Intro to Neural Networks

Advanced ML

Outline for today

- Data Competition
- Recap of Learning Nonlinearities
- Proto-neural nets: Regression, logistic regression, and multi-class logistic regression
- Neural Network Intuition

Data Competition

Learning Nonlinearities

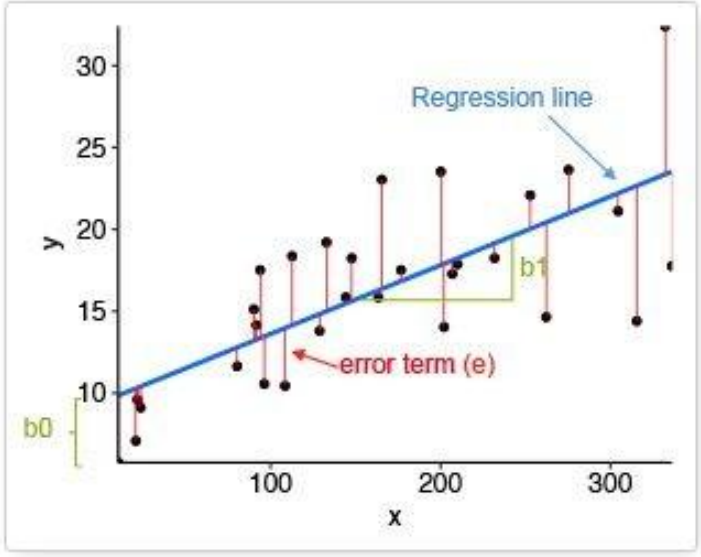
Simplest Neural Nets

- Linear Regression
- Logistic Regression
- Multi-class logistic regression

Review of linear regression

- `x` square footage of a house
- `y` house value

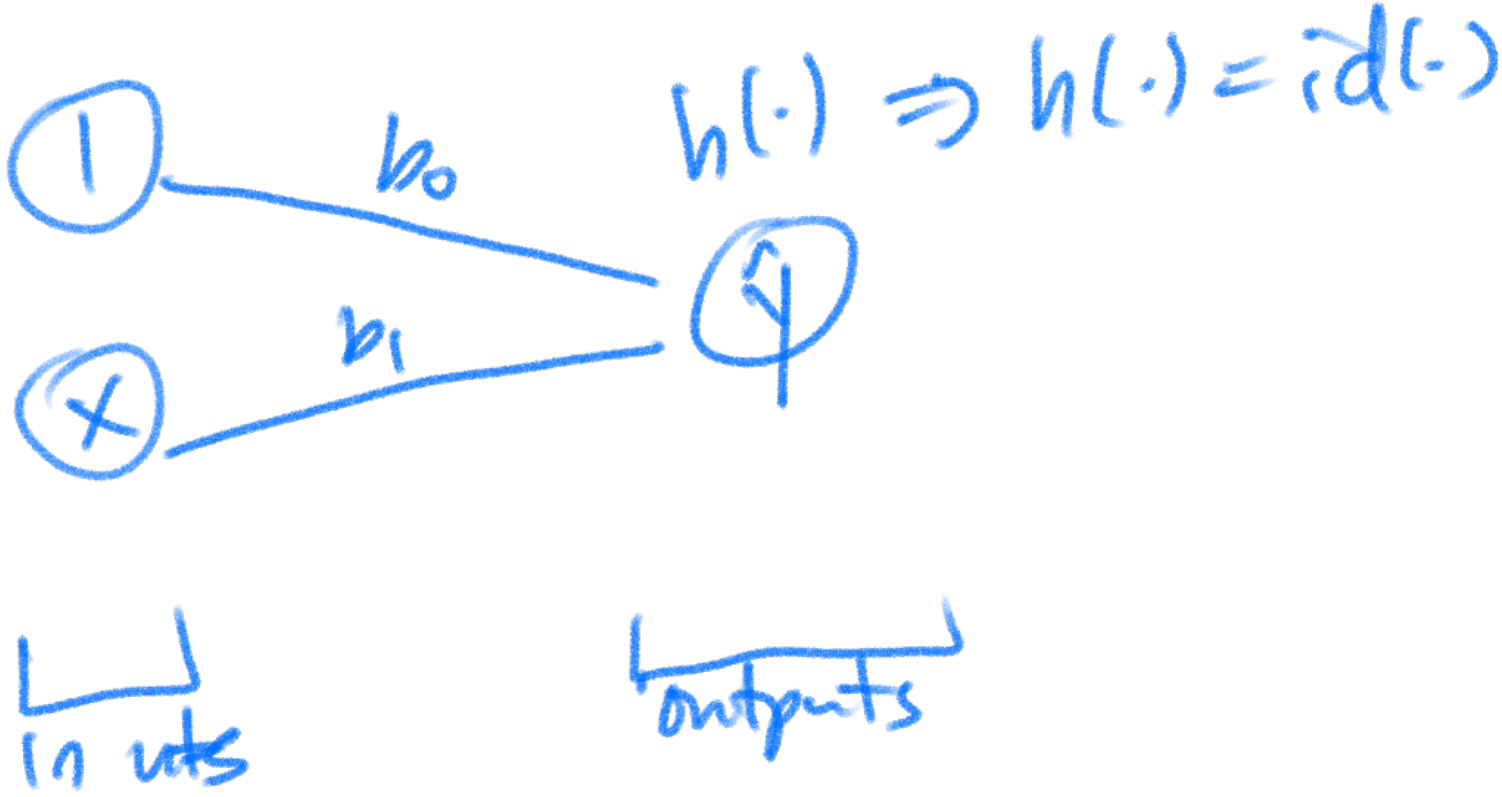
$$\hat{y} = b_0 + b_1 \cdot x$$



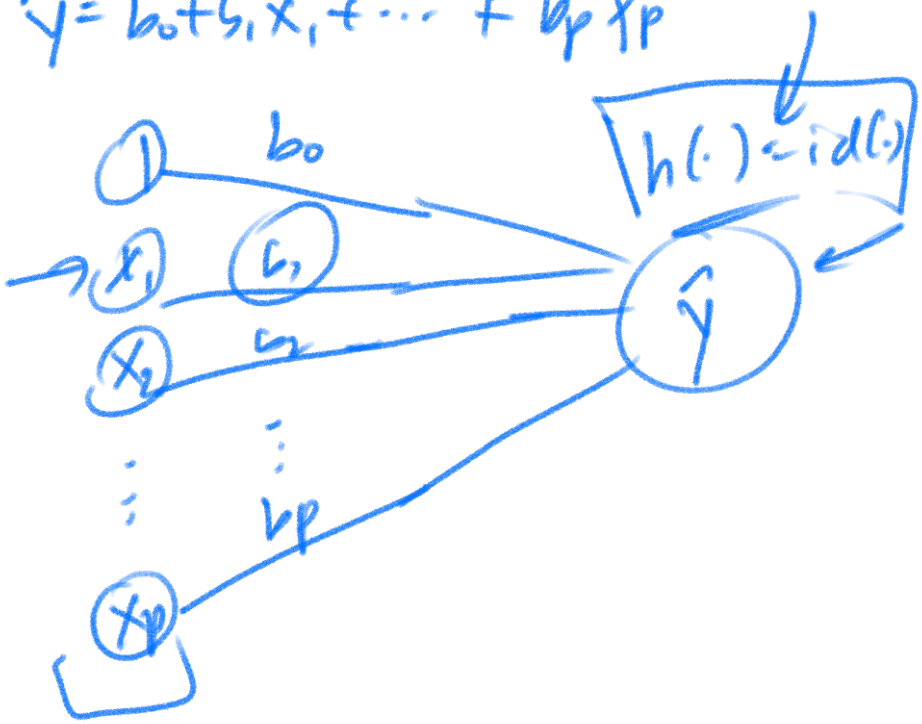
Exercise:

$$\hat{y} = b_0 + b_1 \cdot x$$

Sketch the linear regression model as a neural network diagram.

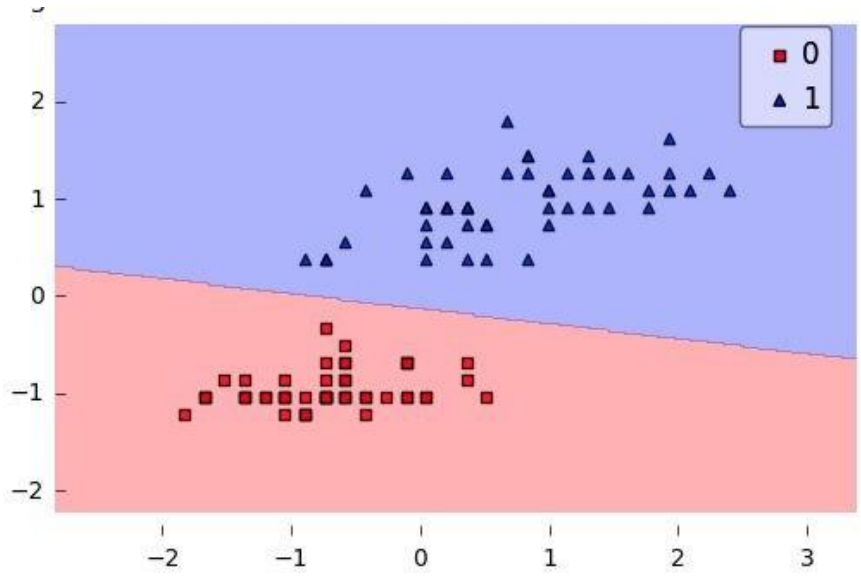


$$\hat{y} = b_0 + b_1 x_1 + \cdots + b_p x_p$$



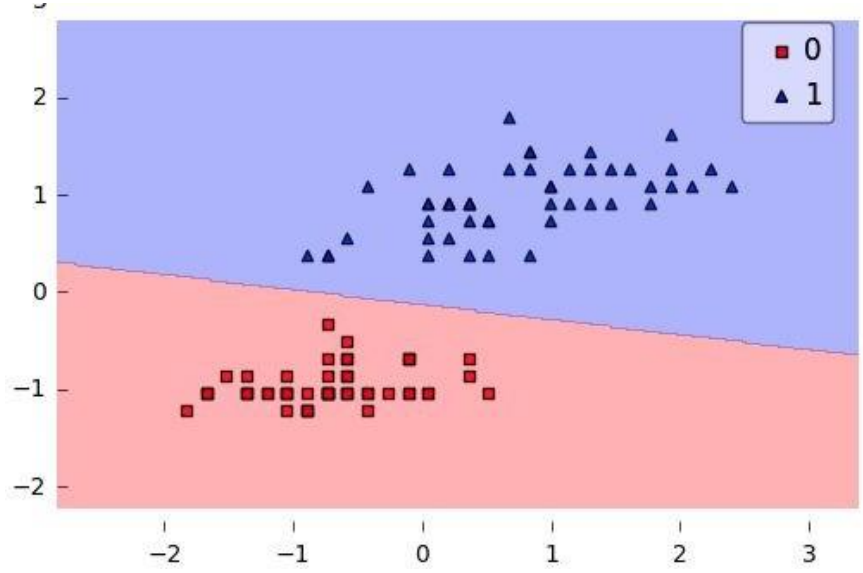
Review of logistic regression

- Model classification problems
- In picture: two features and the two labels (`0` or `1`)
- Linear classifier
- Objective: find the line that separates `0`s from `1`s



Logistic regression: Model

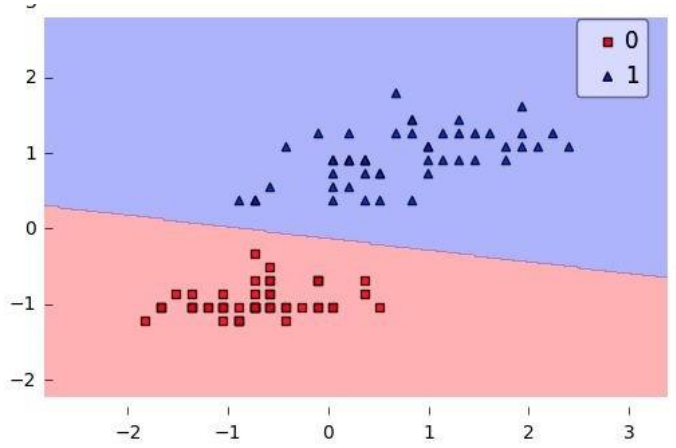
- Model classification problems
- Model predicts the probability of class 1
- E.g. predict dogs versus cats



$$\hat{y} = \sigma(b_0 + b_1 \cdot x)$$

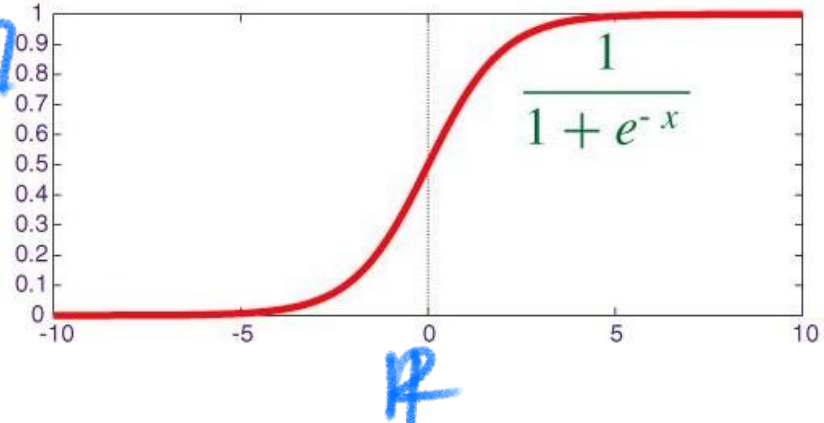
Logistic regression: Model

- Model classification problems
- Model predicts the probability of class 1



$$\hat{y} = \sigma(b_0 + b_1 \cdot x)$$

[0,1]



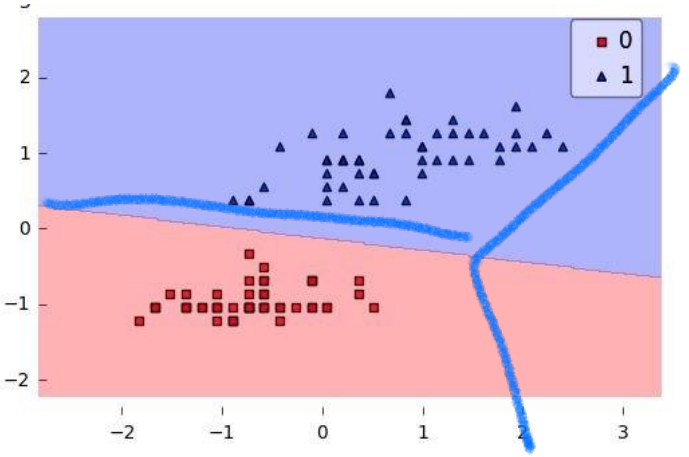
Logistic regression

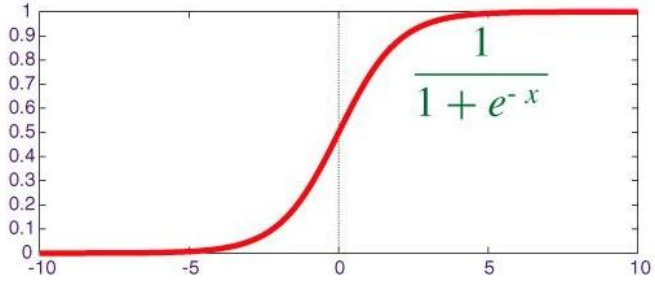
Model:

$$\hat{y} = \sigma(b_0 + b_1 \cdot x)$$

Loss function: $-[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$

Cost function: Average the loss function over the training points.

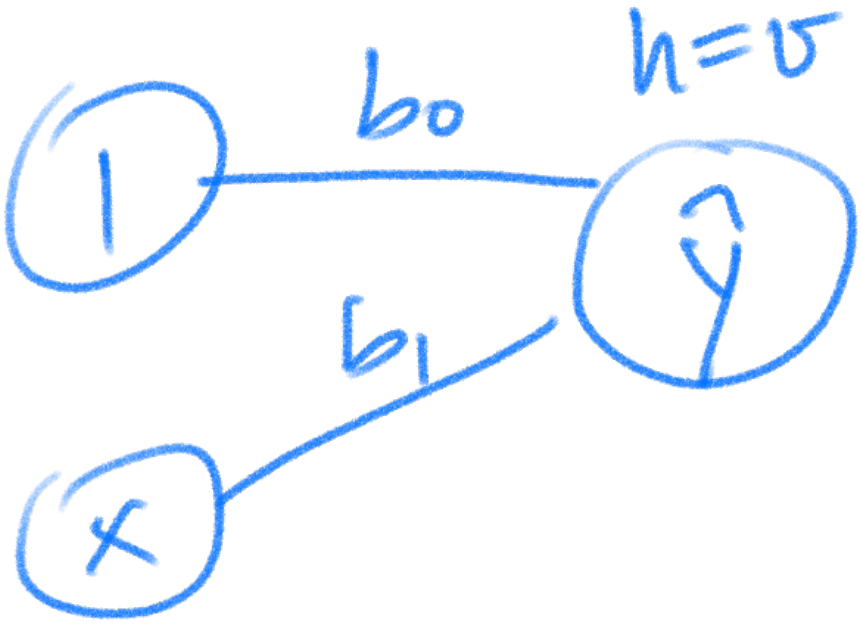




Exercise:

$$\hat{y} = \sigma(b_0 + b_1 \cdot x)$$

Sketch the logistic regression model as a neural network diagram.



Multi-class logistic regression

- Classification problem
- Instead of two classes we have `K` classes
  - Predict dog breed from images
- Generalization of logistic regression
  - Need `K` lines instead of one

Multi-class logistic regression

Instead of two classes we have `K` classes

- Predict dog breed from images

$\hat{y} = (\hat{y}_1, \ldots, \hat{y}_K)$ prediction is a tuple

$\sum_{k=1}^{K} \hat{y}_k = 1$ sum of all predictions is one

Loss:

$$-\sum_{k=1}^{K=2} y_k \log \hat{y}_k$$

Multi-class logistic regression

For multi-class logistic regression we need K linear equations.

$$
\begin{aligned}
z_1 &= b_1 + w_1 \cdot x \\
z_2 &= b_2 + w_2 \cdot x \\
&\dots \\
z_K &= b_K + w_K \cdot x
\end{aligned}
$$

Dot product

$\in \mathbb{R}$

To get the soft predictions $\hat{y} = (\hat{y}_1, \dots, \hat{y}_K)$ from $z = (z_1, \dots, z_K)$ we use the **softmax** function to get probabilities.

$$\hat{y}_k = P(y = k|x) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\sum_{k=1}^{K} \hat{y}_k = 1$$

Multi-class logistic regression

$$z_1 = b_1 + w_1 \cdot x$$
$$z_2 = b_2 + w_2 \cdot x$$
$$\dots$$
$$z_K = b_K + w_K \cdot x$$

Cam be written as

$$z = W \cdot x + b$$

Where $z = (z_1, z_2, \dots, z_K)^T$, $b = (b_1, b_2, \dots, b_K)^T$ and w_i are the rows of W. If $x \in \mathbb{R}^D$, W is KxD.

Multi-class logistic regression

$$z = W \cdot x + b$$

$$\hat{y} = softmax(z)$$

softprd.

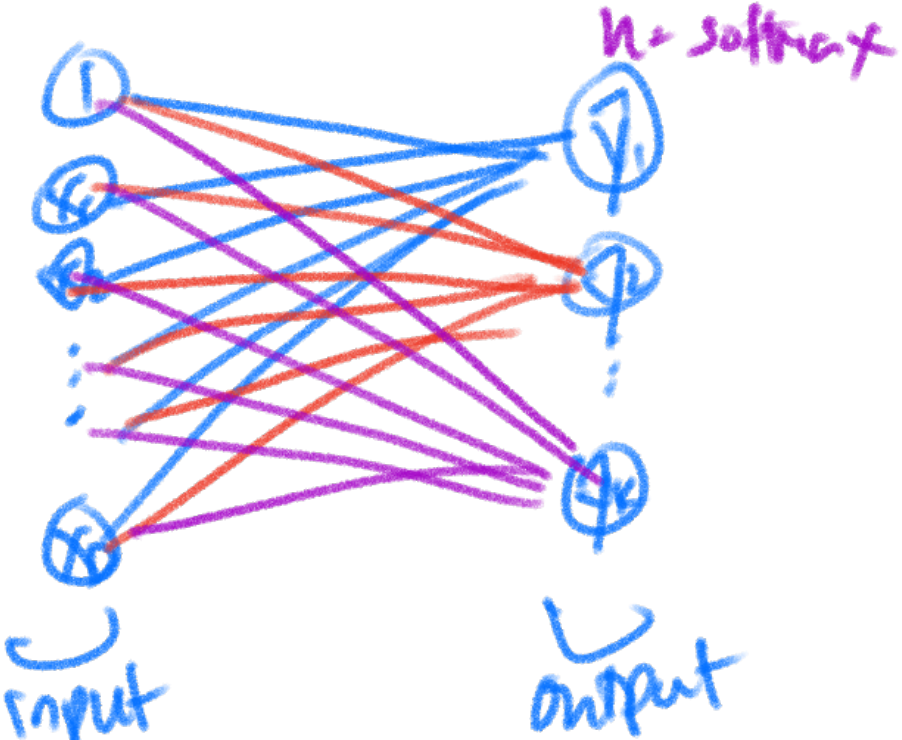
$$\hat{y}_{H} = \underset{K}{\operatorname{argmax}}\ \hat{y}_{K}$$

Exercise:

$$z = W \cdot x + b$$

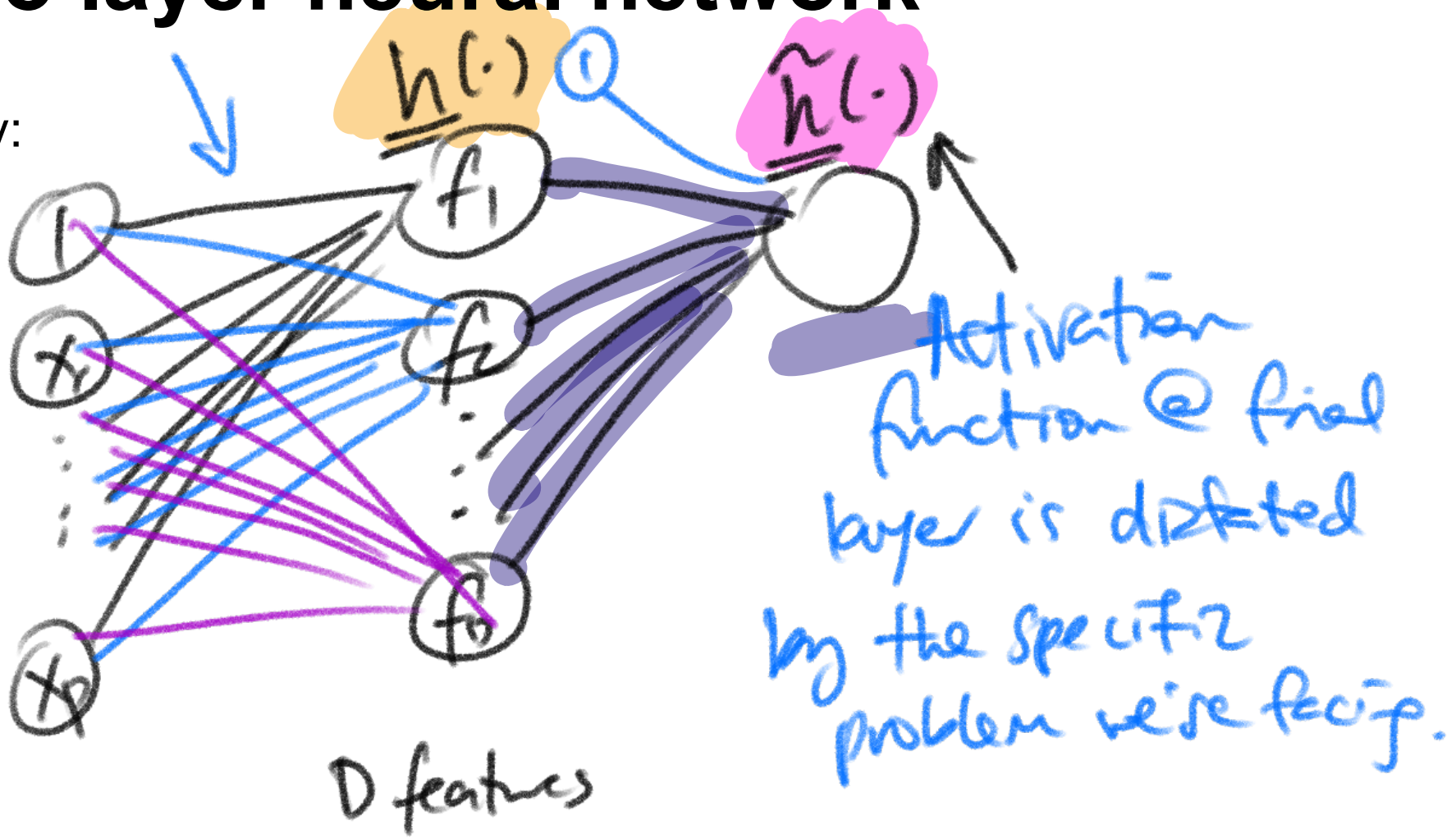
Sketch the multi-class logistic regression model as a neural network diagram.

$$\hat{y} = softmax(z)$$



A two layer neural network

Visually:



Two layer neural network for regression

Algebraically:

$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$
$$a^{[1]} = h(z^{[1]})$$
$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$
$$\hat{y} = z^{[2]} \quad (= \tilde{h}(z^{[2]}) \text{ where } \tilde{h} = \mathrm{id}.)$$

where h is the element-wise activation function.

Two layer neural network binary for classification

$$
\begin{aligned}
z^{[1]} &= W^{[1]} \cdot x + b^{[1]} \\
a^{[1]} &= h(z^{[1]}) \\
z^{[2]} &= W^{[2]} \cdot a^{[1]} + b^{[2]} \\
\hat{y} &= \sigma(z^{[2]})
\end{aligned}
$$

where h is the element-wise activation function.

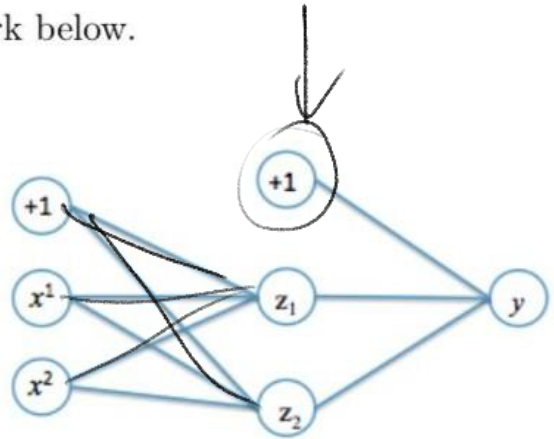
Two layer neural network multi-class classification

$$
\begin{aligned}
z^{[1]} &= W^{[1]} \cdot x + b^{[1]} \\
a^{[1]} &= h(z^{[1]}) \\
z^{[2]} &= W^{[2]} \cdot a^{[1]} + b^{[2]} \\
\hat{y} &= softmax(z^{[2]})
\end{aligned}
$$

where h is the element-wise activation function.

Example

Consider the network below.



$$\sigma(c) = h(c) = \begin{cases} 1 & \text{if } c > 0 \\ 0 & \text{otherwise.} \end{cases}$$

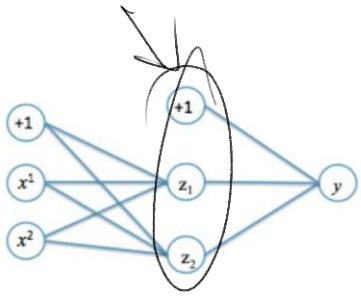
x_1	x_2	$z_1^{[1]}$	$z_2^{[1]}$	$a_1^{[1]}$	$a_2^{[1]}$	$z^{[2]}$	$\hat{y}$
0	0	-30	10	0	1	10	1
0	1	-10	-10	0	0	-10	0
1	0						0
1	1						1

Assume the following network weights:

$b_1^{[1]}$	$w_{11}^{[1]}$	$w_{12}^{[1]}$	$b_2^{[1]}$	$w_{21}^{[1]}$	$w_{22}^{[1]}$	$b^{[2]}$	$w_1^{[2]}$	$w_2^{[2]}$
-30	20	20	10	-20	-20	-10	20	20

$\hat{y}$ is approximating $1 - \text{XOR}(x_1, x_2)$

What function is this network computing?



$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$$a^{[1]} = z^{[1]}$$

$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

$$\hat{y} = softmax(z^{[2]})$$

$$z^{[2]} = W^{[2][1]} z + b^{[2]}$$

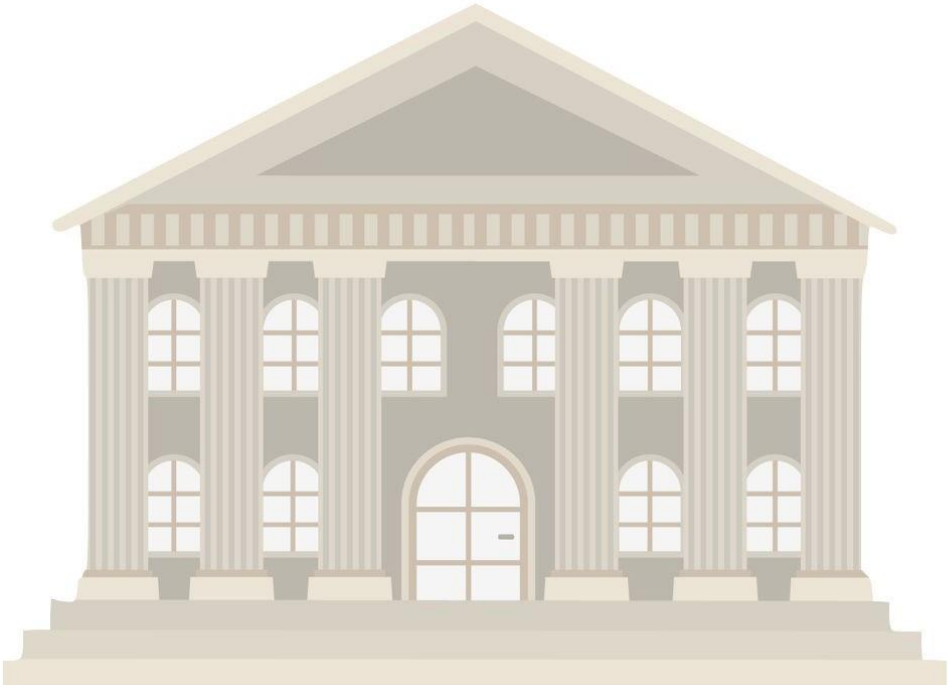
$$= W^{[2]}\left[W^{[1]}x + b^{[1]}\right] + b^{[2]}$$

$$= \underbrace{W^{[2]}W^{[1]}}_{\tilde{W}}x + \underbrace{\left(W^{[2][1]}b + b^{[2]}\right)}_{\tilde{b}}$$

$$= \tilde{W}x + \tilde{b}$$

Neural Networks – Intuition

Admission Office



Exam

Grades

Admission Office

Exam

Grades

Student 1
Exam: 9/10
Grades: 8/10

Student 2
Exam: 3/10
Grades: 4/10

Student 3
Exam: 7/10
Grades: 6/10

Would you admit Student 3?

Admission Office

Exam

Grades

Student 1
Exam: 9/10
Grades: 8/10

Student 2
Exam: 3/10
Grades: 4/10

Student 3
Exam: 7/10
Grades: 6/10

Model 1:
Score = Exam + Grades
Score > 10

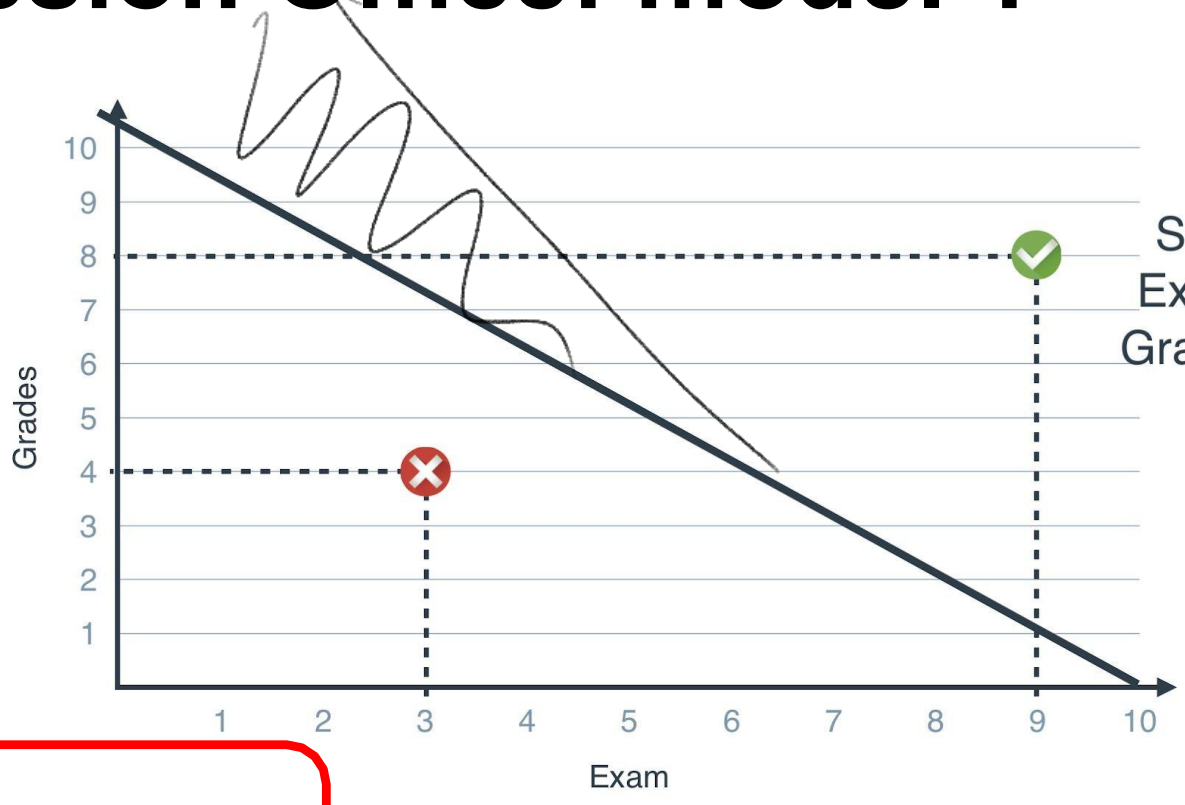
Model 2:
Score = Exam + 2*Grades
Score > 18

Model 3:
Exam > 5
Grades > 5

Admission Office: model 1

Student 1
Exam: 9/10
Grades: 8/10

Student 2
Exam: 3/10
Grades: 4/10



Student 3
Exam: 7/10
Grades: 6/10

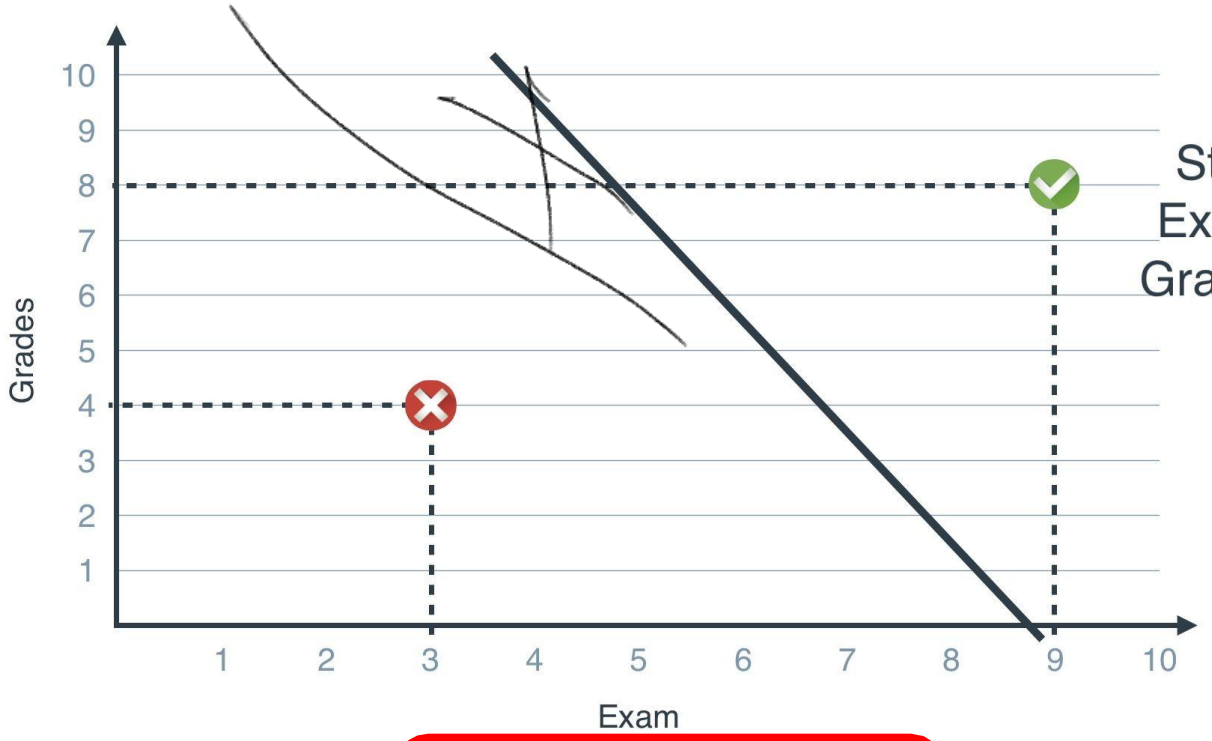
Score = Exam + Grades
Score > 10

Score = Exam + 2*Grades
Score > 18

Admission Office: model 2

Student 1
Exam: 9/10
Grades: 8/10

Student 2
Exam: 3/10
Grades: 4/10



Student 3
Exam: 7/10
Grades: 6/10

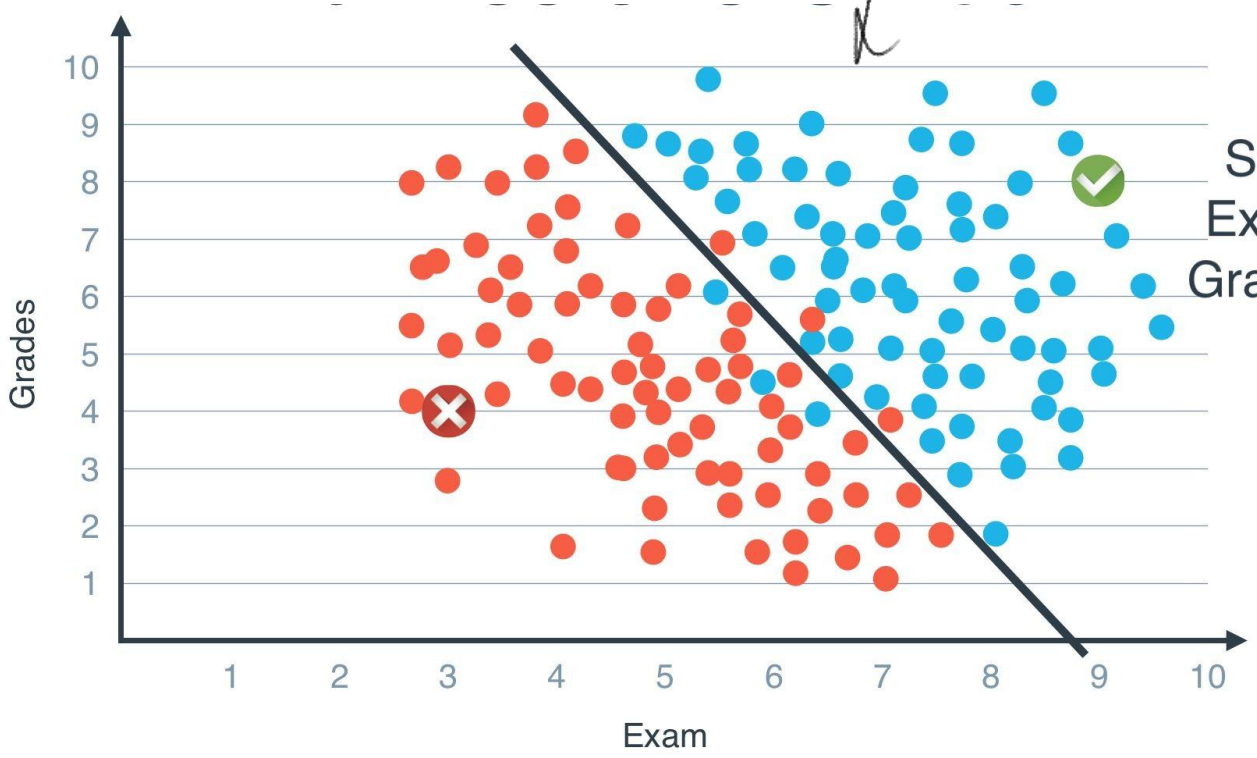
Score = Exam + Grades
Score > 10

Score = Exam + 2*Grades
Score > 18

Does the student get accepted?

Student 1
Exam: 9/10
Grades: 8/10

Student 2
Exam: 3/10
Grades: 4/10

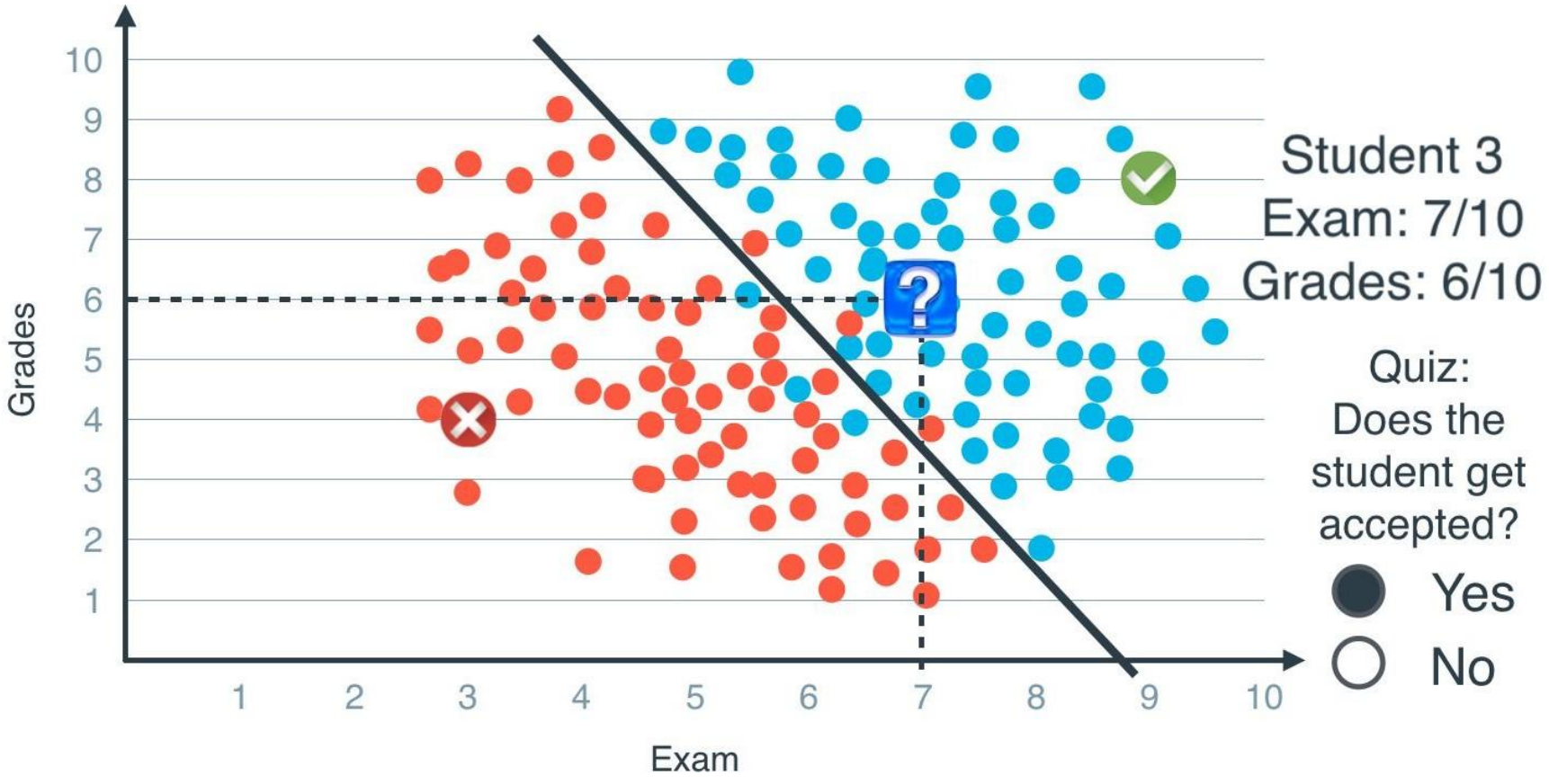


Student 3
Exam: 7/10
Grades: 6/10

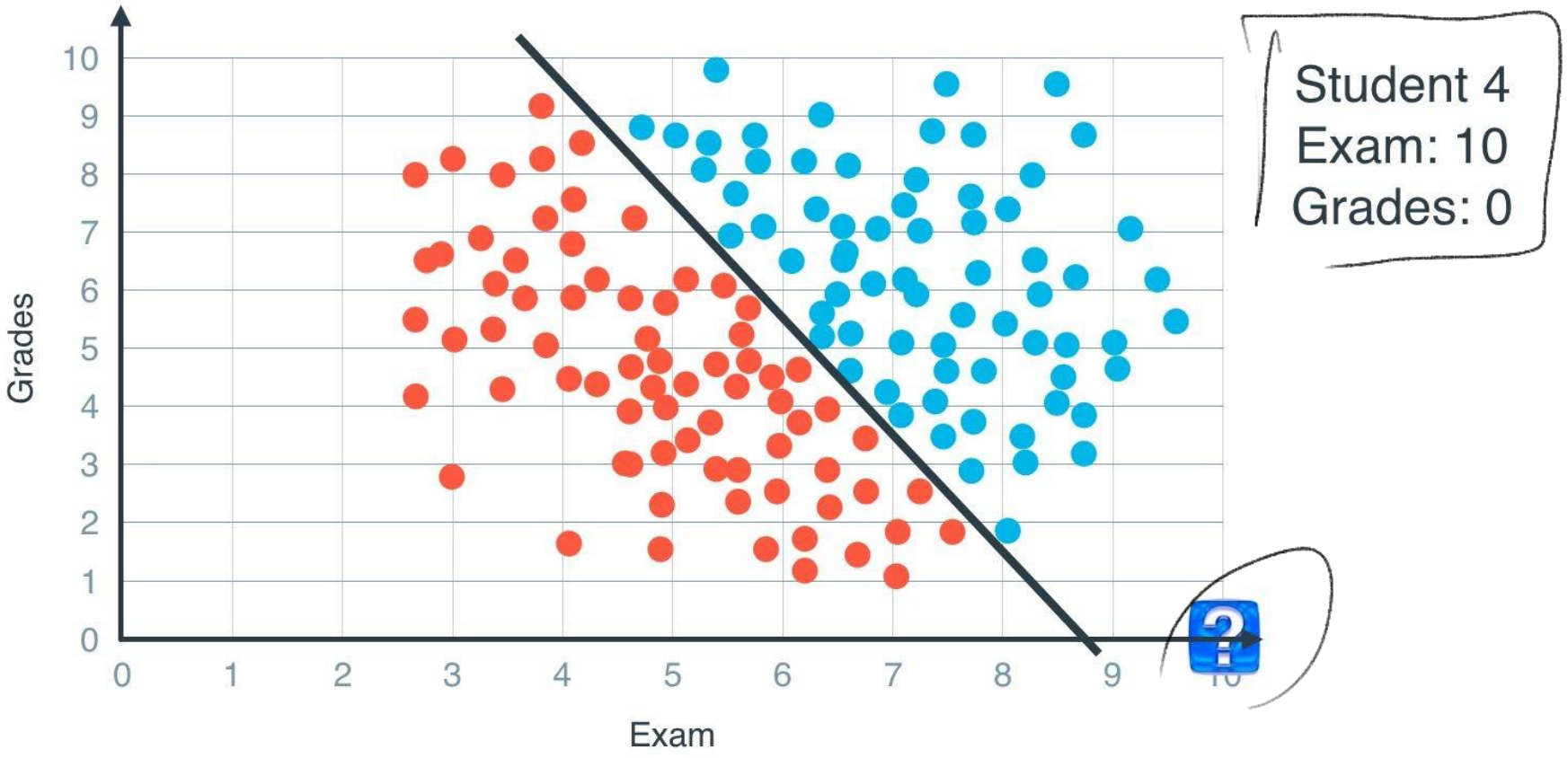
Score = Exam + Grades
Score > 10

Score = Exam + 2*Grades
Score > 18

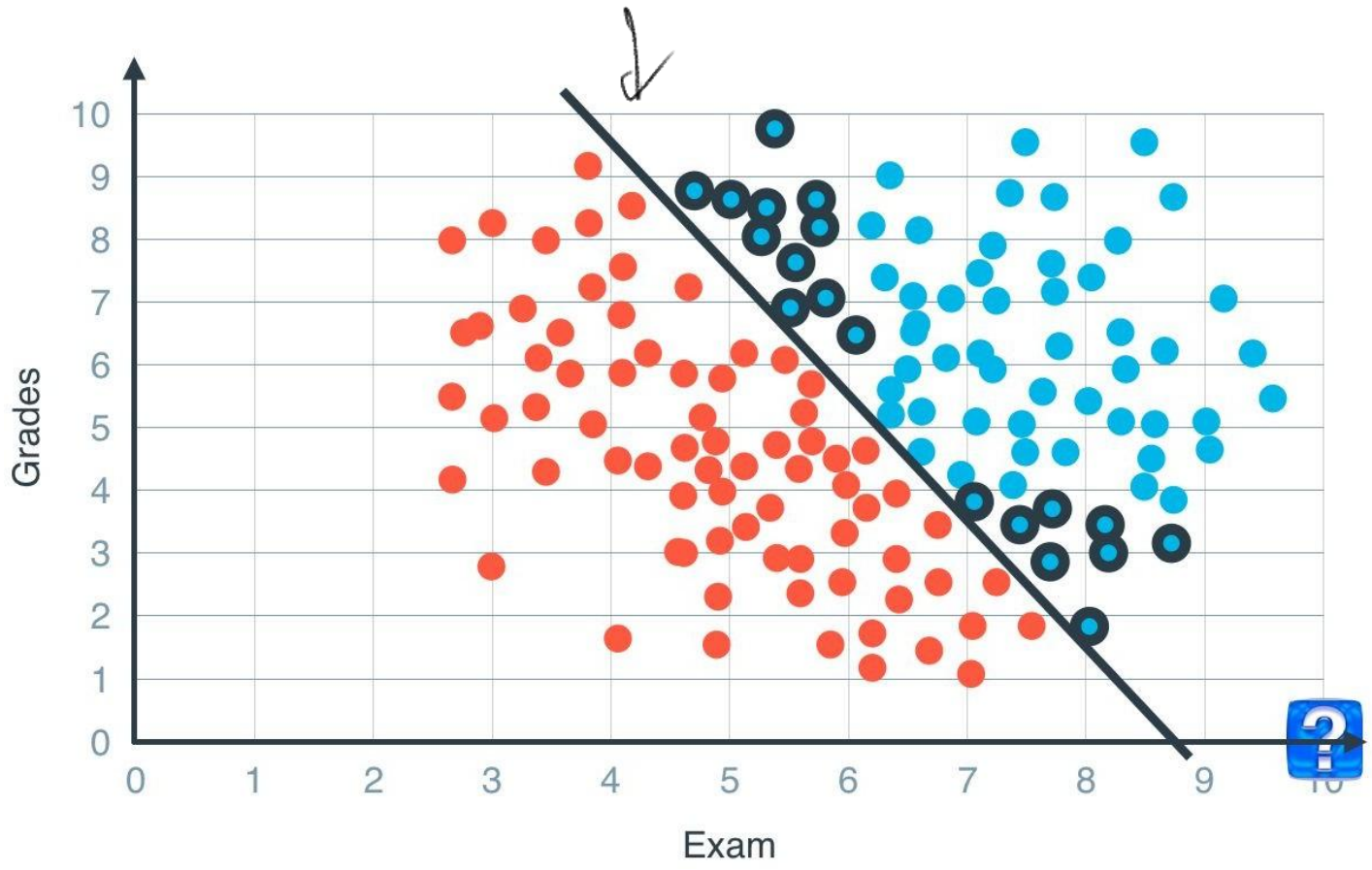
Given this model: yes



Would you accept this student?

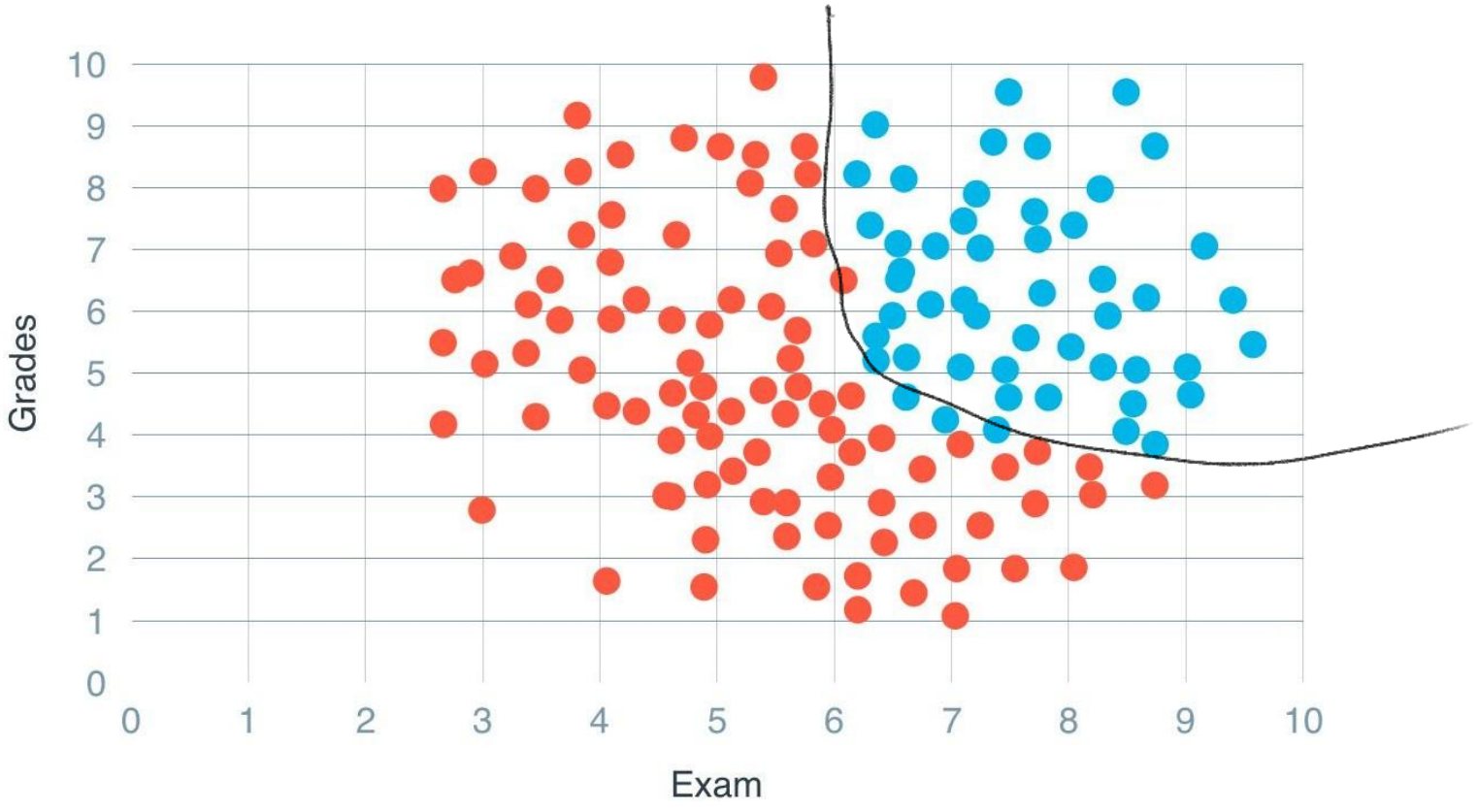


Real data is often non-linear

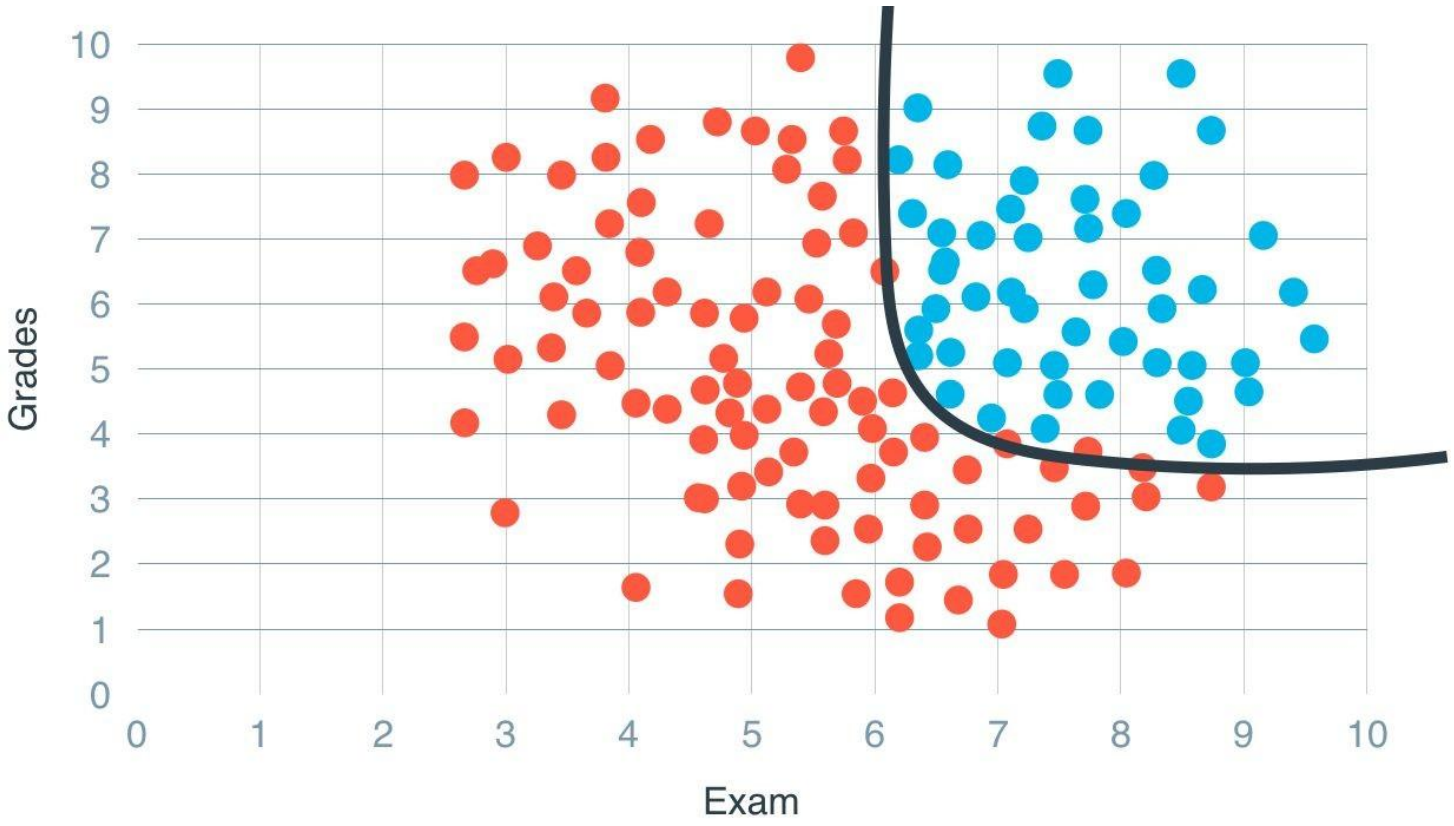


Student 4
Exam: 10
Grades: 0

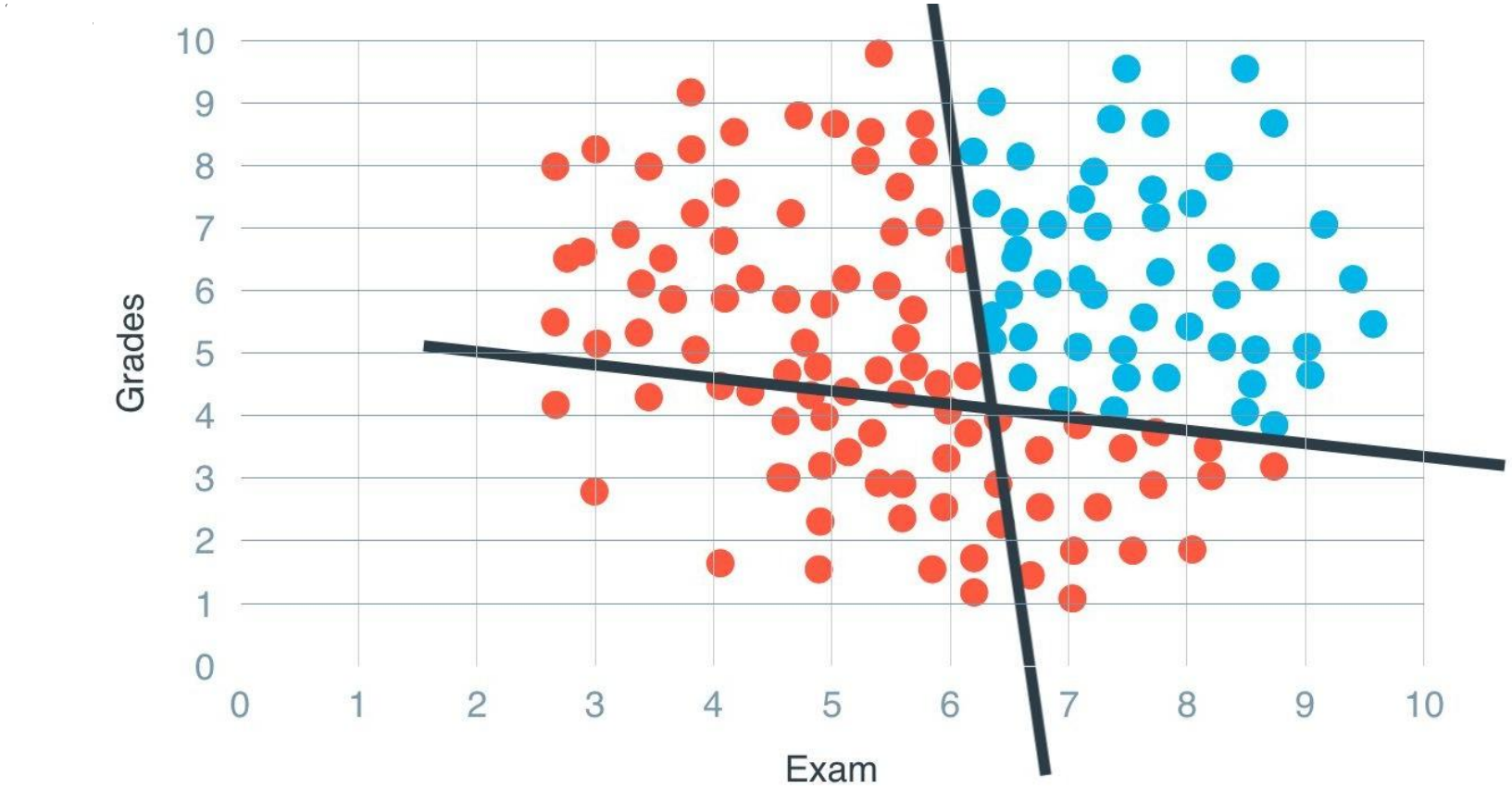
Real data is often non-linear



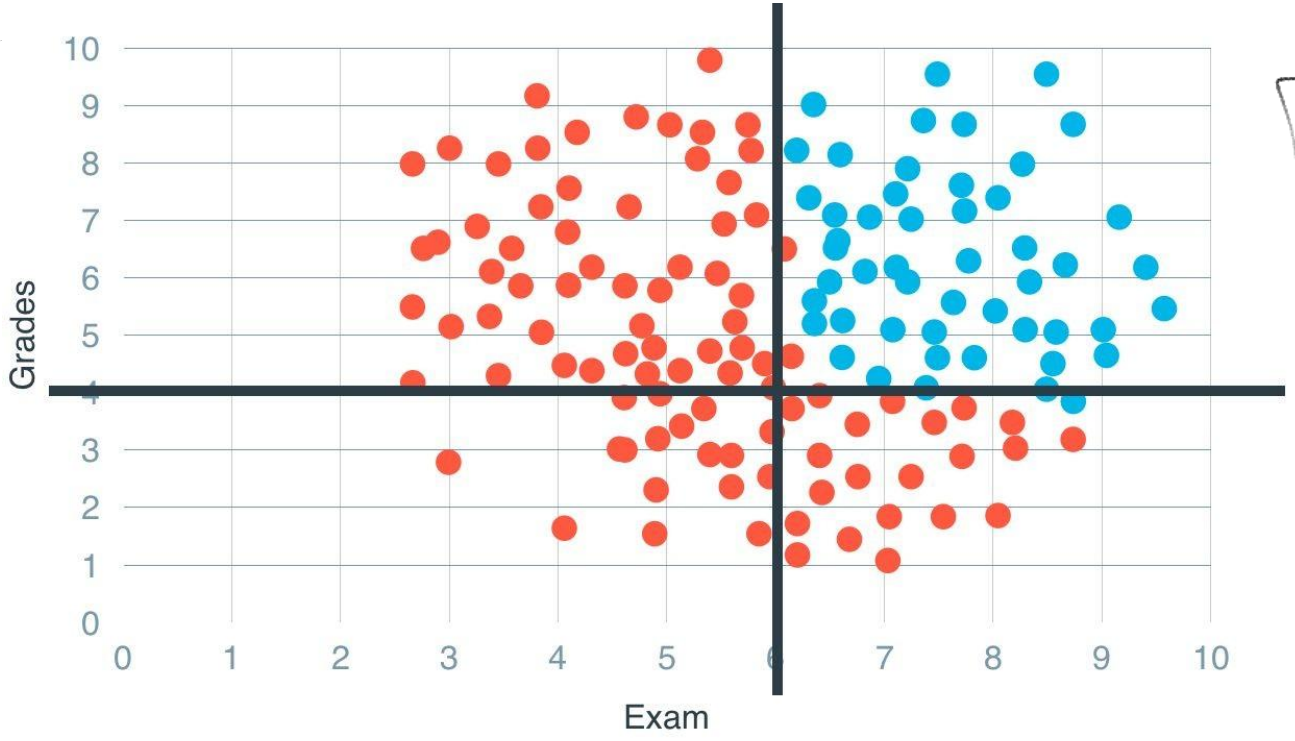
Real data is often non-linear



How to get non-linear from linear

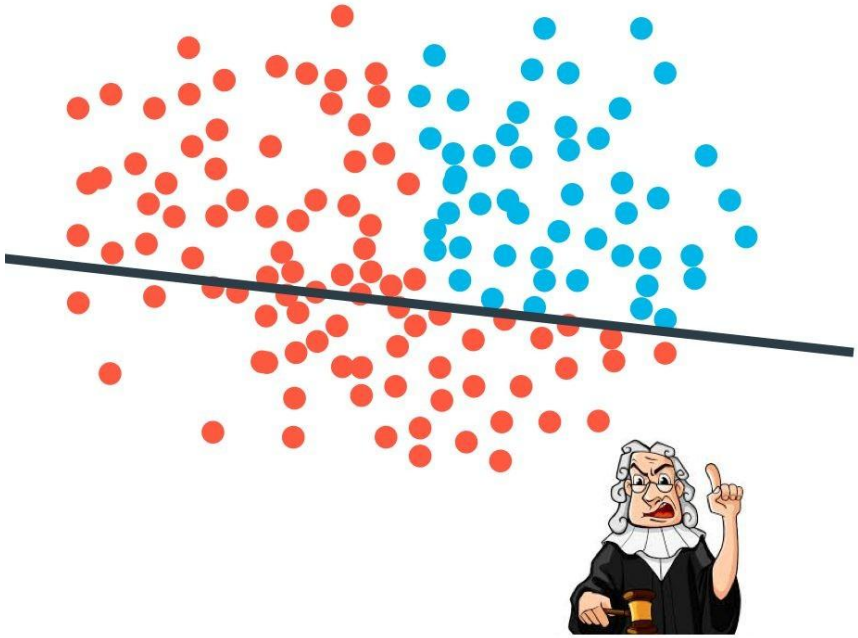


How to get non-linear from linear

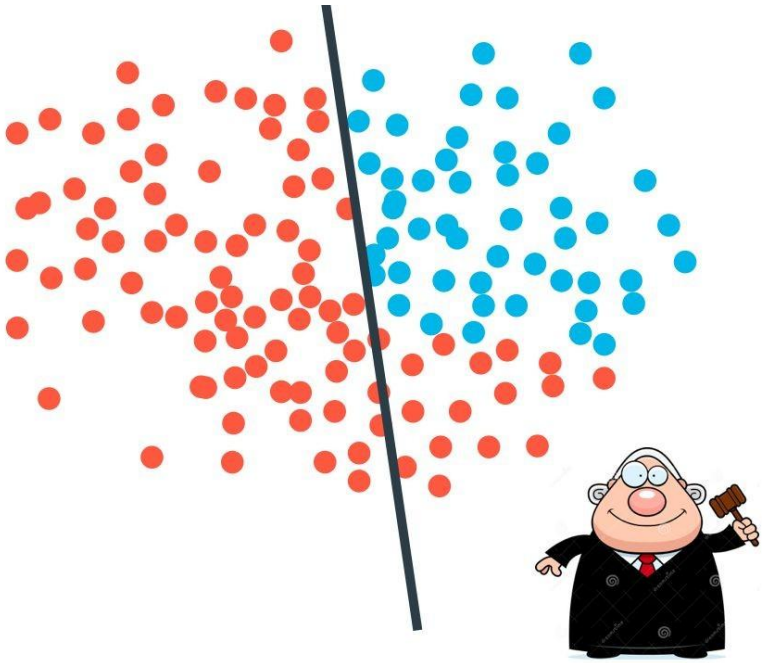


Exam > 6
Grades > 4

What’s the consensus??

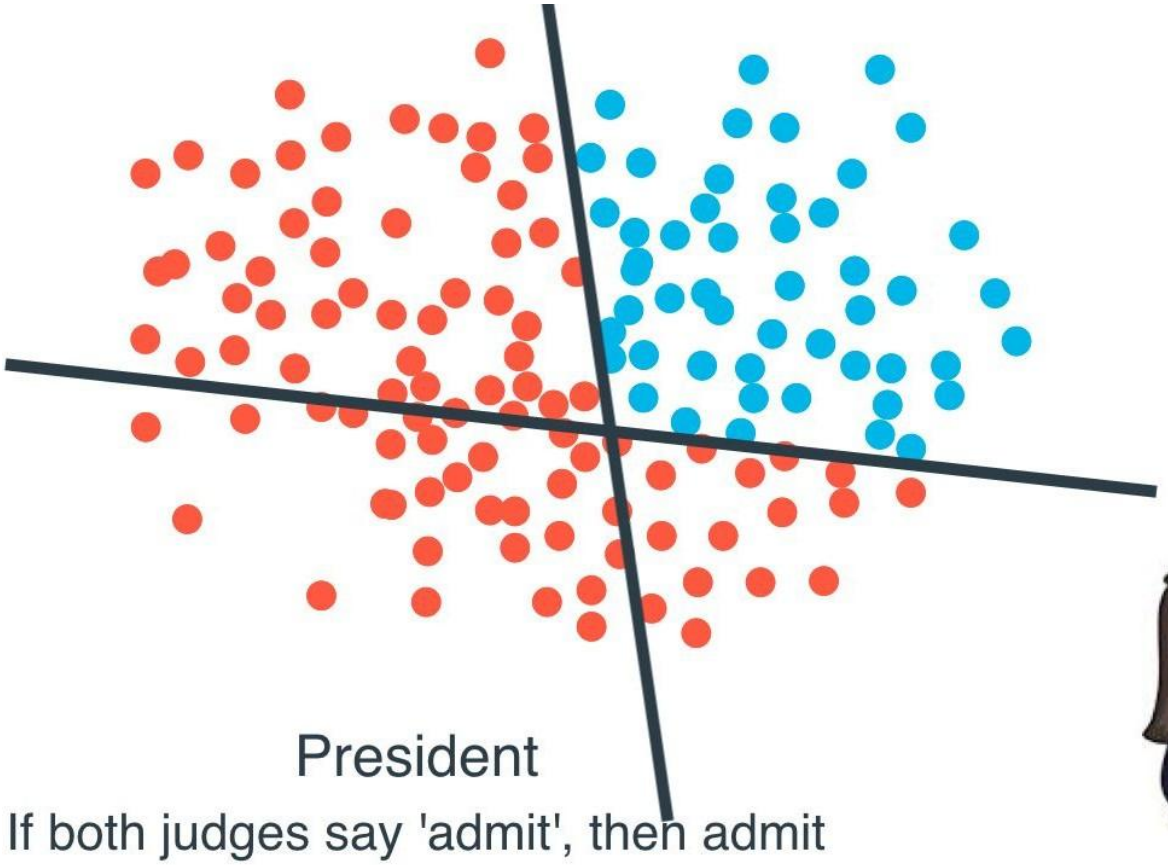


Judge 1

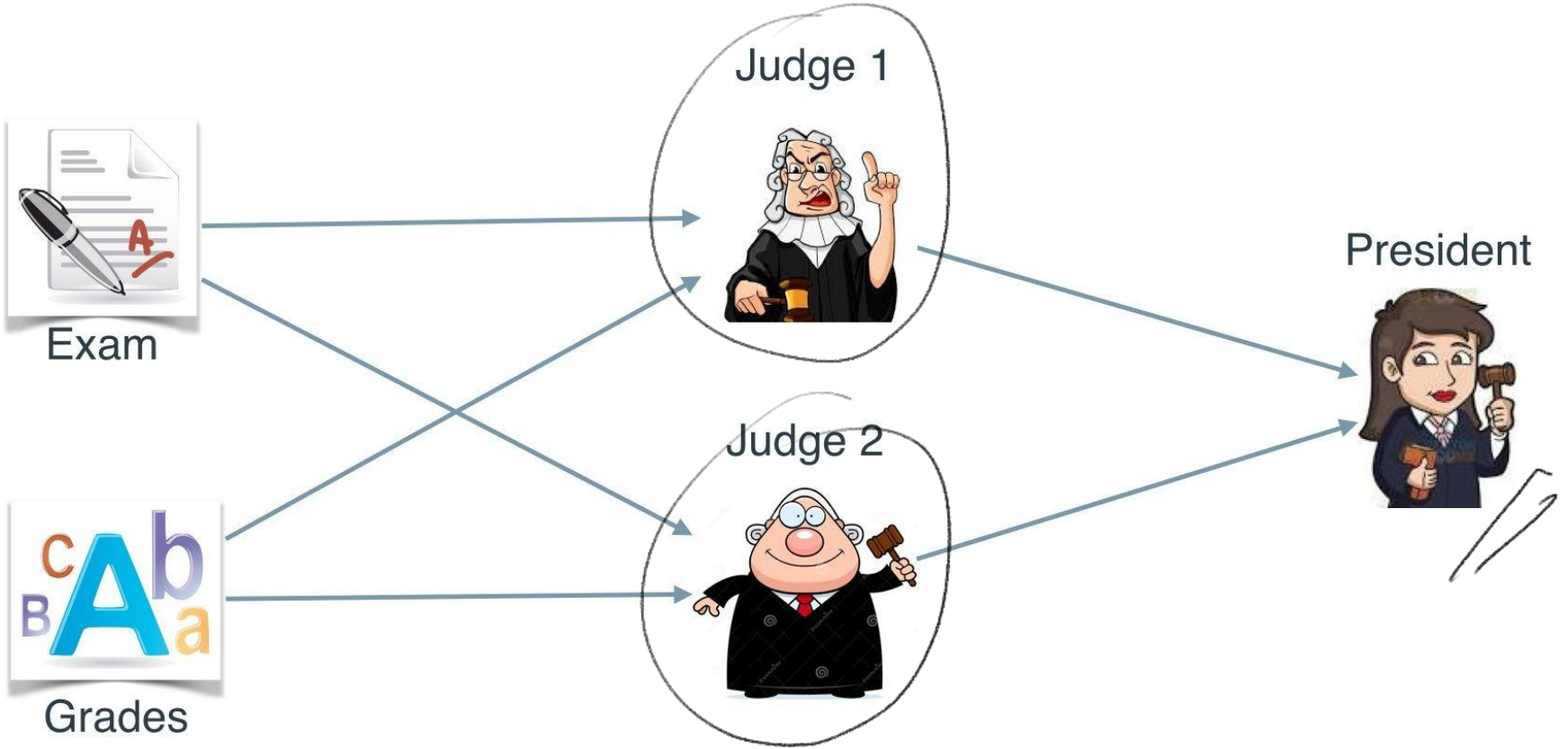


Judge 2

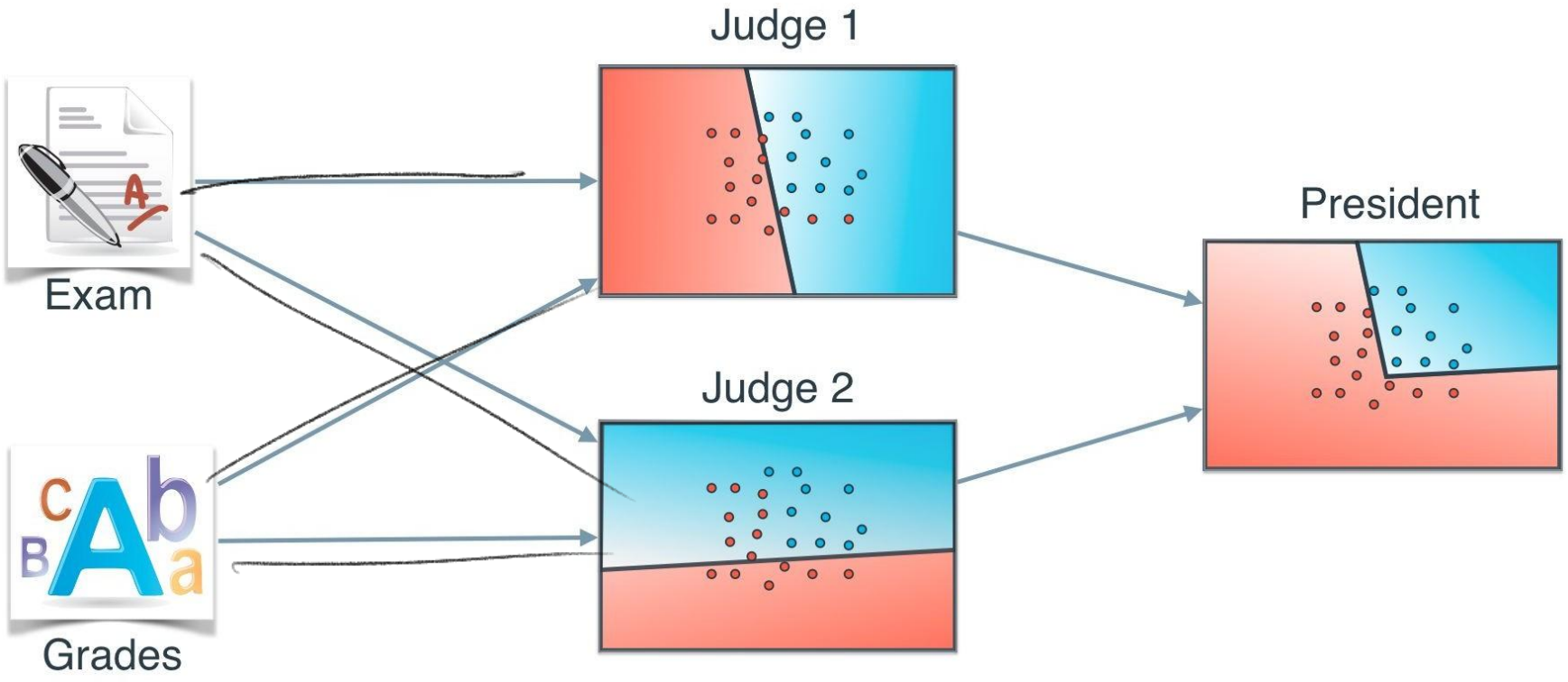
How the president makes decisions



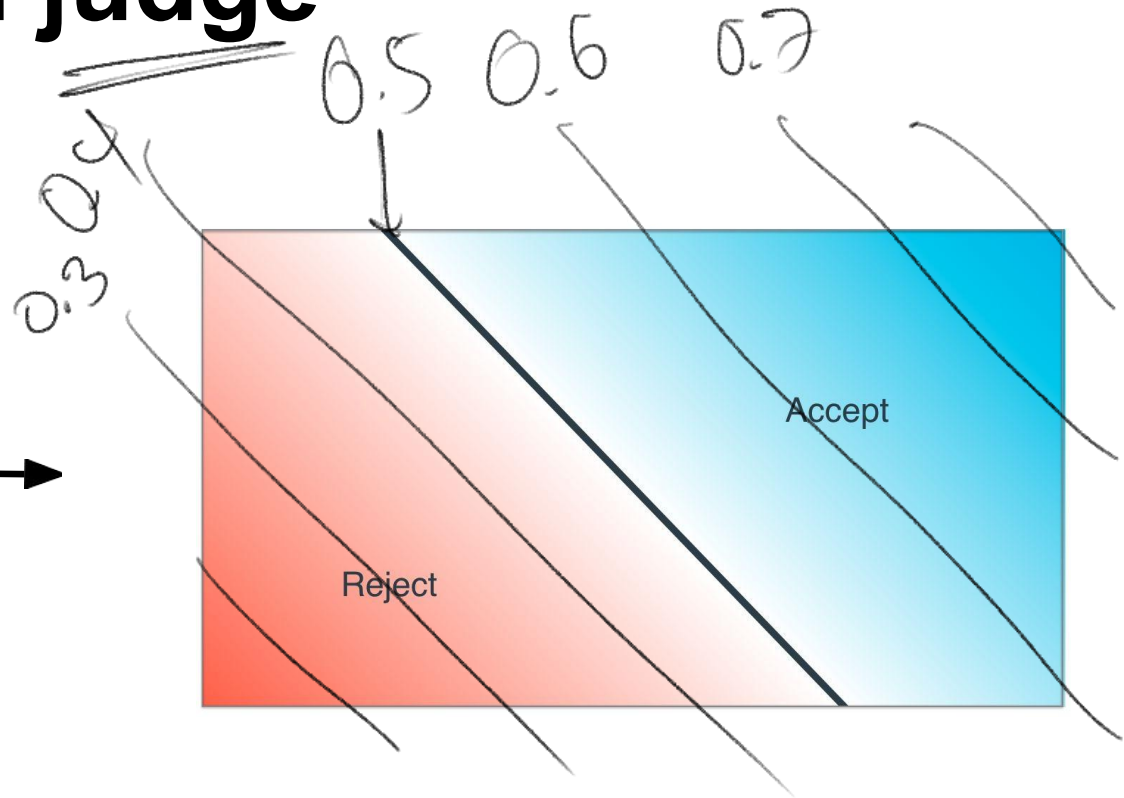
Neural Network



Neural Network

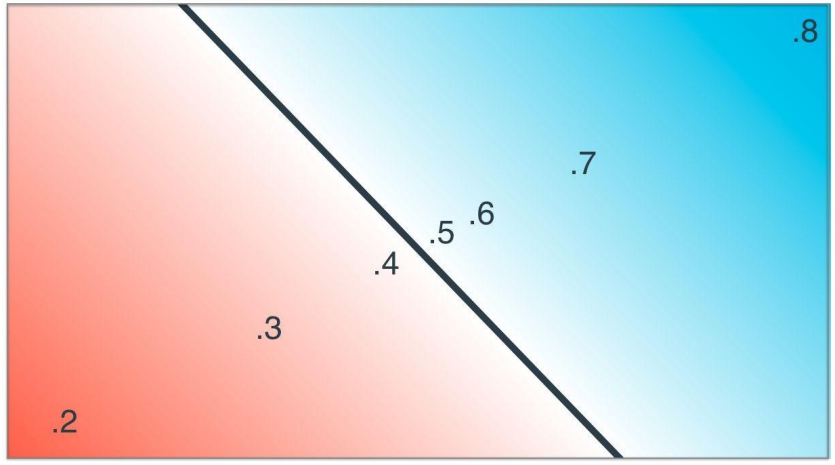


Normalizing each judge

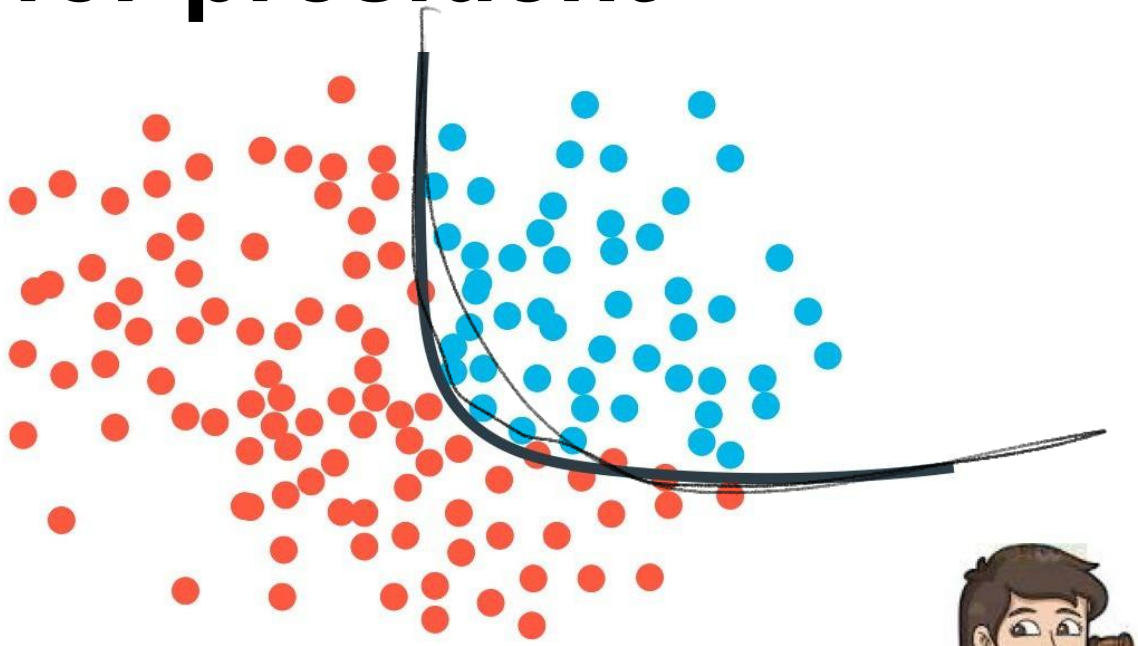


The sigmoid function maps raw scores into normalized scores (e.g. probabilities).

sigmoid



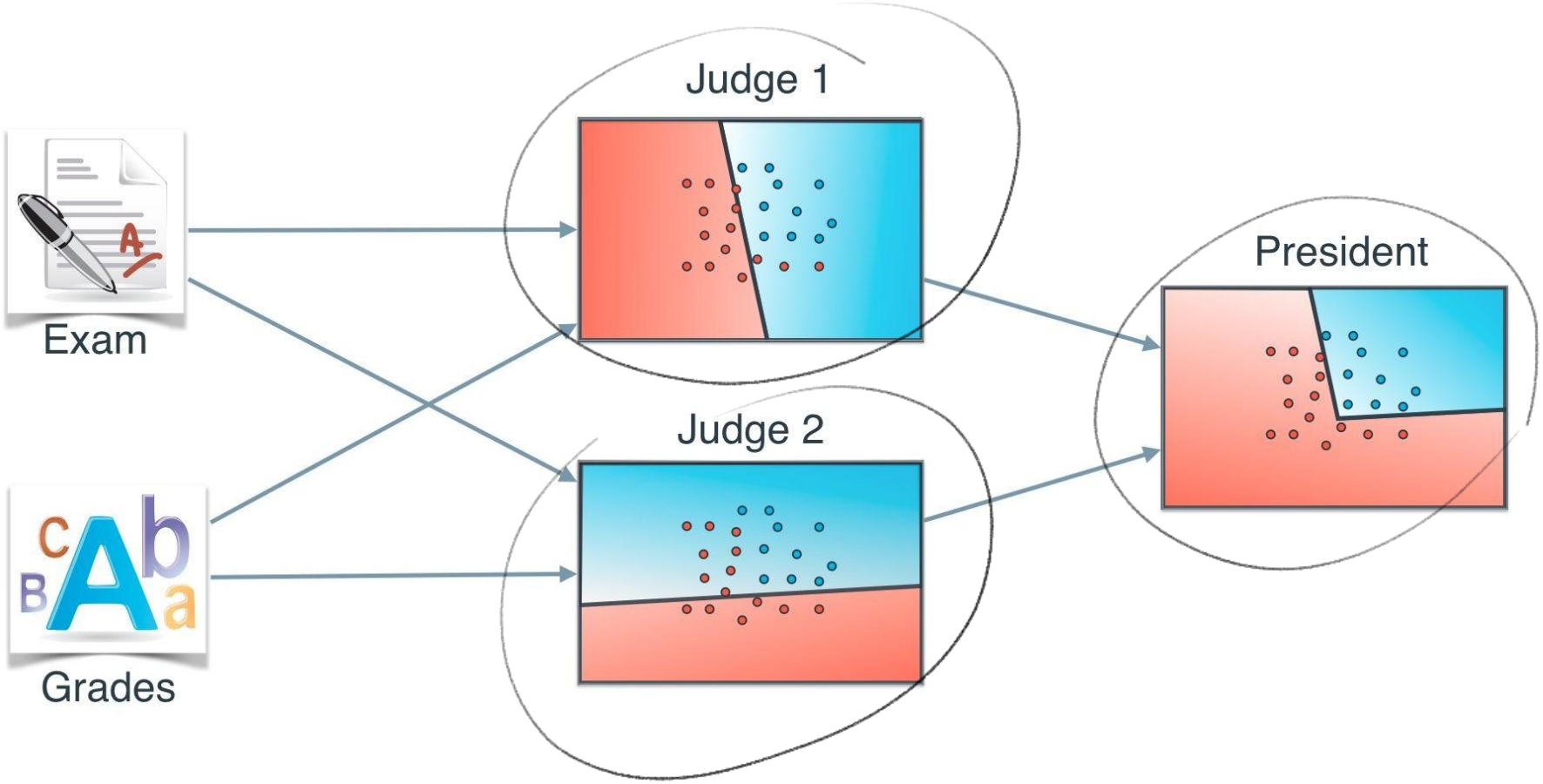
Formula for president



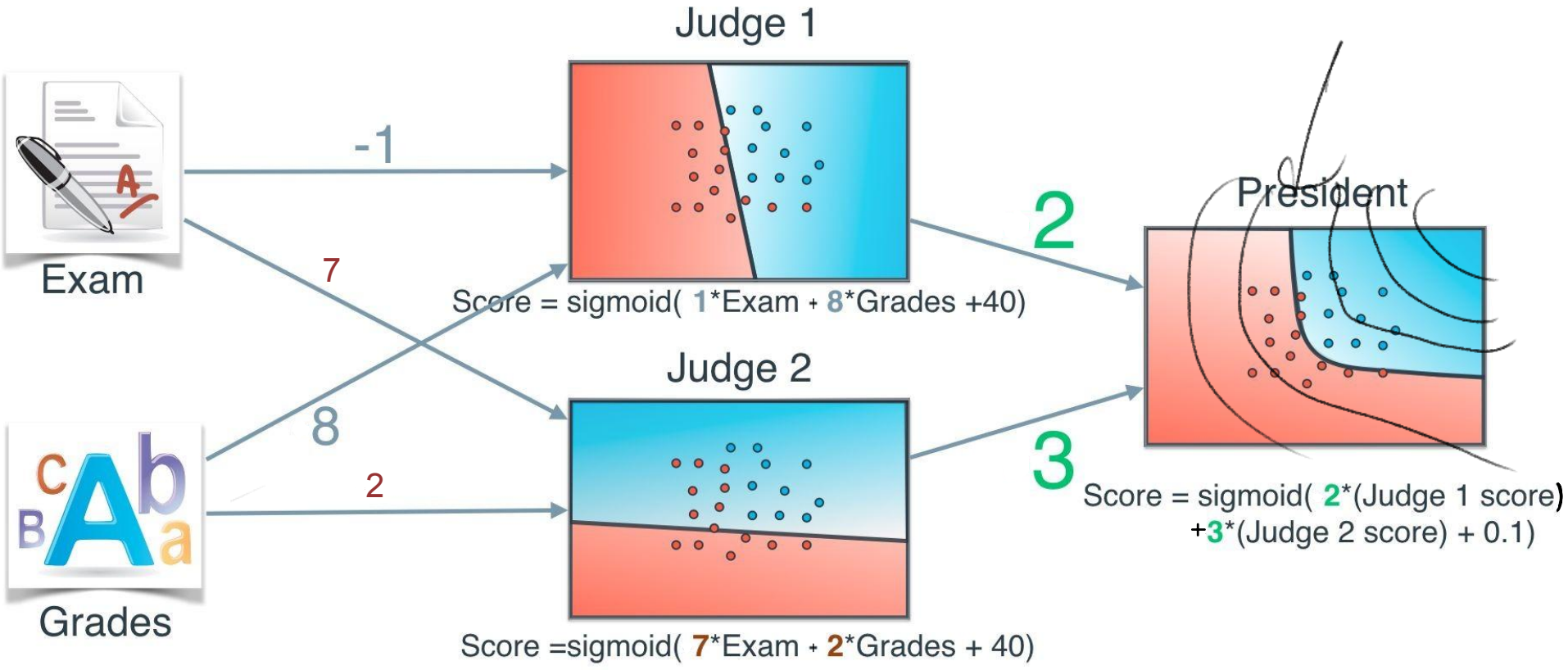
President

Score = 2*(Judge 1 Score) + 3*(Judge 2 Score)

Neural Network



Full network details



Break