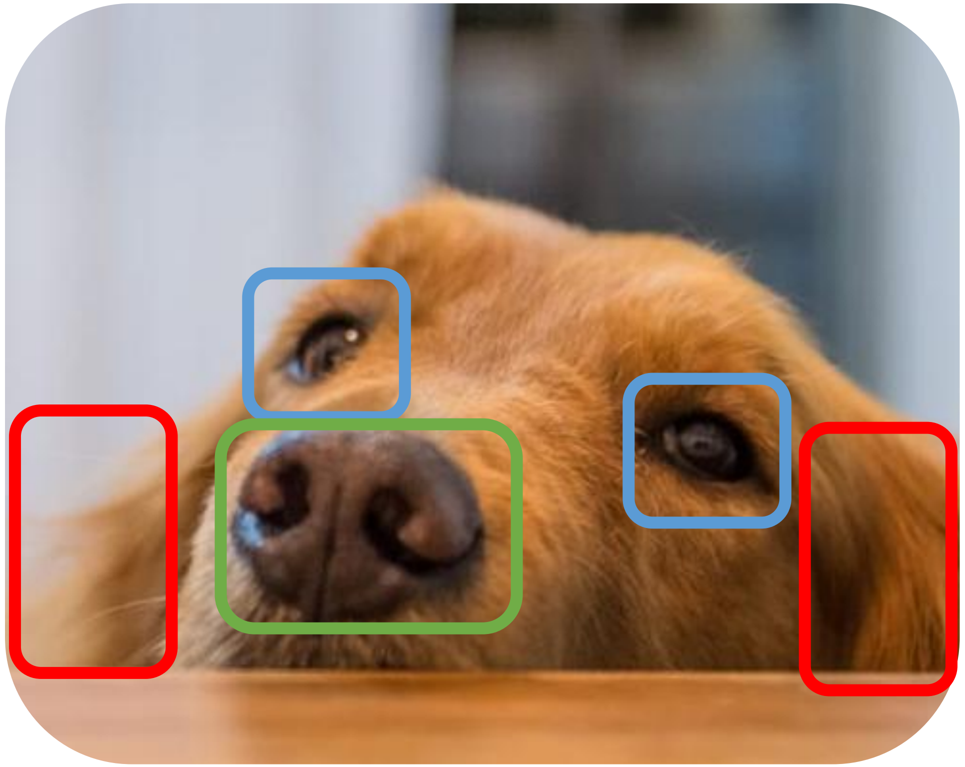
How Do You Tell If It's A Dog or a Cat?



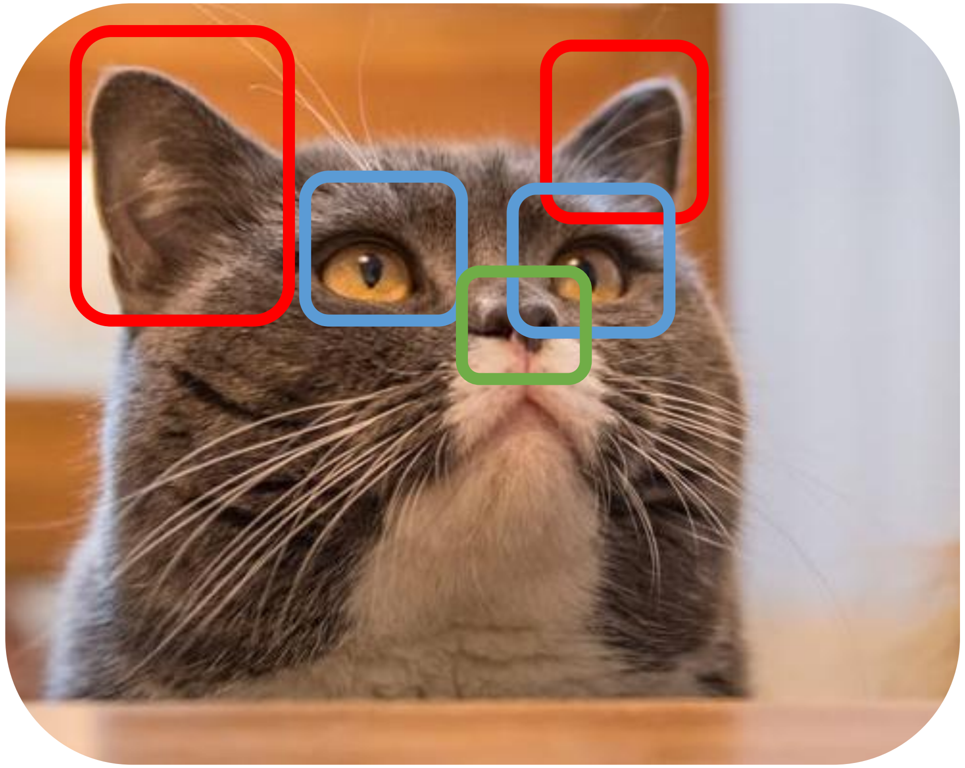


Features

$$X = \begin{bmatrix} \text{Non} - \text{pointy ears} \\ \text{Round and large nose} \\ \text{Eyes without pointy pupil} \end{bmatrix}$$

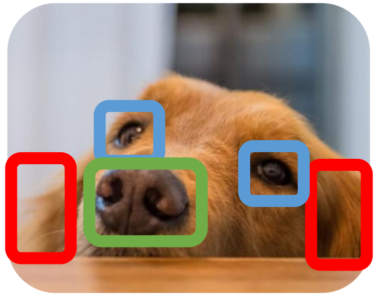


$$\begin{bmatrix} 0.21 \\ 0.99 \\ 0.86 \end{bmatrix}$$

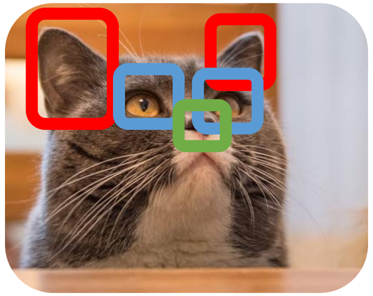


$$\begin{bmatrix} 0.79 \\ 0.01 \\ 0.14 \end{bmatrix}$$

Features

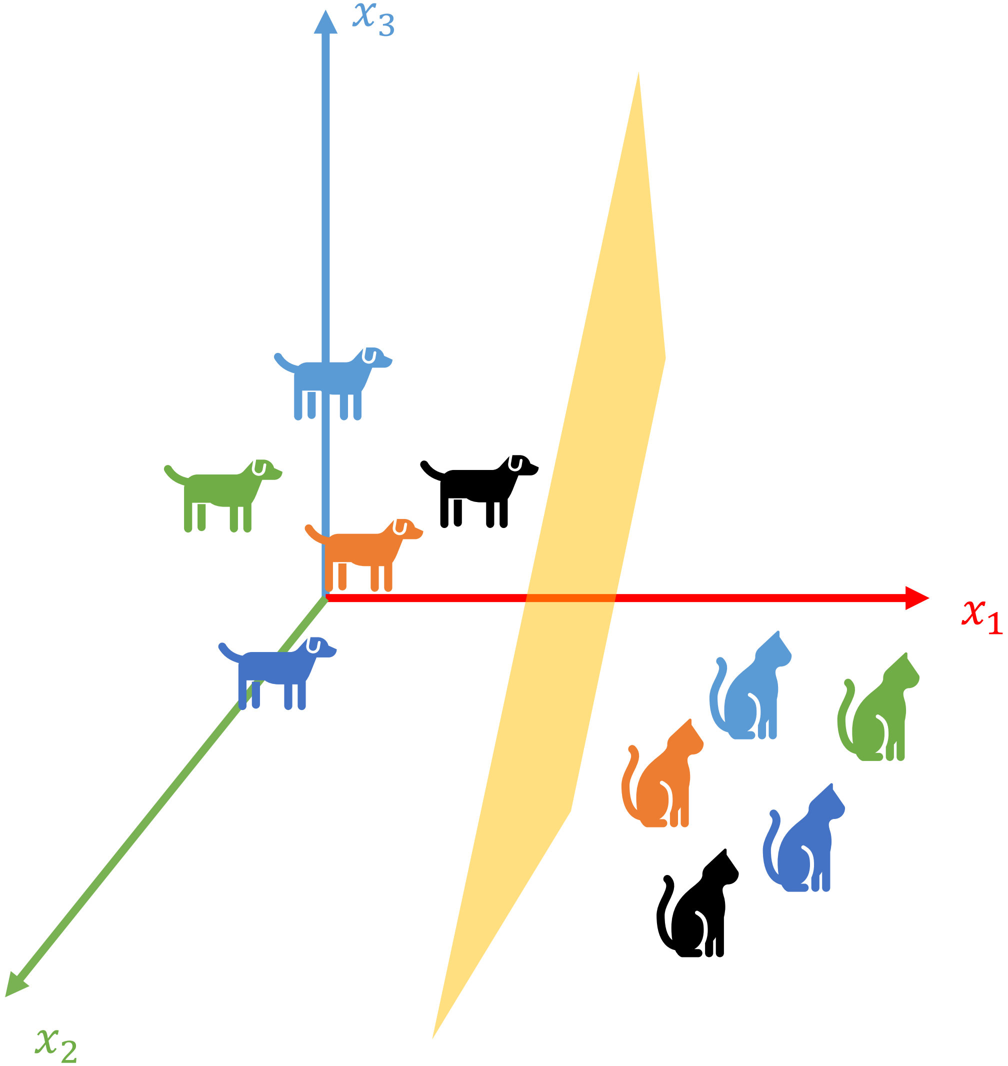


$$\begin{bmatrix} 0.21 \\ 0.99 \\ 0.86 \end{bmatrix}$$



$$\begin{bmatrix} 0.79 \\ 0.01 \\ 0.14 \end{bmatrix}$$

$$X = \begin{bmatrix} \text{Non} - \text{pointy ears} \\ \text{Round and large nose} \\ \text{Eyes without pointy pupil} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$



Linear Regression

$Y = W_0^T X + B$

or $Y = W^T \begin{bmatrix} X \\ 1 \end{bmatrix}, W = [W_0^T \; B]$

Given $\{X, Y\}_i$ find W

Mathematically,

$\underset{W}{\operatorname{argmin}} \, \mathbb{E}\left(\|\hat{Y} - \tilde{Y}\|_2 \right)$

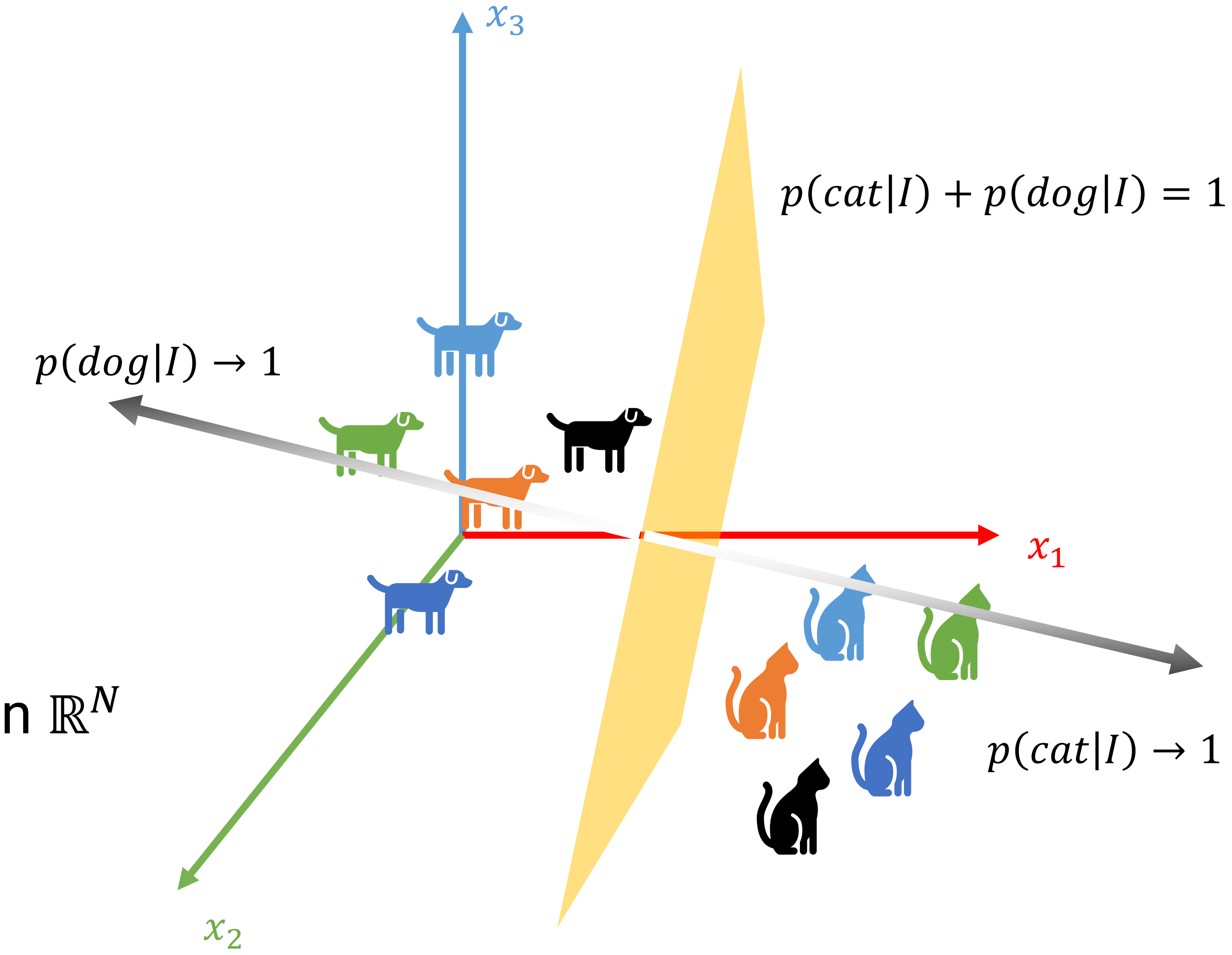
or $\underset{W}{\operatorname{argmin}} \, \mathbb{E}\left(\|\hat{Y} - f(X, W)\|_2 \right)$

We are fitting a linear hyperplane in $\mathbb{R}^N$

How do you get p from Y?

You need some function!

$p = g(\tilde{Y})$



Linear Regression

$Y = W^T \begin{bmatrix} X \\ 1 \end{bmatrix}$

$Y = AW$

W$= A^{-1}Y$

When is this valid?

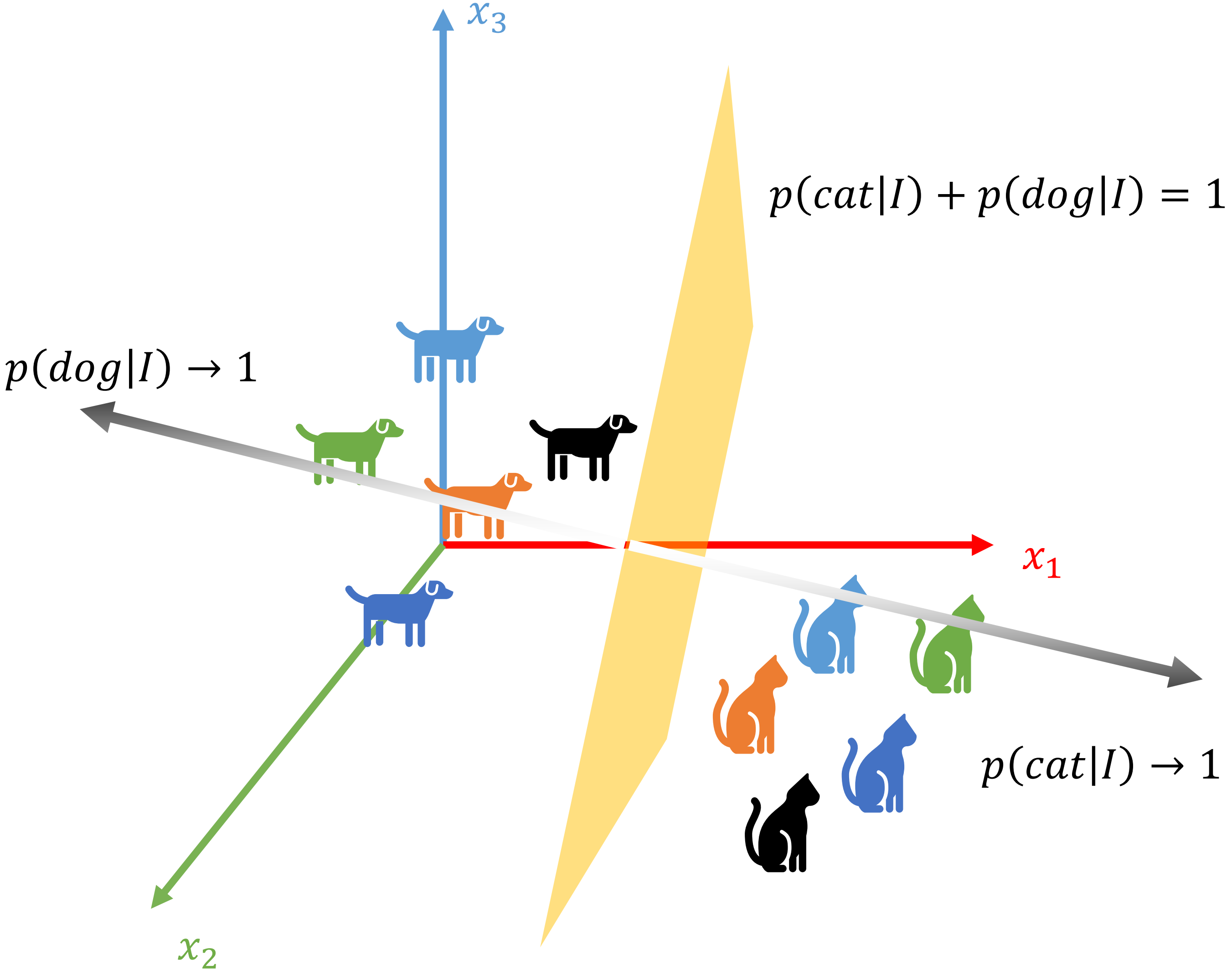
A is square and invertible!

What if A is not square?

$A^{-1} \approx A^{\dagger}$

How do you get the pseudo-inverse?

SVD!

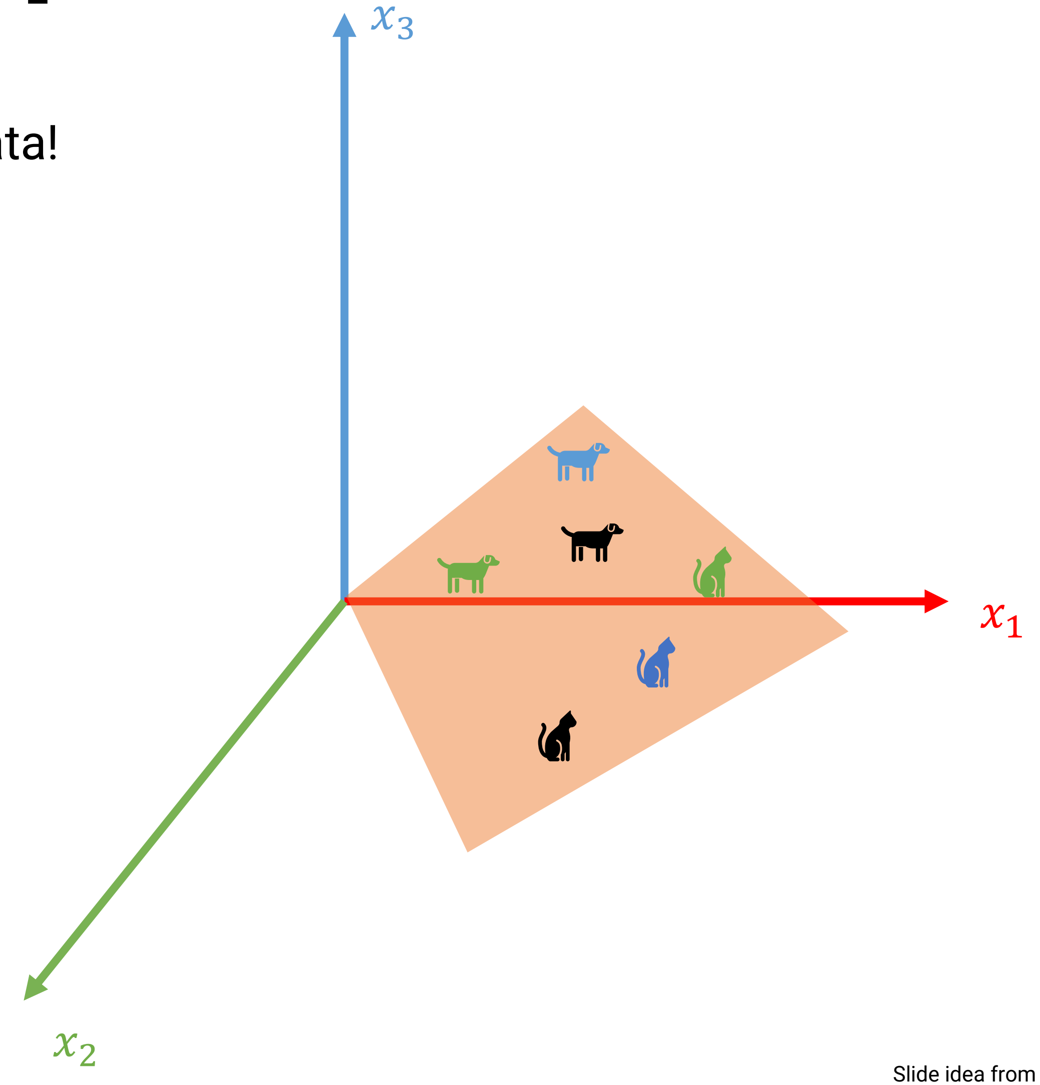


Singular Value Decomposition

Dimensionality of Data
Min. num. independent dimensions to represent **all** data!

How do we know this mathematically?
Rank of A!

In our example, what is the dimensionality?
2! But it is represented in 3D.



Slide idea from Stanford's CS426

Singular Value Decomposition

Σ can also be shown as D

$$A = U\Sigma V^T$$

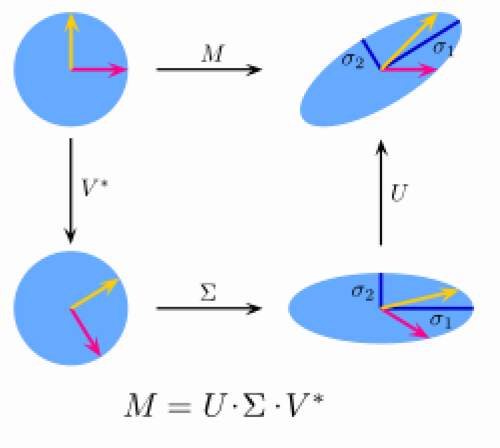
$M \times N$ $M \times M$ $N \times N$

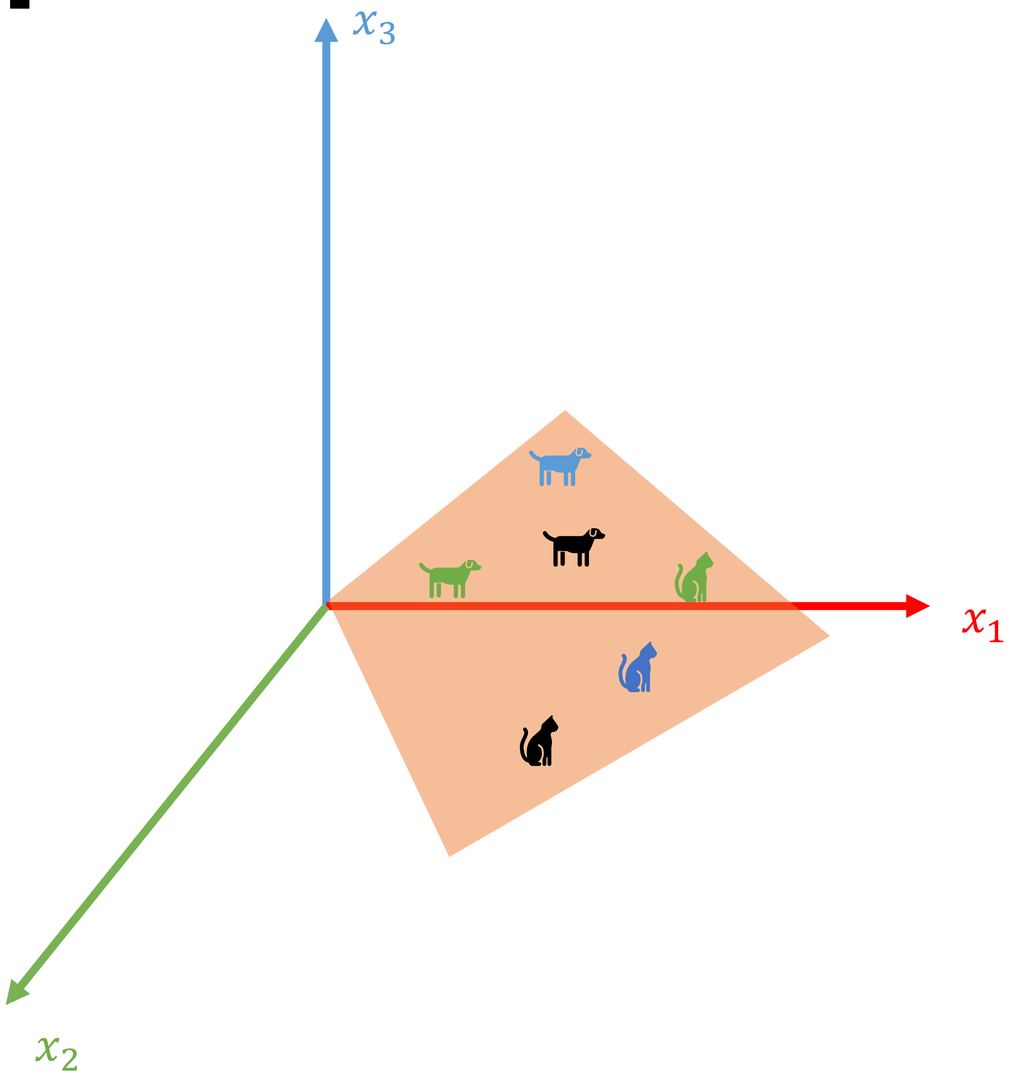
$M \times N$

Shows the spread of data!

$$UU^T = \mathbb{I}_M$$

$$VV^T = \mathbb{I}_N$$





SVD

$A\mathbf{x} \approx \mathbf{b}$

What do you think num. red squares is?

The **true** dimensionality!
The number of actual min. dim. needed to represent data!
Also called Num. Principal Components in PCA

$$A = U \, D \, V^T$$

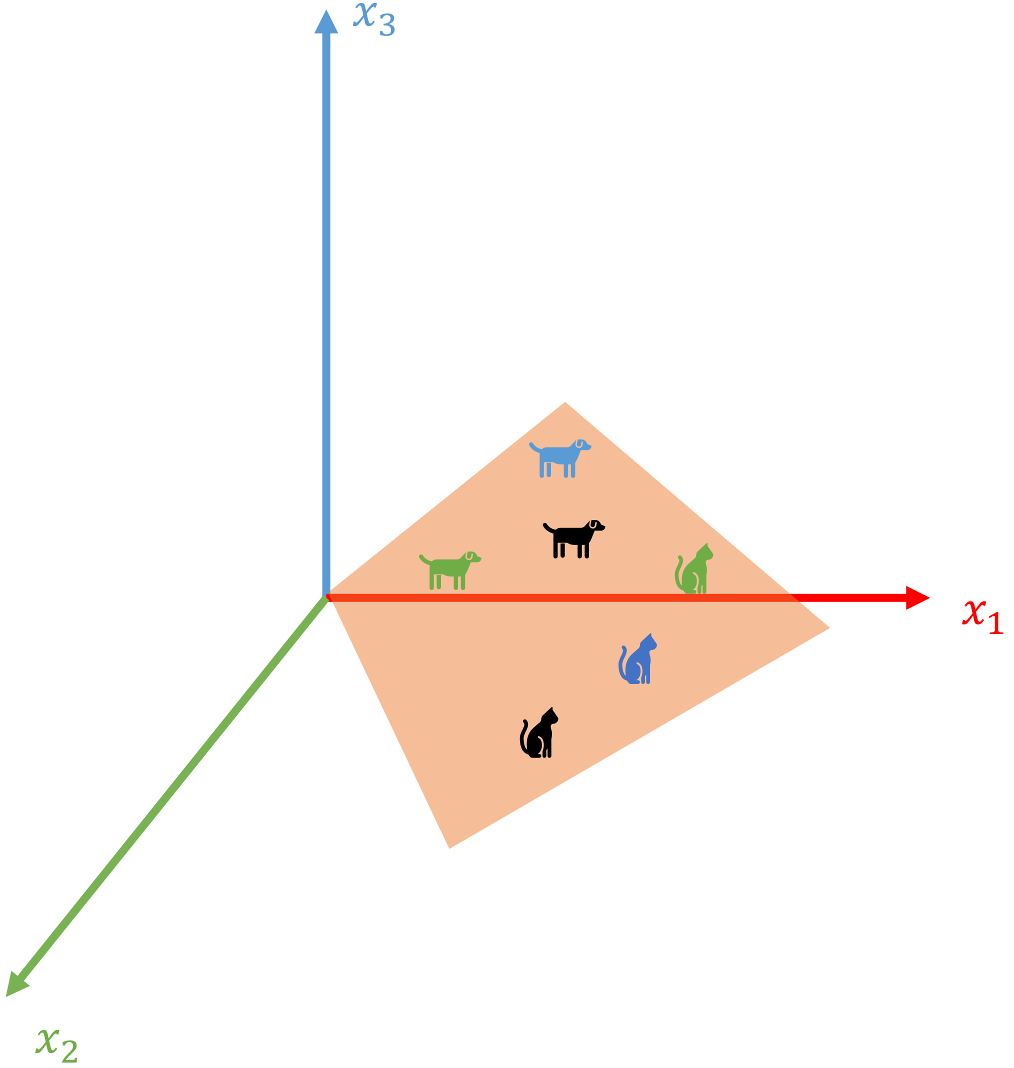
A: $M \times N$

U: $M \times M$, Orthogonal matrix

$$UU^T = \mathbb{I}_M$$

D: $M \times N$, Diagonal matrix

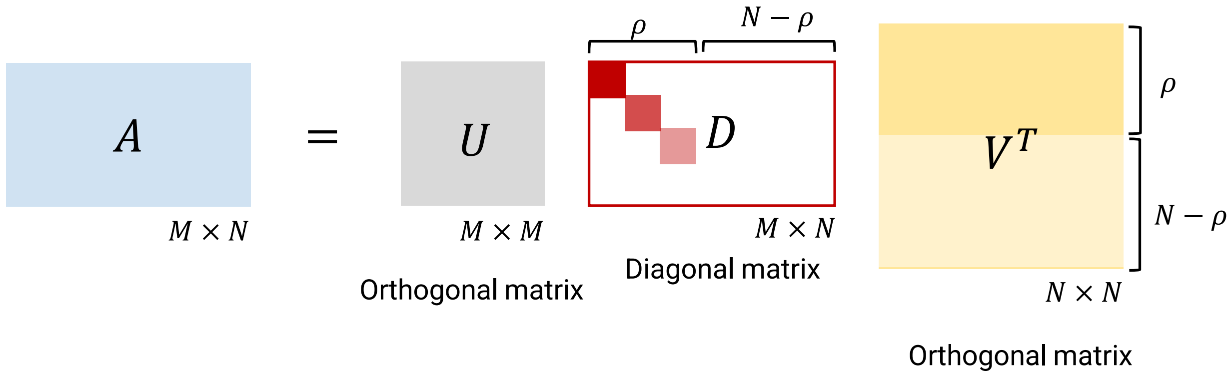
V^T: $N \times N$, Orthogonal matrix



Nullspace

$$\underset{M \times N}{A}\ \underset{N \times 1}{\mathbf{x}} = \underset{M \times 1}{\mathbf{0}}$$

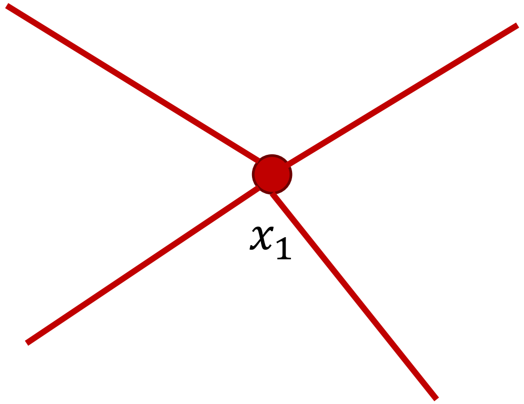
$\mathbf{x} = \mathrm{null}(A)$, the set of all vectors x that satisfy the above equation (the rank of A is $\rho \leq \min(M, N)$)



$$V_{\rho+1:N} = \mathrm{null}(A)$$

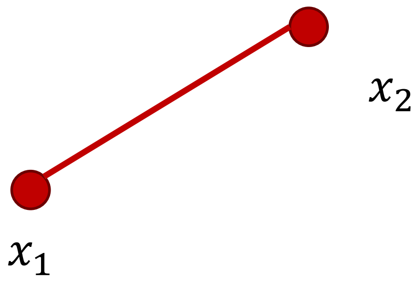
If $\rho = N$, the matrix has full rank, and the null space is unique (i.e., 0)

Line Fitting 101



Under constrained problem: ∞ Solutions

Line Fitting 101



$$ax + by + c = 0$$
$$ax_1 + by_1 + c = 0$$
$$ax_2 + by_2 + c = 0$$

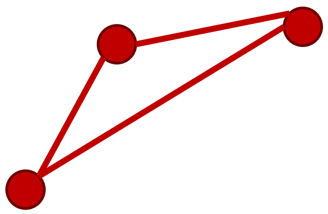
Form $A\mathbf{x} = \mathbf{0}, \mathbf{x} = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

Perfectly constrained problem: Unique Solution

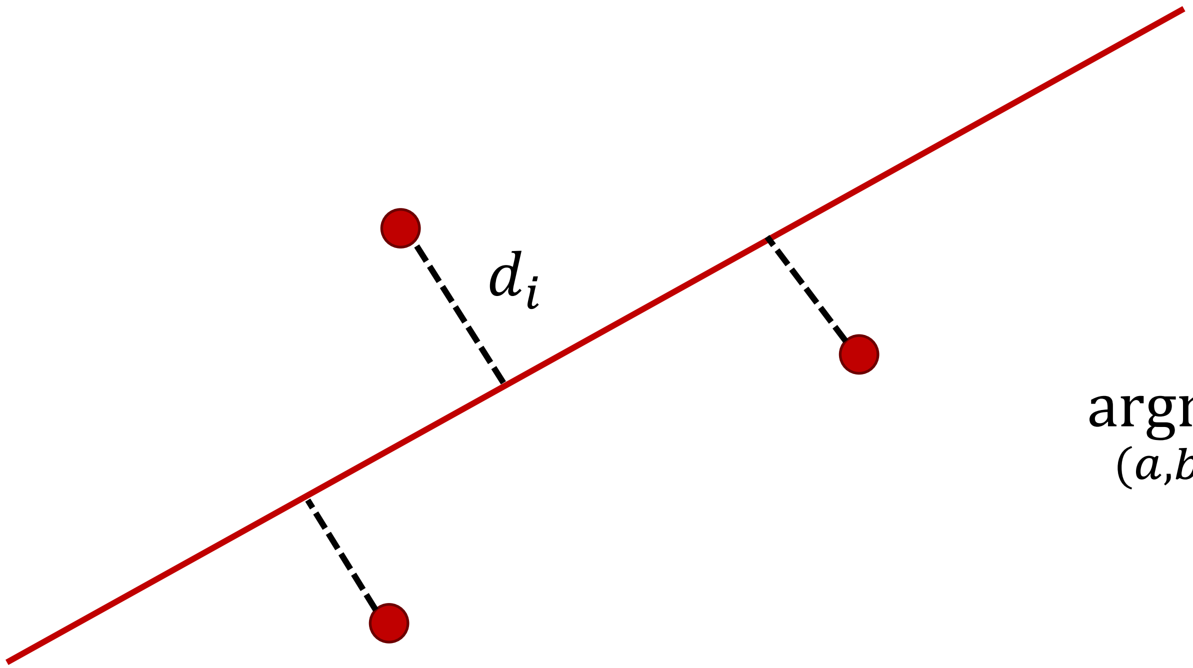
Can be written as $\underset{x=(a,b,c)}{\operatorname{argmin}} \|A\mathbf{x}\|_2^2 \; s.t. \|\mathbf{x}\|_2 = 1$

Avoids trivial solution

Line Fitting 101



Line Fitting 101



$$\underset{(a,b,c)}{\operatorname{argmin}} \sum_{\forall i} \|d_i\|_2^2$$

$A\mathbf{x} = \mathbf{b}$

```
x = A\b;
```

MATLAB

Instead of $ax + by + c = 0$
Let's use $y = mx + d$

$A \; \mathbf{x} \approx \mathbf{b}$

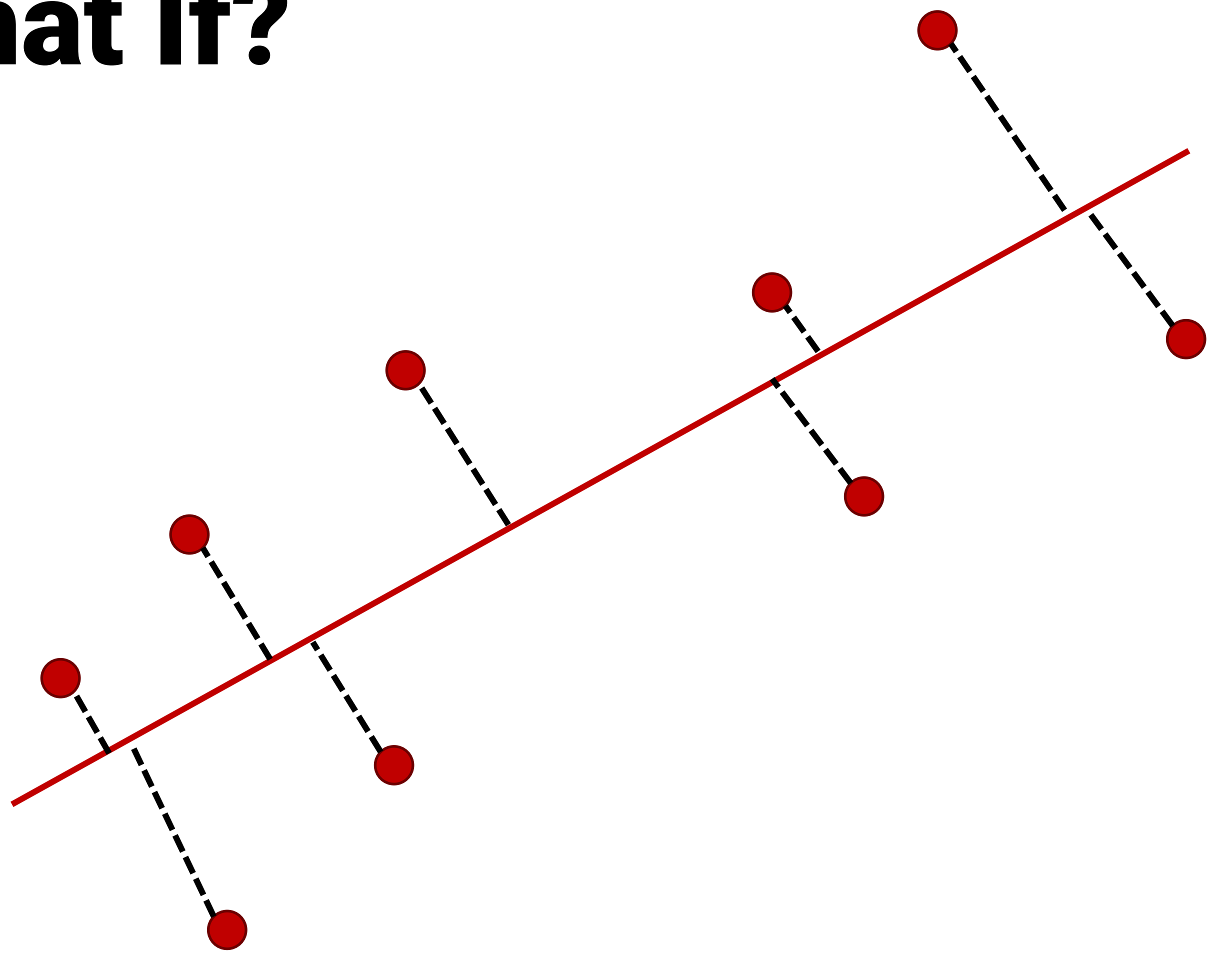
How do we solve this?
Use a loss function (metric) that needs to be minimized (optimized)

Let's minimize $\text{argmin}_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}||_2^2$

How do you obtain closed form solution?
Differentiate and set to zero!

Solved using $\mathbf{x} = \left(A^T A\right)^{-1} A^T \mathbf{b}$
Called **Ordinary Least Squares (OLS)**

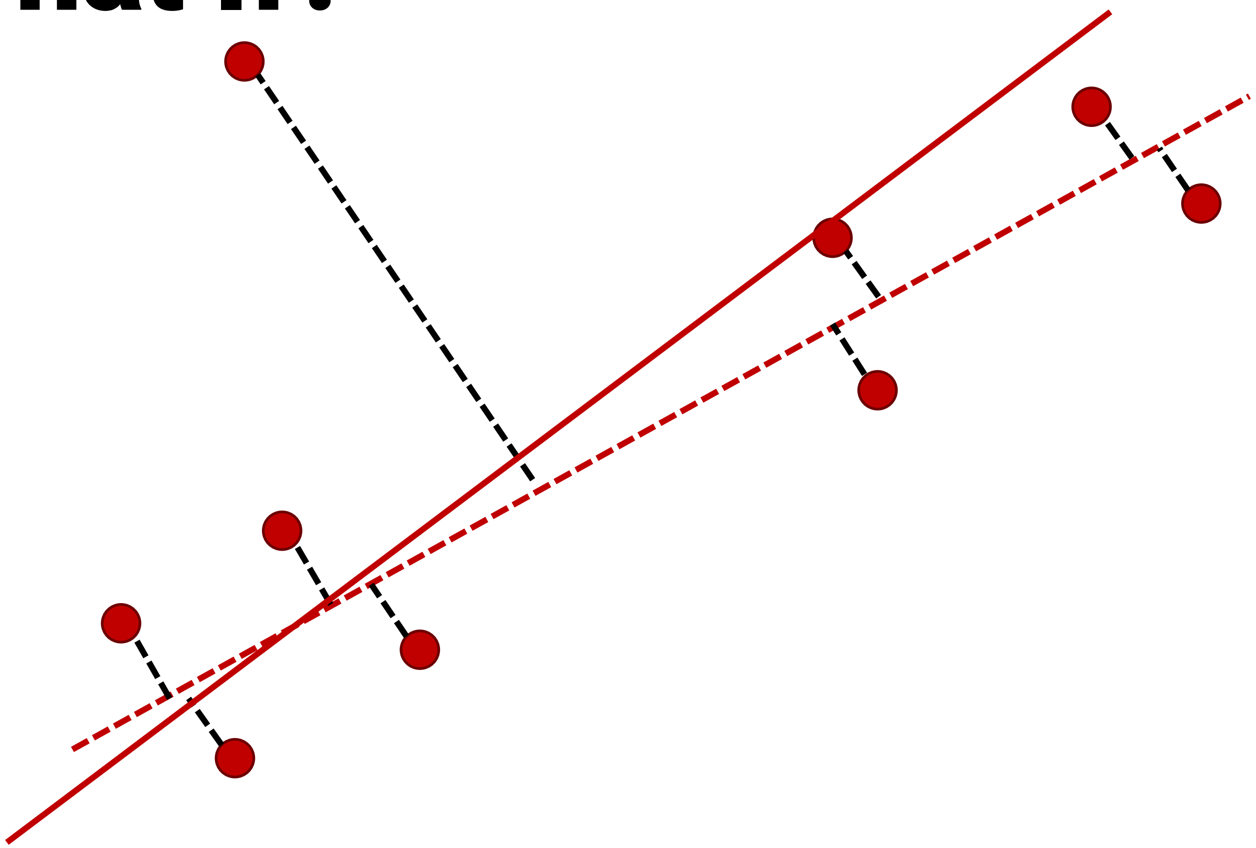
What If?



$$\underset{(a,b,c)}{\operatorname{argmin}} \sum_{\forall i} \|d_i\|_2^2$$

Same as before!

What If?



$$\underset{(a,b,c)}{\operatorname{argmin}} \sum_{\forall i} \|d_i\|_2^2$$

Outliers!

You can regularize!

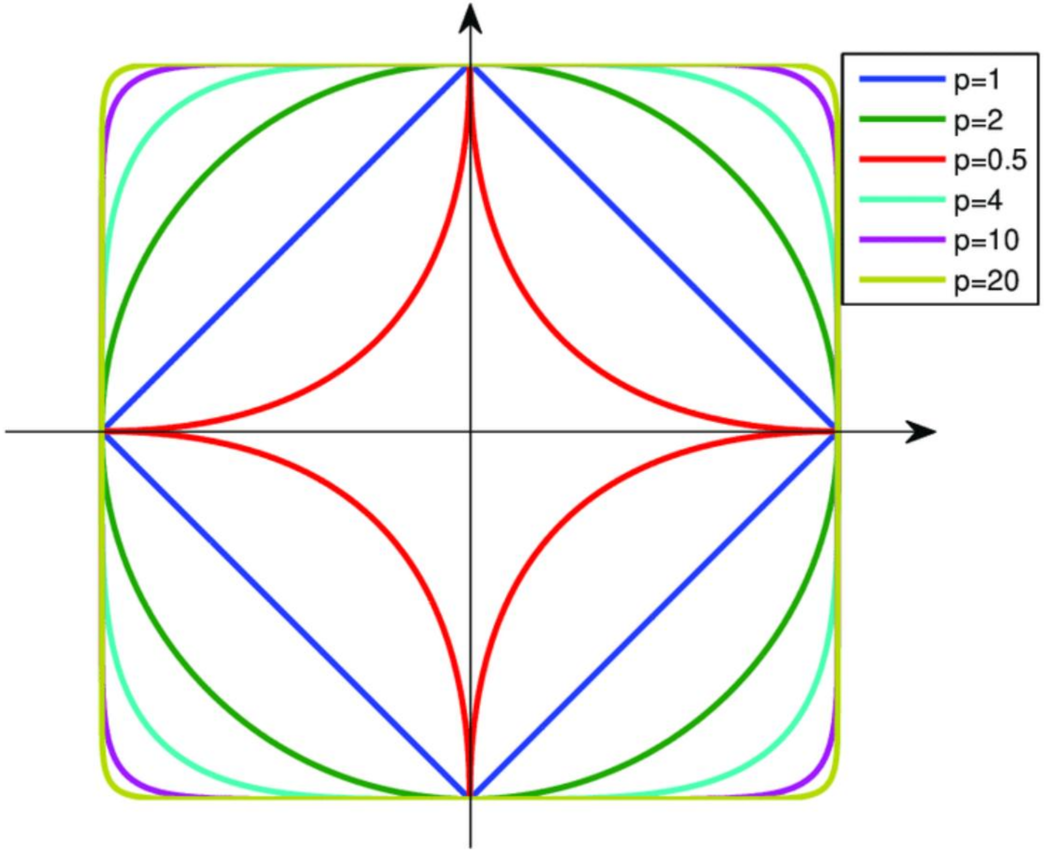
$$\underset{(a,b,c)}{\operatorname{argmin}} \sum_{\forall i} \|d_i\|_2^2 + \lambda \left\| \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right\|_2^2$$

Penalize for large values in model! (This penalty discourages the model from assigning large coefficients to features)

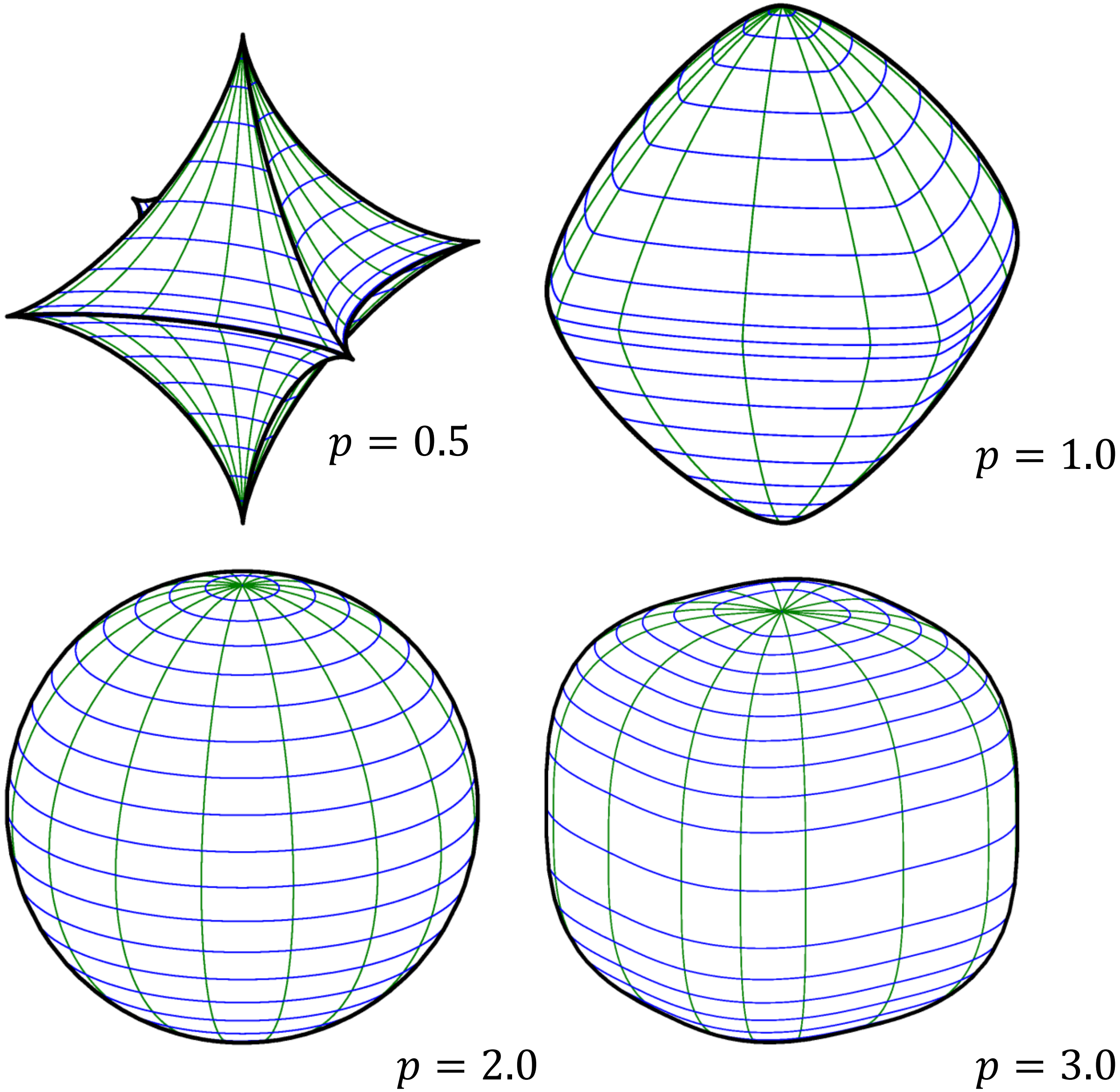
Penalty or Regularization can be l_p norm

l_p norm: $\|\mathbf{x}\|_p = \left(\sum_{\forall i} x_i^p\right)^{\frac{1}{p}}$

World Of Norms!



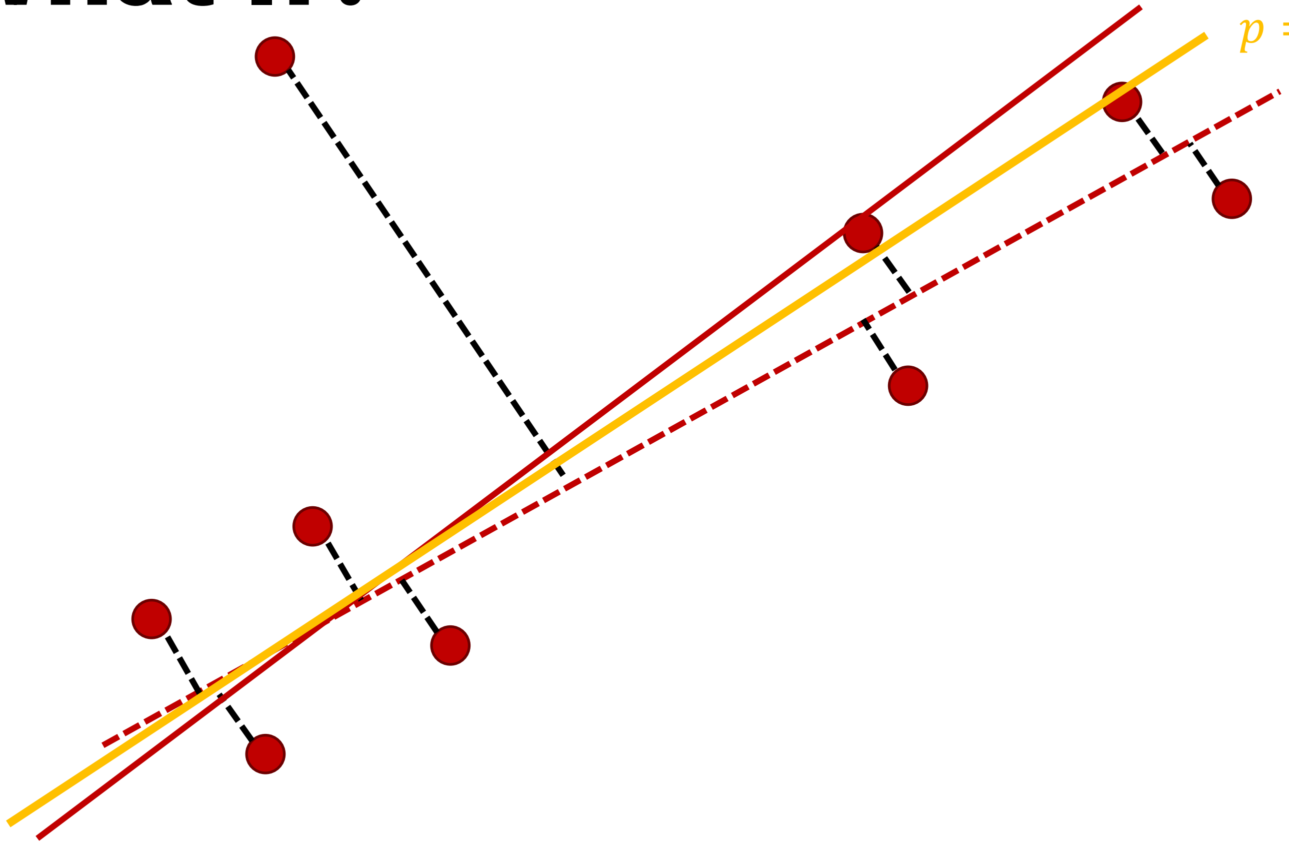
Help alleviate different kinds of noise!



What If?

$p = 2$ Called **Ridge Regression**

$p = 1$ Called **Lasso Regression**



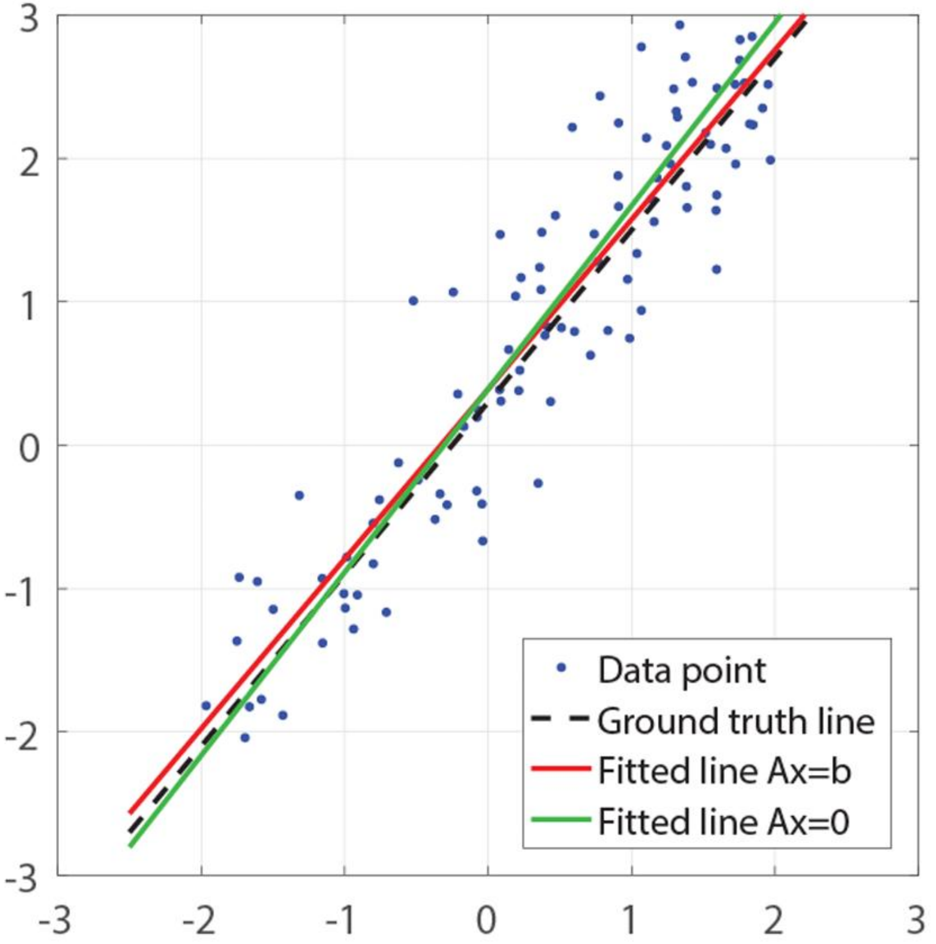
$$\underset{(a,b,c)}{\operatorname{argmin}} \sum_{\forall i} \|d_i\|_2^2 + \lambda \left\| \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right\|_p$$

l_p norm: $\|\mathbf{x}\|_p = \left(\sum_{\forall i} x_i^p\right)^{\frac{1}{p}}$

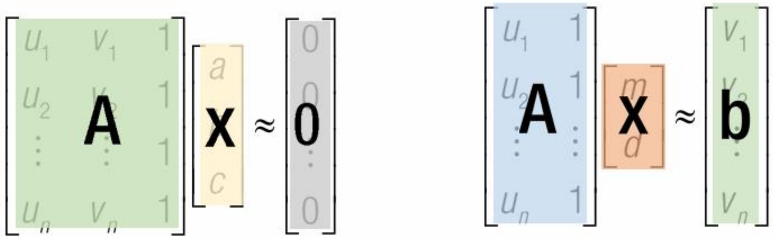
Sensitivity to outliers is controlled by p!

When p goes down, you also reject variation in signal!
Careful tradeoff between **sensitivity** to changing in signal and **robustness** to noise!

$A\mathbf{x} = \mathbf{0}$ Vs $A\mathbf{x} = \mathbf{b}$



$$\begin{aligned} au_1 + bv_1 + c &\approx 0 \\ au_2 + bv_2 + c &\approx 0 \\ &\vdots \\ au_n + bv_n + c &\approx 0 \end{aligned} \quad \longrightarrow \quad \begin{aligned} v_1 &\approx mu_1 + d \\ v_2 &\approx mu_2 + d \\ &\vdots \\ v_n &\approx mu_n + d \end{aligned}$$



$A\mathbf{x} = 0$

- No sol. Inconsistent
- Unique sol.
- ∞ sols.

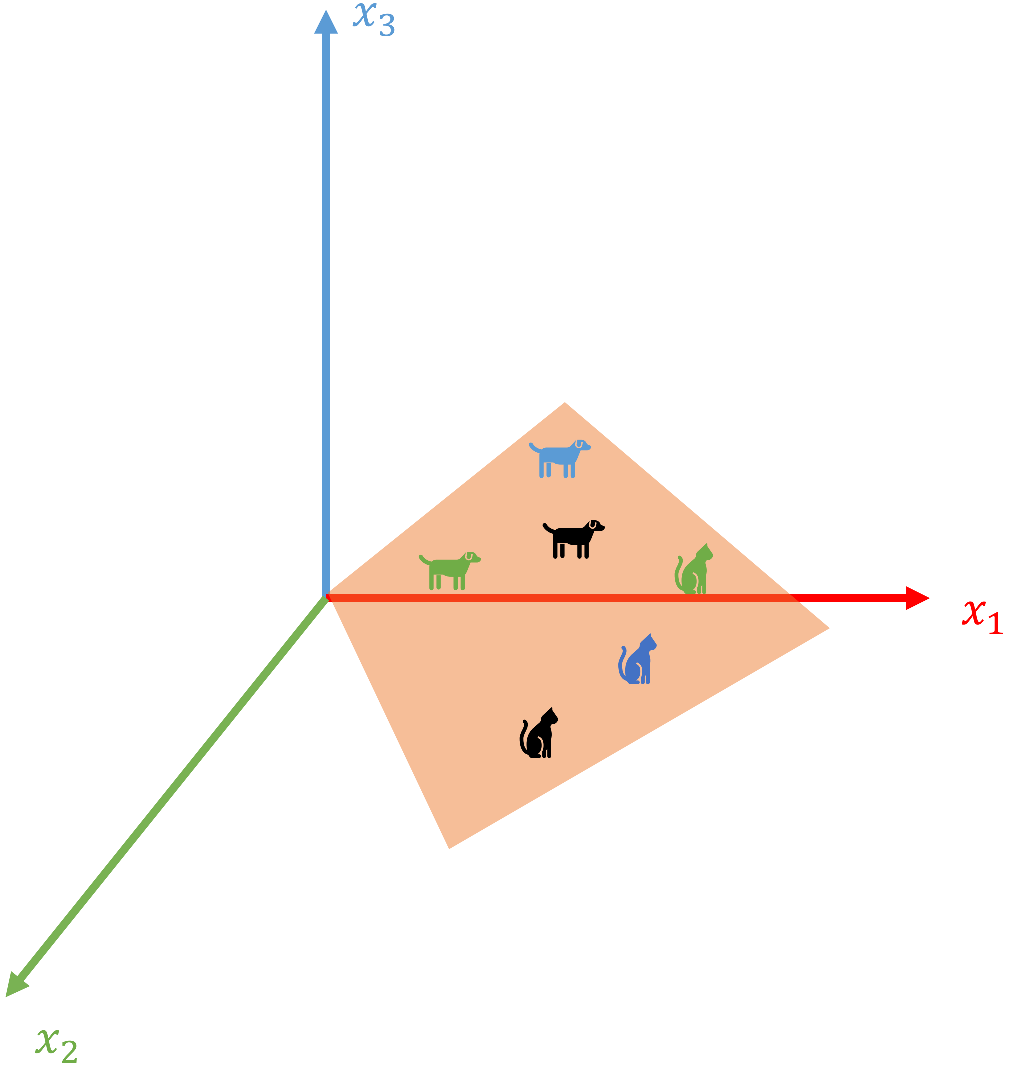
(Unique sol. and ∞ sols.: Consistent)

$A\mathbf{x} = \mathbf{b}$

- Unique sol. Trivial sol. $\mathbf{x} = \mathbf{0}$
- ∞ sols. Non-trivial sol. $\mathbf{x} \neq \mathbf{0}$

Slide adapted from UMN's CSCI5563

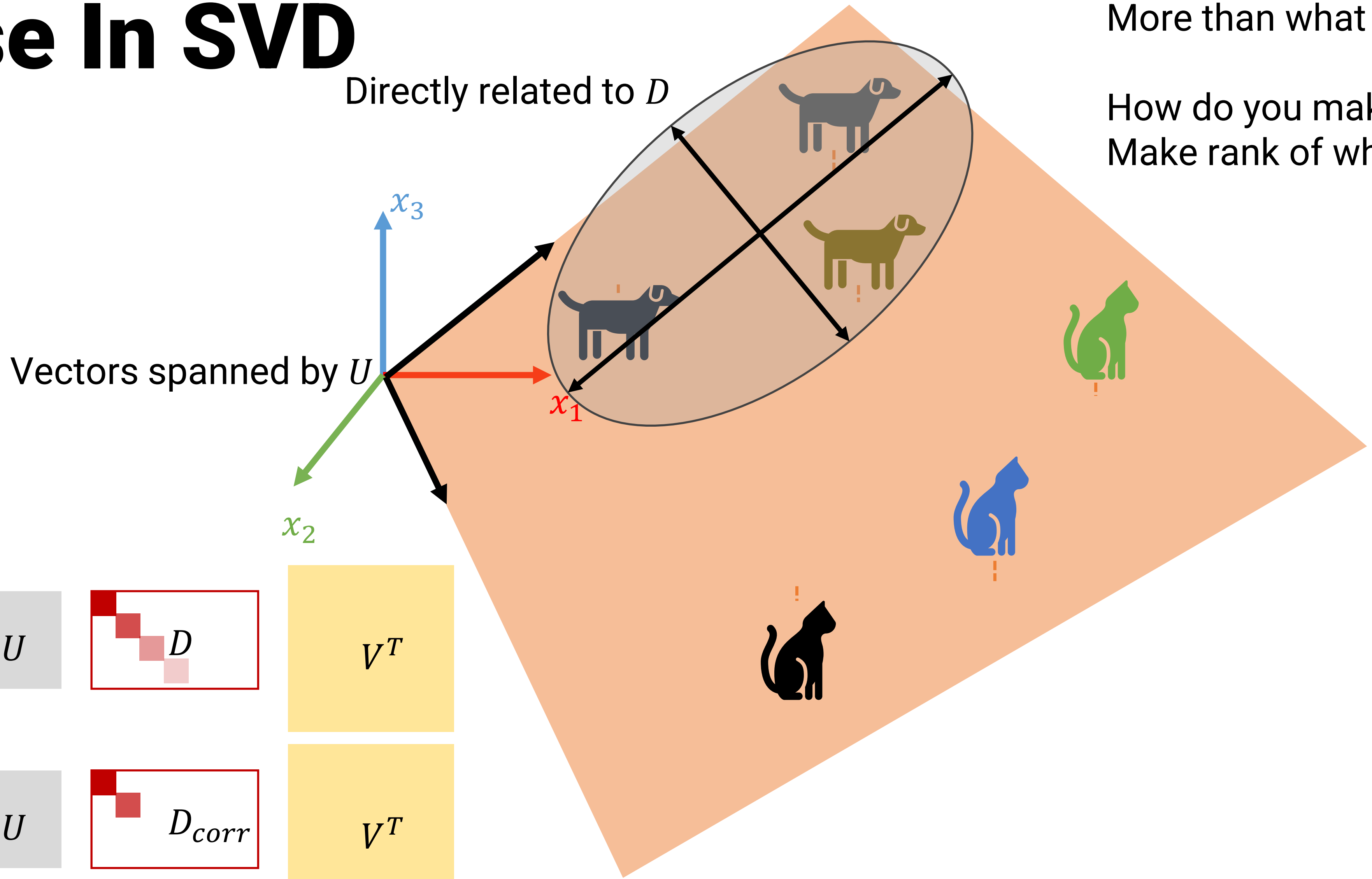
Noise In SVD



Noise In SVD

Not all one plane!
What will be rank of D?
More than what it should be!

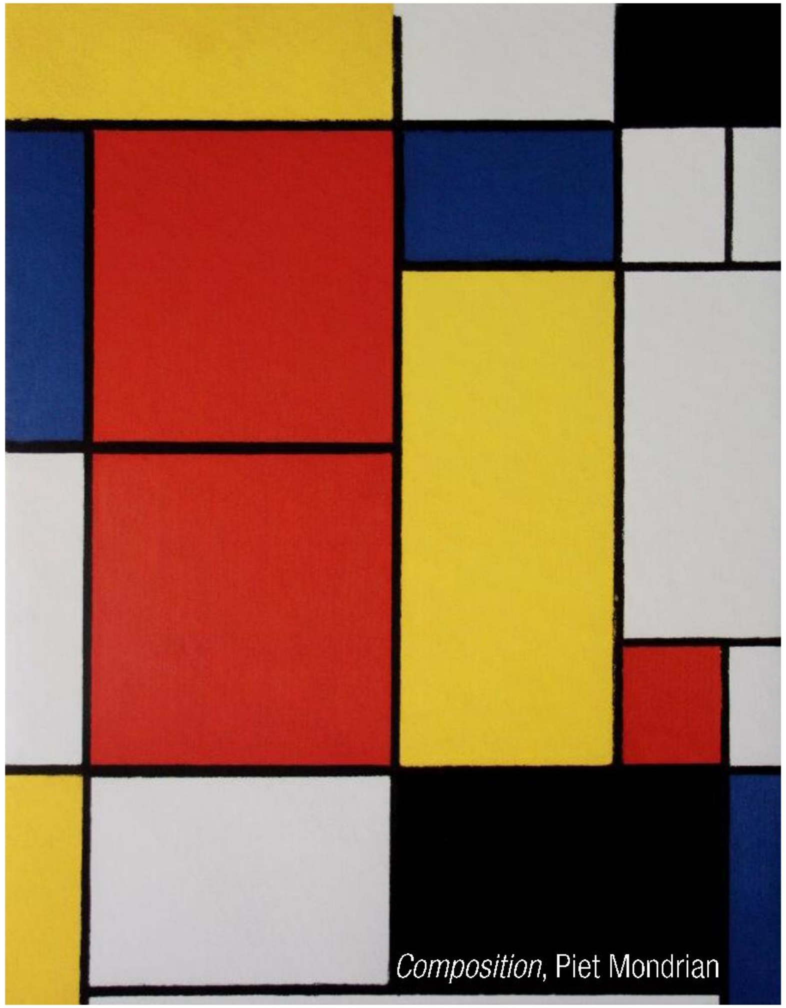
How do you make it work?
Make rank of what it should be!

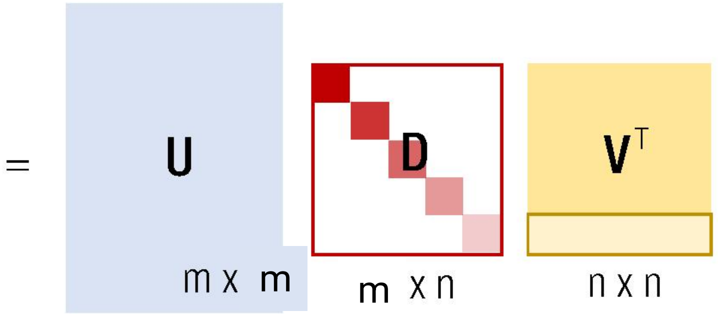


$A = U \; D \; V^T$

$A_{corr} = U \; D_{corr} \; V^T$

SVD On Images

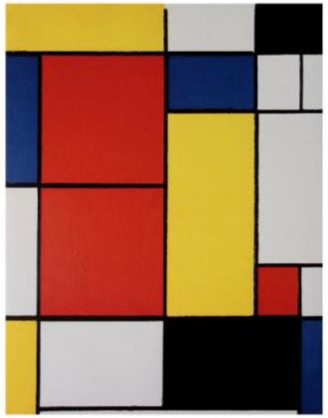

Composition, Piet Mondrian



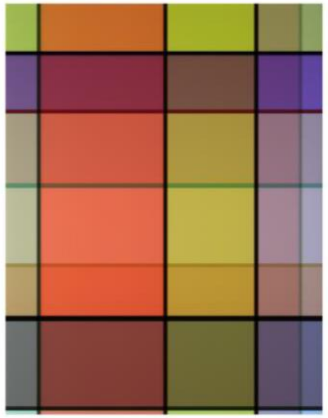
Slide adapted from UMN's CSCI5563

SVD On Images

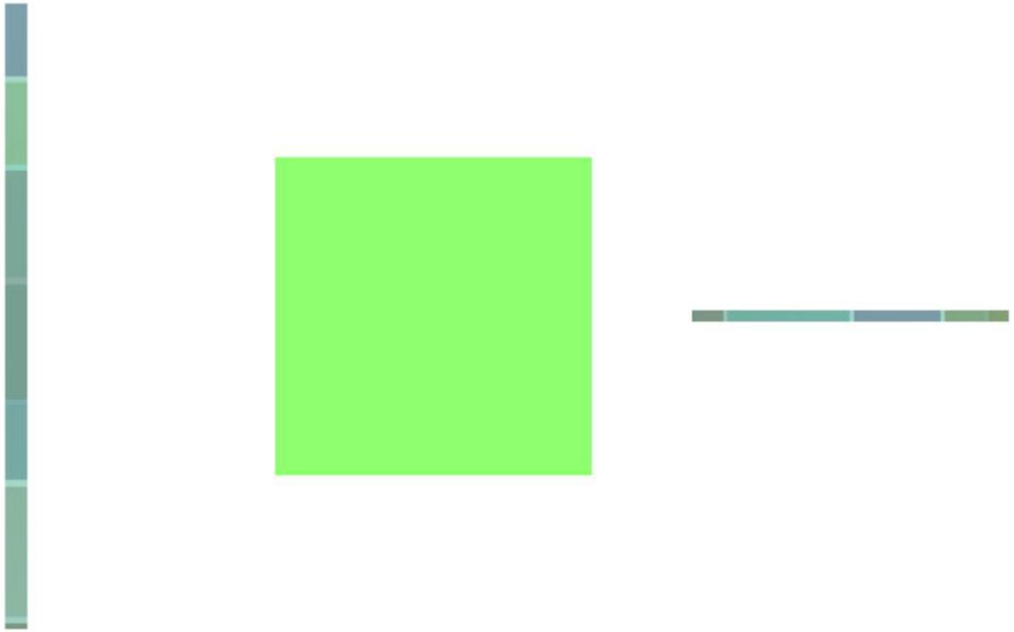
MondrianSVD.m



Ground truth



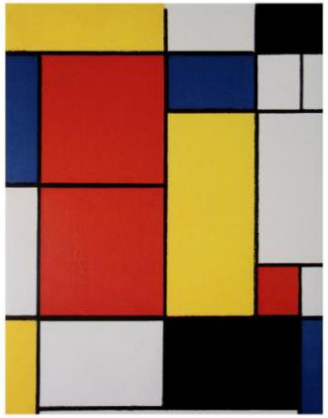
Number of basis: 1



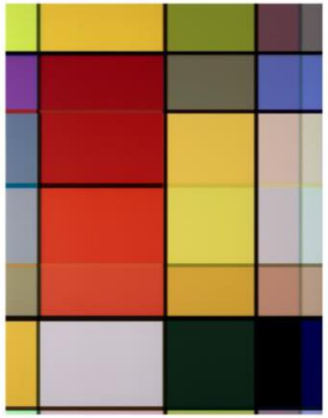
$$A = U \; D \; V^T$$

SVD On Images

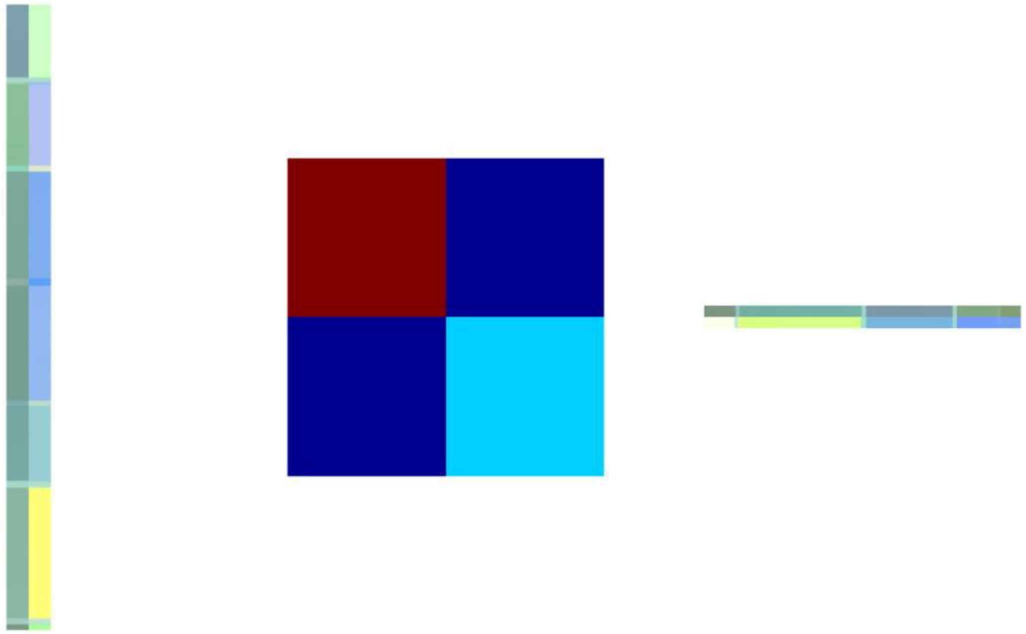
MondrianSVD.m



Ground truth



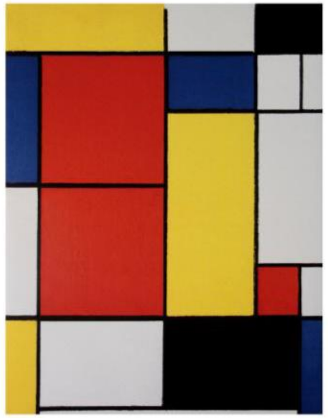
Number of basis: 2



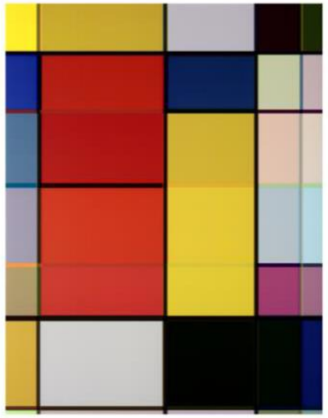
$$A = U \quad D \quad V^T$$

SVD On Images

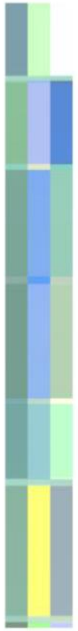
MondrianSVD.m

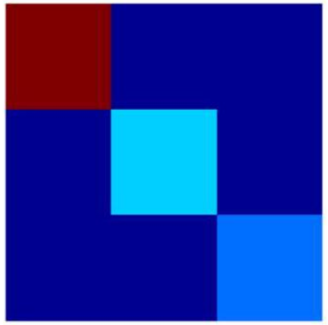


Ground truth



Number of basis: 3

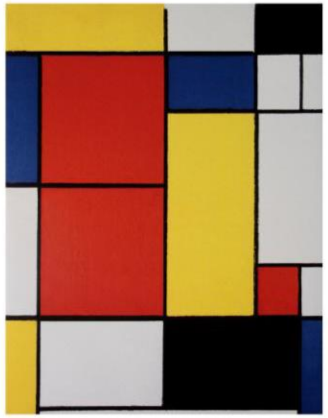




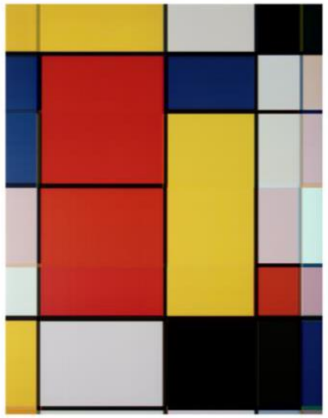
$$A = U \quad D \quad V^T$$

SVD On Images

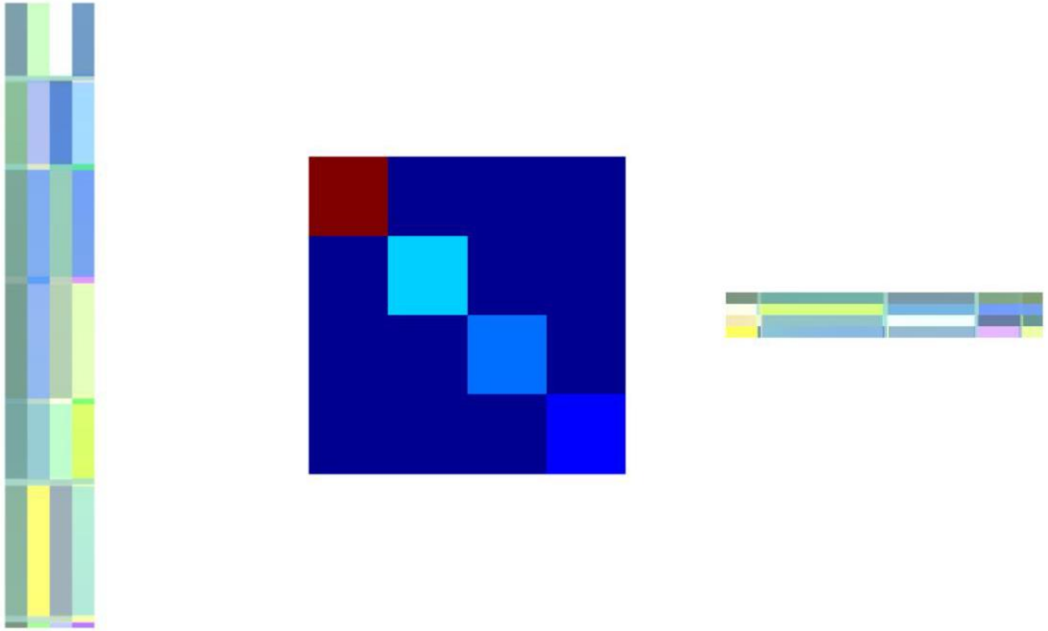
MondrianSVD.m



Ground truth



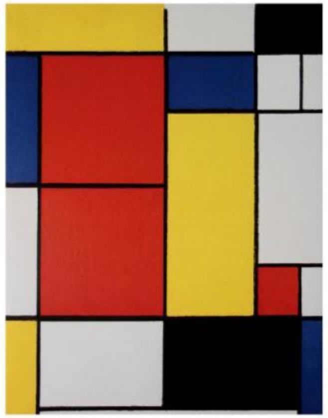
Number of basis: 4



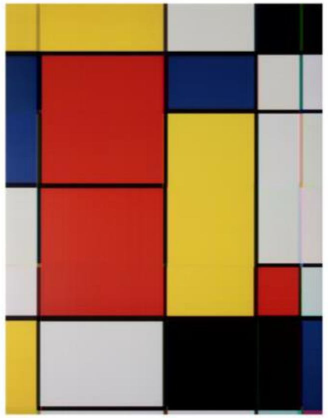
$$A = U \quad D \quad V^T$$

SVD On Images

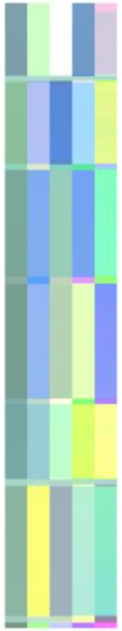
MondrianSVD.m

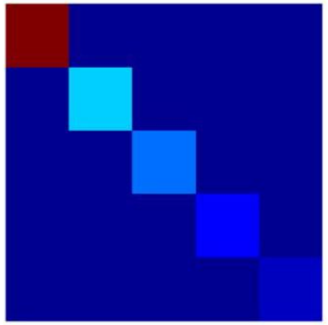


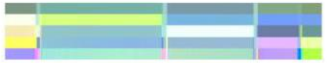
Ground truth



Number of basis: 5

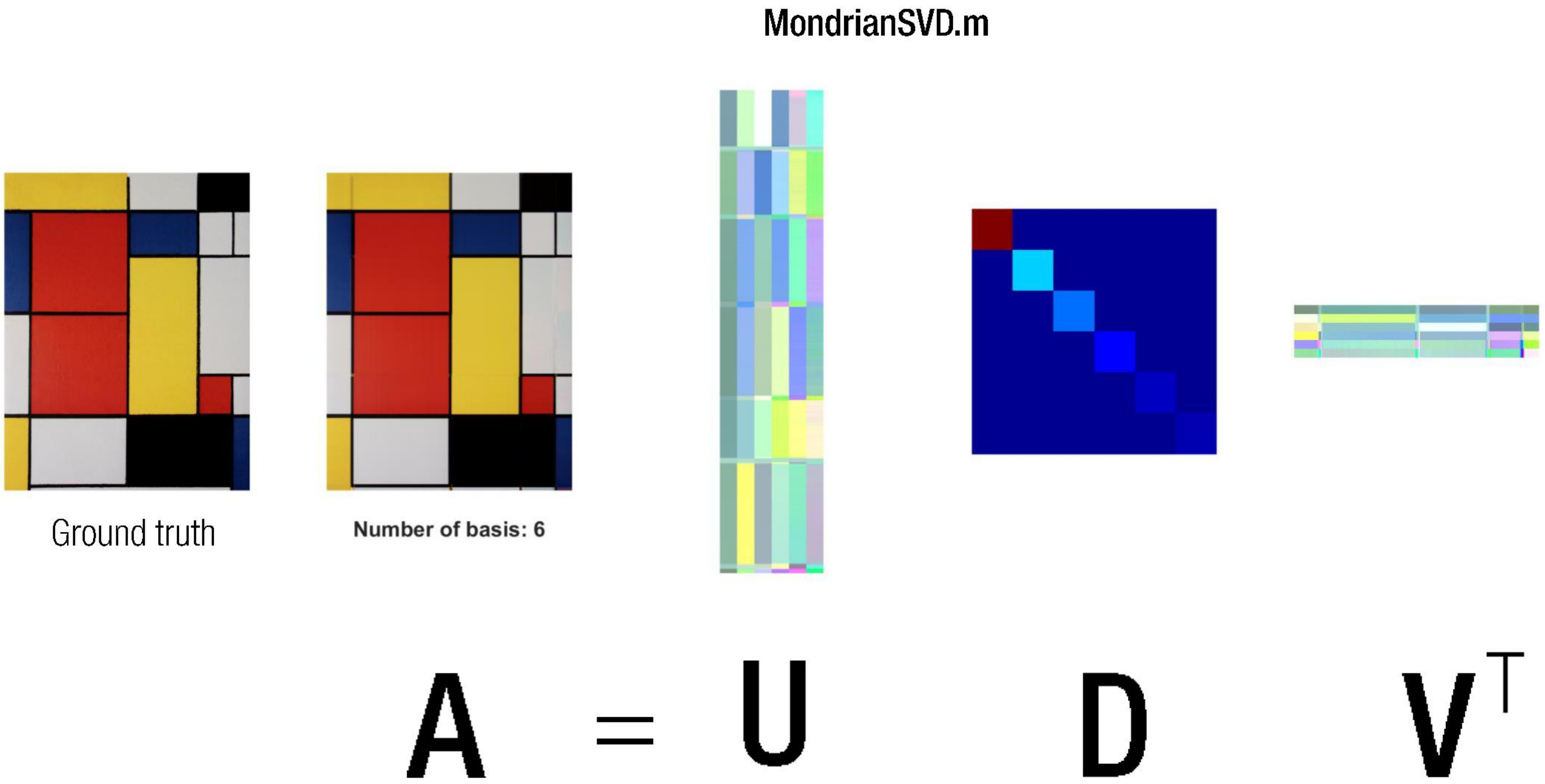






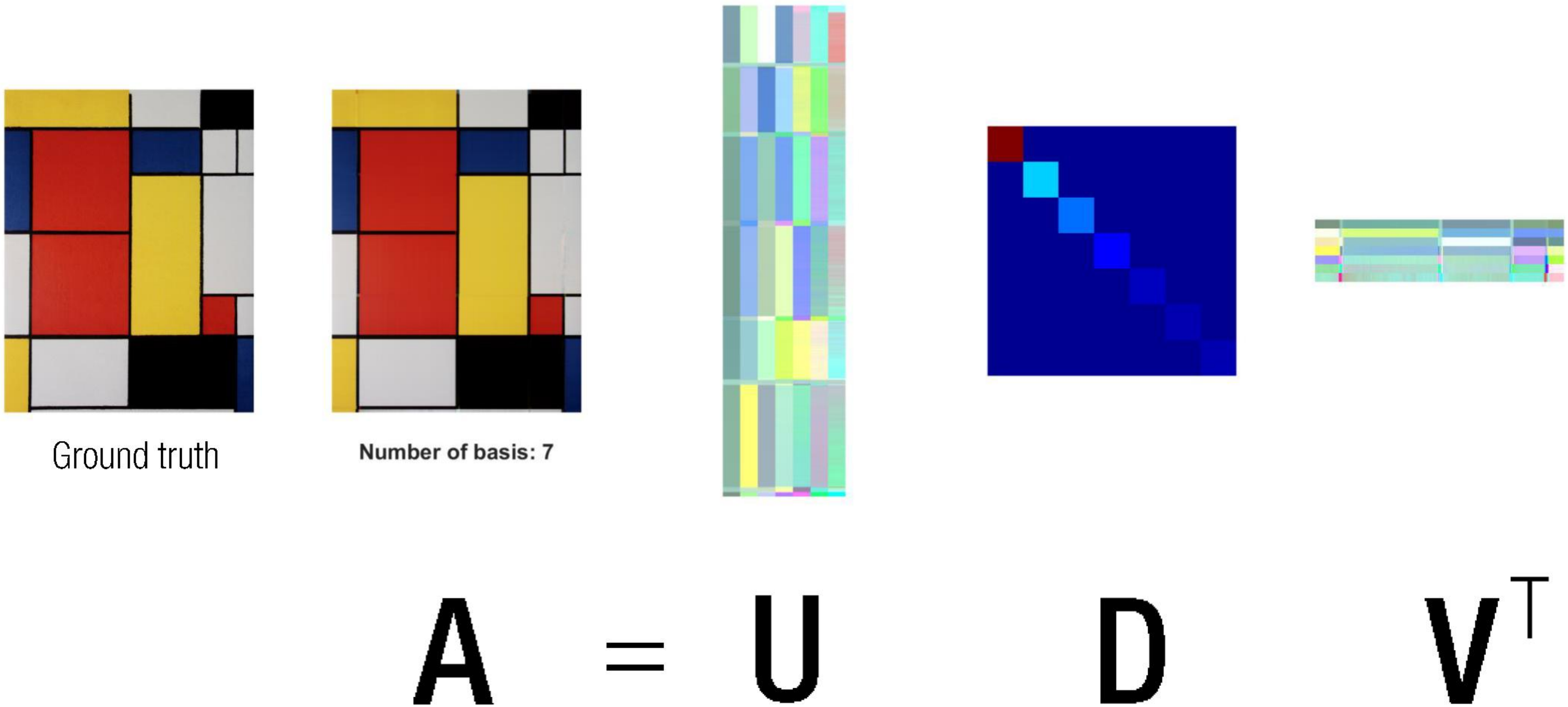
$$A = U \quad D \quad V^T$$

SVD On Images

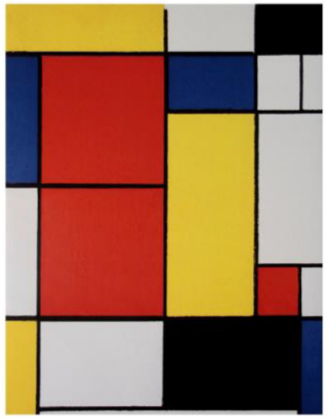


SVD On Images

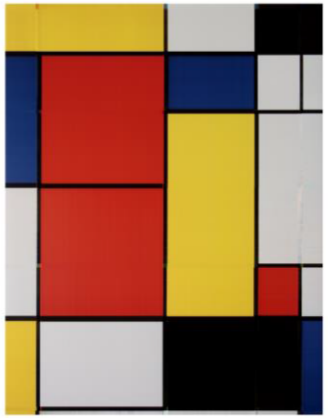
MondrianSVD.m



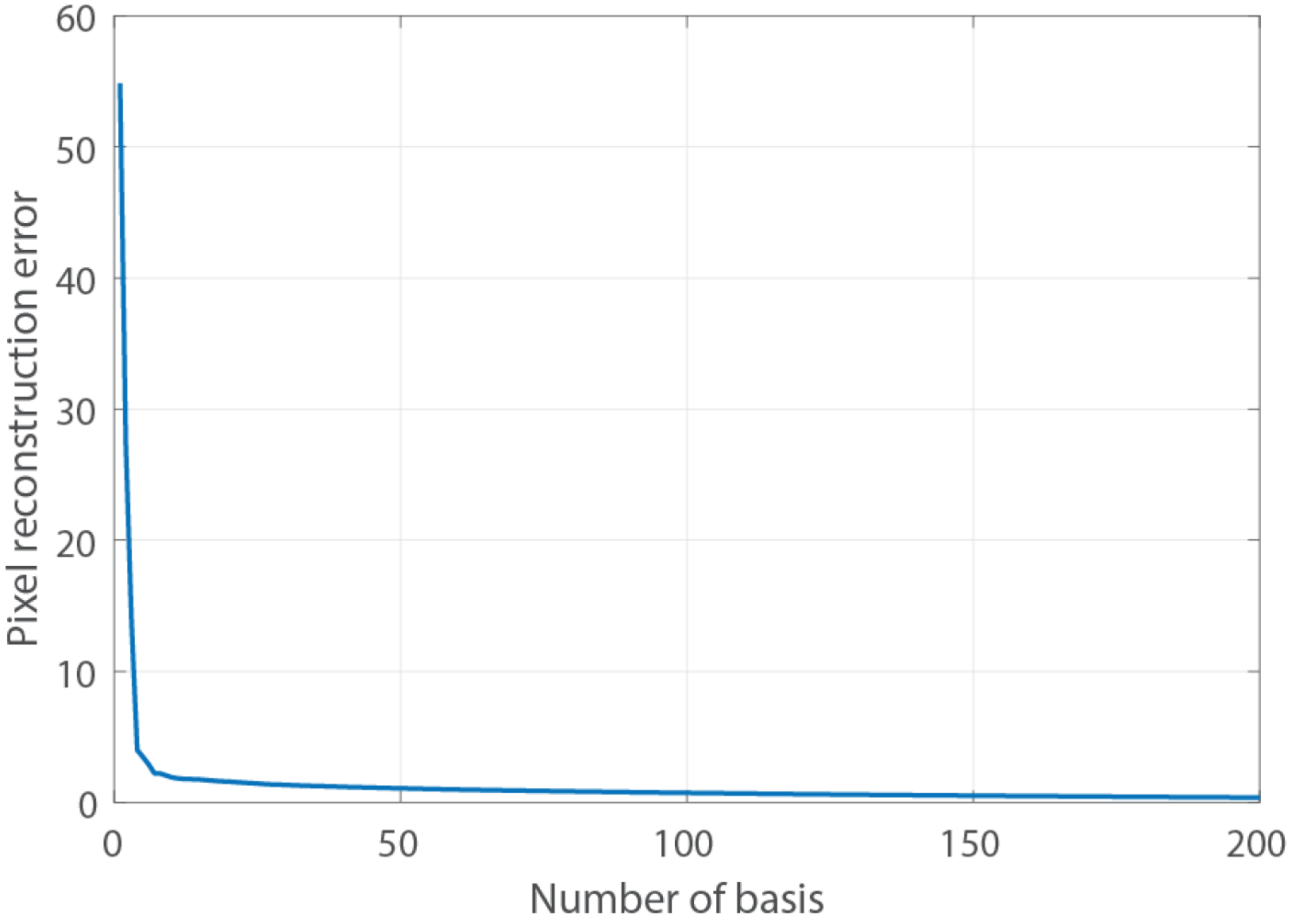
Reconstruction Error



Ground truth



Number of basis: 7



A

How do you get p from Y?
You need some function!
$p = g(\tilde{Y})$

Back To Linear Regression

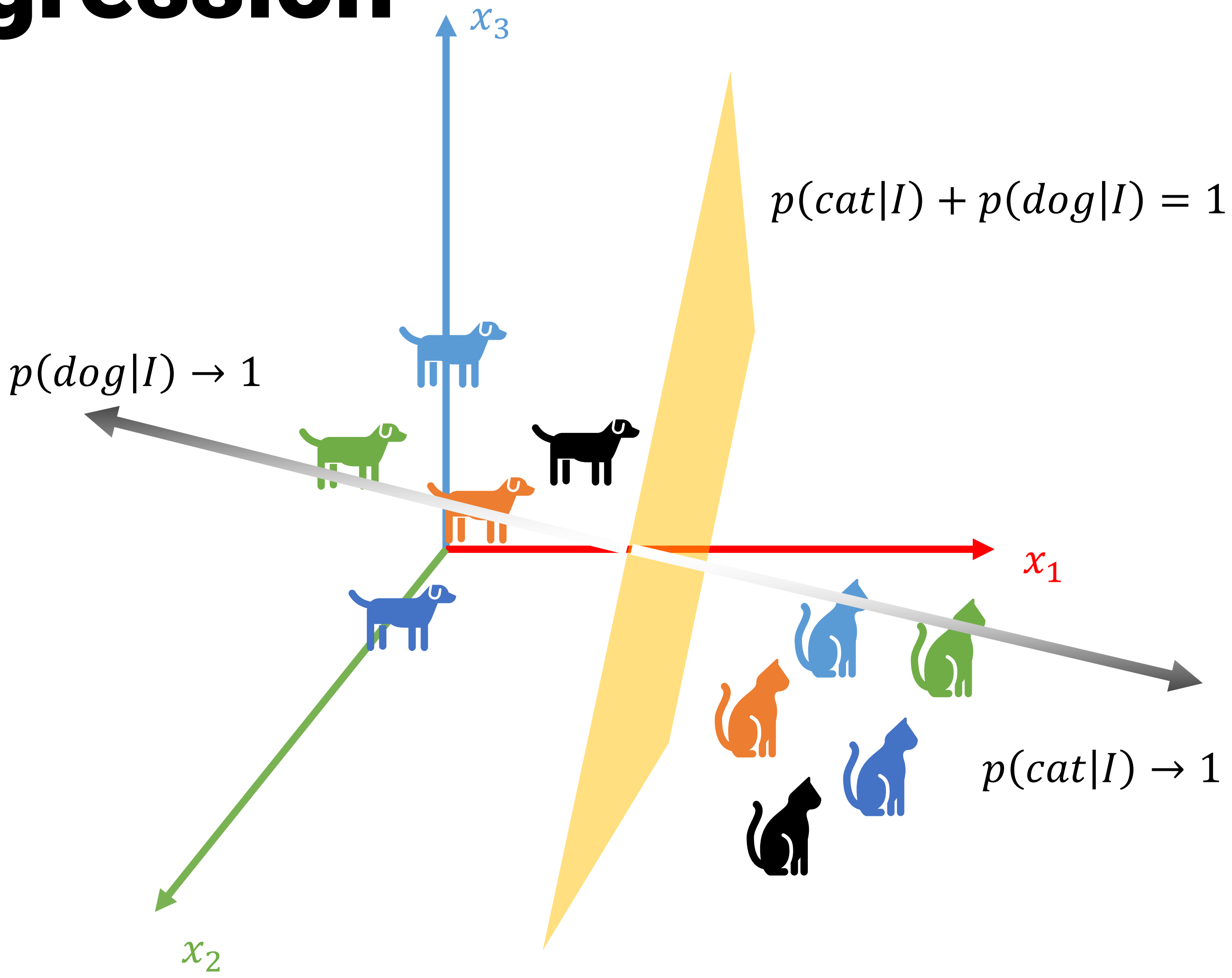
$Y = W^T \begin{bmatrix} X \\ 1 \end{bmatrix}$
$Y = AX$
$X = A^{-1}Y$

When is this valid?
A is square and invertible!

What if A is not square?
$A^{-1} \approx A^{\dagger}$

How do you get the pseudo-inverse?
SVD!

$$A^{+} = VD^{+}U^{*}$$

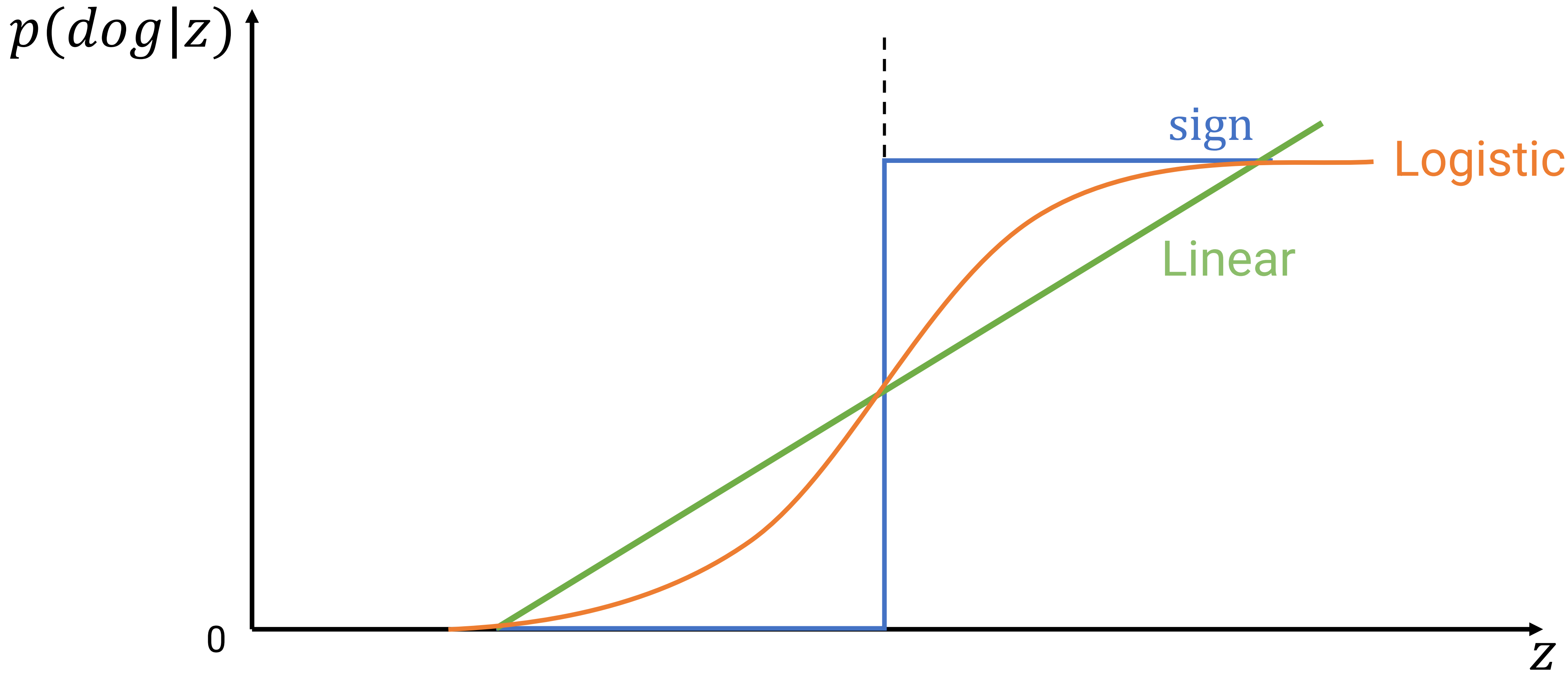


$\tilde{Y} \rightarrow p$

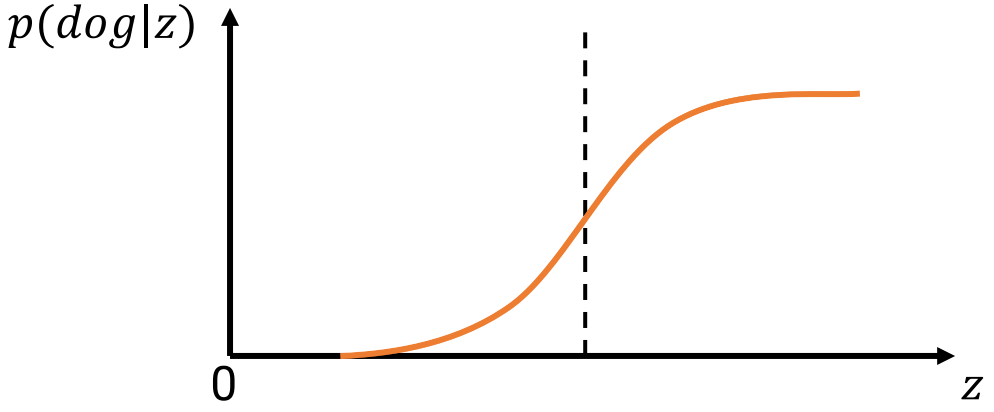
$\mathbb{R} \rightarrow [0,1]$

Function To Get Probabilities

Some squashing function

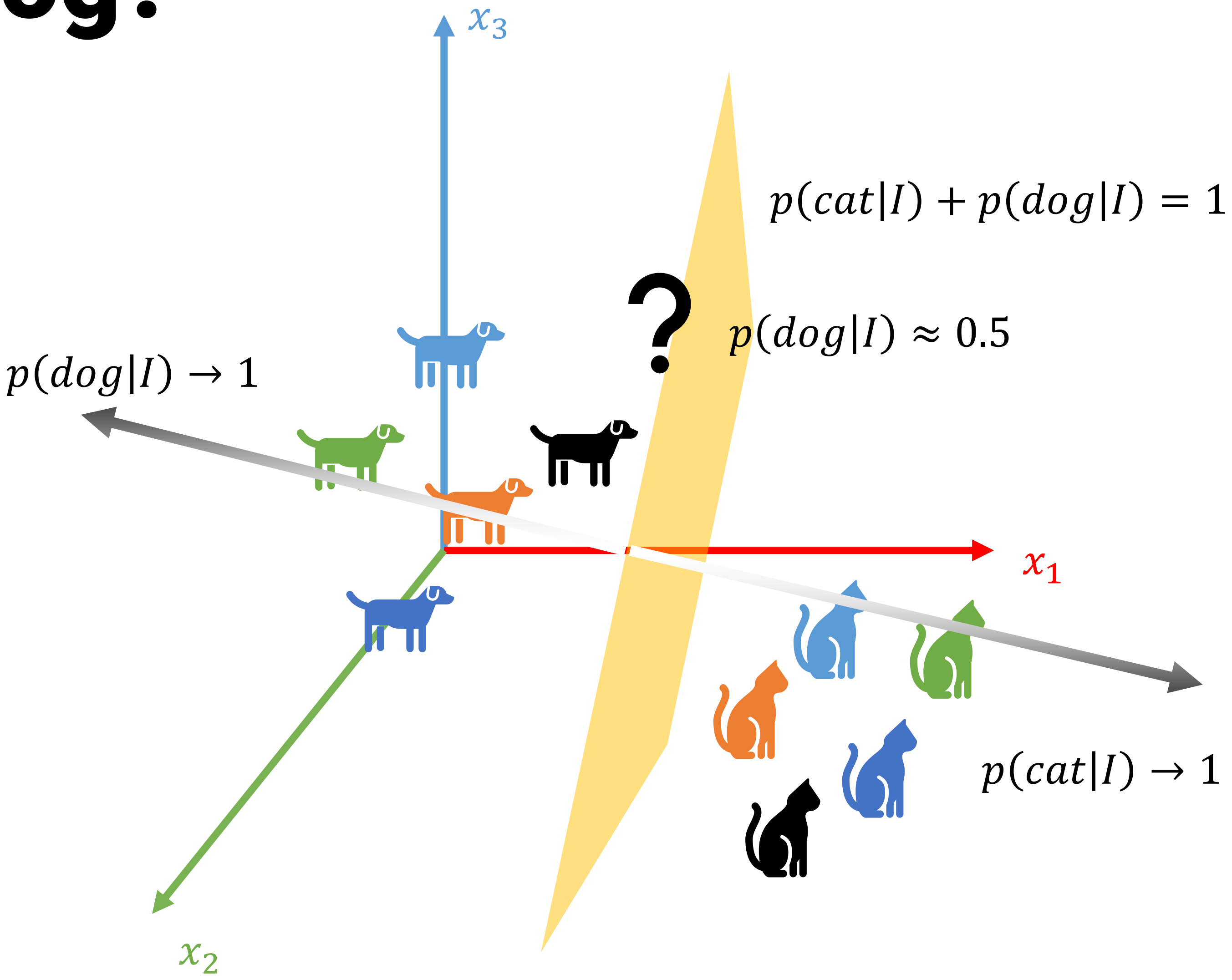


Logistic Regression

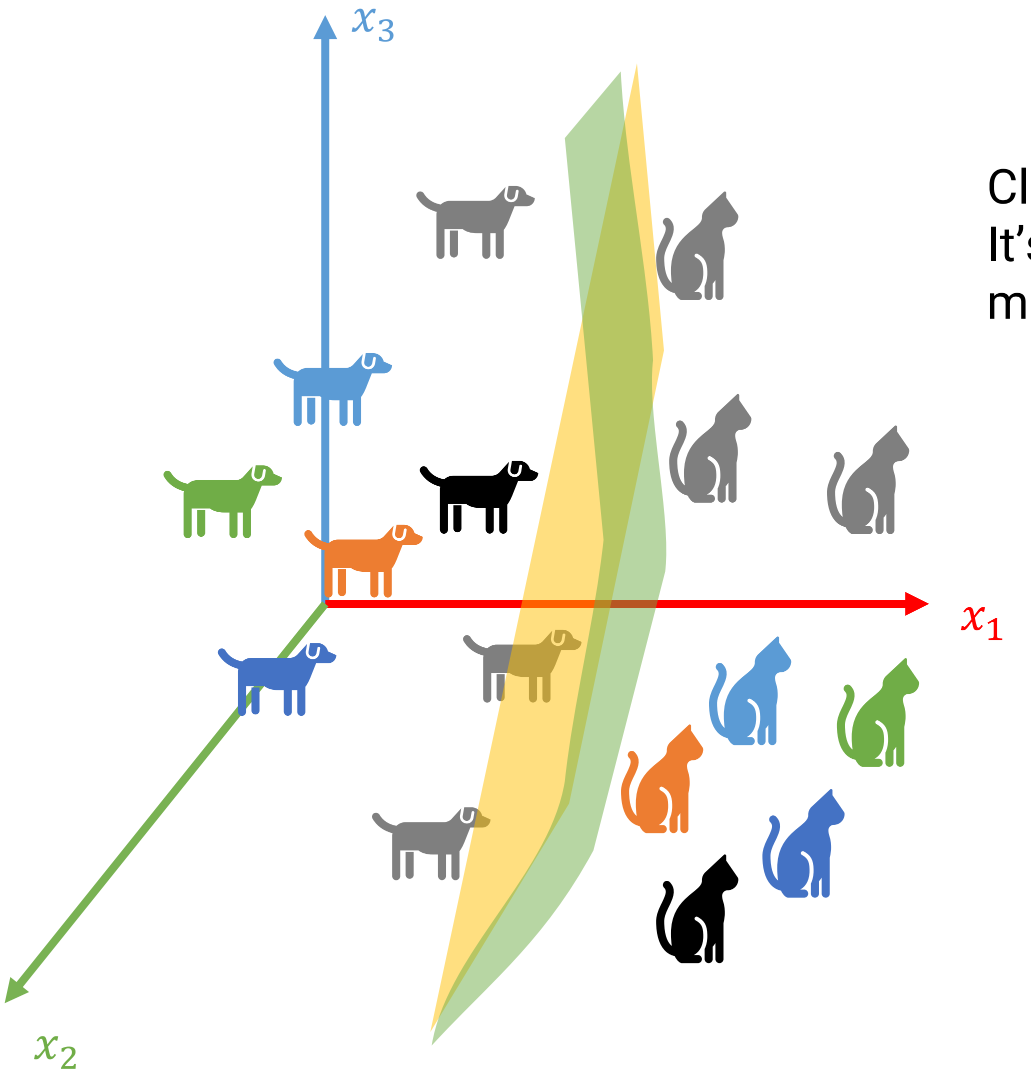


- **Sigmoid function** to obtain probabilities of classification
- Mathematically, $p = \frac{1}{1+e^{-(W^T X)}}$
- Both class probabilities are related using Log-Odds $z = \log\frac{p}{1-p} = W^T X$(Also called **Logits**)
- Sometimes also called **Perceptron**

Is This a Cat or a Dog?

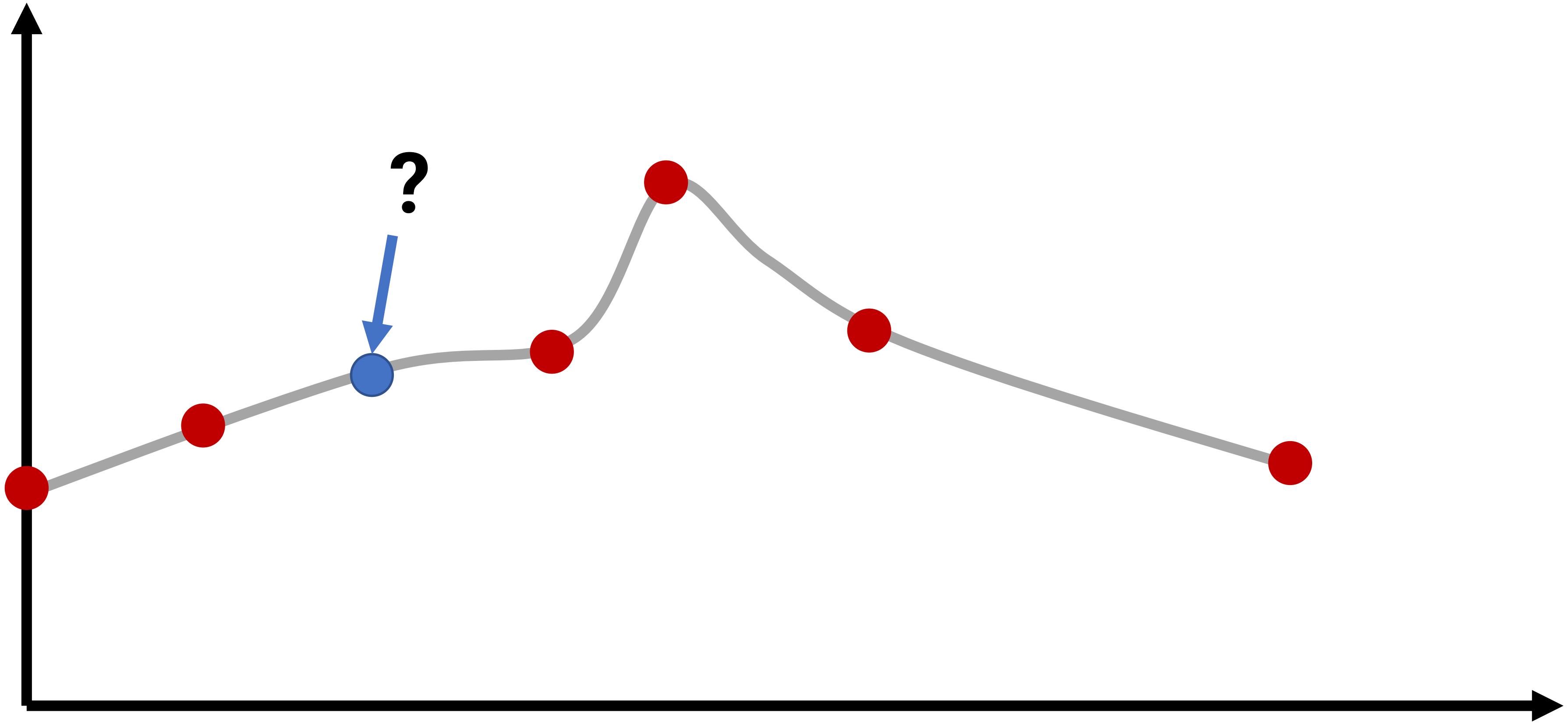


Why Did This Happen?



Classification boundary is **not linear**! It's actually a non-linear function map or a **manifold**!

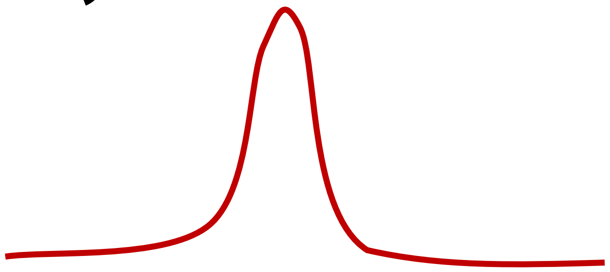
Interpolation



Radial Basis Function (RBF)

Instead use a Kernel

From Machine Learning/Math lingo!



- Smooth
- "Simple"
- Nice properties you want!

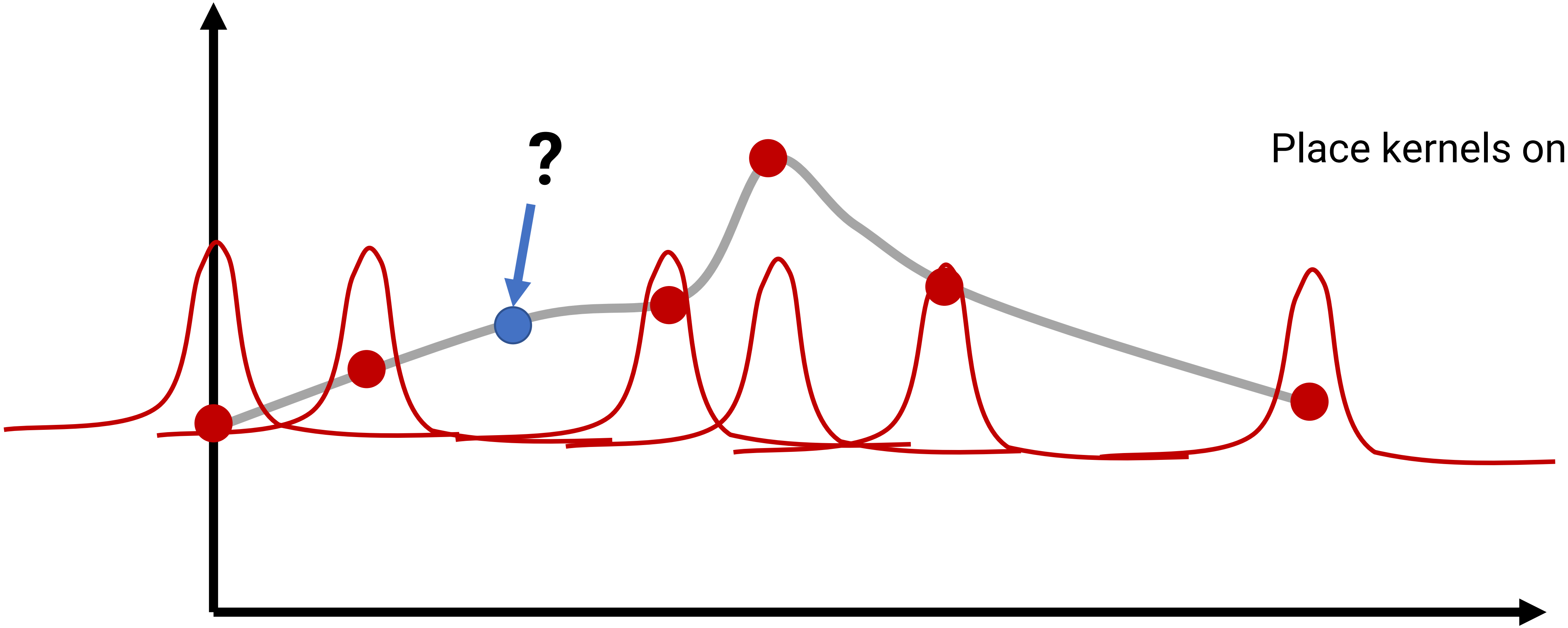
Mathematically,
$\phi: [0, \infty) \to \mathbb{R}$
Only depends on **distance** between input and fixed point!
$\phi = \hat{\phi}(||\mathbf{x} - \mathbf{c}||)$

What is a distance metric?
Say "how far things are"!

Mathematically,
$D(\mathbf{x}, \mathbf{x}) = 0$
$D(\mathbf{x}, \mathbf{y}) \geq 0$
$D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$
$D(\mathbf{x}, \mathbf{z}) \leq D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z})$ (Triangle Inequality)

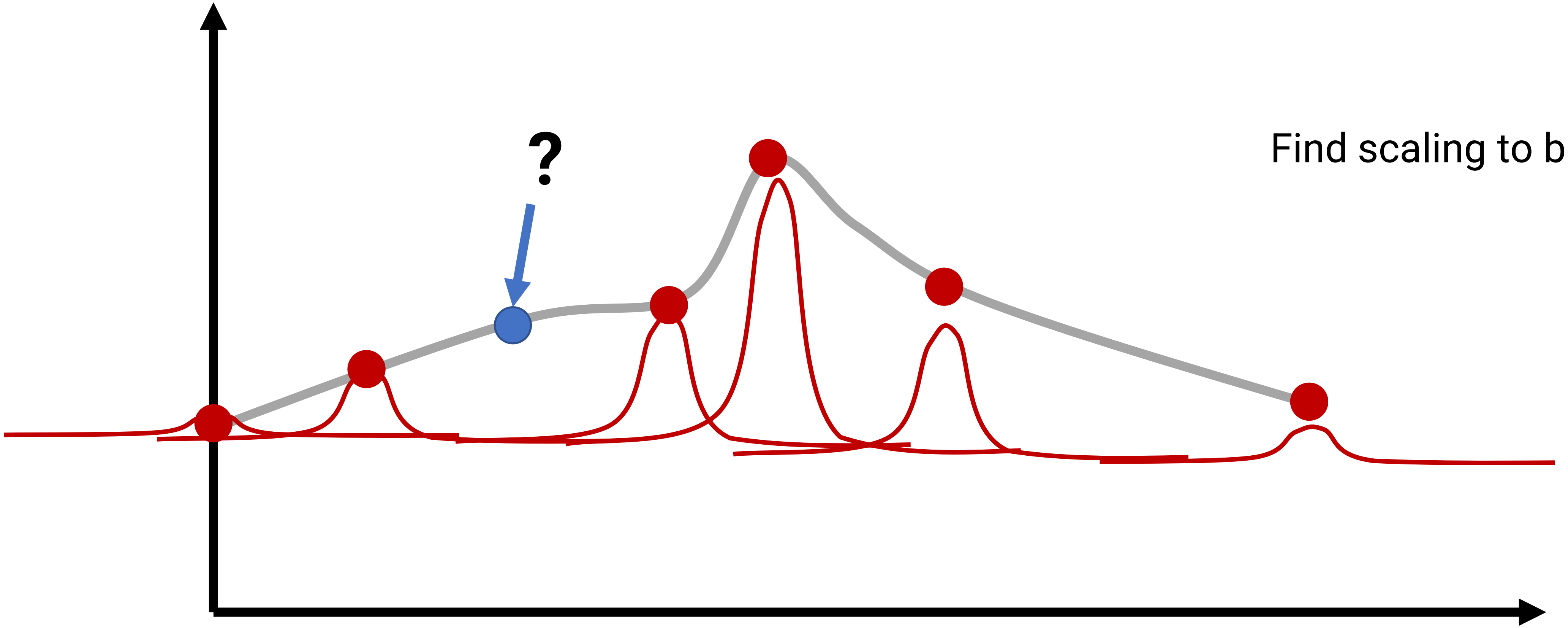
Common distance metrics?
Norms!

How To Interpolate?



Place kernels on each sample!

How To Interpolate?

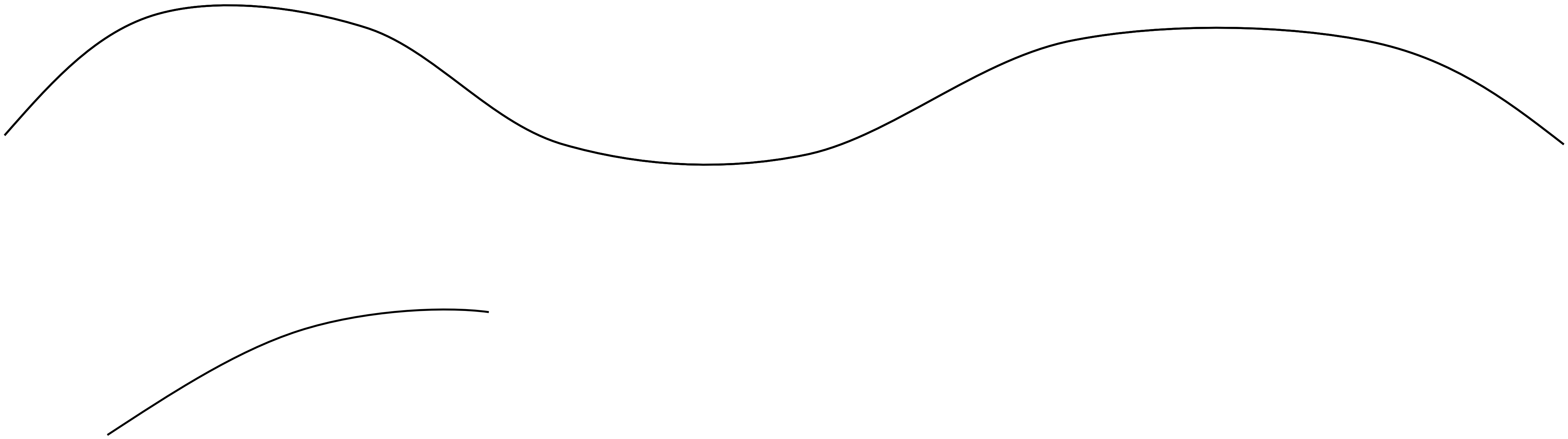


Find scaling to best fit data!

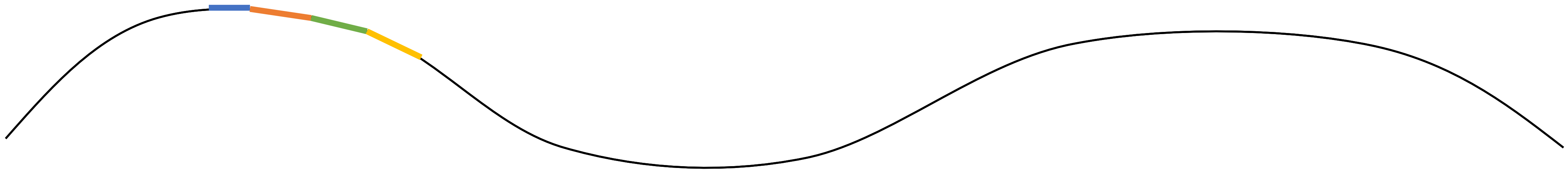
$f(z) = \sum_{\forall i} \alpha_i R(z, x_i)$ given $f(x_i) = y_i$

Take A Step Back!

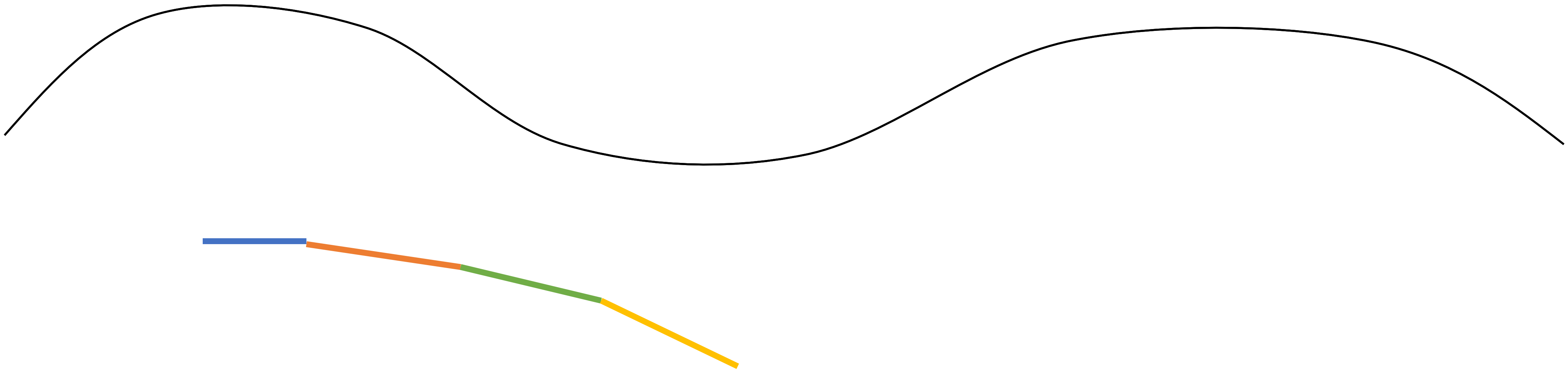
Take A Step Back!



Take A Step Back!

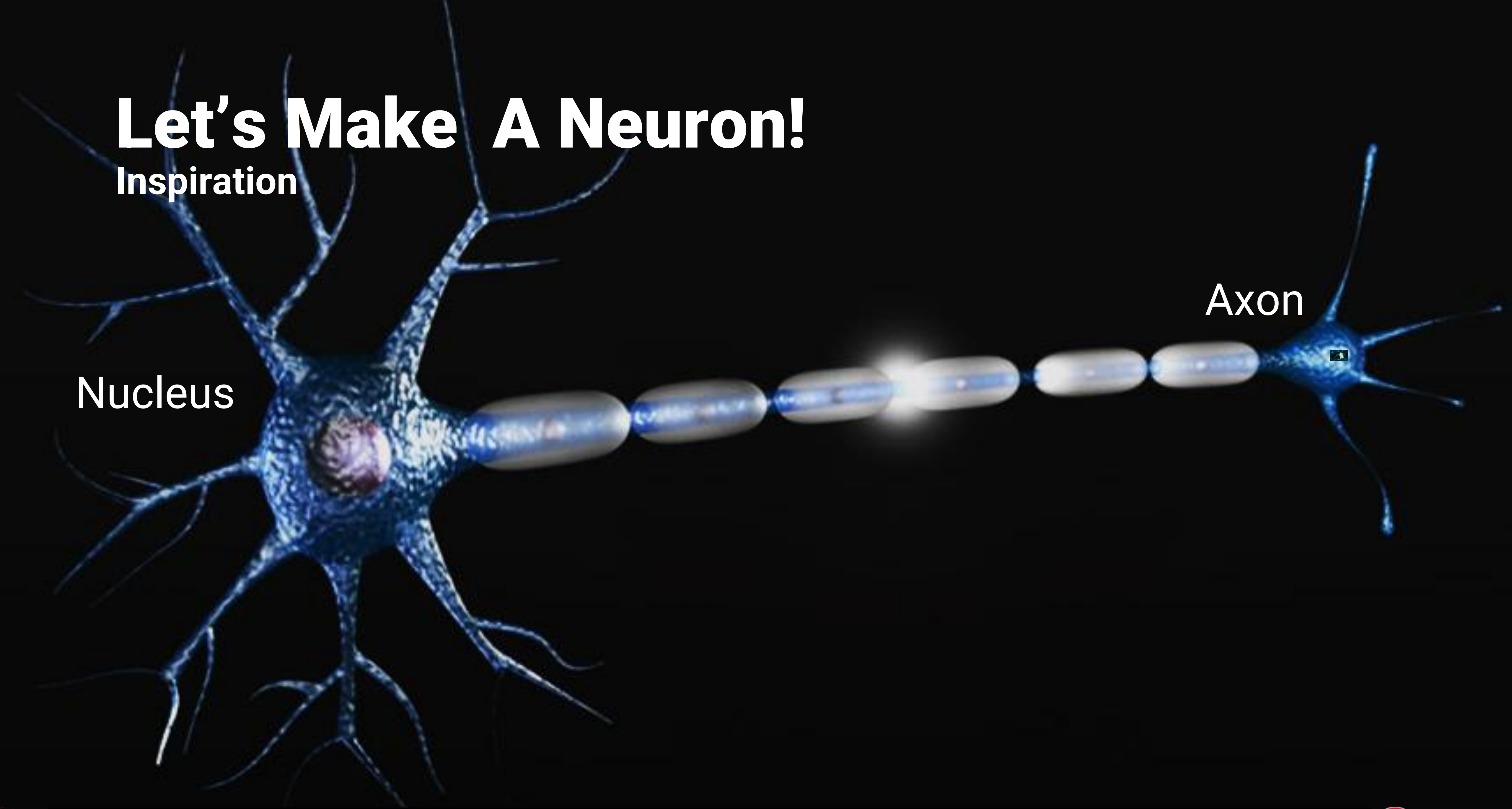


Take A Step Back!



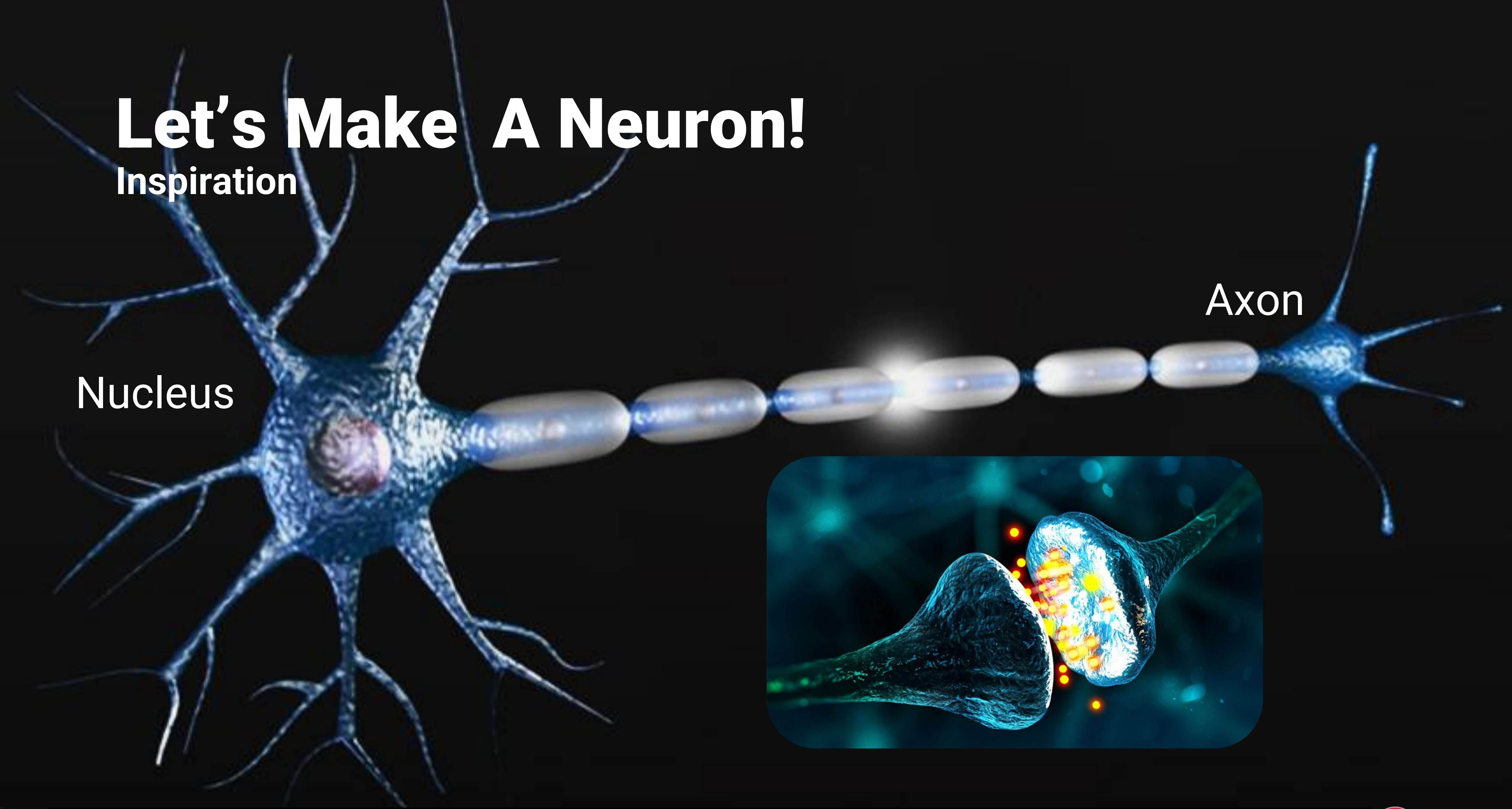
Can approximate any function with multiple simpler functions!

Let’s Make A Neuron!
Inspiration



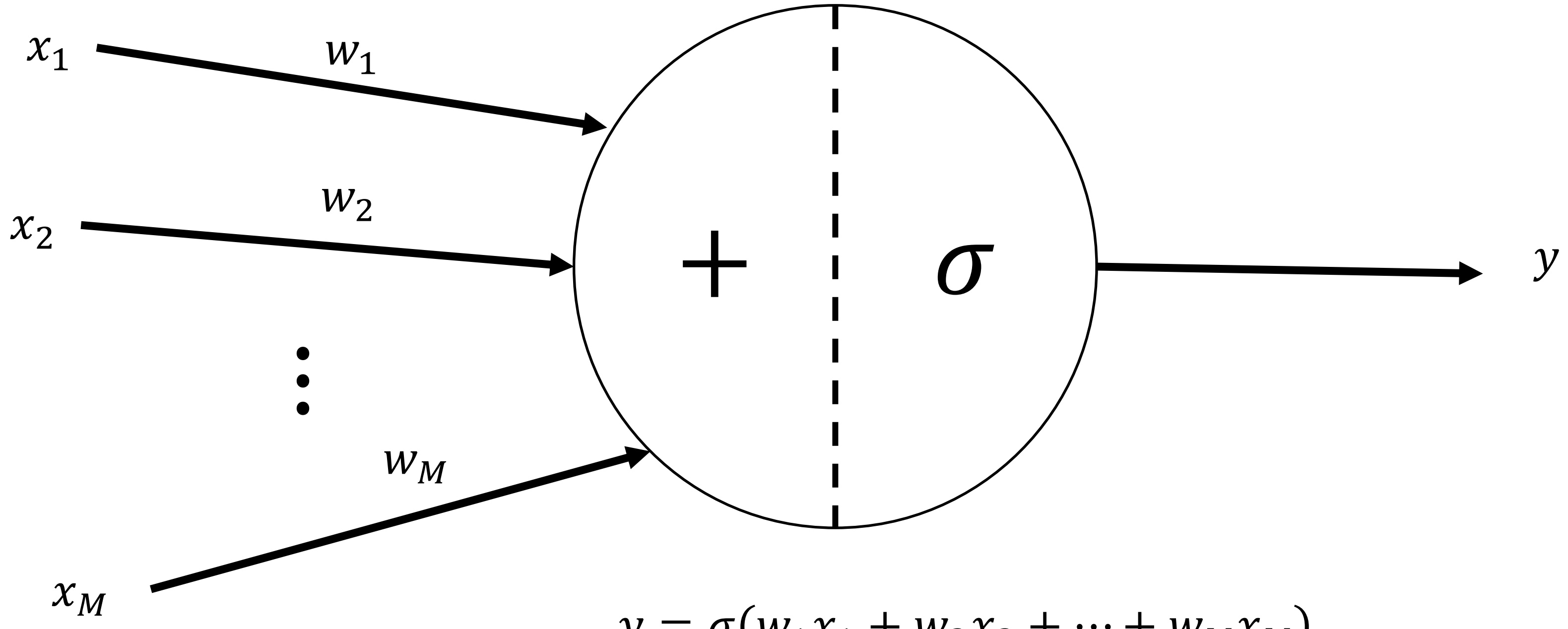
Let’s Make A Neuron!

Inspiration



Let's Make A Neuron!

Artificial Neuron AKA Perceptron



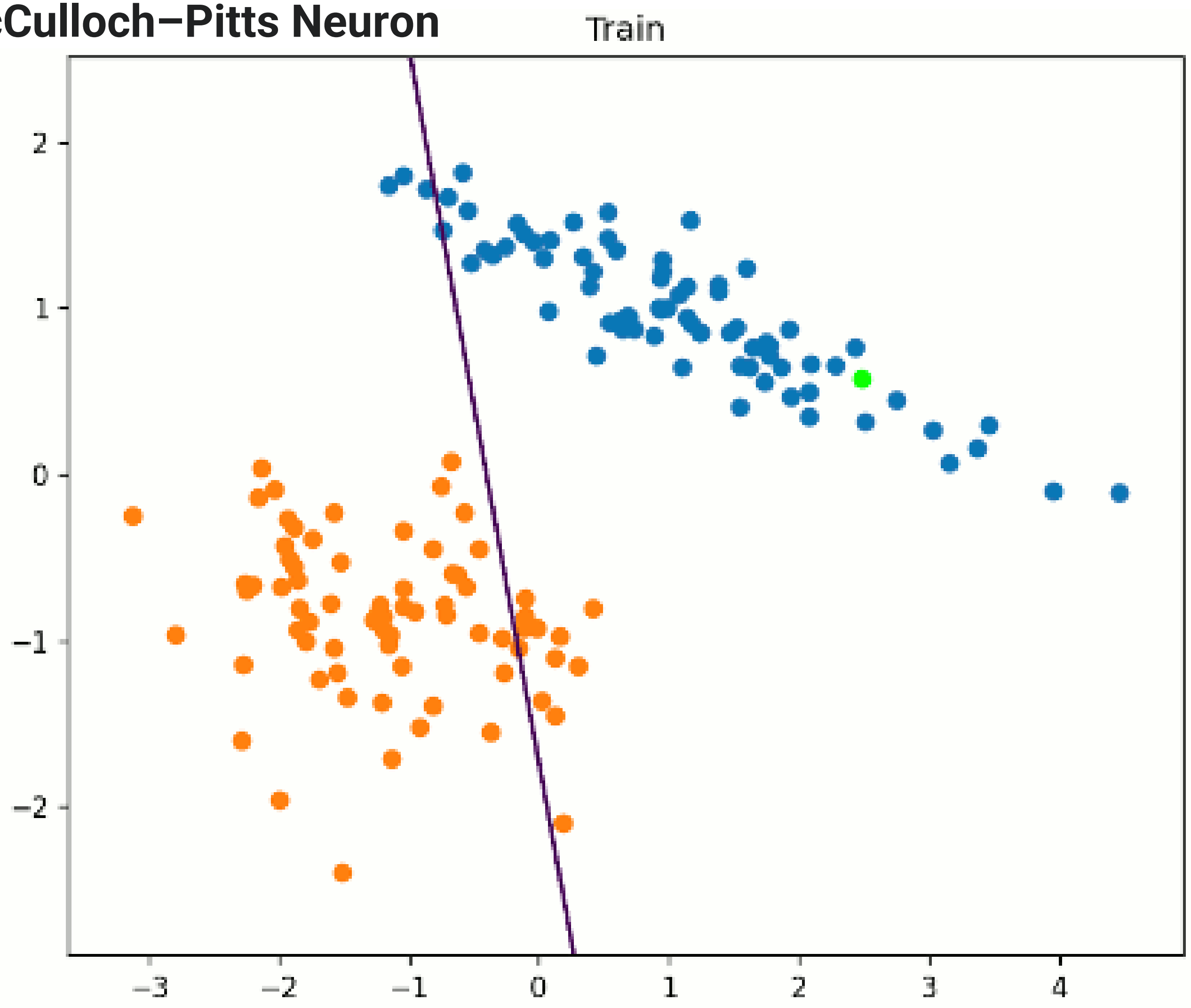
$$y = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_M x_M)$$

$$y = \sigma\left(W^T X\right)$$ **Recall Logistic Regression!**

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix} X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}$$

Perceptron Update

McCulloch–Pitts Neuron



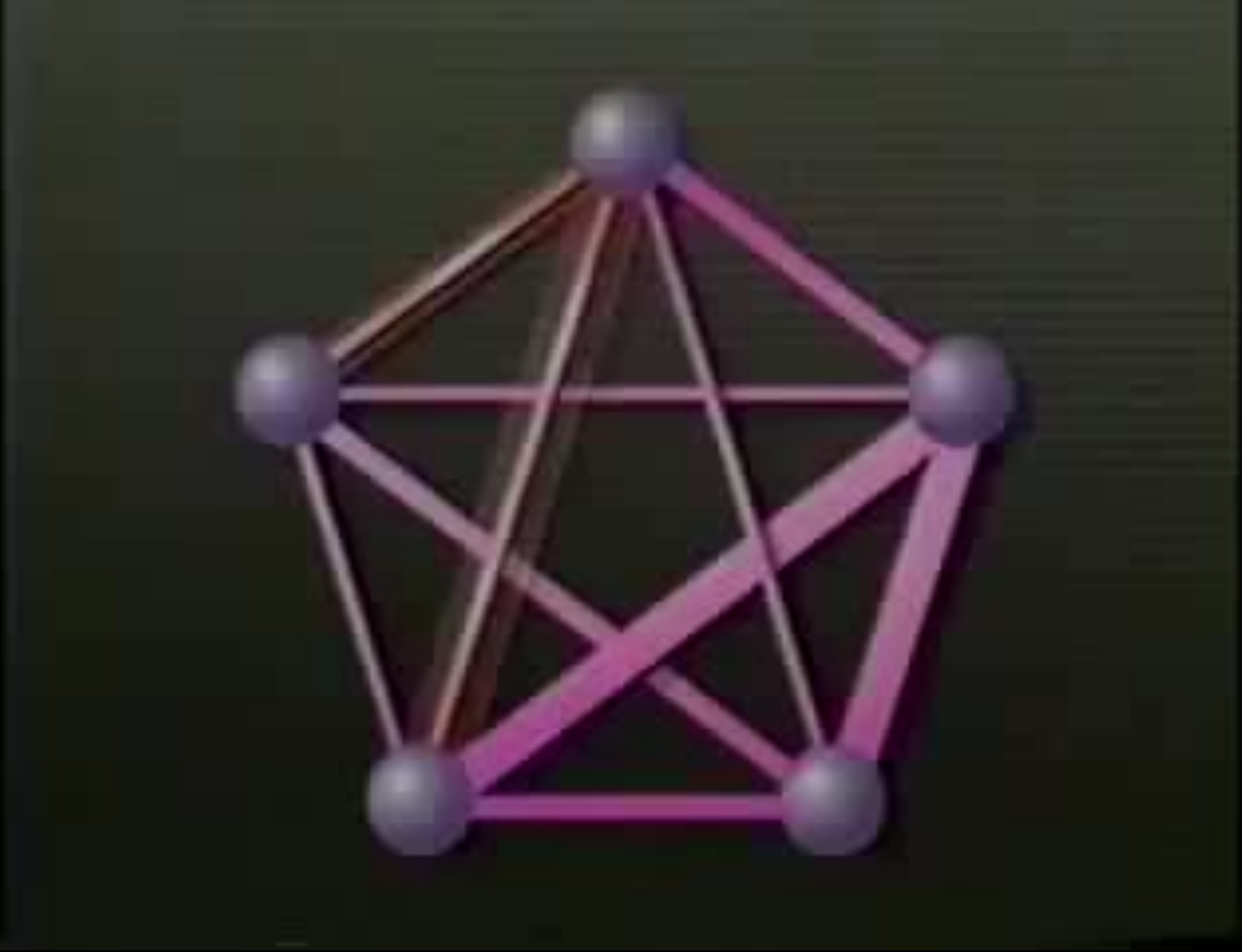
For each train sample x_i

- Compute $y_i = \sigma\left(W^T X_i\right)$
- Update weight $W(t+1) = W(t) + \alpha(\hat{y}_i - y_i)x_i$

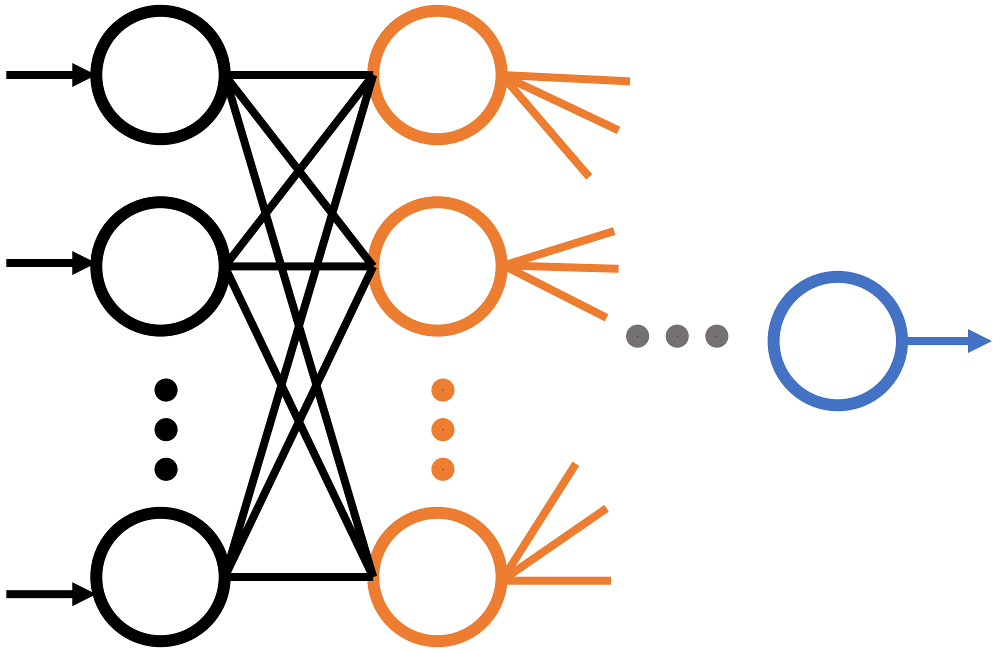
Desired Output/Label

Learning Rate

perceptron/perceptron.ipynb at main · lazuxd/perceptron (github.com)



Let’s Make A Neural Network!



If σ was linear!

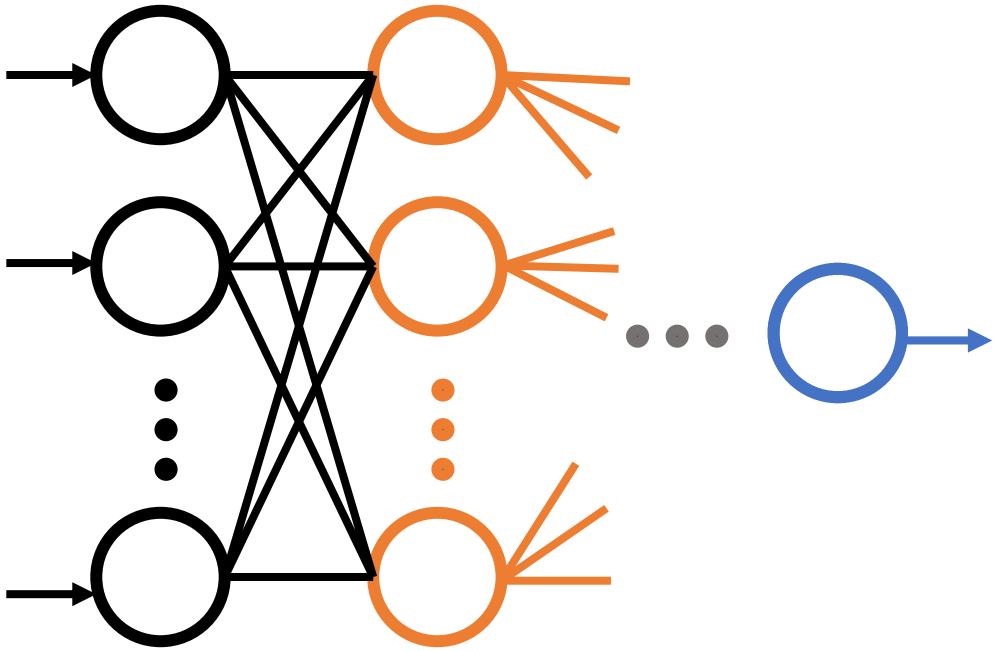
$$y_1 = \sigma(W_1 x_1) = AW_1 x_1$$

$$y_2 = \sigma(W_2 y_1) = \sigma\big(\sigma(W_1 x_1)\big) = A^2 W_2 W_1 x_1 = W_2' x_1$$

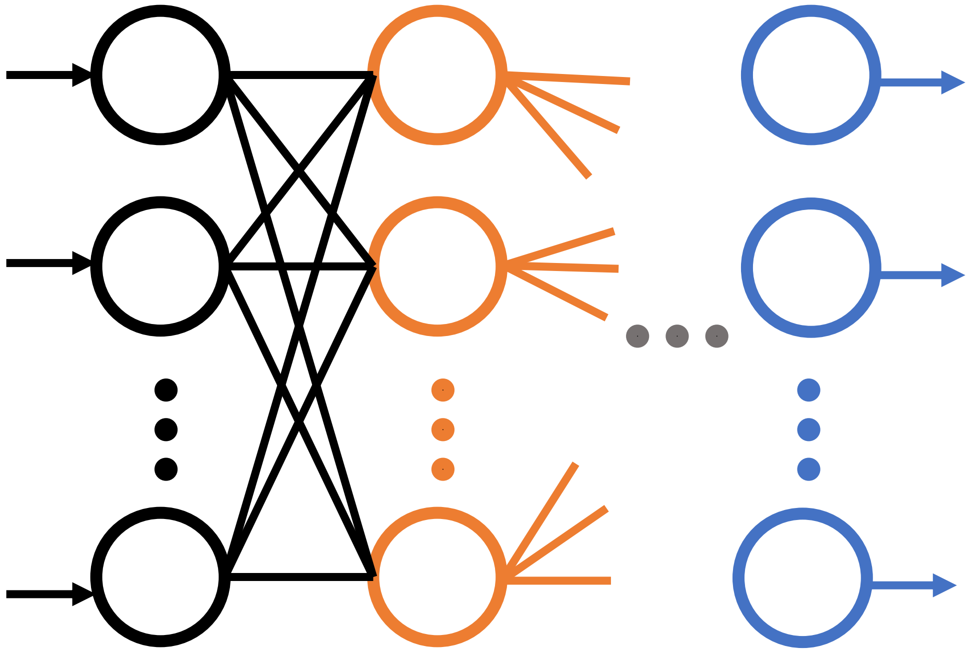
$$y_n = \prod_{\forall i} y_{n-i} \cdots AW_1 x_1 = W_n' x_1$$

This is the same as one neuron!

Let's Make A Neural Network!



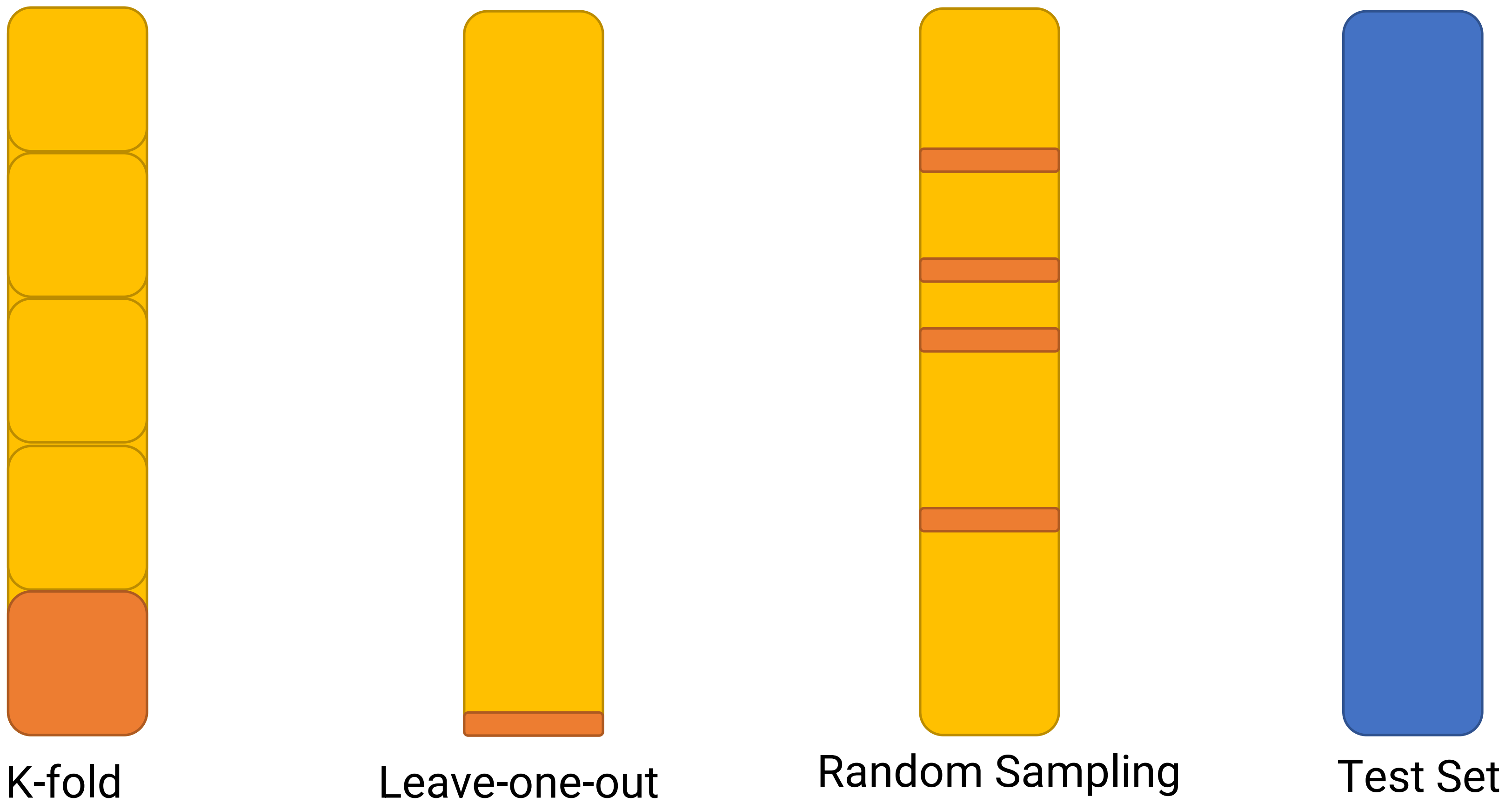
Let's Make A Neural Network!



Training-Validation-Test Sets

- **Training**: A labeled group of examples which is fed to the NN during training (weight updating/backprop etc.)
- **Validation**: A group of unseen examples used to test your NN to see how it performs
- **Test:** The set where your model is deployed on!

Cross-Validation



Let's Get Some Math Done!

- Derivative is defined by
- $D_x f = \lim_{\|h\| \to 0} \frac{f(x+h) - f(x)}{\|h\|}$
- $Df = (\partial_{x_1} f \quad \partial_{x_2} f \quad \partial_{x_3} f)$ (assuming variables are column vectors)
- **Gradient** is the **adjoint** of derivative!

$Df = (\partial_{x_1} f \quad \partial_{x_2} f \quad \partial_{x_3} f)$ and $\nabla f = \begin{bmatrix} \partial_{x_1} f \\ \partial_{x_2} f \\ \partial_{x_3} f \end{bmatrix}$

For matrices:

$\nabla f = \begin{bmatrix} \partial_{11} f & \partial_{12} f & \partial_{13} f \\ \partial_{21} f & \partial_{22} f & \partial_{23} f \\ \partial_{31} f & \partial_{32} f & \partial_{33} f \end{bmatrix}$ where $x = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$

What Is The Gradient?

Function	Gradient
$f(x) = Ax$	$\nabla f(x) = A^T$
$f(x) = \|x\|^2$	$\nabla f(x) = 2x$
$f(x) = \frac{1}{2}x^T Ax$ Symmetric A	$\nabla f(x) = Ax$

Remember:
If $h(x) = f(Ax) \Rightarrow \nabla h(x) = A^T f'(Ax)$
And if $h(x) = f(xA) \Rightarrow \nabla h(x) = f'(xA)A^T$

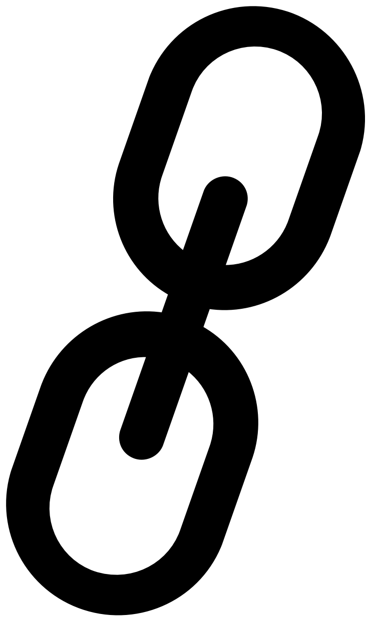
Recall Chain Rule!

$h(x) = f \circ g(x) = f(g(x))$

For single variables:
$\partial f\big(g(x)\big) = f'(g)g'(x)$

For multiple-variables:
$Df\big(g(x)\big) = Df \circ D(g(x))$

For gradients:
$\nabla f\big(g(x)\big) = \nabla g(x)\nabla f(x)$



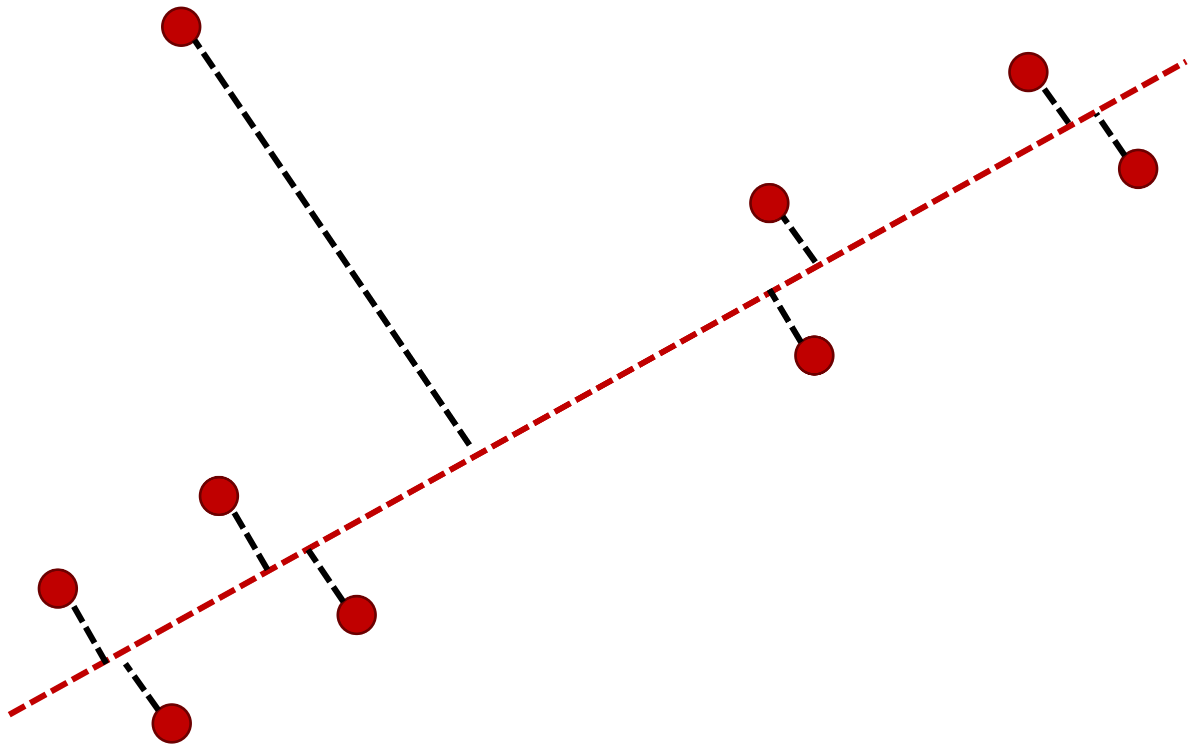
Recall Ridge Regression

Least Squares With Penalty

$$f(x) = \frac{1}{2}\|Ax - b\|^2 + \frac{\lambda}{2}\|x\|^2$$

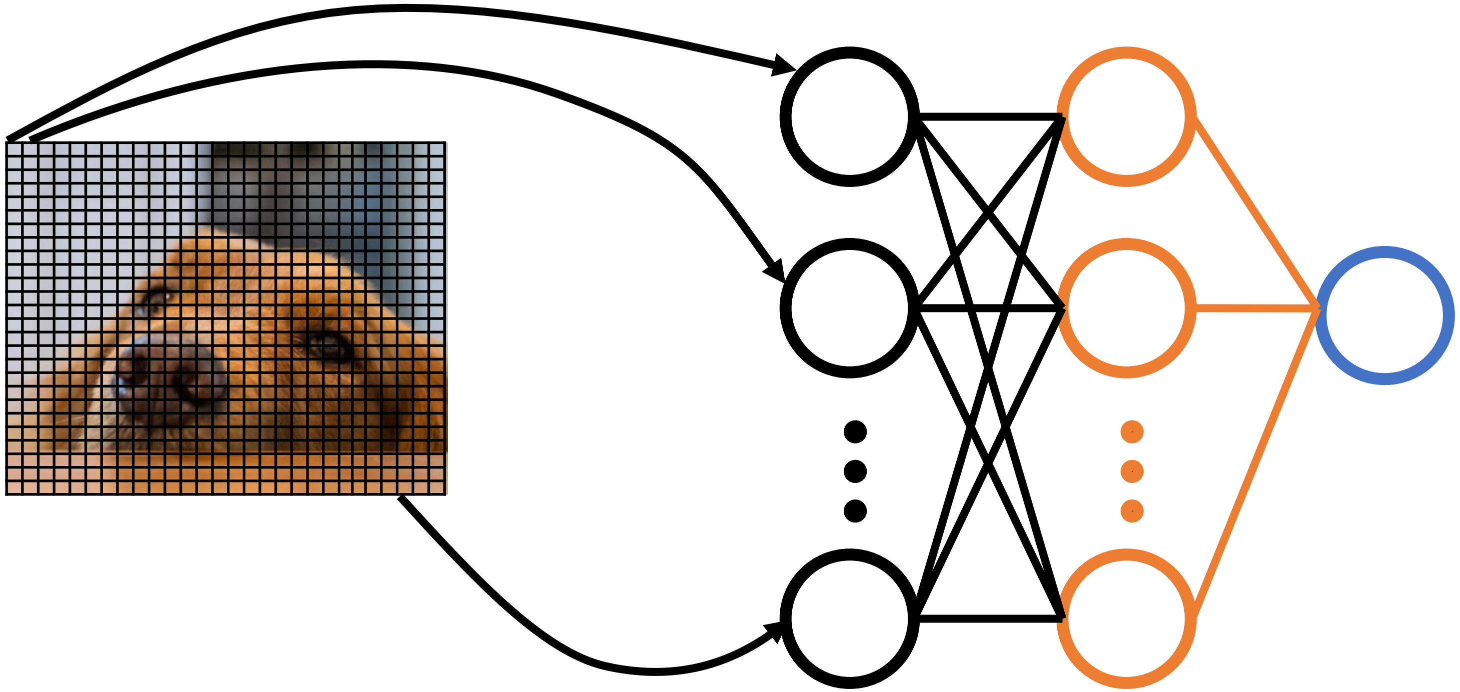
How do you obtain optimal solution?
Set gradient to zero!

$$\nabla f(x) = \lambda x + A^T(Ax - b) = 0$$
$$\left(A^T A + \lambda I\right)x = A^T b$$
$$x^\star = \left(A^T A + \lambda I\right)^{-1} A^T b$$



Key To Training: Error Back Propagation

AKA Backprop



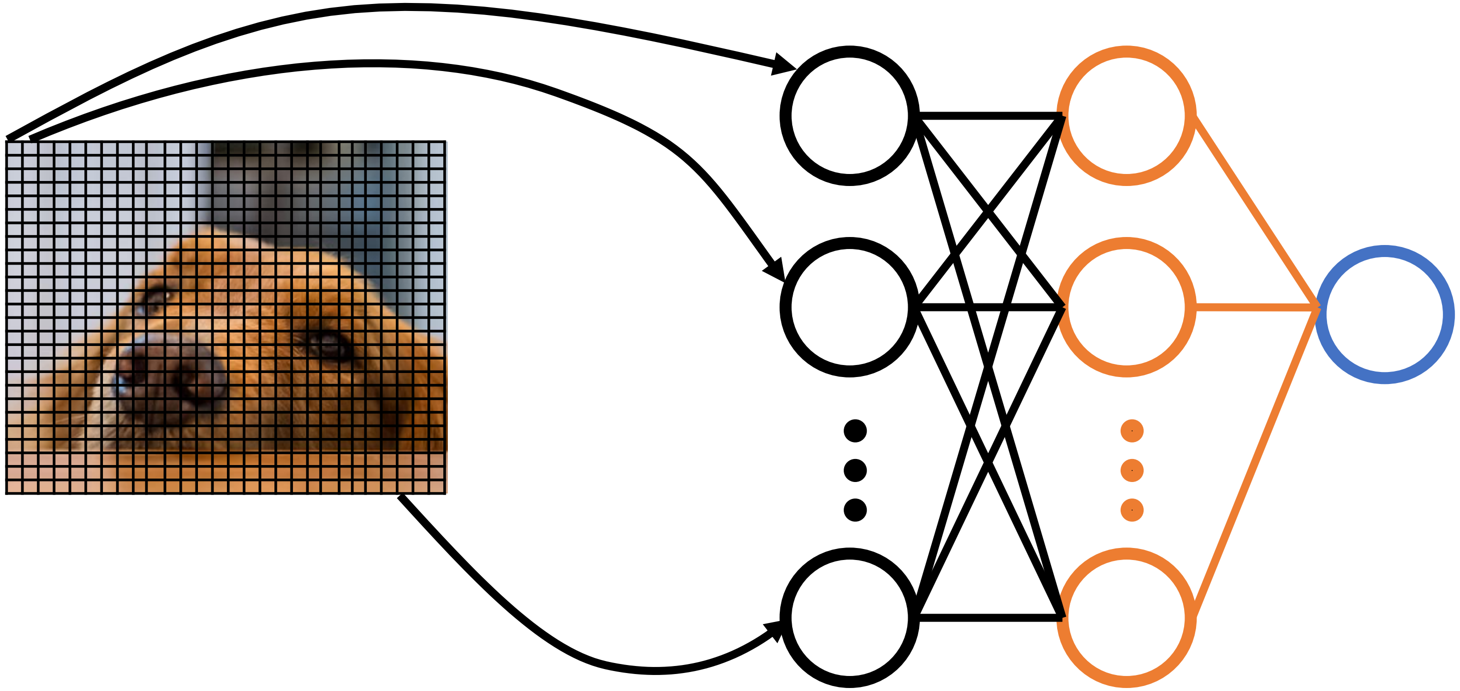
x_i
Input Layer

$y_1 = \sigma(x_i W_1)$
Hidden Layer 1

$y_2 = \sigma(y_1 W_2)$
Hidden Layer 2

$y_3 = \sigma(y_2 W_3)$
Output Layer

Forward Pass



Training Data
(Row Vector)

$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

x_i: Training Data (Row Vector)

z_1: "Pre-activations"

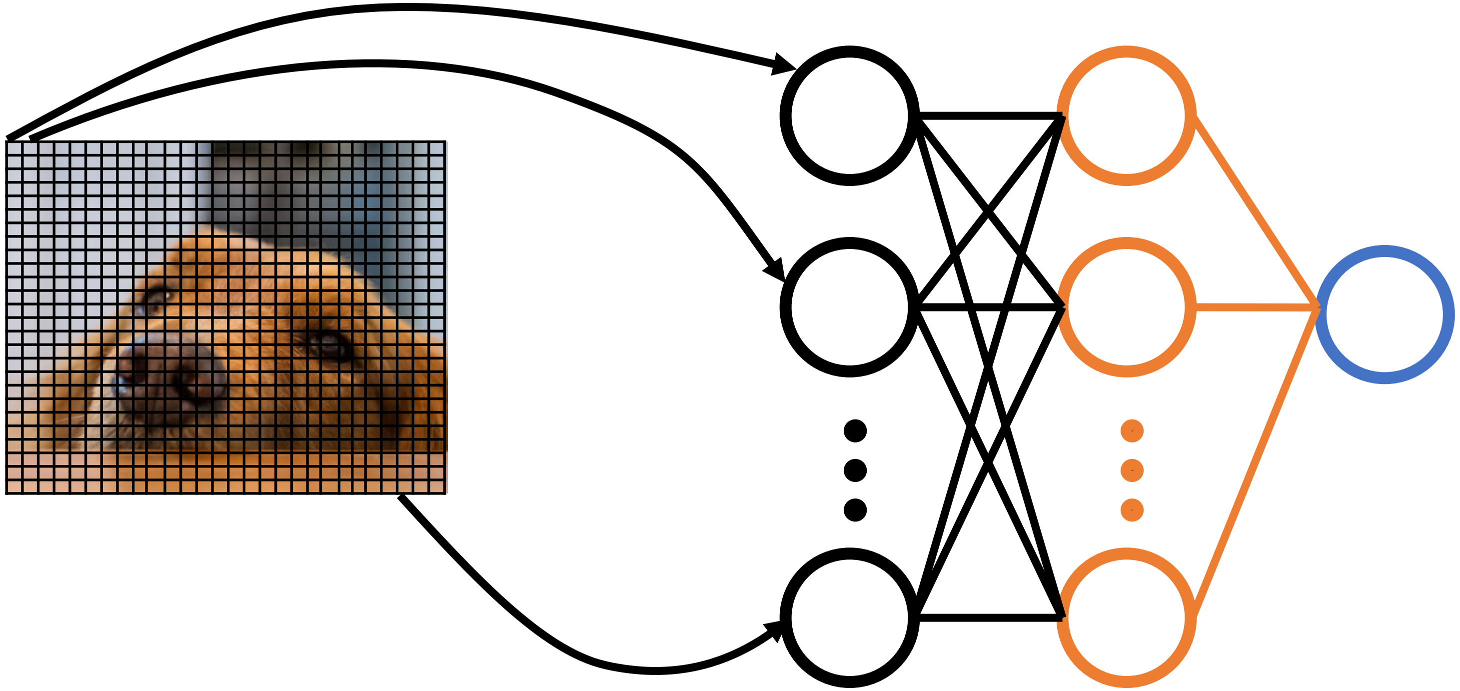
y_1: Activations

Save y's for later

Slide inspired from UMD's CMSC764

Backward Pass

We need derivative of $\mathcal{L}$ with respect to x and W

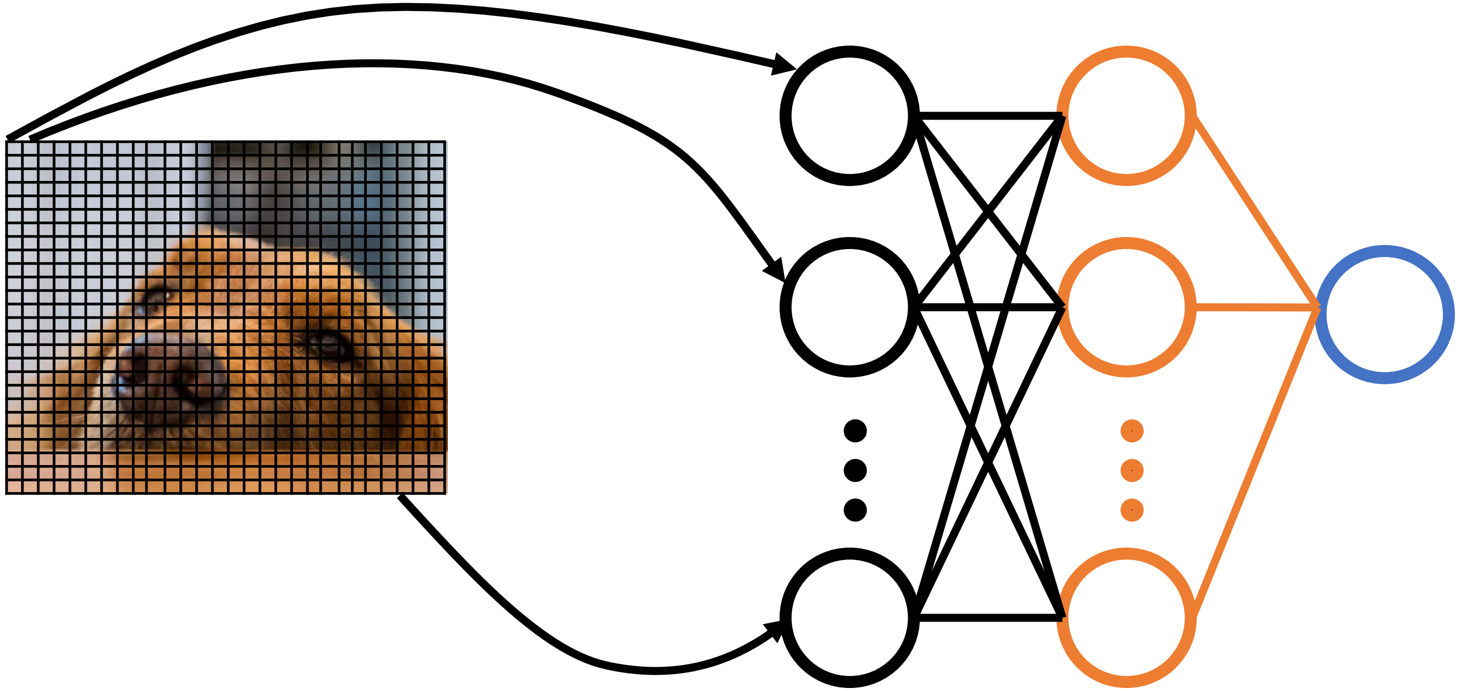


$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative here is $\dfrac{\partial \mathcal{L}}{\partial z_3}$

Backward Pass

We need derivative of $\mathcal{L}$ with respect to x



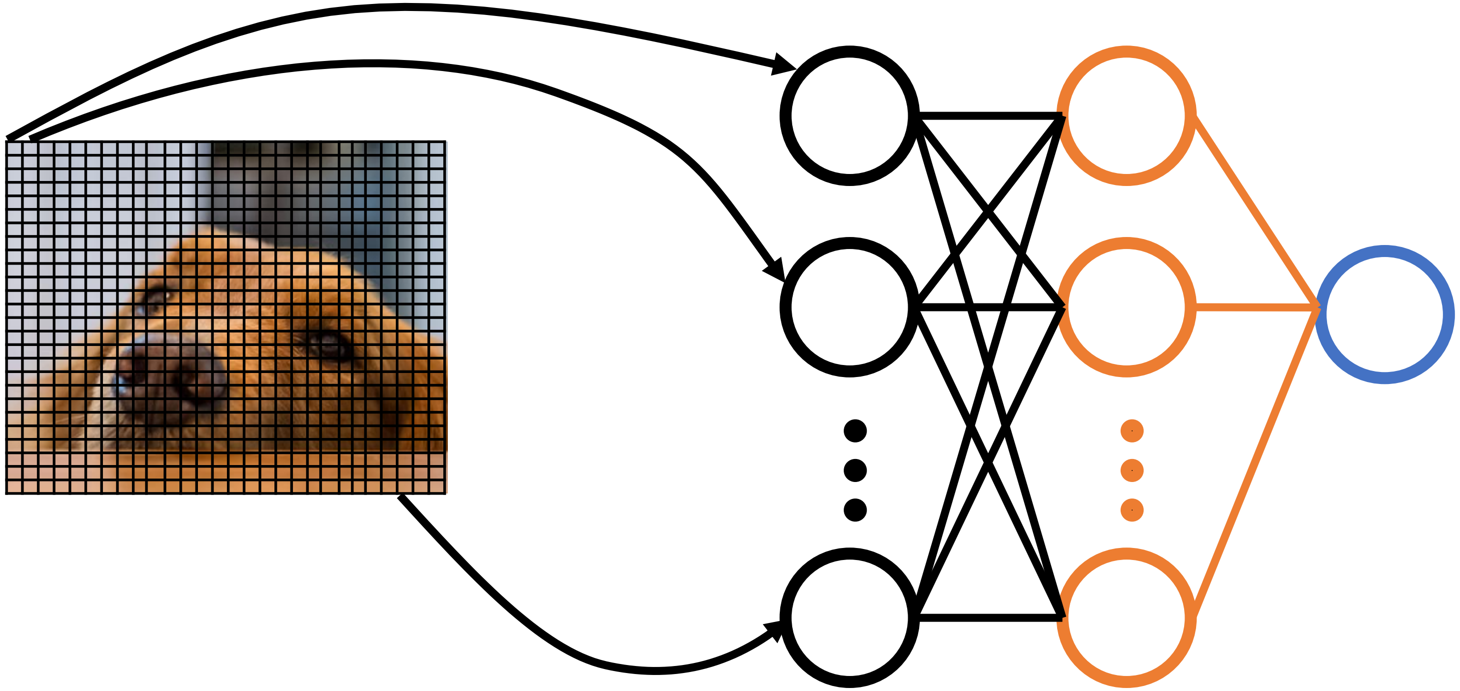
$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative here is $W_3 \dfrac{\partial \mathcal{L}}{\partial z_3}$

Scaling property!

Backward Pass

We need derivative of $\mathcal{L}$ with respect to x



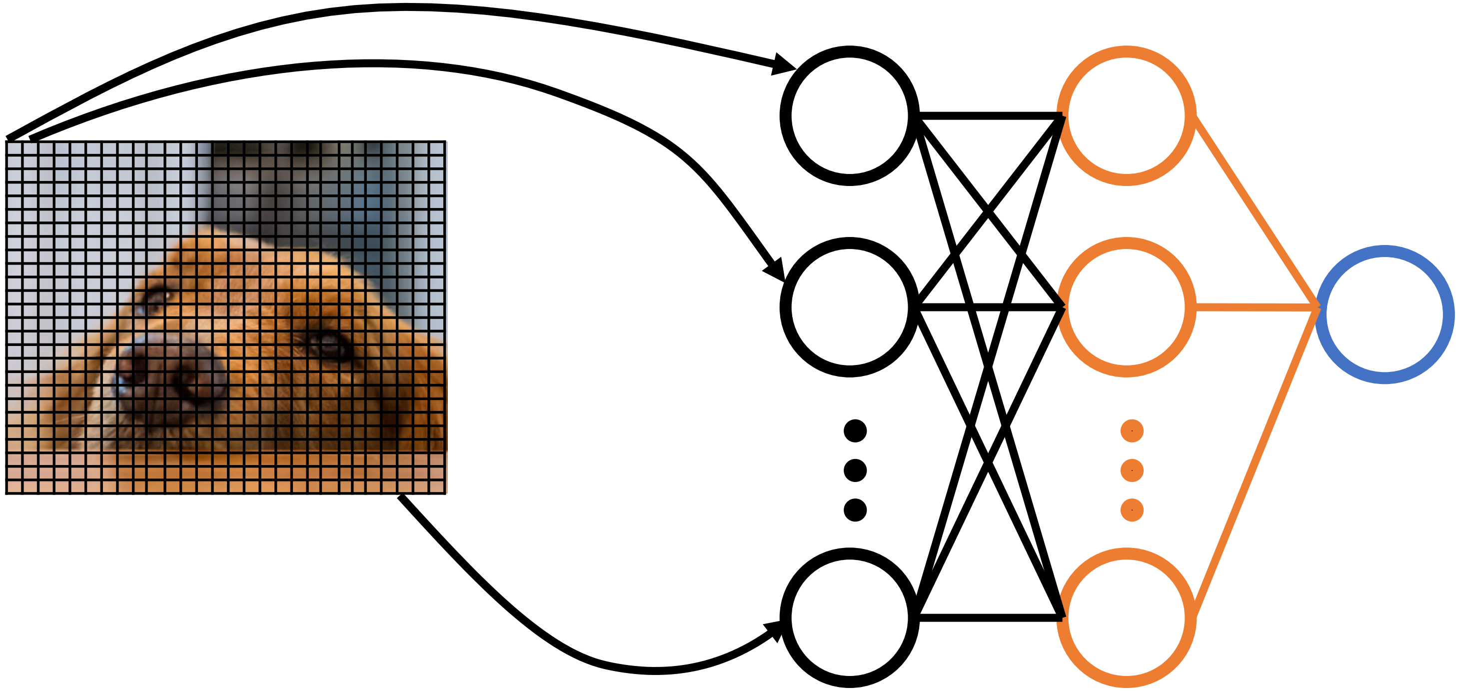
$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative here is $\sigma_2{}' \mathrm{W}_3 \dfrac{\partial \mathcal{L}}{\partial z_3}$

Chain Rule!

Backward Pass

We need derivative of $\mathcal{L}$ with respect to x

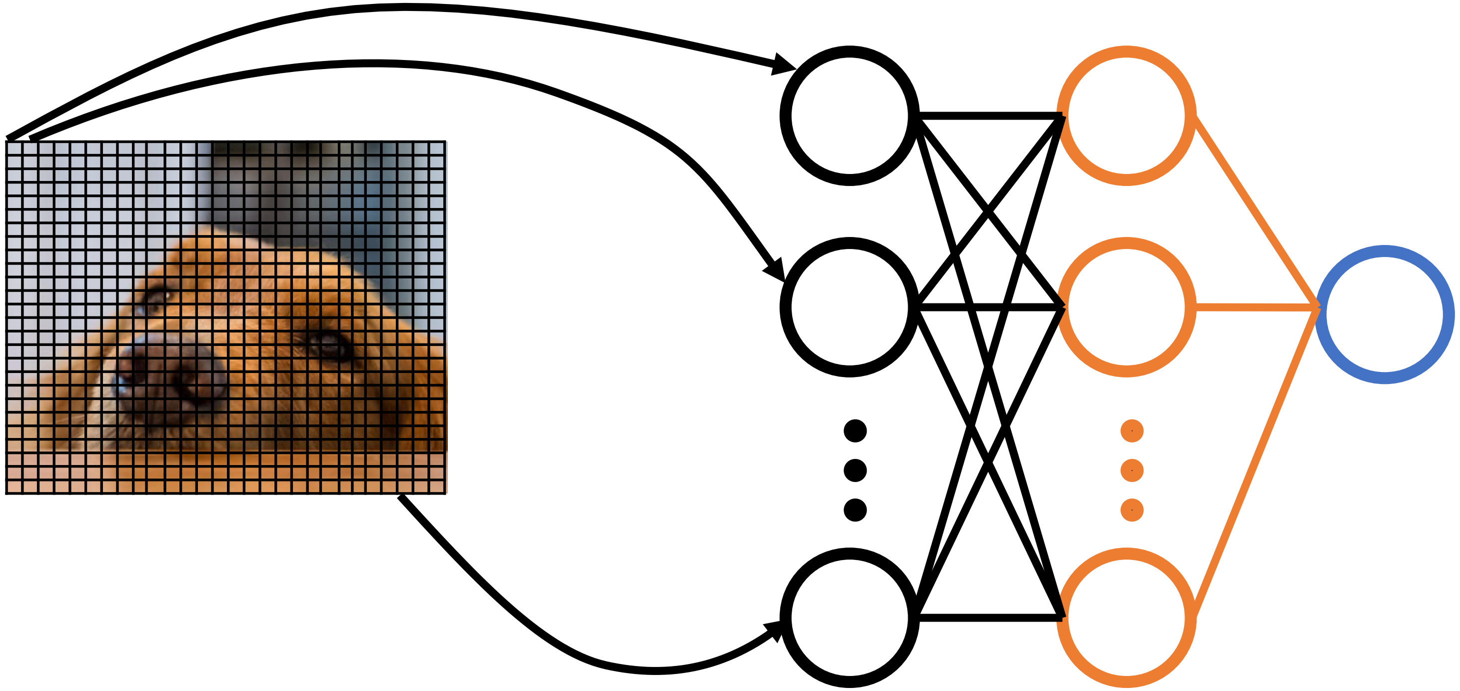


$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative here is $W_2 \sigma_2{}' \mathrm{W}_3 \dfrac{\partial \mathcal{L}}{\partial z_3}$

Backward Pass

We need derivative of $\mathcal{L}$ with respect to x

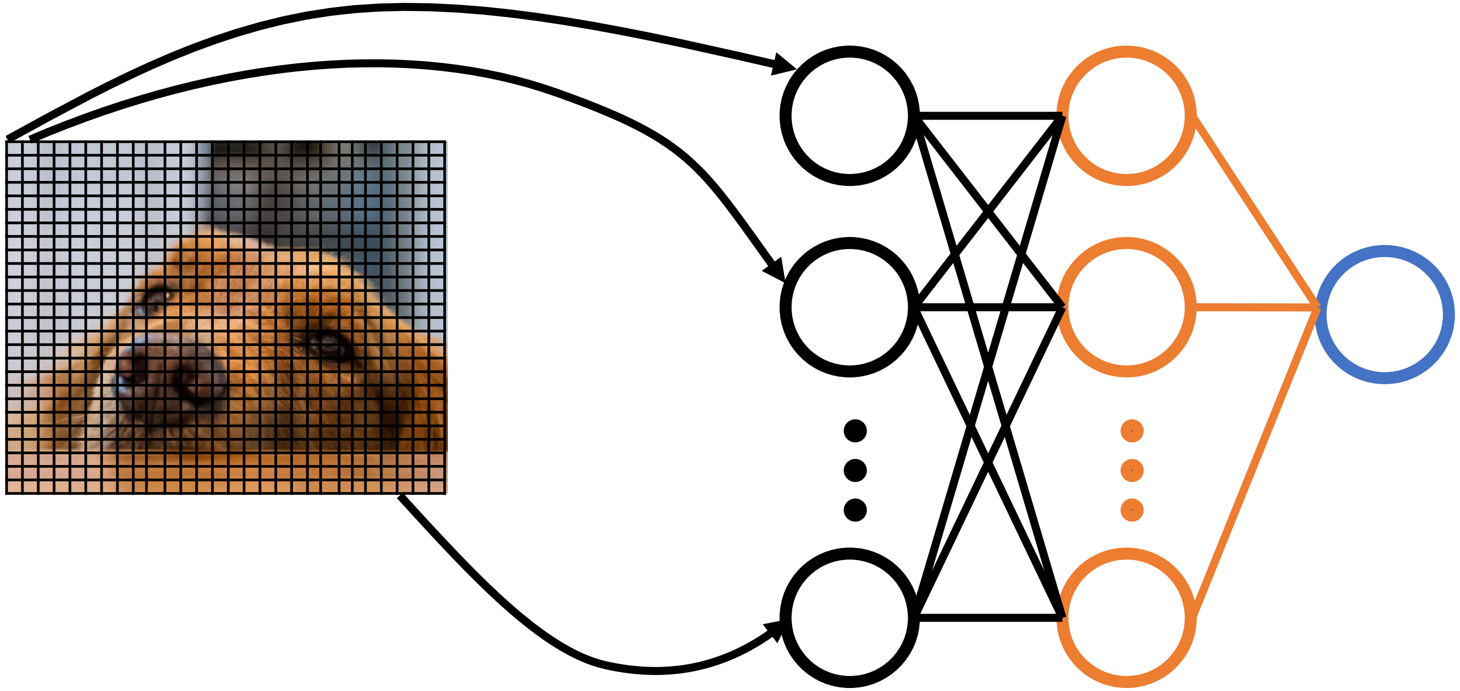


$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative here is $\sigma_1{}' W_2 \sigma_2{}' \mathrm{W}_3 \dfrac{\partial \mathcal{L}}{\partial z_3}$

Backward Pass

We need derivative of $\mathcal{L}$ with respect to x



$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative here is $\mathrm{W}_1 \sigma_1{}' W_2 \sigma_2{}' \mathrm{W}_3 \frac{\partial \mathcal{L}}{\partial z_3}$

Derivative With Respect To W_2

Derivative of loss $\mathcal{L}$ with respect to W_2

$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{\boxed{W_2}} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Use Chain Rule

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial z_2} \frac{\partial z_2}{\partial W_2}$$

$$\nabla_{W_2} \mathcal{L} = \nabla_{W_2} z_2 \nabla_{z_2} \mathcal{L}$$

Derivative With Respect To W_2

Derivative of loss $\mathcal{L}$ with respect to W_2

$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{\boxed{W_2}} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

Derivative with respect to z_2

$$\sigma_2' W_3 \frac{\partial \mathcal{L}}{\partial z_3}$$

Derivative With Respect To W_2

Derivative of loss $\mathcal{L}$ with respect to W_2

$$\begin{tikzcd} x_i \arrow[r, "W_1"] & z_1 \arrow[r, "\sigma_1"] & y_1 \arrow[r, "\boxed{W_2}"] & z_2 \arrow[r, "\sigma_2"] & y_2 \arrow[r, "W_3"] & z_3 \arrow[r, "\text{Loss}"] & \mathcal{L} \end{tikzcd}$$

Derivative with respect to z_2

$$\sigma_2' W_3 \frac{\partial \mathcal{L}}{\partial z_3}$$

$$\frac{\partial z_2}{\partial W_2}$$

$$\sigma_2' W_3 \frac{\partial \mathcal{L}}{\partial z_3} y_1$$

Complete Backprop

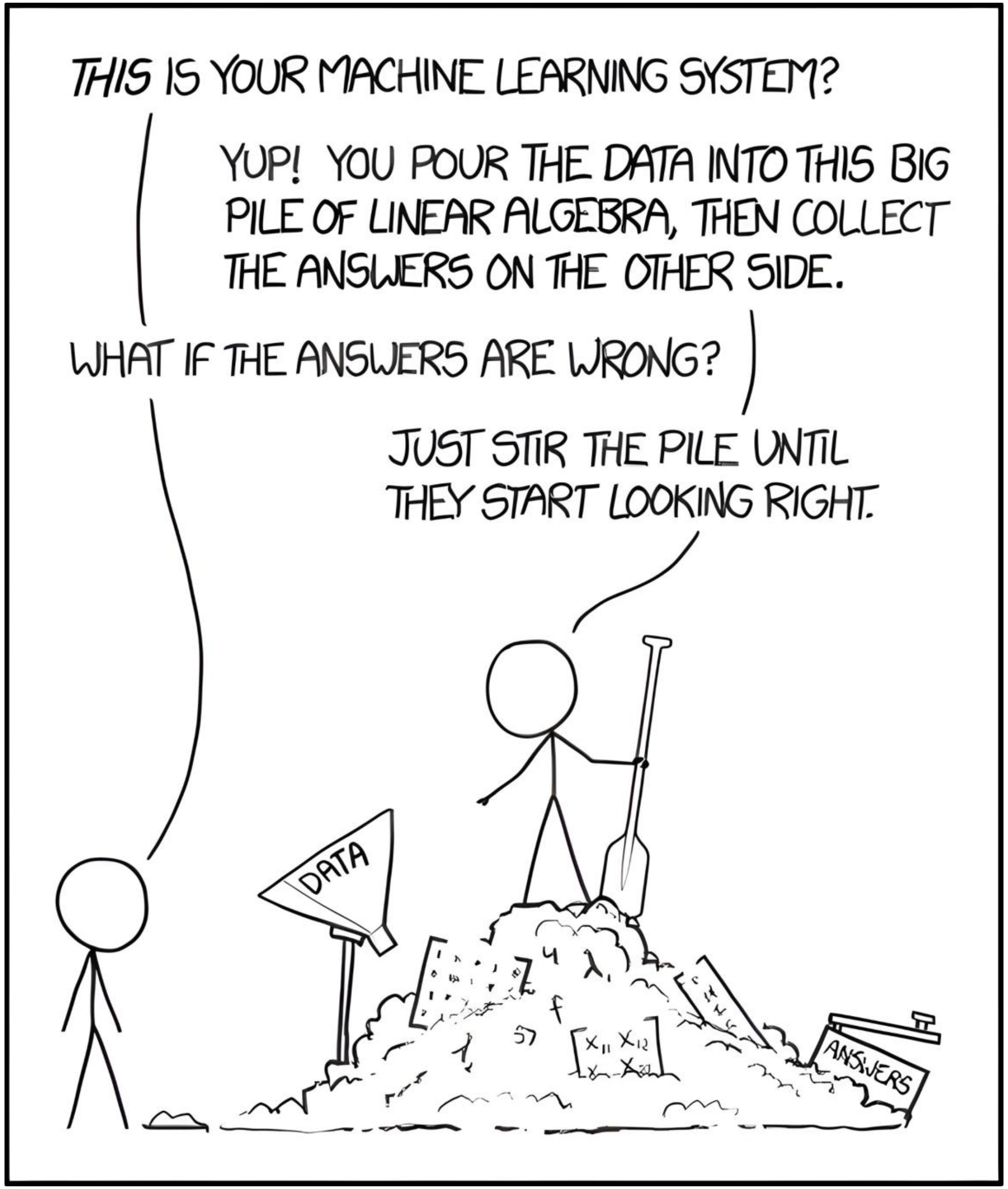
$$x_i \xrightarrow{W_1} z_1 \xrightarrow{\sigma_1} y_1 \xrightarrow{W_2} z_2 \xrightarrow{\sigma_2} y_2 \xrightarrow{W_3} z_3 \xrightarrow{\text{Loss}} \mathcal{L}$$

$\nabla_{z_3}\mathcal{L}$ $\qquad$ $\nabla_{W_3}\mathcal{L} = y_2^T \nabla_{z_3}\mathcal{L}$

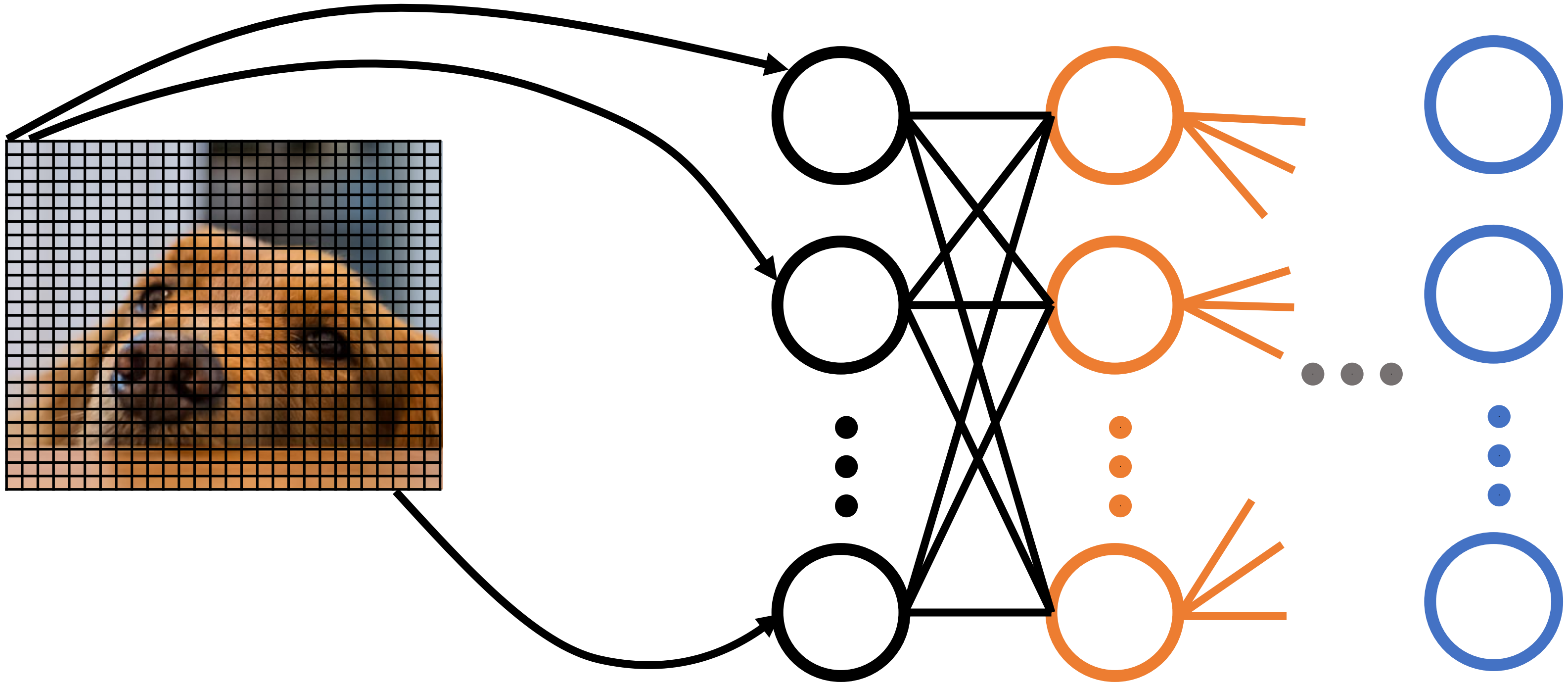
$\nabla_{z_2}\mathcal{L} = \nabla_{z_3}\mathcal{L} W_3^T \sigma_2'$ $\qquad$ $\nabla_{W_2}\mathcal{L} = y_1^T \nabla_{z_2}\mathcal{L}$

$\nabla_{z_1}\mathcal{L} = \nabla_{z_3}\mathcal{L} W_3^T \sigma_2' W_2^T \sigma_1'$ $\qquad$ $\nabla_{W_1}\mathcal{L} = x_i^T \nabla_{z_1}\mathcal{L}$

Essentially!

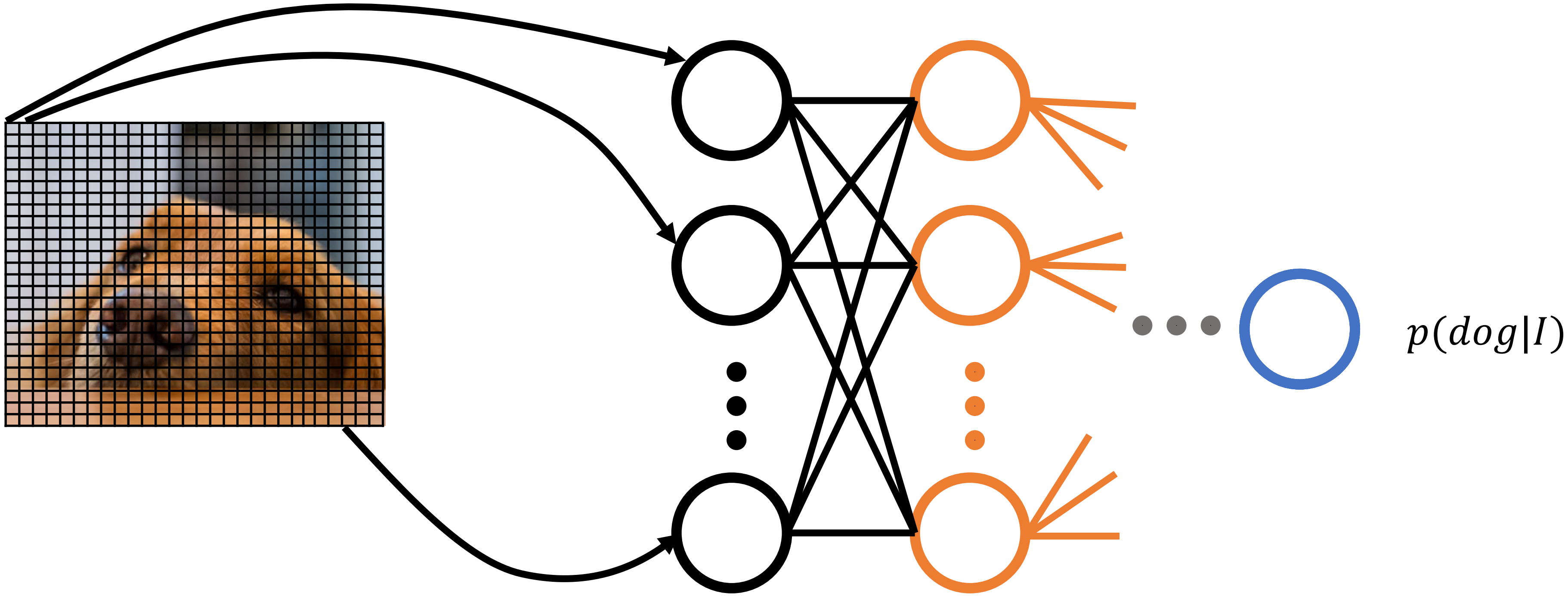


We Did Something Cool!



Dimension of Output?
1
Why?
$p(dog|I)$ **as**
$p(cat|I) = 1 - p(dog|I)$

We Did Something Cool!



Fully Connected Network or Multi Layer Perceptron (MLP)

Gradient Vs Derivative

$$\frac{\partial \mathcal{L}}{\partial x_1} = \mathrm{W}_1 \sigma_1{}' W_2 \sigma_2{}' \mathrm{W}_3 \frac{\partial \mathcal{L}}{\partial z_3}$$ Derivative

$$\nabla \mathcal{L}_{x_1} = \nabla_{z_3} \mathcal{L} W_3^T \sigma_2' W_2^T \sigma_1' W_1^T$$ Gradient

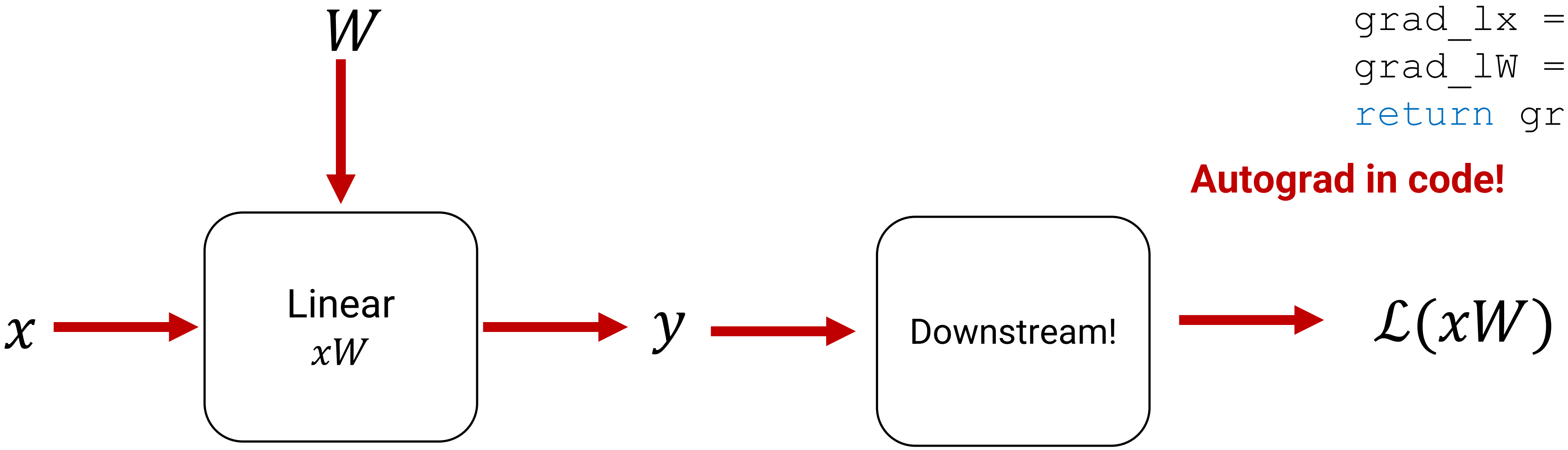
Backward Pass!

Let’s Do A Trick!

```
def dorward(x,W):
        return x@W

def backward(x, W, grad_ly):
        grad_lx = grad_ly@W.T
        grad_lW = x.T@grad_ly
        return grad_lx, grad_lW
```

Autograd in code!



Gradient with respect to input?

$\nabla_x \mathcal{L}(xW) = \mathcal{L}'(xW)W^T$

$\nabla_W \mathcal{L}(xW) = x^T \mathcal{L}'(xW)$

Since if $h(x) = f(Ax) \Rightarrow \nabla h(x) = A^T f'(Ax)$

And if $h(x) = f(xA) \Rightarrow \nabla h(x) = f'(xA)A^T$

Next Class!

NN Tuning, Image Filtering And Convolutional Neural Networks