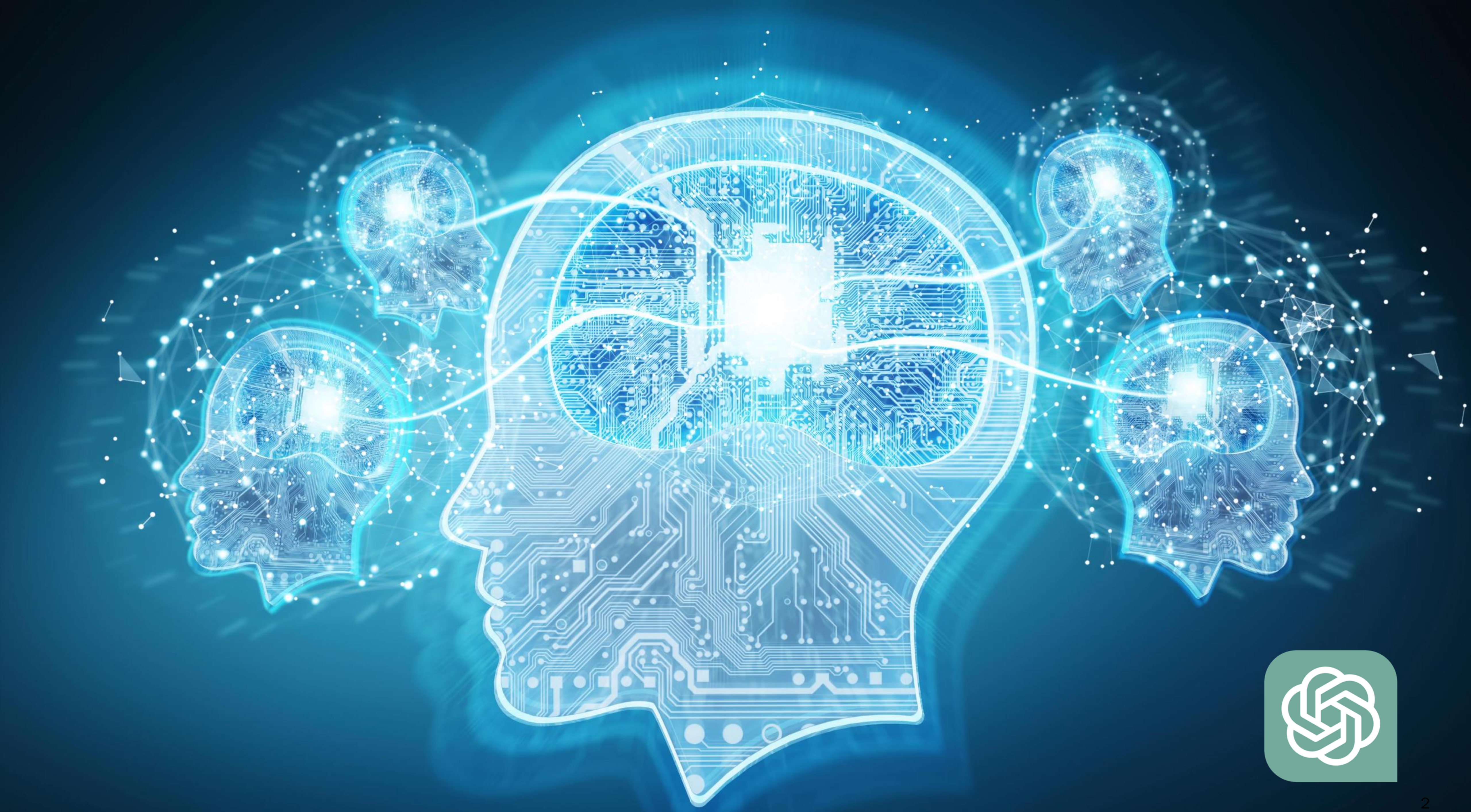
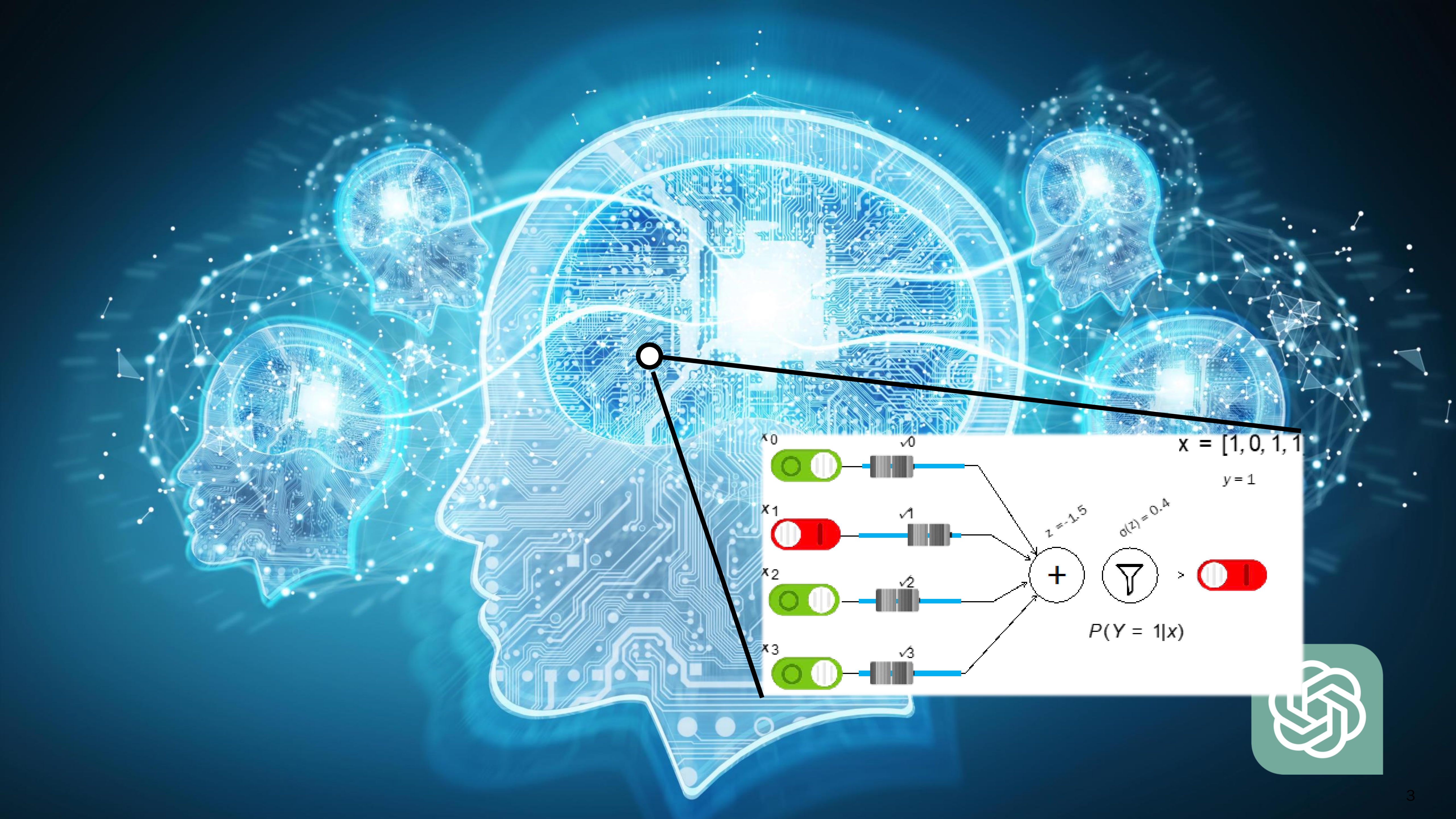
Logistic Regression

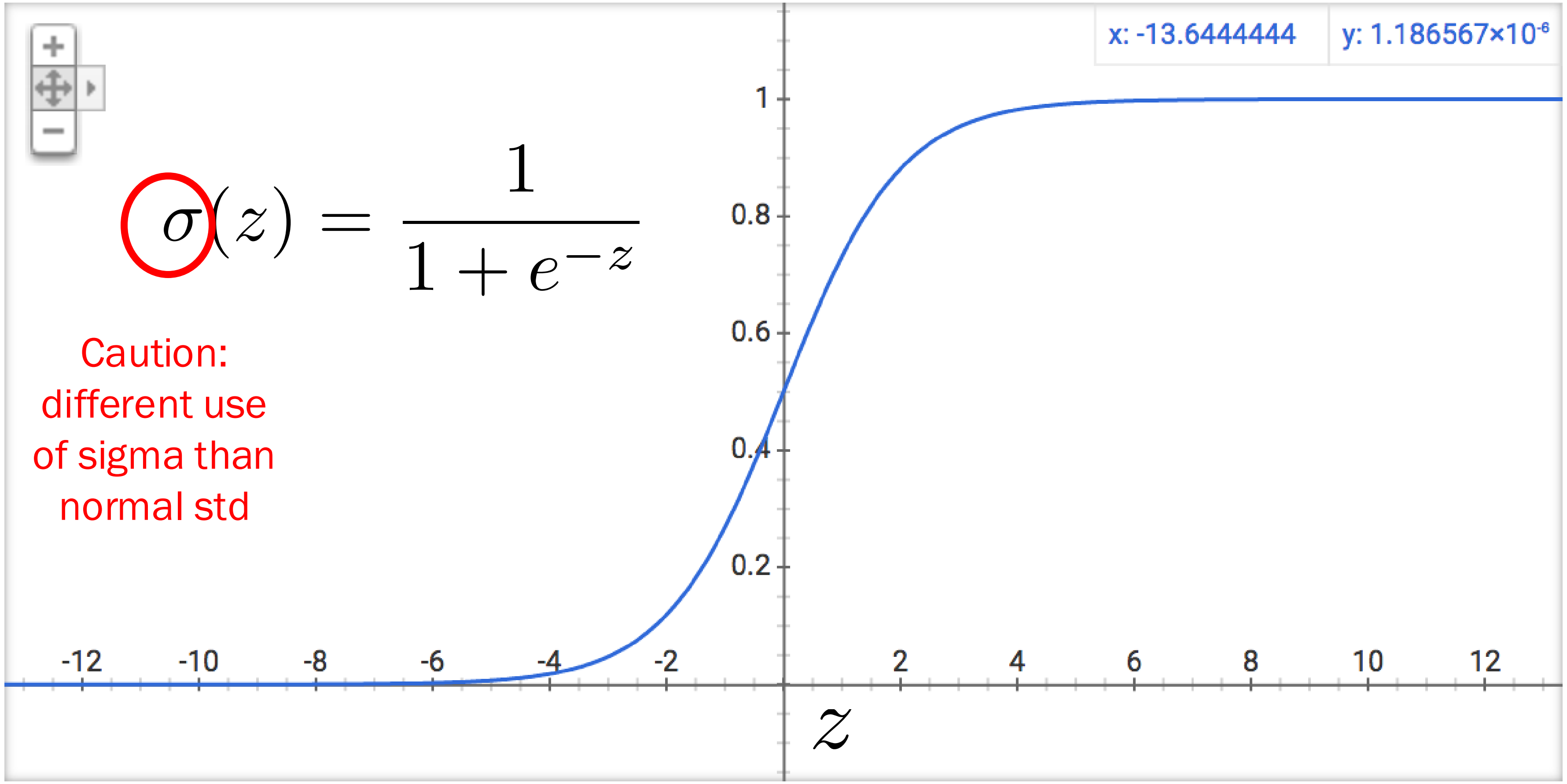
Chris Piech
CS109, Stanford University




x0
√0
x = [1, 0, 1, 1
y = 1
x1
√1
z = -1.5
σ(z) = 0.4
x2
√2
>
P(Y = 1|x)
x3
√3

Review

Background: Sigmoid Function



The sigmoid function squashes z to be a number between 0 and 1

Background: Key Notation

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

Sigmoid function

$$\theta^T \mathbf{x} = \sum_{i=1}^{n} \theta_i x_i$$

$$= \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Weighted sum
(aka dot product)

$$\sigma(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

Sigmoid function of weighted sum

Background: Chain Rule

Who knew calculus would be so useful?

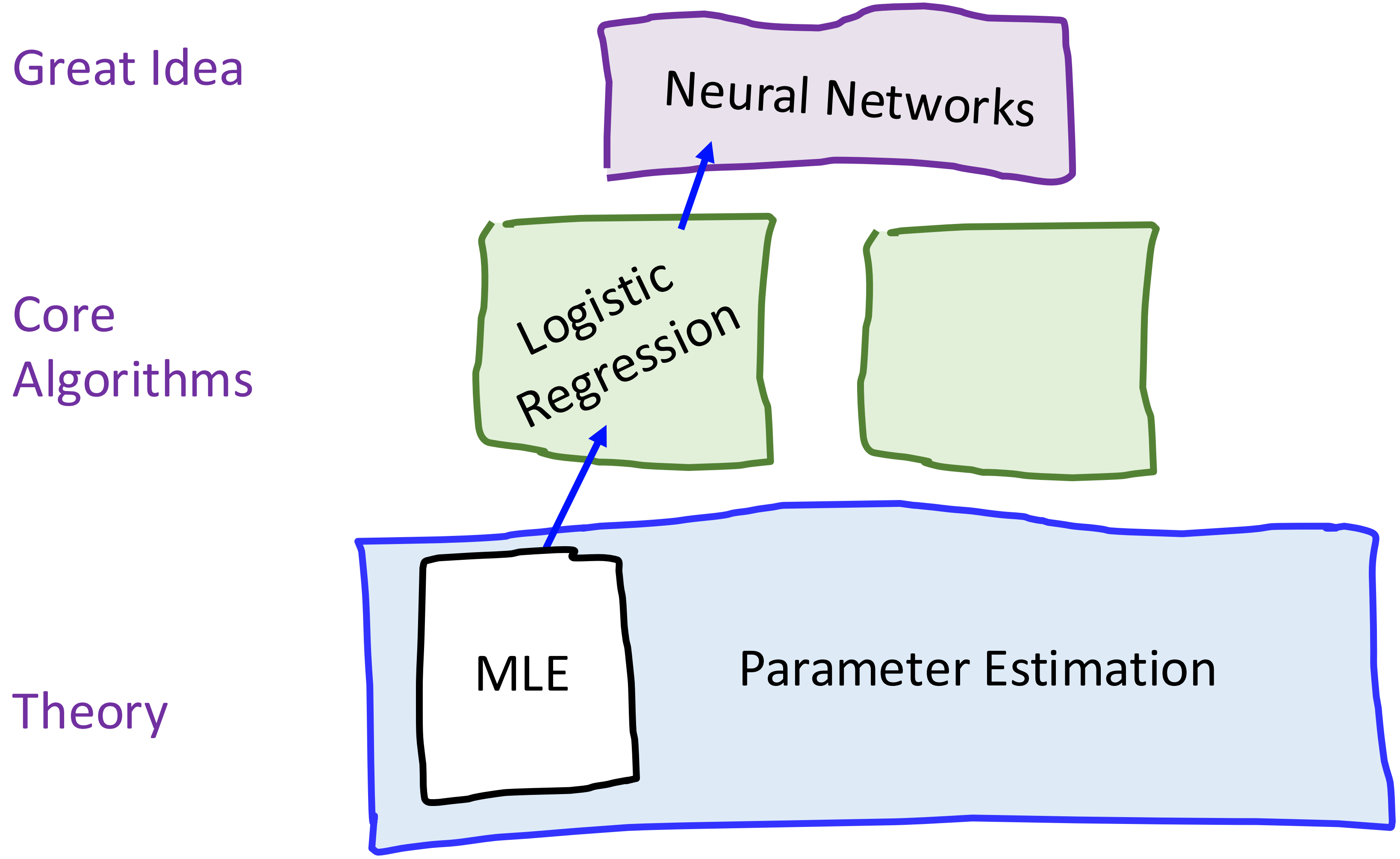
$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(z)}{\partial z} \cdot \frac{\partial z}{\partial x}$$

Aka decomposition of composed functions

$$f(x) = f(z(x))$$

Machine Learning
(aka Applied Probability)

Machine Learning in CS109



MLE Idea: Chose params that make the data look likely

Data = [6.3 , 5.5 , 5.4, 7.1, 4.6, 6.7, 5.3 , 4.8, 5.6, 3.4, 5.4, 3.4, 4.8, 7.9, 4.6, 7.0, 2.9, 6.4, 6.0 , 4.3]

Estimate the Parameters

Parameter μ: 5.6 Parameter σ: 1.4

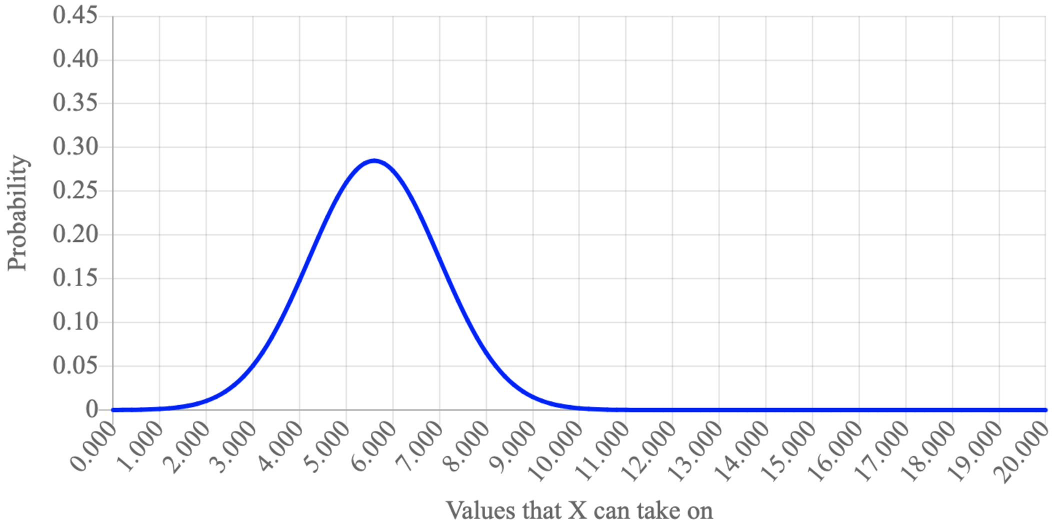
Likelihood

Likelihood: 1.9542923784106326e-15

Log Likelihood: -301.9

Best Seen: -301.9

PDF Graph



Likelihood Definition

Wikipedia:

Likelihood function

Article Talk

From Wikipedia, the free encyclopedia

The **likelihood function** (often simply called the **likelihood**) is the joint probability (or probability density) of observed data viewed as a function of the parameters of a statistical model.[1] [2] [3]

A generalized term for “PDF / PMF / Joint” of data as a function of parameters

Maximum Likelihood

That maximize likelihood

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \; f(x^{(1)}, \ldots, x^{(n)} | \theta)$$

Chose the params

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

Define likelihood, use independence.

$$LL(\theta) = \sum_{i=1}^{n} \log f(x_i|\theta)$$

Define the loglikelihood

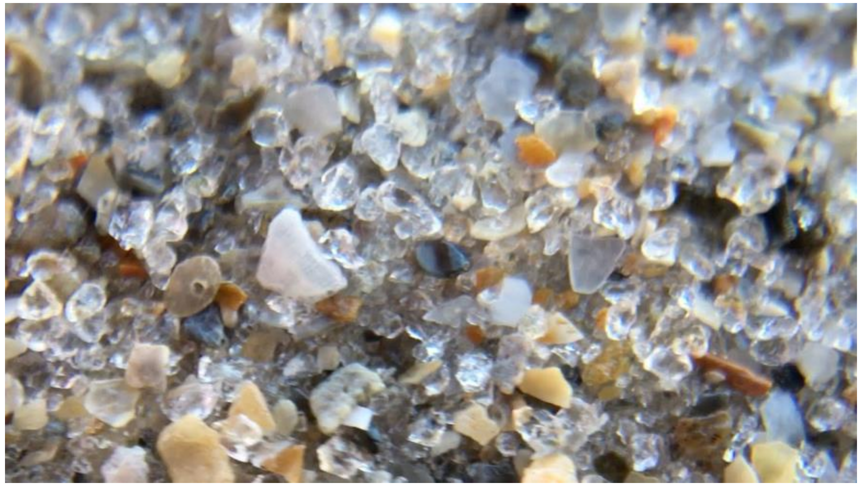
$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \; LL(\theta)$$

Use LL to chose params

MLE for a Pareto

```python
observations = [1.677, 3.812, 1.463, 2.641, 1.256, 1.678, 1.157,
1.146, 1.323, 1.029, 1.238, 1.018, 1.171, 1.123, 1.074, 1.652,
1.873, 1.314, 1.309, 3.325, 1.045, 2.271, 1.305, 1.277, 1.114,
1.391, 3.728, 1.405, 1.054, 2.789, 1.019, 1.218, 1.033, 1.362,
1.058, 2.037, 1.171, 1.457, 1.518, 1.117, 1.153, 2.257, 1.022,
1.839, 1.706, 1.139, 1.501, 1.238, 2.53 , 1.414, 1.064, 1.097,
1.261, 1.784, 1.196, 1.169, 2.101, 1.132, 1.193, 1.239, 1.518,
2.764, 1.053, 1.267, 1.015, 1.789, 1.099, 1.25 , 1.253, 1.418,
1.494, 1.015, 1.459, 2.175, 2.044, 1.551, 4.095, 1.396, 1.262,
1.351, 1.121, 1.196, 1.391, 1.305, 1.141, 1.157, 1.155, 1.103,
1.048, 1.918, 1.889, 1.068, 1.811, 1.198, 1.361, 1.261, 4.093,
2.925, 1.133, 1.573]
```

```python
def estimate_alpha(observations):
    print('your code here')
```



We know sand is distributed as a pareto with PDF

$$f(x) = \frac{\alpha}{x^{\alpha+1}}$$

$$\alpha_{\text{mle}} = \frac{n}{\sum_i \log x_i}$$

MLE for a Pareto

Consider I.I.D. random variables $X_1, X_2, \ldots, X_n$

- $X_i \sim \text{Pareto}(\alpha)$. **Use Maximum Likelihood to estimate α.**

1. What is the likelihood of all the *data*

2. What is the log-likelihood all the *data*

3. Find the value of α which maximizes log likelihood

MLE for a Pareto

Consider I.I.D. random variables $X_1, X_2, \ldots, X_n$

- $X_i \sim \text{Pareto}(\alpha)$. **Use Maximum Likelihood to estimate α.**
- Likelihood:

$$L(\alpha) = \prod_{i=1}^{n} \frac{\alpha}{x_i^{\alpha+1}}$$

2. What is the log-likelihood all the *data*

3. Find the value of α which maximizes log likelihood

MLE for a Pareto

Consider I.I.D. random variables $X_1, X_2, \ldots, X_n$

- $X_i \sim$ Pareto(α). **Use Maximum Likelihood to estimate α.**
- Likelihood:

$$L(\alpha) = \prod_{i=1}^{n} \frac{\alpha}{x_i^{\alpha+1}}$$

- Log-likelihood:

$$LL(\alpha) = \sum_{i=1}^{n} \log \alpha - (\alpha + 1) \log x_i = n \log \alpha - (\alpha + 1) \sum_{i=1}^{n} \log x_i$$

3. Find the value of **α** which maximizes log likelihood

MLE for a Pareto

Consider I.I.D. random variables $X_1, X_2, \ldots, X_n$

- $X_i \sim \text{Pareto}(\alpha)$. **Use Maximum Likelihood to estimate α.**
- Likelihood:

$$L(\alpha) = \prod_{i=1}^{n} \frac{\alpha}{x_i^{\alpha+1}}$$

- Log-likelihood:

$$LL(\alpha) = \sum_{i=1}^{n} \log \alpha - (\alpha+1)\log x_i = n \log \alpha - (\alpha+1)\sum_{i=1}^{n} \log x_i$$

- Chose α to be the argmax of LL:

$$\frac{\partial LL(\alpha)}{\partial \alpha} = \frac{n}{\alpha} - \sum_{i=1}^{n} \log x_i$$

Argmax Option #1: set the derivative to 0, and solve for alpha

End Review

Two Issues with MLE to resolve

1

2

We need a more general argmax

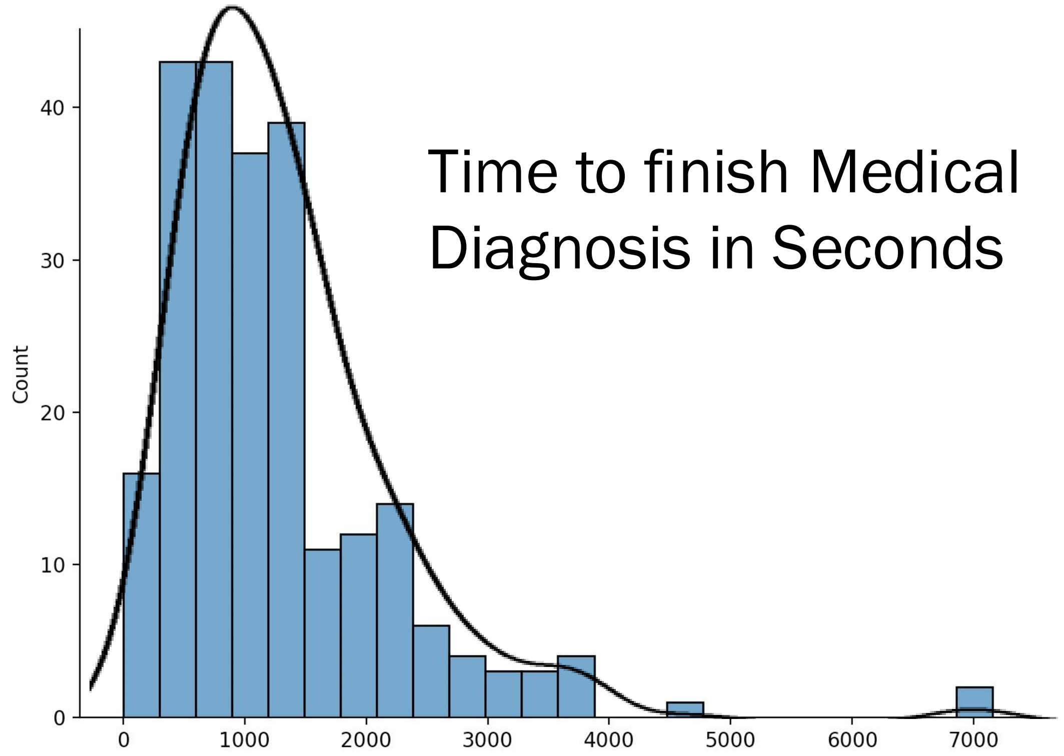
Its not always possible to set a derivative to 0 and solve

How can we incorporate prior beliefs about our parameters?

Inference vs MLE

MLE for Erlang

```
[3.002, 0.983, 2.186, 1.624, 3.997, 1.777,
2.809, 0.42, 0.515, 1.582, 0.948, 0.458, 1.
066, 0.8, 2.398, 0.794, 2.561, 2.61, 0.
595, 3.897, 1.852, 1.182, 3.043, 0.905, 1.
45, 0.405, 0.445, 2.103, 1.425, 3.12, 0.
973, 1.056, 3.715, 2.952, 1.817, 2.686, 4.
173, 0.358, 2.185, 2.581, 7.134, 0.206, 2.
049, 0.896, 2.095, 4.39, 2.199, 3.434, 5.
696, 0.819, 0.416, 1.571, 1.337, 2.79, 2.
701, 3.061, 4.677, 0.671, 1.594, 3.586, 2.
708, 1.417, 1.799, 1.137, 1.771, 2.12, 0.
93, 6.835, 3.213, 2.541, 2.505, 1.257, 1.
99, 1.5, 0.014, 3.856, 0.979, 2.413, 2.
596, 1.653, 0.881, 4.457, 0.717, 3.305, 2.
456, 3.462, 1.737, 0.968, 0.528, 0.18, 1.
626, 2.224, 1.466, 1.6, 1.572, 0.12, 2.86,
1.062, 2.139, 1.217]
```

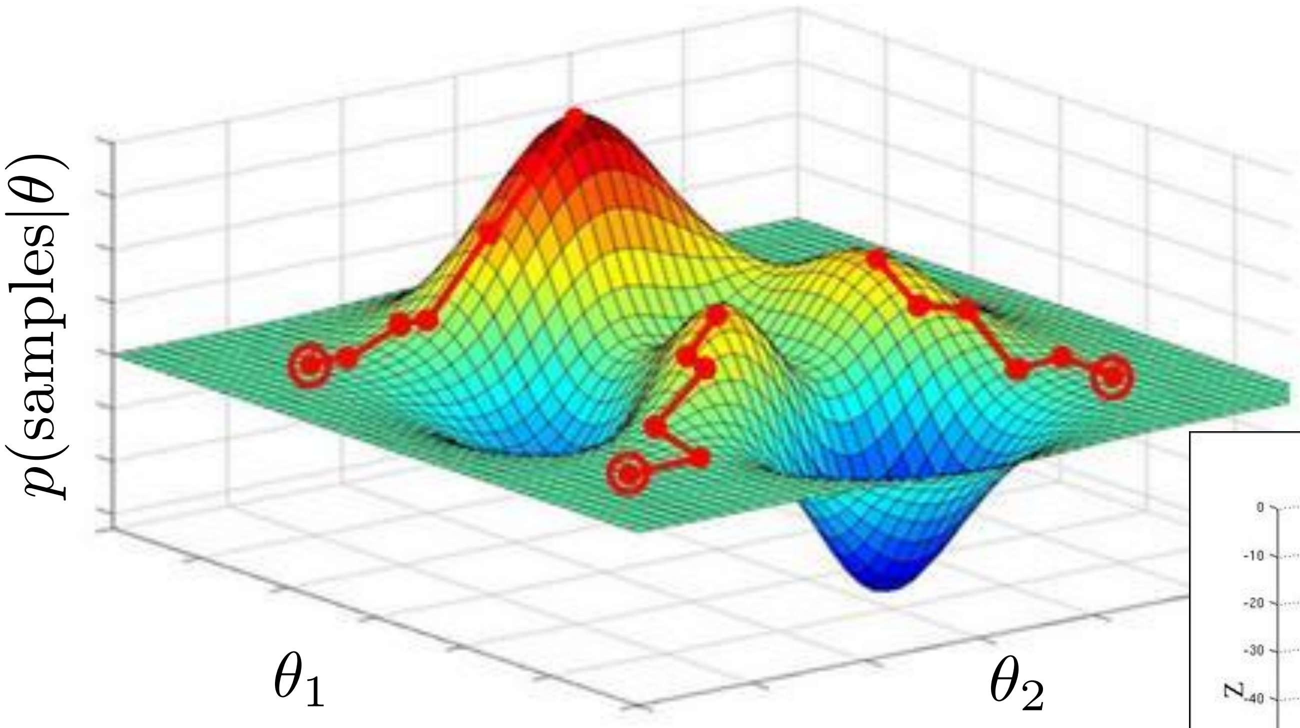


Time to finish Medical Diagnosis in Seconds

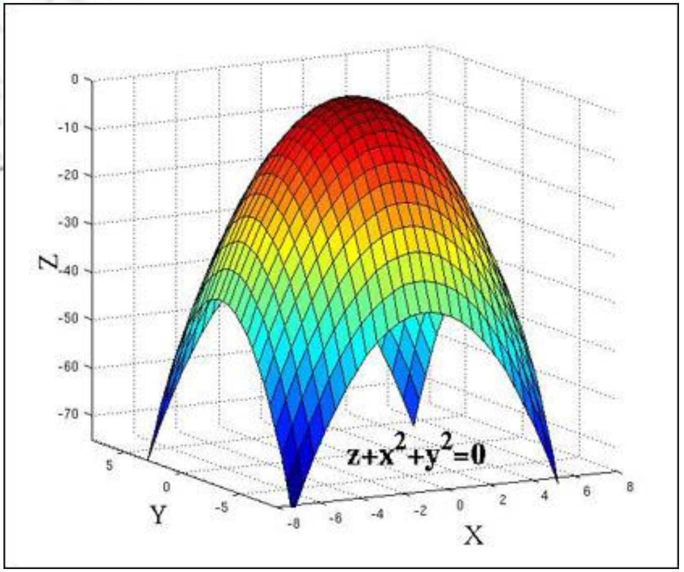
$$f(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{\Gamma(k)}$$


arg max

Gradient Ascent



Especially good if function is convex



Walk uphill and you will find a local maxima
(if your step size is small enough)

Gradient Ascent

Repeat many times

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{ old}})}{\partial \theta_j^{\text{old}}}$$

Step size constant

This is some **profound** life philosophy

Walk uphill and you will find a local maxima
(if your step size is small enough)

Gradient Ascent

`Initialize:` θ_j `= random for all 0` $\leq j \leq m$

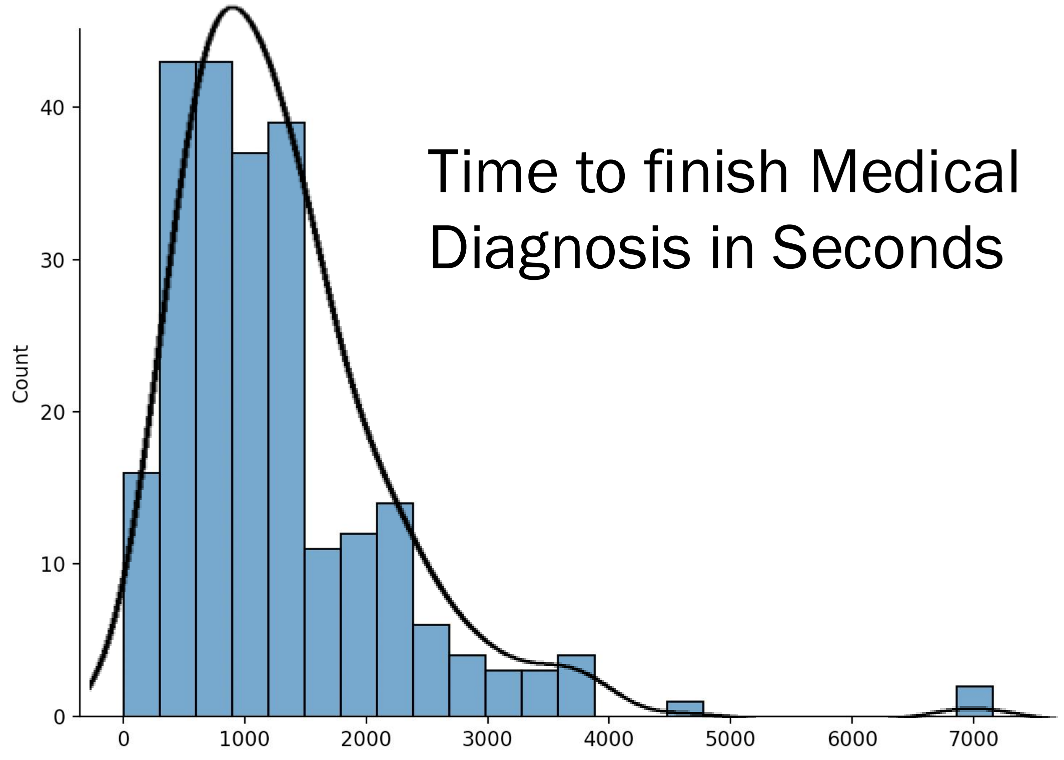
`Repeat many times:`

Calculate all `gradient[j]`*'s based on data*

θ_j `+=` η `* gradient[j] for all 0` $\leq j \leq m$

MLE of Erlang

```
[3.002, 0.983, 2.186, 1.624, 3.997, 1.777,
2.809, 0.42, 0.515, 1.582, 0.948, 0.458, 1.
066, 0.8, 2.398, 0.794, 2.561, 2.61, 0.
595, 3.897, 1.852, 1.182, 3.043, 0.905, 1.
45, 0.405, 0.445, 2.103, 1.425, 3.12, 0.
973, 1.056, 3.715, 2.952, 1.817, 2.686, 4.
173, 0.358, 2.185, 2.581, 7.134, 0.206, 2.
049, 0.896, 2.095, 4.39, 2.199, 3.434, 5.
696, 0.819, 0.416, 1.571, 1.337, 2.79, 2.
701, 3.061, 4.677, 0.671, 1.594, 3.586, 2.
708, 1.417, 1.799, 1.137, 1.771, 2.12, 0.
93, 6.835, 3.213, 2.541, 2.505, 1.257, 1.
99, 1.5, 0.014, 3.856, 0.979, 2.413, 2.
596, 1.653, 0.881, 4.457, 0.717, 3.305, 2.
456, 3.462, 1.737, 0.968, 0.528, 0.18, 1.
626, 2.224, 1.466, 1.6, 1.572, 0.12, 2.86,
1.062, 2.139, 1.217]
```



$$f(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{\Gamma(k)}$$

To the code!

Gradient Ascent for MLE of Erlang

```python
def fit_erlang(observations, n_iterations=10, initial_step_size=0.0001):
    step_size = initial_step_size
    k = 1.0  # Initial guess for shape parameter
    lambda_ = 2.0  # Initial guess for rate parameter
    n = len(observations)

    for i in range(n_iterations):
        # To debug: Calculate log-likelihood
        # ll = calc_log_likelihood_erlang(observations, k, lambda_)
        # print(f"Log-Likelihood at iteration {k, lambda_}: {ll}")

        # Calculate gradients
        # gradient_lambda = (n * k / lambda_) - sum(observations)
        gradient_lambda = 0
        for x_i in observations:
            gradient_lambda += k / lambda_ - x_i
        gradient_k = n * math.log(lambda_) + sum(math.log(x) for x in observations) - n * digamma(k)

        # Update parameters
        lambda_ += step_size * gradient_lambda
        k += step_size * gradient_k

    return k, lambda_
```

Derived these partial derivatives of log likelihood

Two Issues with MLE to resolve

1

2

We need a more general argmax

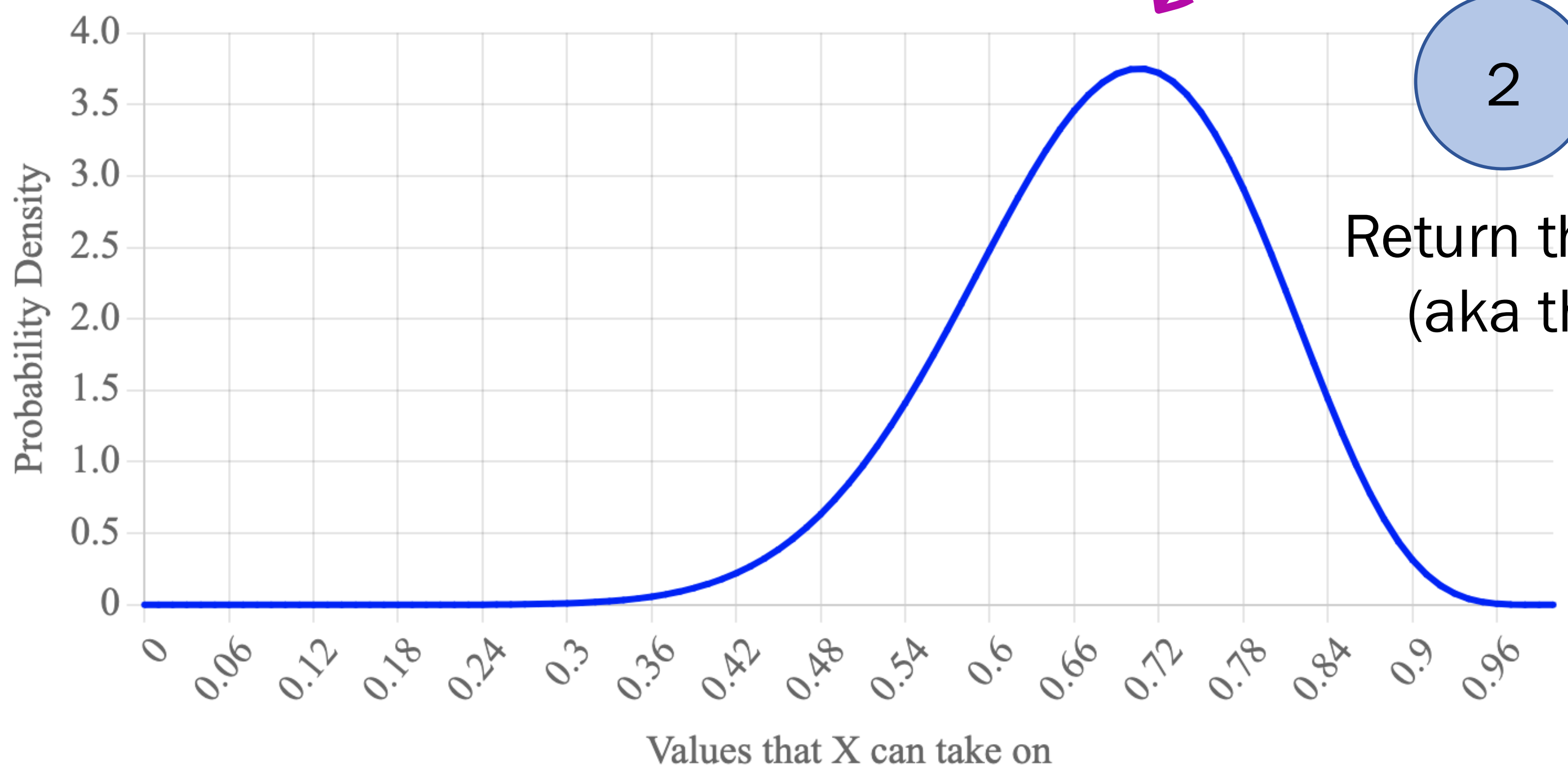
Its not always possible to set a derivative to 0 and solve

How can we incorporate prior beliefs about our parameters?

Inference vs MLE

Maximum A Posteriori (MAP)

1 Infer the posterior belief in params



2 Return the argmax (aka the mode)

MLE vs MAP

Data: $x^{(1)}, \ldots, x^{(n)}$

Maximum Likelihood Estimation

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\ f(x^{(1)}, \ldots, x^{(n)} | \theta)$$
$$= \underset{\theta}{\operatorname{argmax}} \left(\sum_i \log f(x^{(i)} | \theta) \right)$$

Maximum A Posteriori

This is your prior on params

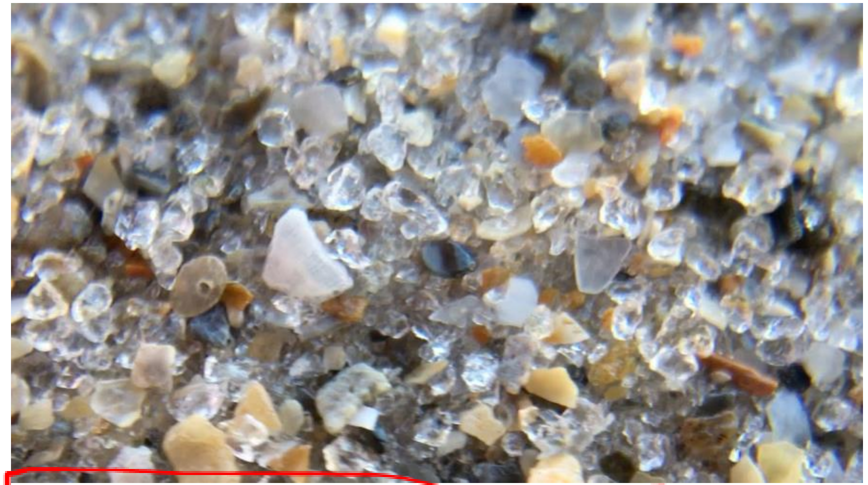
$$\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\ f(\theta | x^{(1)}, \ldots, x^{(n)})$$
$$= \underset{\theta}{\operatorname{argmax}} \left(\log f(\theta) + \sum_i \log f(X^{(i)} = x^{(i)} | \theta) \right)$$

In case you are curious:

MLE for a Pareto

```python
observations = [1.677, 3.812, 1.463, 2.641, 1.256, 1.678, 1.157,
1.146, 1.323, 1.029, 1.238, 1.018, 1.171, 1.123, 1.074, 1.652,
1.873, 1.314, 1.309, 3.325, 1.045, 2.271, 1.305, 1.277, 1.114,
1.391, 3.728, 1.405, 1.054, 2.789, 1.019, 1.218, 1.033, 1.362,
1.058, 2.037, 1.171, 1.457, 1.518, 1.117, 1.153, 2.257, 1.022,
1.839, 1.706, 1.139, 1.501, 1.238, 2.53 , 1.414, 1.064, 1.097,
1.261, 1.784, 1.196, 1.169, 2.101, 1.132, 1.193, 1.239, 1.518,
2.764, 1.053, 1.267, 1.015, 1.789, 1.099, 1.25 , 1.253, 1.418,
1.494, 1.015, 1.459, 2.175, 2.044, 1.551, 4.095, 1.396, 1.262,
1.351, 1.121, 1.196, 1.391, 1.305, 1.141, 1.157, 1.155, 1.103,
1.048, 1.918, 1.889, 1.068, 1.811, 1.198, 1.361, 1.261, 4.093,
2.925, 1.133, 1.573]
```

```python
def estimate_alpha(observations):
    print('your code here')
```



We know sand is distributed as a pareto with PDF

$$f(x) = \frac{\alpha}{x^{\alpha+1}}$$

Prior: $\alpha \sim N(\mu = 2.5, \sigma^2 = 3)$

MAP for Pareto

Prior: $\alpha \sim N(\mu = 2.5, \sigma^2 = 3)$

- X_i ~ Pareto(α). **Use MAP to estimate α.**
- MAP function:

$$= \log g(\alpha) + n \log \alpha - (\alpha + 1) \sum_{i=1}^{n} \log x_i$$

Where g is the prior likelihood

$$= \log \frac{1}{\sqrt{3}\sqrt{2\pi}} e^{\frac{-(\alpha-2)^2}{6}} + n \log \alpha - (\alpha + 1) \sum_{i=1}^{n} \log x_i$$

$$= K + \frac{-(\alpha - 2)^2}{6} + n \log \alpha - (\alpha + 1) \sum_{i=1}^{n} \log x_i$$

- Choose α which is the argmax of this function

$$\frac{\partial \text{MAP}(\alpha)}{\partial \alpha} = -2\alpha + 4 + \frac{n}{\alpha} - \sum_{i=1}^{n} \log x_i$$

You need to know MLE and Inference.
MAP is something you should recognize!

Two Issues with MLE to resolve

1

2

We need a more general argmax

Its not always possible to set a derivative to 0 and solve

How can we incorporate prior beliefs about our parameters?

Inference vs MLE

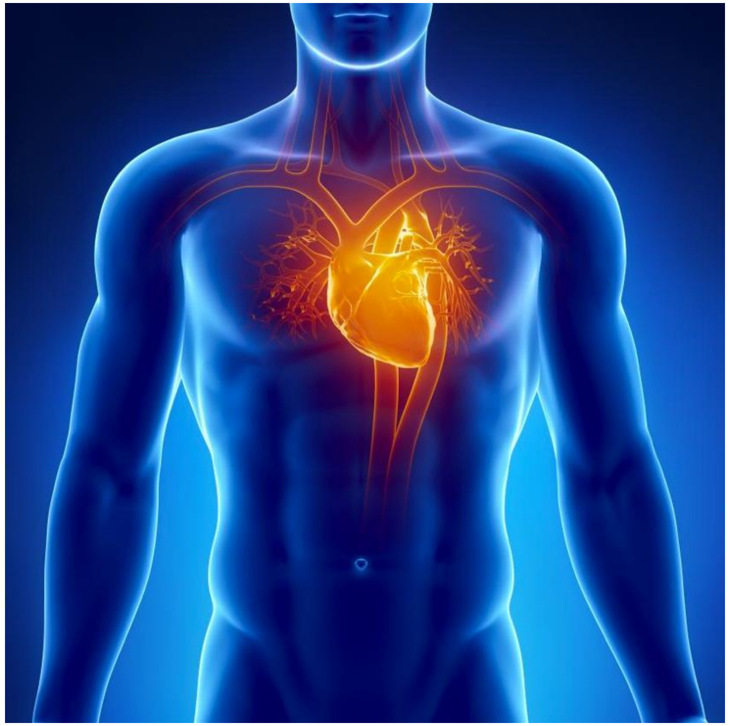
It is time….

Today a very special MLE problem

Classification

Example Datasets

Heart



Ancestry

23andMe

Netflix

NETFLIX

Training Data

Training Data: assignments all random variables **X** and Y

Assume IID data:

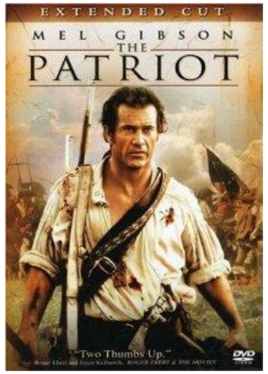
n training datapoints

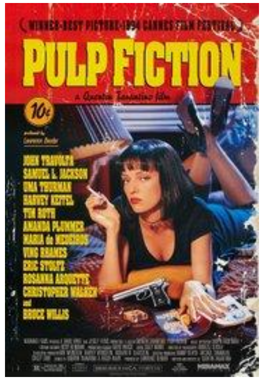
$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots (\mathbf{x}^{(n)}, y^{(n)})$$

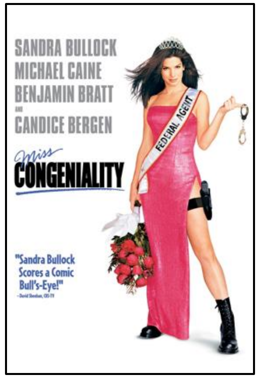
$$m = \left|\mathbf{x}^{(i)}\right|$$

Each datapoint has m features and a single output

Single Feature Value



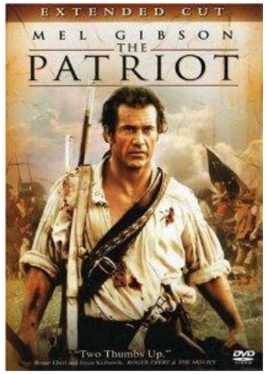


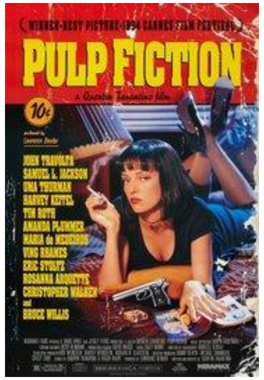


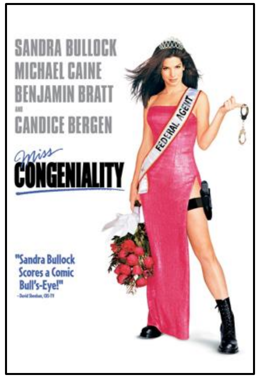
	Movie 1	Movie 2	…	Movie m	Output
User 1	1	0		1	1
User 2	1	1		0	0
			⋮		⋮
User n	0	0		1	1

$$(\mathbf{x}^{(i)}, y^{(i)}) \text{ such that } 1 \le i \le n$$

Single Feature Value



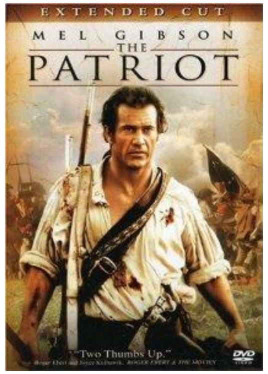


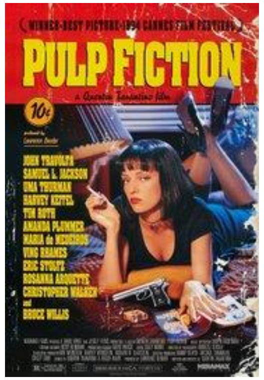


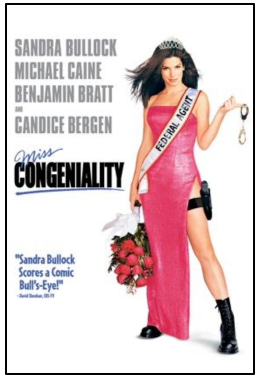
	Movie 1	Movie 2	...	Movie m	Output
User 1	1	0		1	1
User 2	1	1		0	0
⋮			⋮		⋮
User n	0	0		1	1

$$(\mathbf{x}^{(i)}, y^{(i)}) \text{ such that } 1 \leq i \leq n$$

Single Feature Value





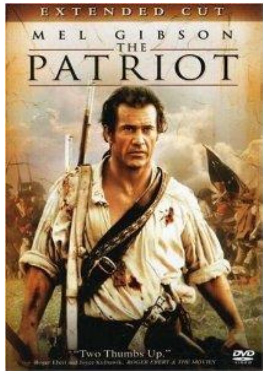


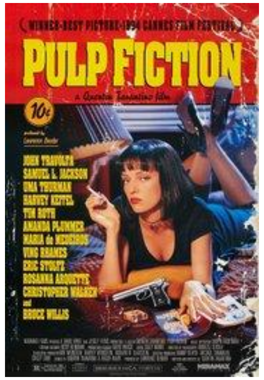
	Movie 1	Movie 2	...	Movie m	Output
User 1	1	0		1	1
User 2	1	1		0	0
⋮			⋮		⋮
User n	0	0		1	1

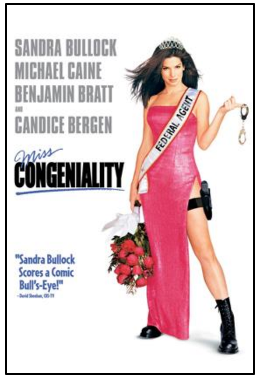
$$(\boxed{\mathbf{x}^{(i)}}, y^{(i)}) \text{ such that } 1 \leq i \leq n$$

Stanford University

Single Feature Value



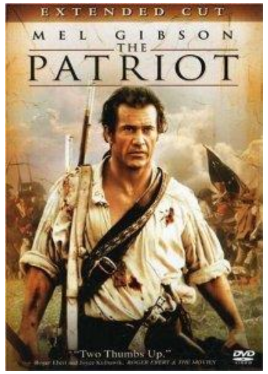


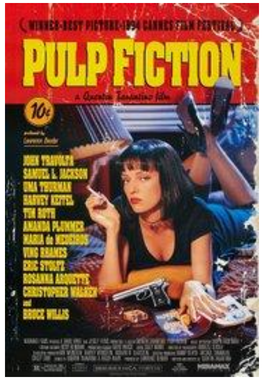


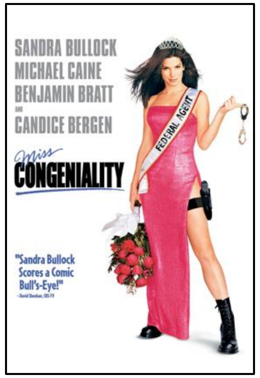
	Movie 1	Movie 2	...	Movie m	Output
User 1	1	0		1	1
User 2	1	1		0	$\boxed{0}$
⋮		⋮			⋮
User n	0	0		1	1

$$(\mathbf{x}^{(i)}, \boxed{y^{(i)}}) \text{ such that } 1 \leq i \leq n$$

Single Feature Value



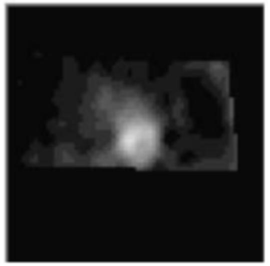
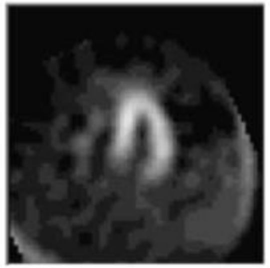




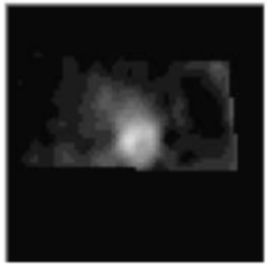
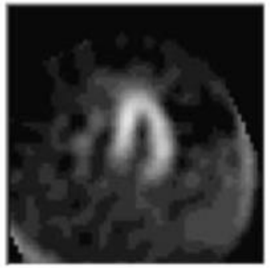
	Movie 1	Movie 2	…	Movie *m*	Output
User 1	1	0		1	1
User 2	1	1		$\boxed{0}$	0
		⋮			⋮
User *n*	0	0		1	1

In general: $\mathbf{x}_j^{(i)}$ In this case: $\mathbf{x}_m^{(2)}$

Healthy Heart Classifier

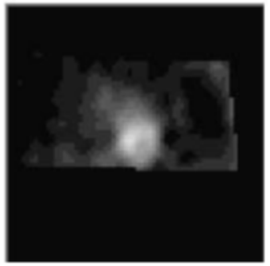
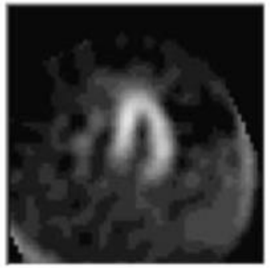
	ROI 1	ROI 2		ROI *m*	Output
			...		
Heart 1	0	1		1	0
Heart 2	1	1		1	0
			⋮		⋮
Heart *n*	0	0		0	1

Healthy Heart Classifier

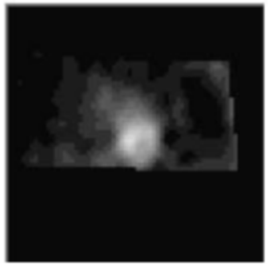
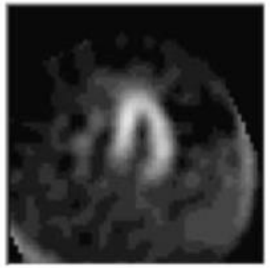



	ROI 1	ROI 2	...	ROI m	Output	
Heart 1	0	1		1	0	
Heart 2	1	1		1	0	$(\mathbf{x}^{(2)}, y^{(2)})$
			⋮		⋮	
Heart n	0	0		0	1	

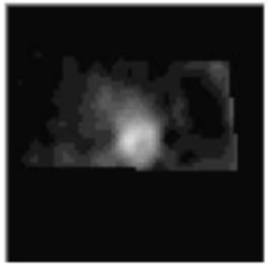
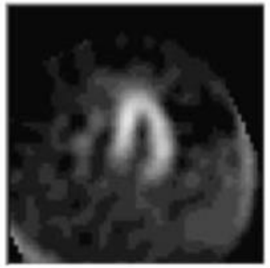
Healthy Heart Classifier

	ROI 1	ROI 2		ROI *m*	Output
			...		
Heart 1	0	1		1	0
Heart 2	1	1	$\mathbf{x}^{(2)}$	1	0
			⋮		⋮
Heart *n*	0	0		0	1

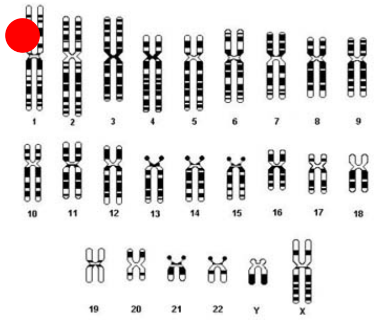
Healthy Heart Classifier

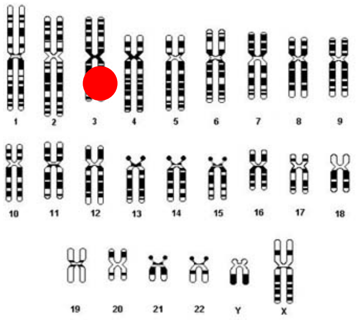
	ROI 1	ROI 2		ROI *m*	Output
			...		
Heart 1	0	1		1	0
Heart 2	1	1		1	$\boxed{0}$ $y^{(2)}$
			⋮		⋮
Heart *n*	0	0		0	1

Healthy Heart Classifier

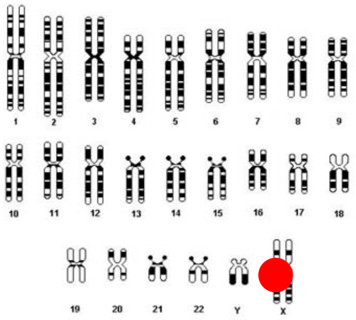
	ROI 1	ROI 2		ROI *m*	Output
			…		
Heart 1	0	$\boxed{1}$ $x_2^{(1)}$		1	0
Heart 2	1	1		1	0
			⋮		⋮
Heart *n*	0	0		0	1

Ancestry Classifier





…



	SNP 1	SNP 2	SNP *m*	Output
User 1	1	0	1	0
User 2	0	0	1	1
⋮				⋮
User *n*	1	1	0	1

Regression: Predicting Real Numbers

...

Points

	Opposing team ELO	Points in last game	...	At Home?	Output
Game 1	84	105		1	120
Game 2	90	102		0	95
		⋮			⋮
Game *n*	74	120		0	115

Training Data

Training Data: assignments all random variables X and Y

Assume IID data:

n training datapoints

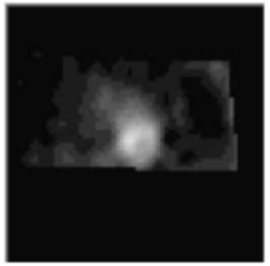
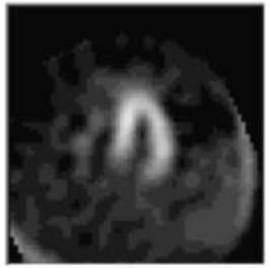
$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots (\mathbf{x}^{(n)}, y^{(n)})$$

$$m = |\mathbf{x}^{(i)}|$$

Each datapoint has m features and a single output

Classification

Healthy Heart Classifier

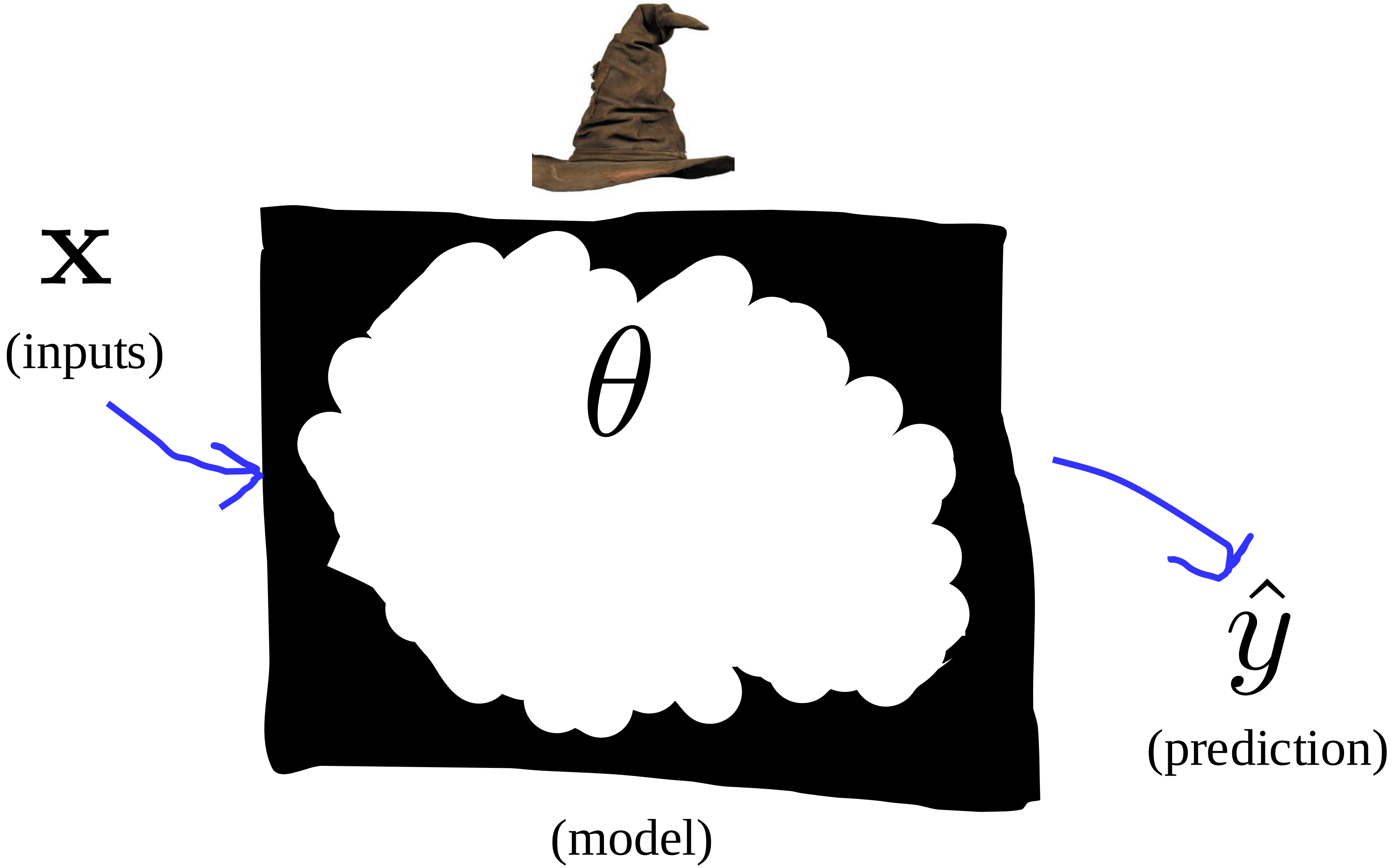
	ROI 1	ROI 2		ROI *m*	Output
			...		
Heart 1	0	1		1	0
Heart 2	1	1		1	0
			⋮		⋮
Heart *n*	0	0		0	1

Classification is Building a Harry Potter Hat

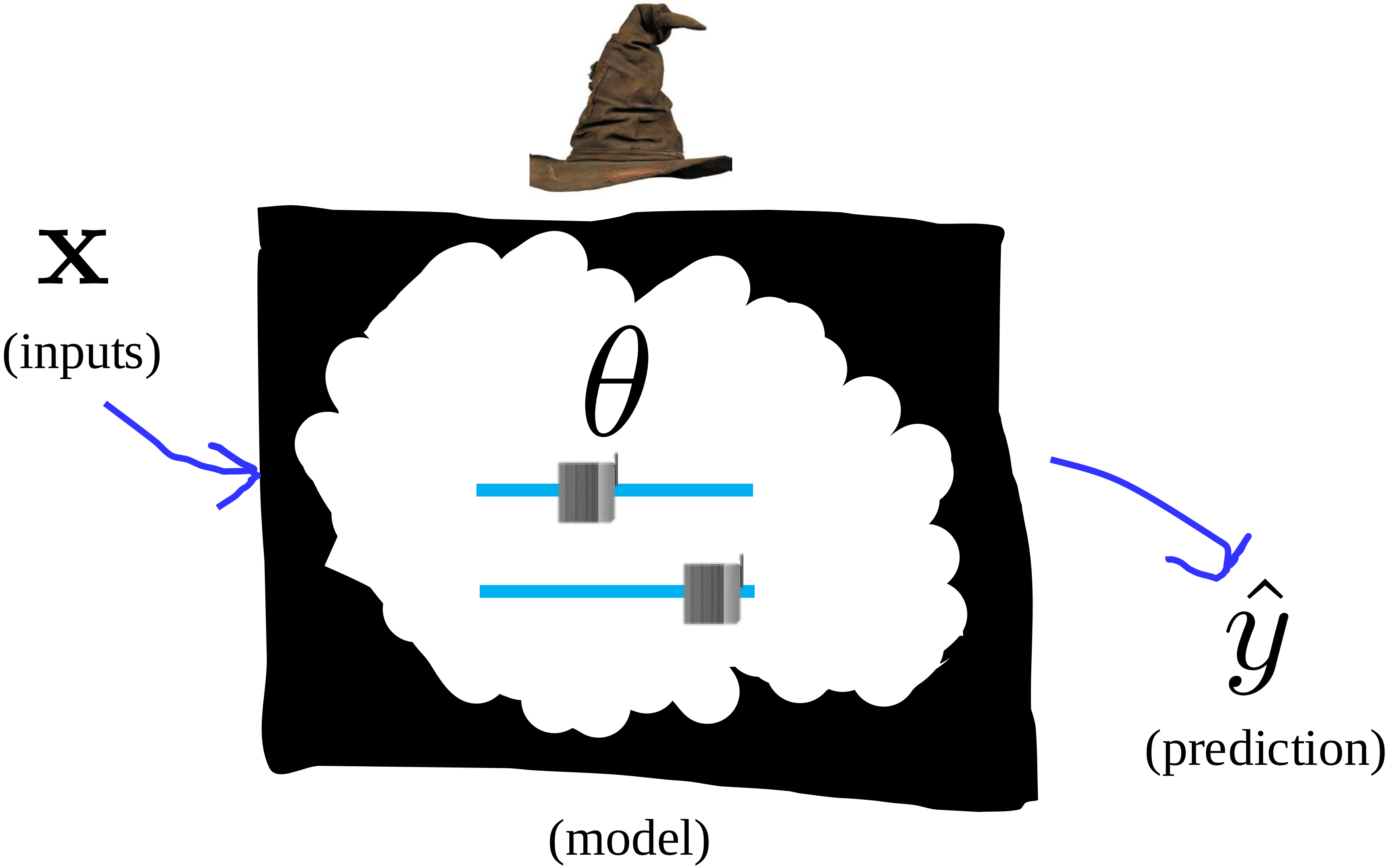
$$\hat{y} = 1$$

$$\mathbf{x} = [0, 1, \ldots, 1]$$

Machine Learning for Classification



Machine Learning for Classification



Logistic Regression

Chapter 1: Big Picture

From Naïve Bayes to Logistic Regression

In classification we care about $P(Y = 1|\mathbf{X} = \mathbf{x})$

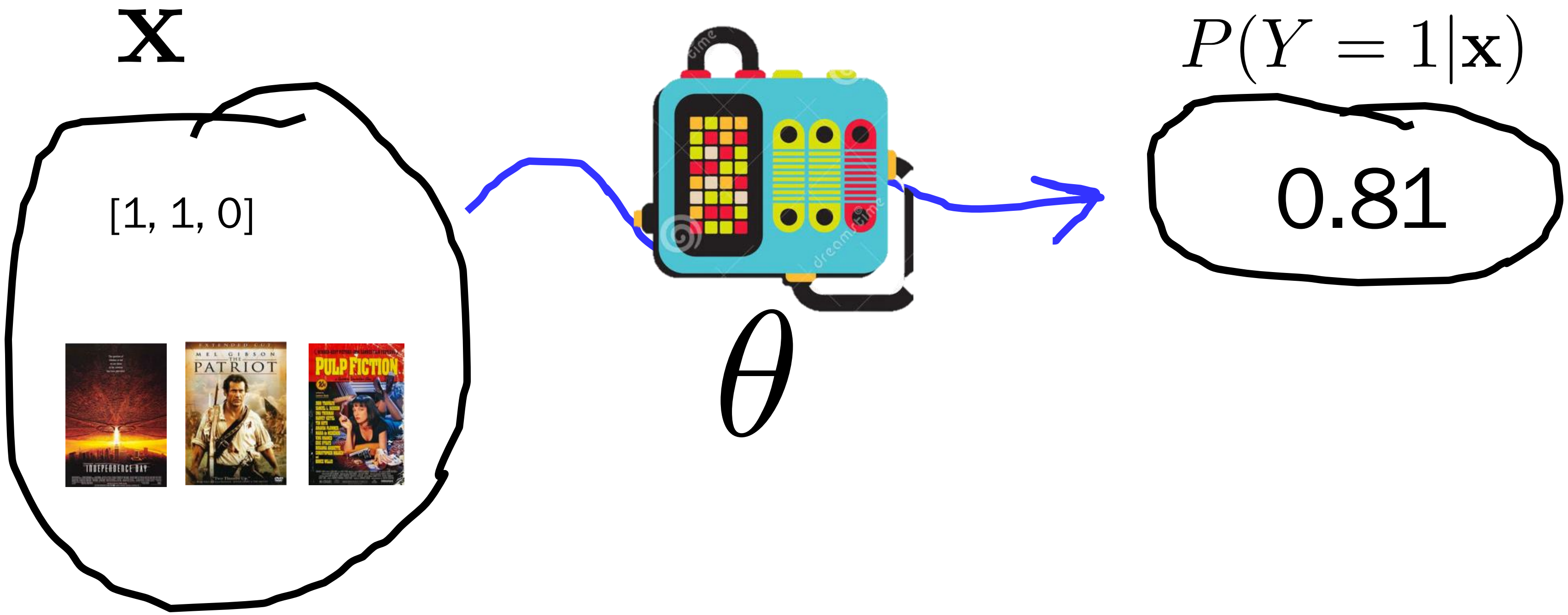
Lets build a machine that can you can put $\mathbf{x}$ into, which then spits out

$$P(Y = 1|\mathbf{X} = \mathbf{x})$$

Logistic Regression Assumption

Could we compute $P(Y = 1|\mathbf{X} = \mathbf{x})$ via a machine?

Welcome our friend: logistic regression!



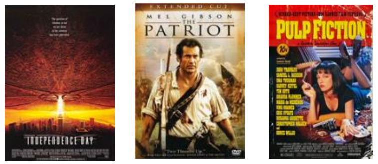
Logistic Regression Assumption

Could we compute $P(Y = 1|\mathbf{X} = \mathbf{x})$ via a machine?

Welcome our friend: logistic regression!

$\mathbf{x}$

[1, 1, 0]



θ

$P(Y = 1|\mathbf{x})$

0.81

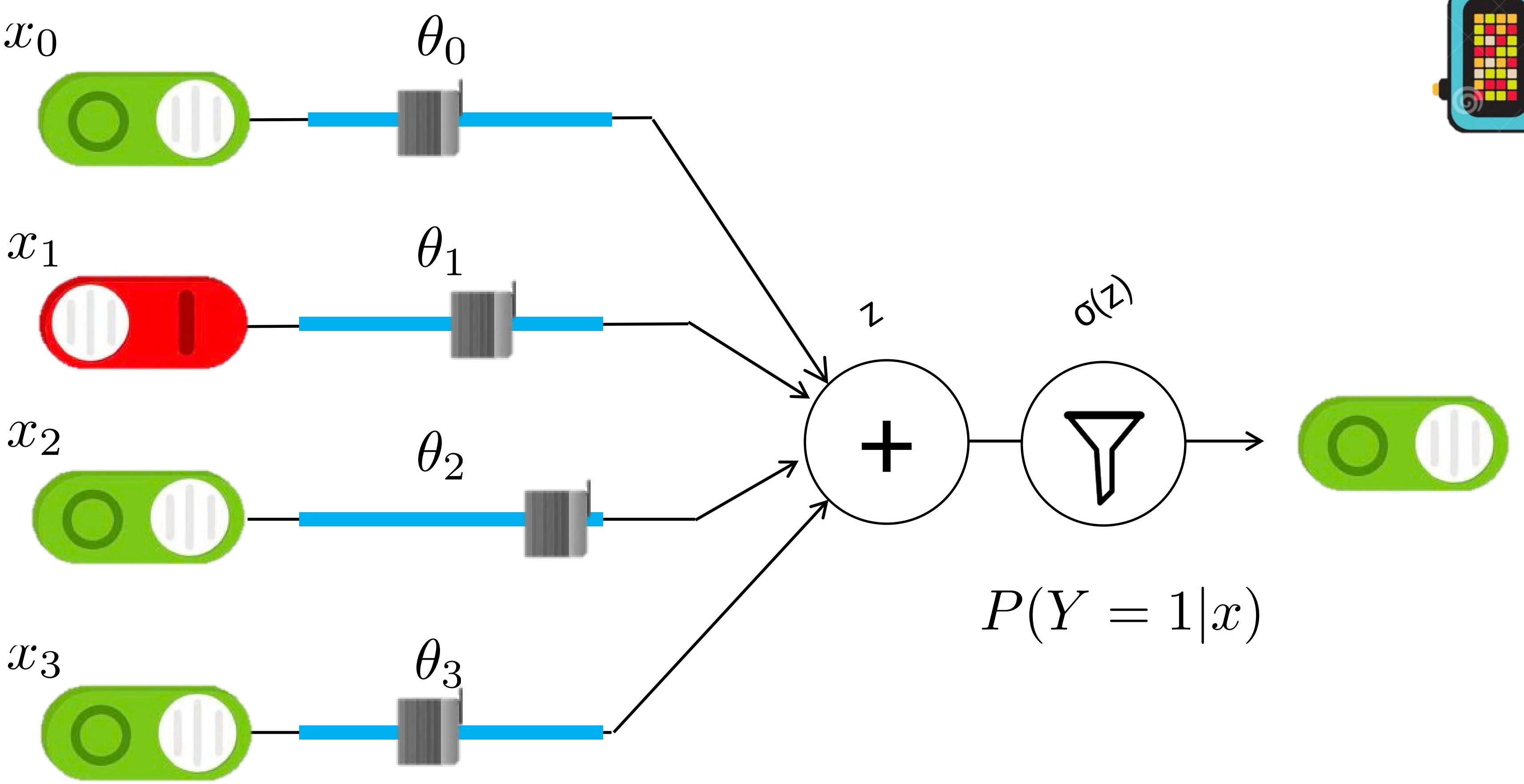
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Logistic Regression Assumption

$$P(Y = 1 | \mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Stanford University

Logistic Regression

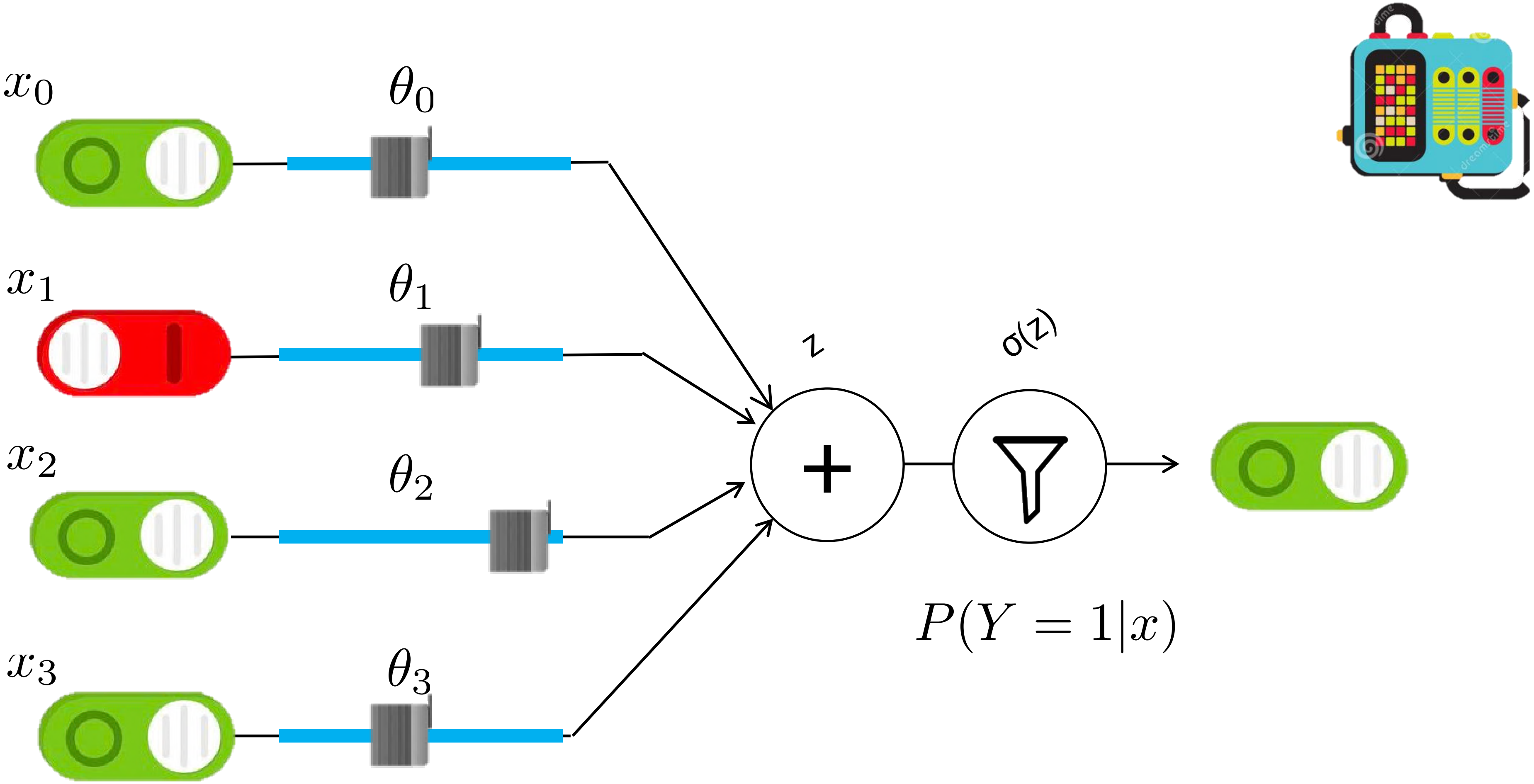


$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Logistic Regression

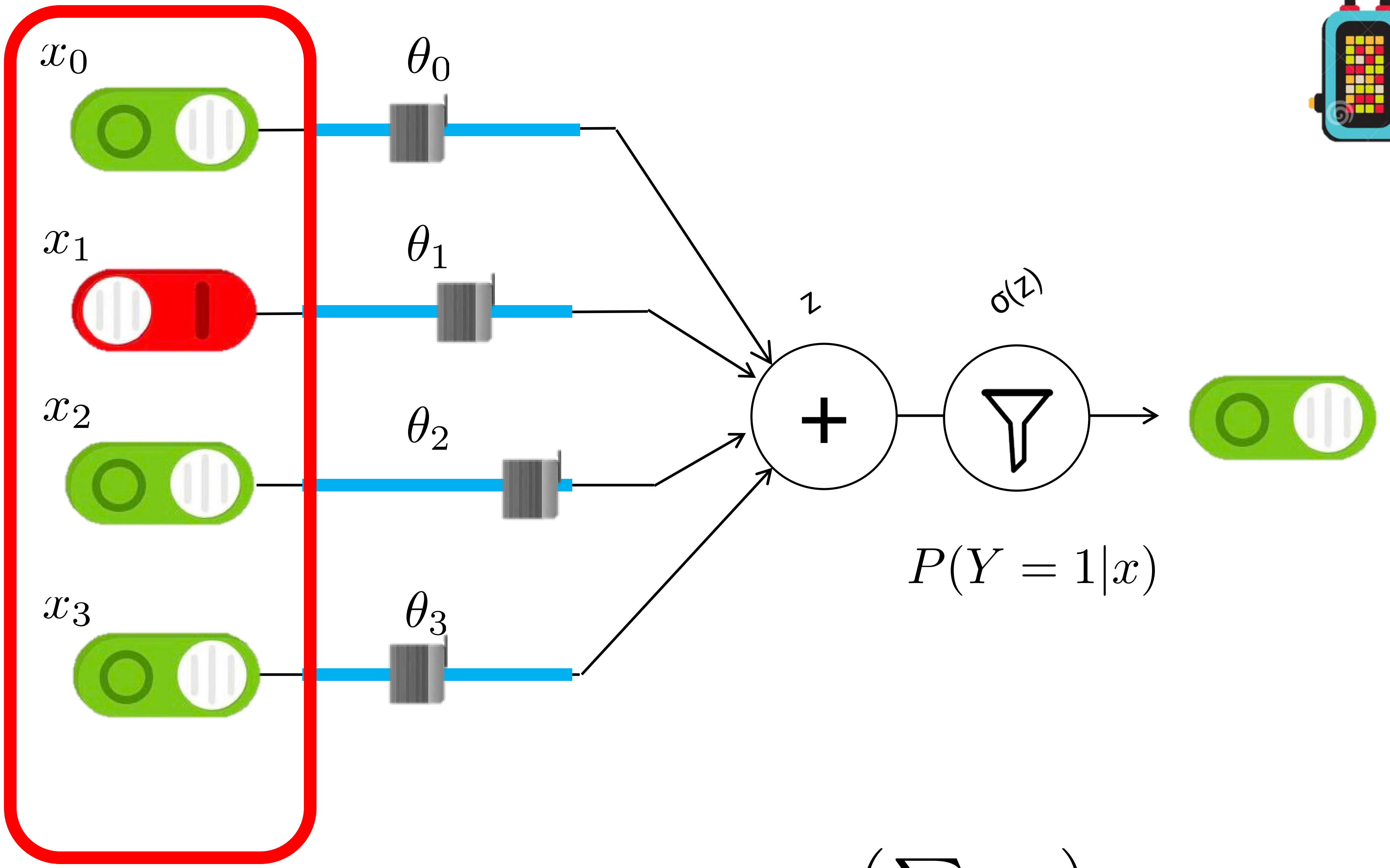
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Logistic Regression Cartoon



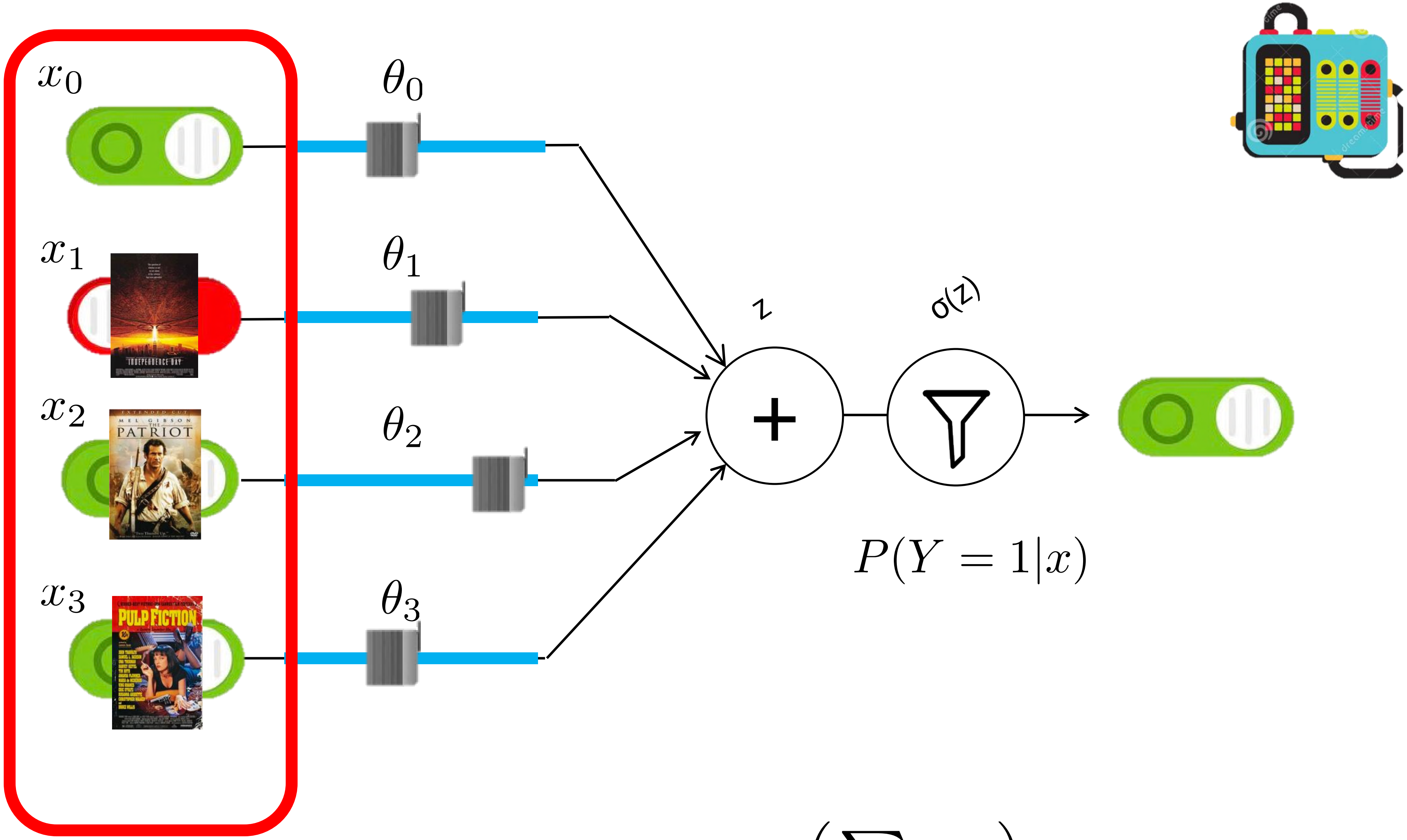
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Inputs x = [0, 1, 1]



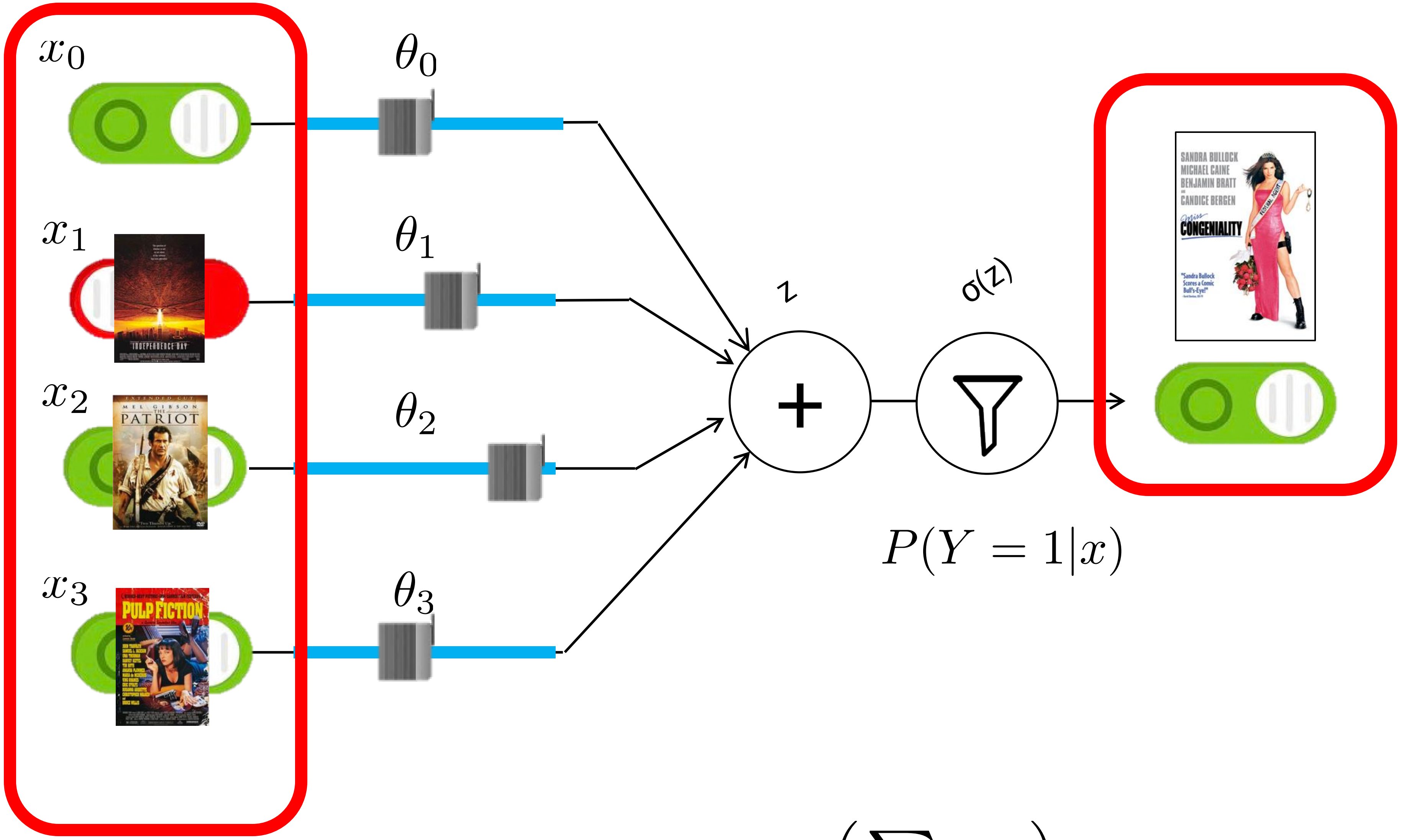
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Inputs



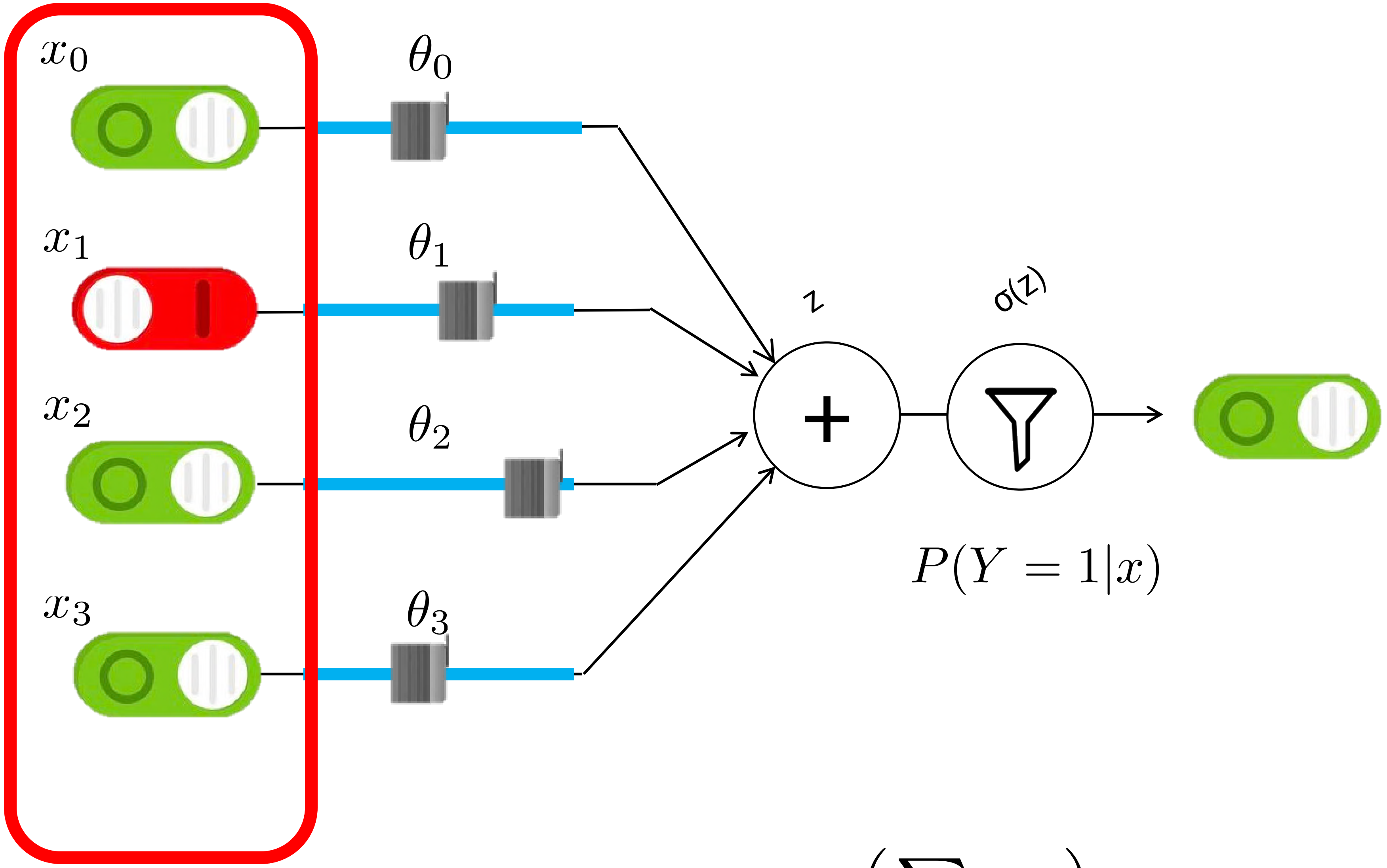
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Inputs + Output



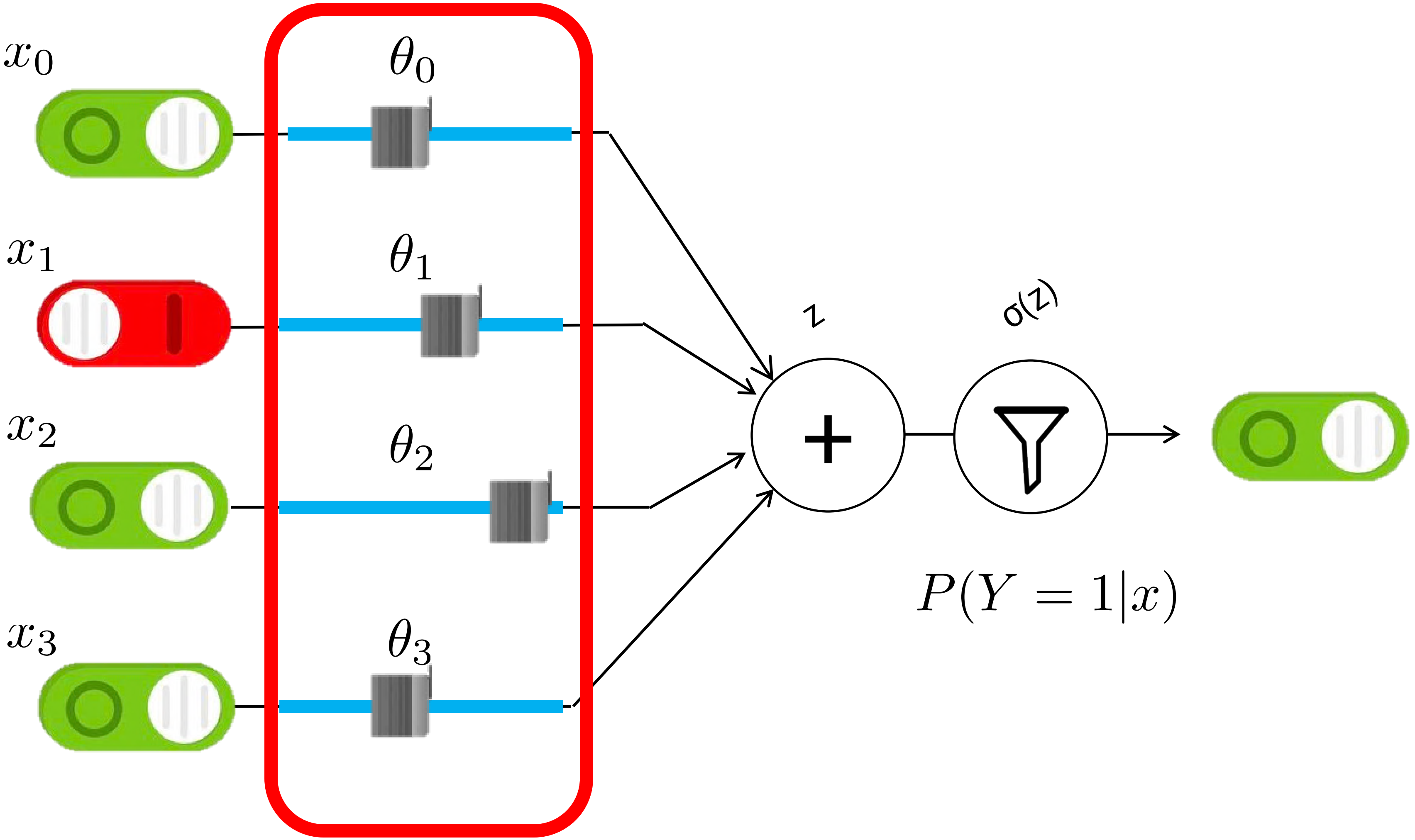
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Inputs



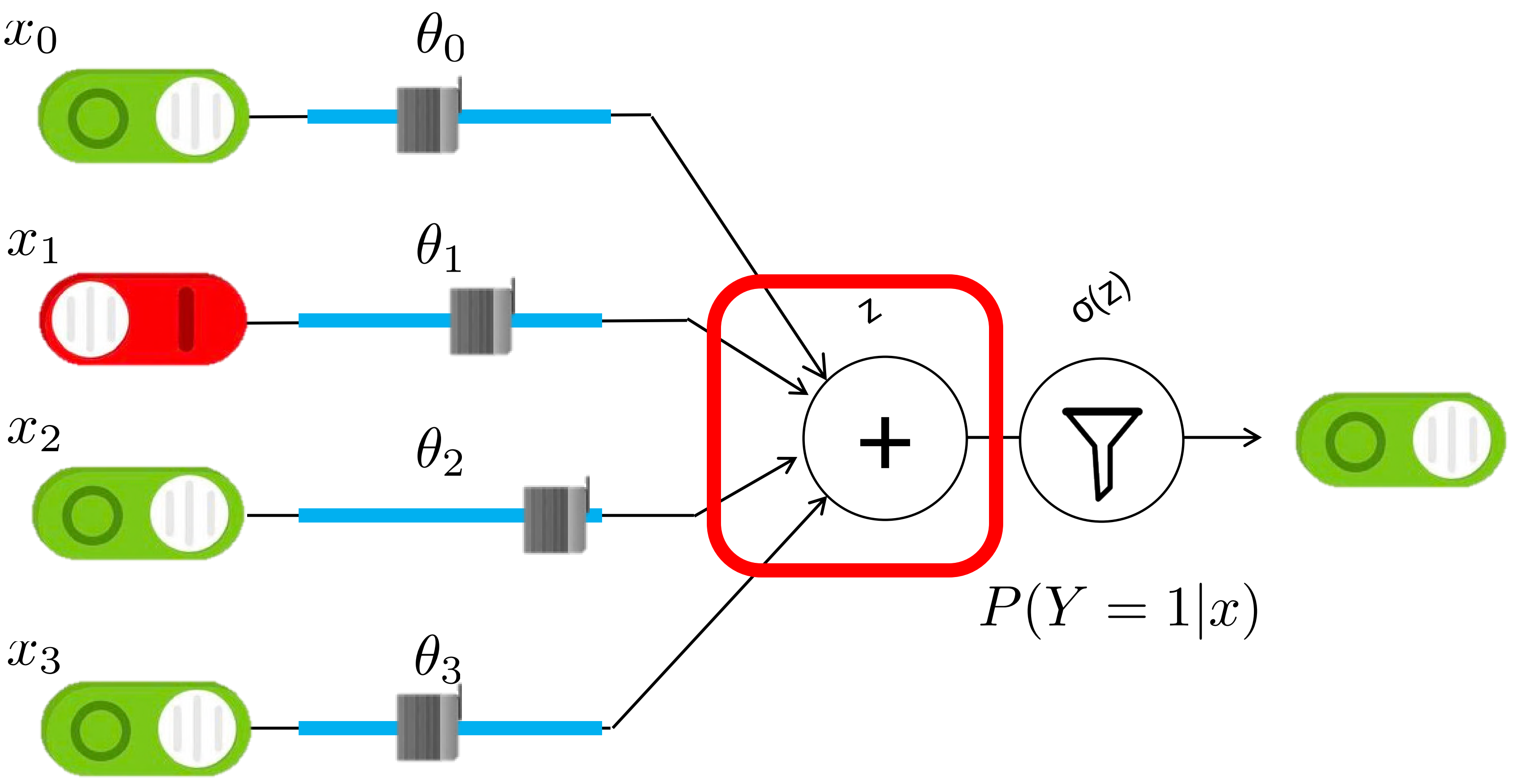
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Weights



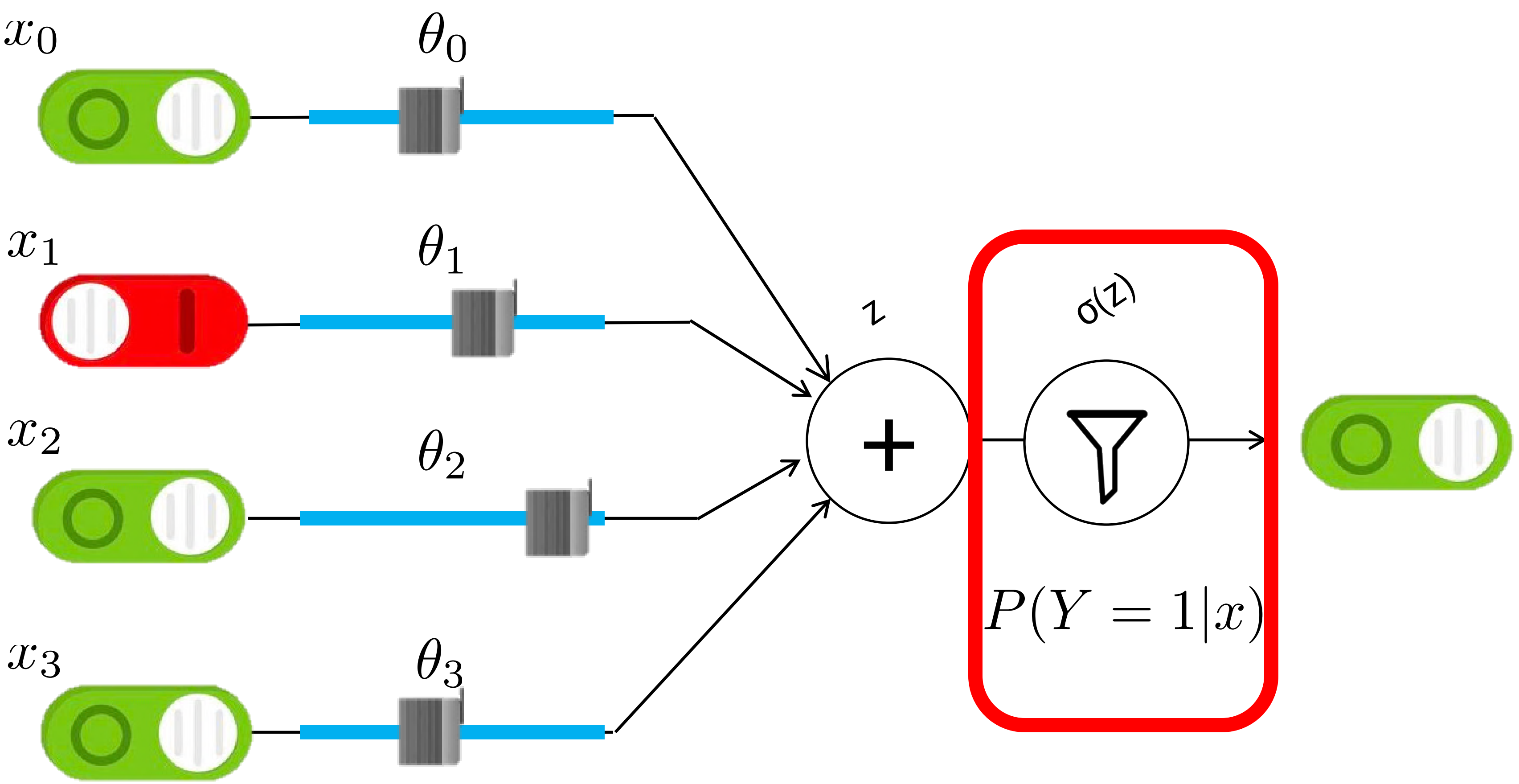
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Weighed Sum



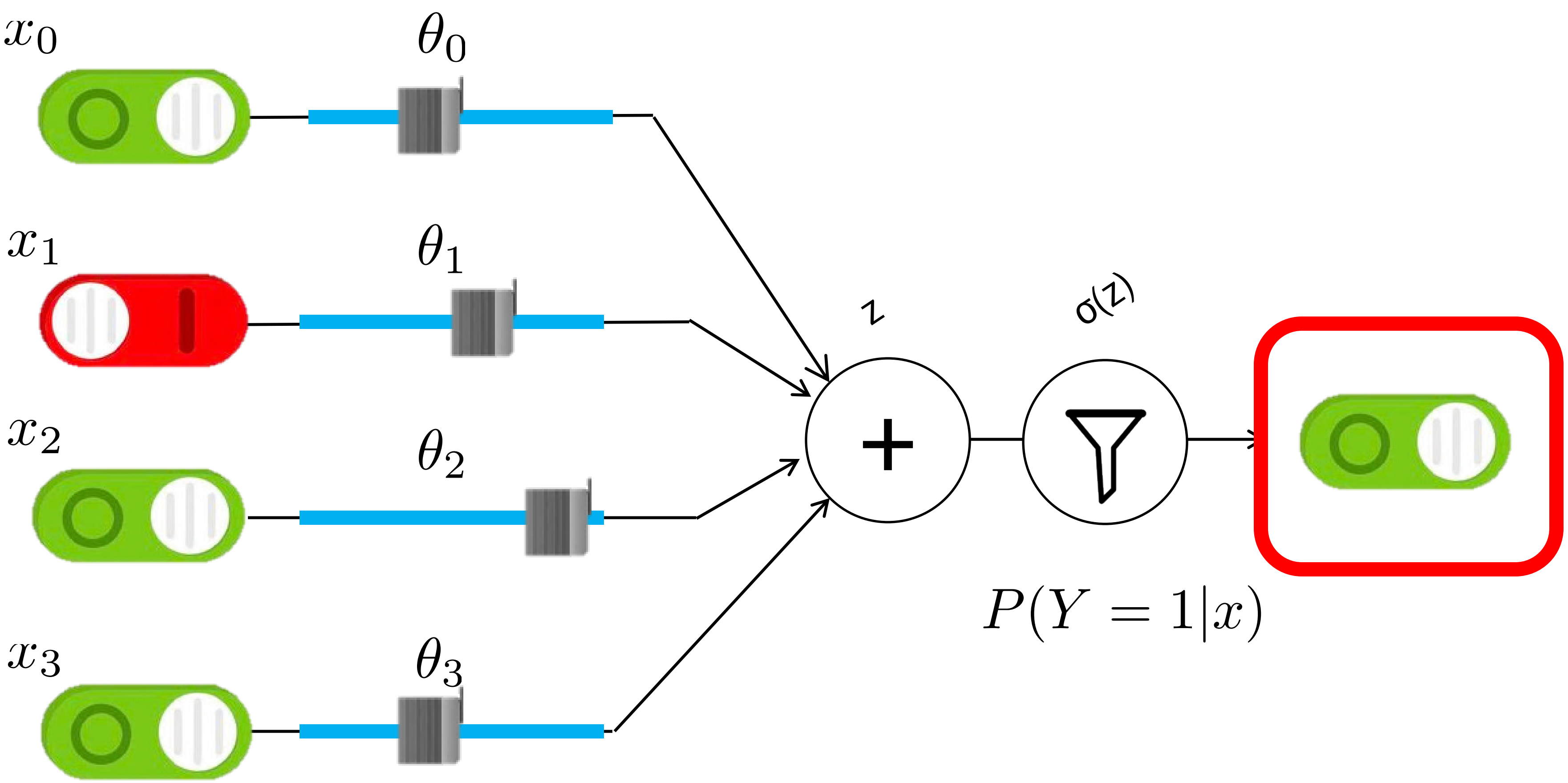
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Squashing Function



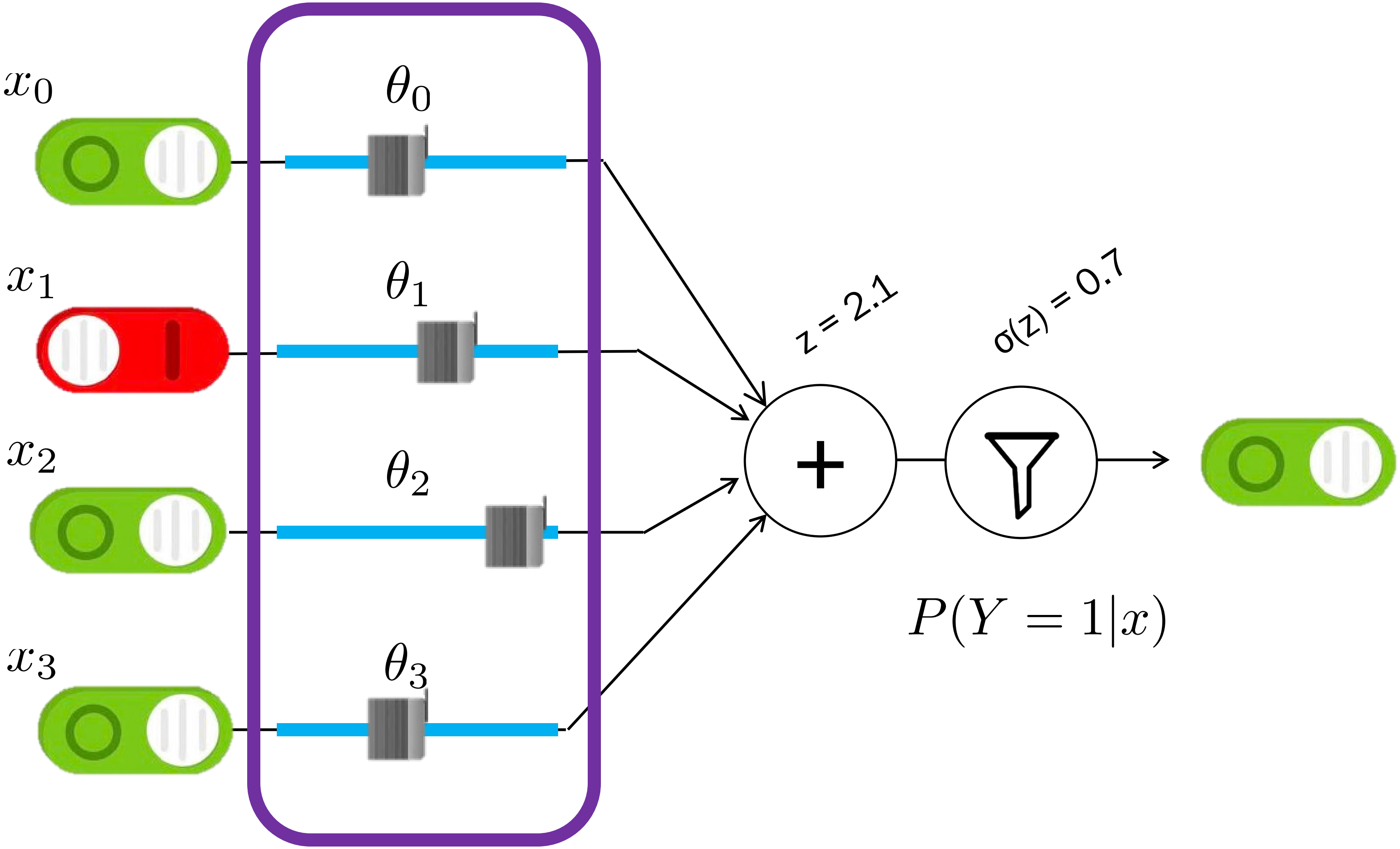
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Prediction



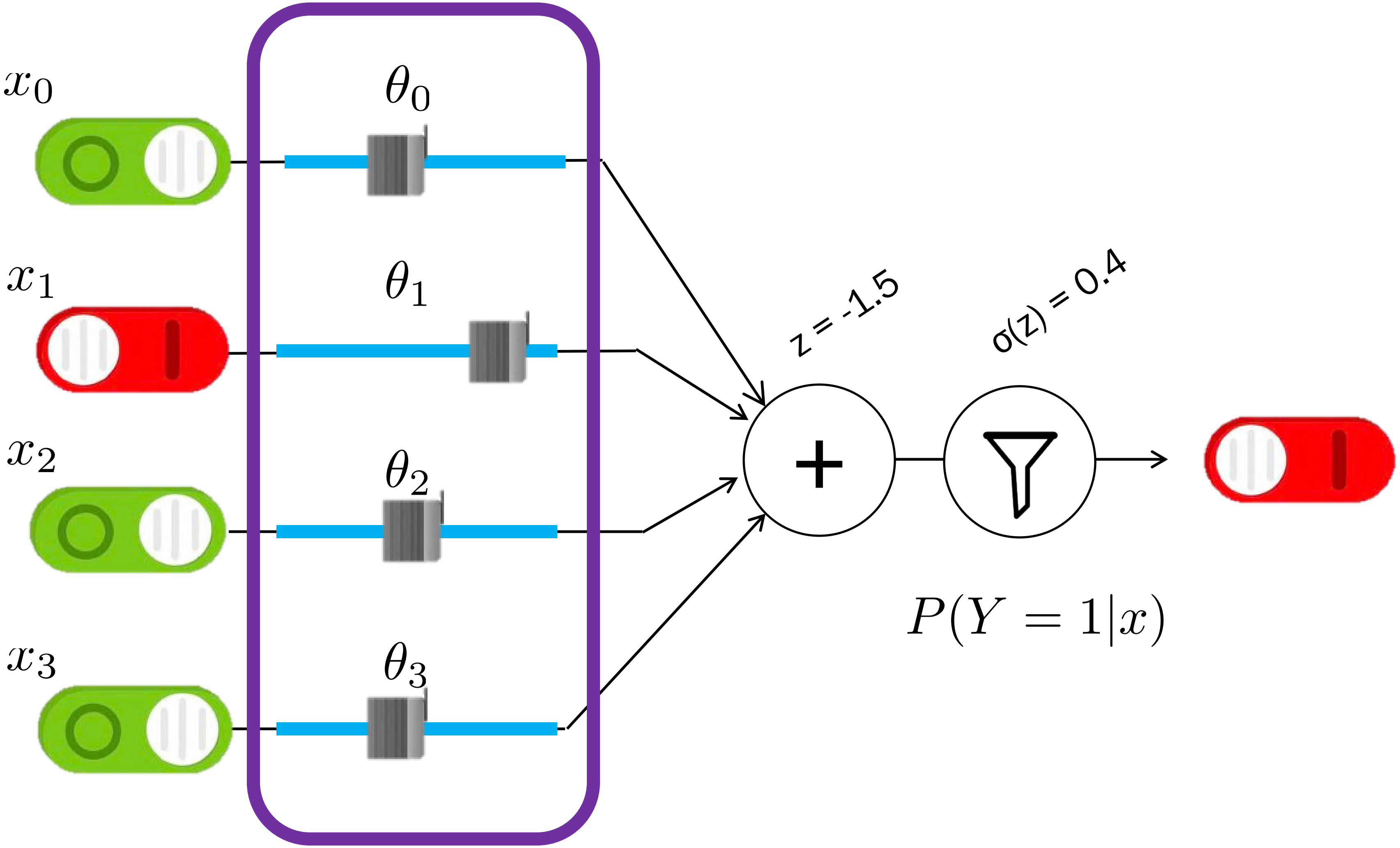
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Parameters Affect Prediction



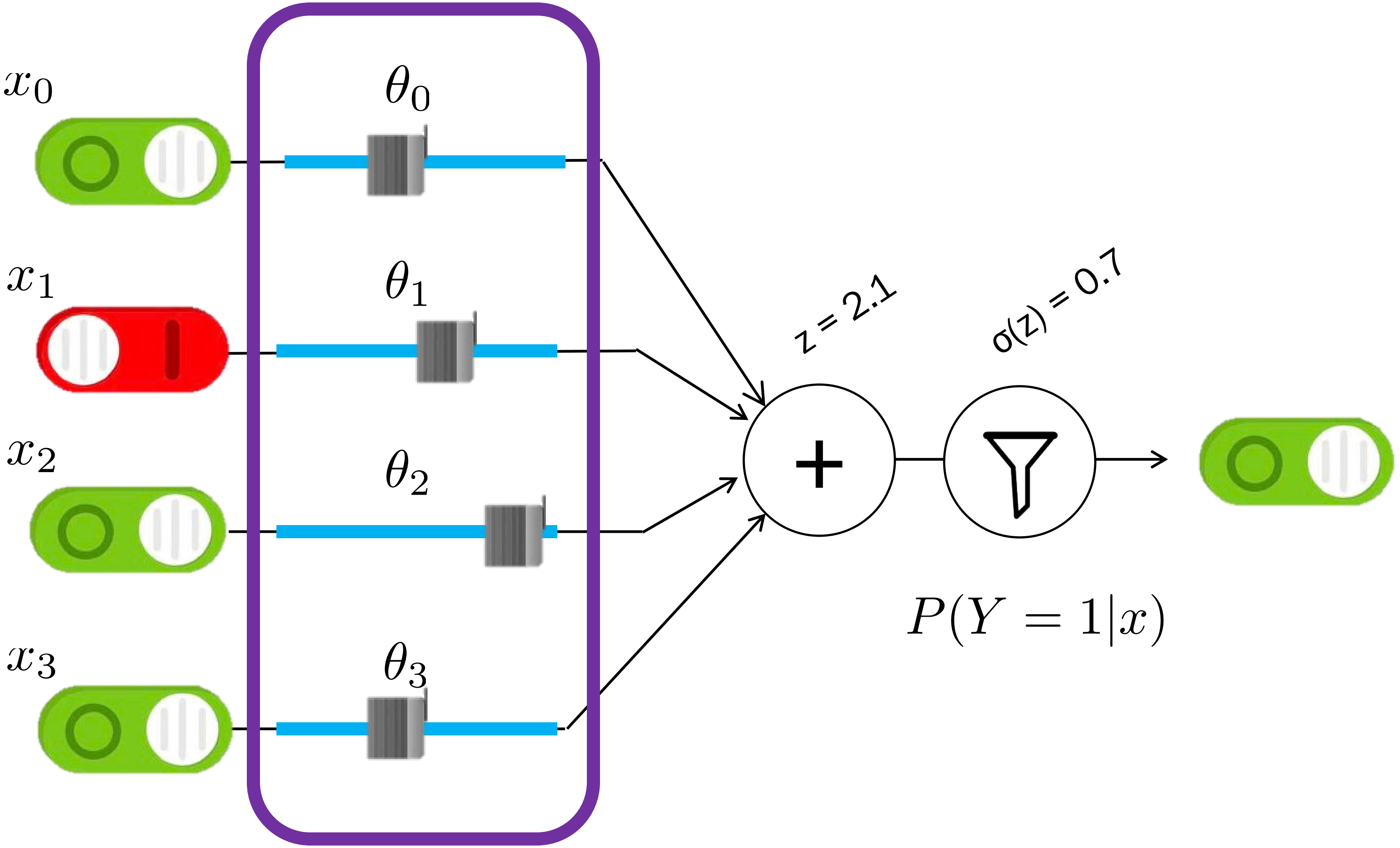
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Parameters Affect Prediction



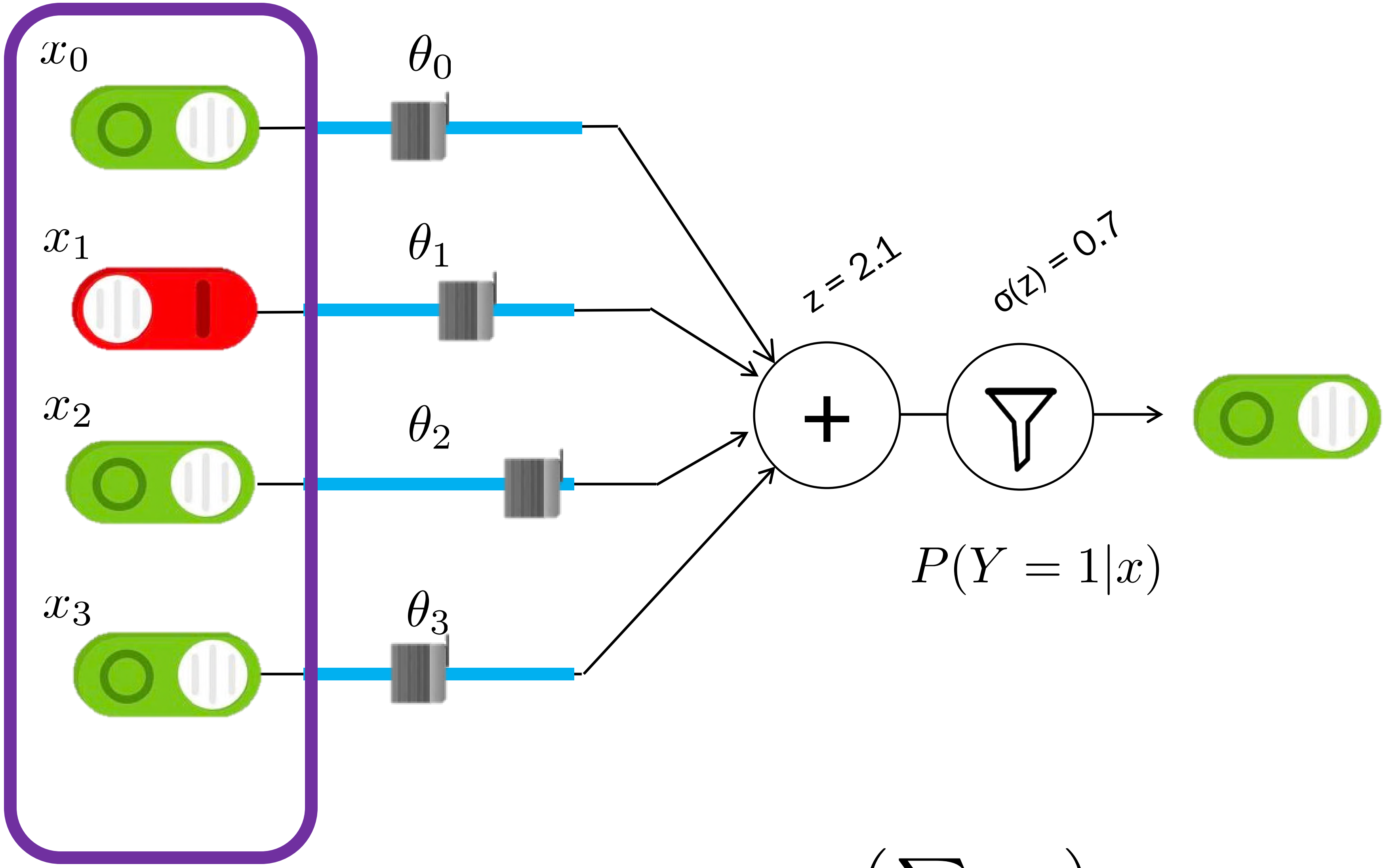
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Parameters Affect Prediction



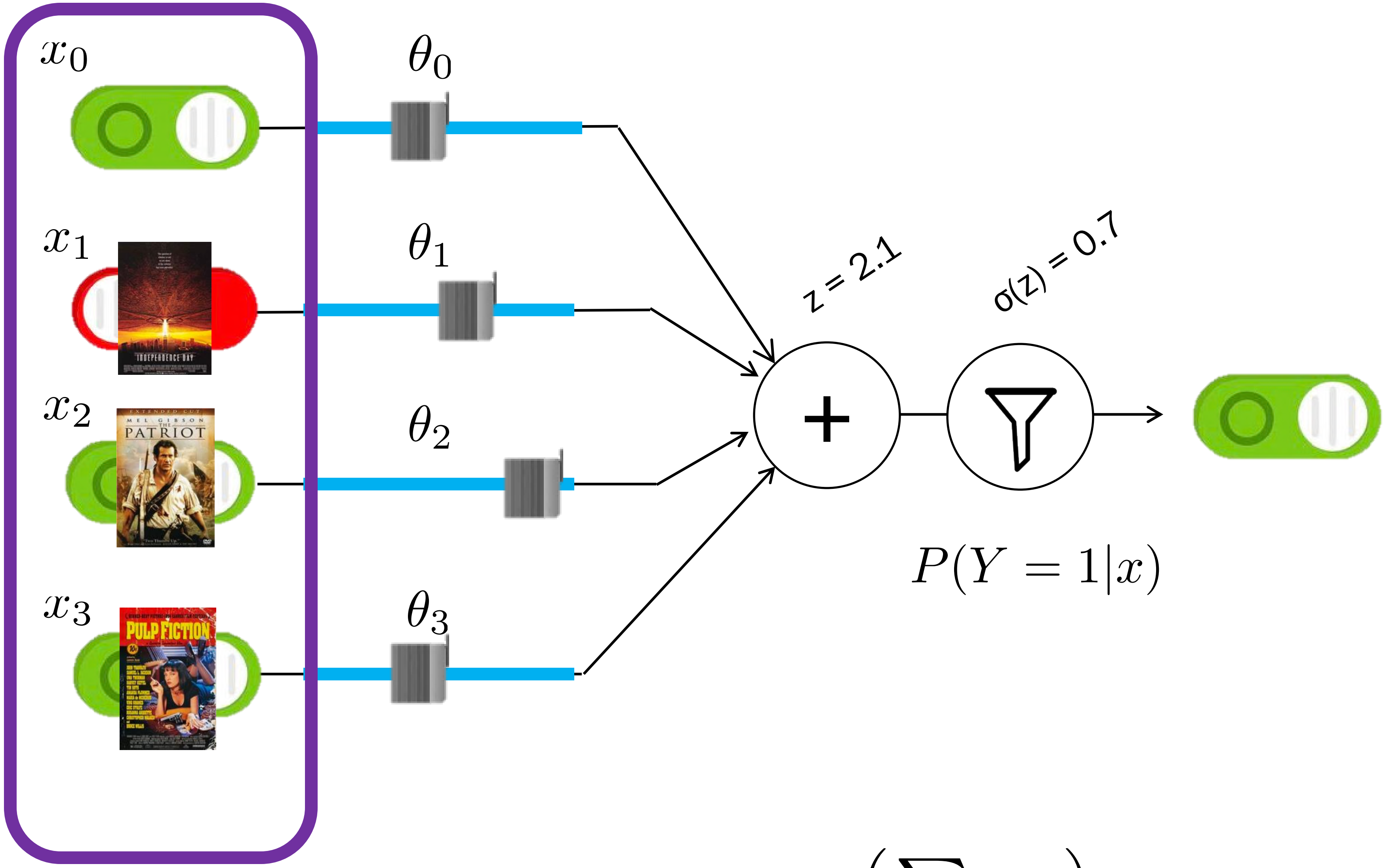
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Different Predictions for Different Inputs



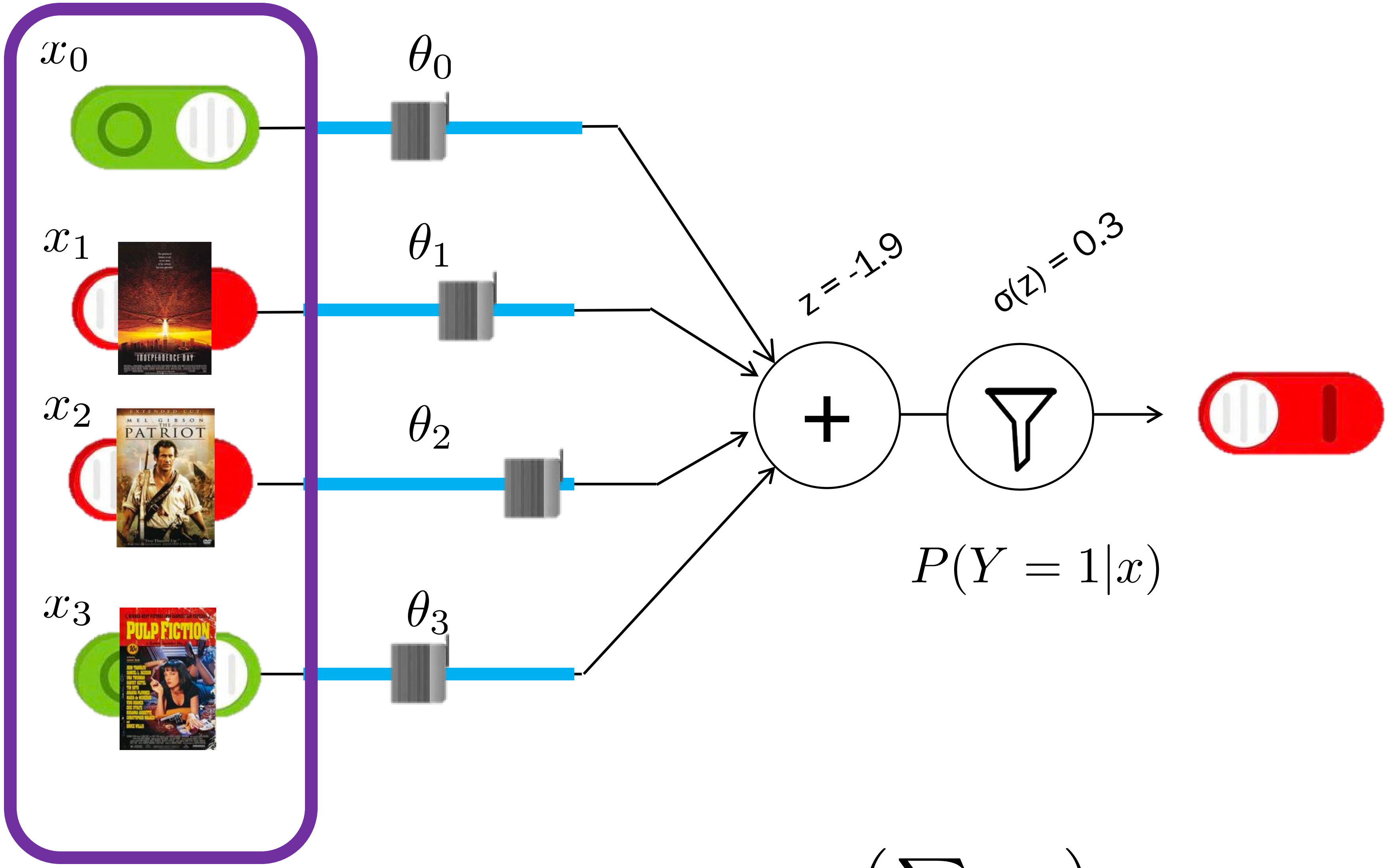
$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\Big(\sum_i \theta_i x_i\Big)$$

Different Predictions for Different Inputs



$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Different Predictions for Different Inputs



$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Handling the Intercept

Model *conditional* likelihood $P(Y=1|\mathbf{X}=\mathbf{x})$

with *logistic* function:

$$P(Y=1|\mathbf{X}) = \sigma(z) \text{ where } z = \theta_0 + \sum_{i=1}^{m} \theta_i x_i$$

- For simplicity define $x_0 = 1$ so $z = \theta^T \mathbf{x}$
- Since P(Y = 0 | $\mathbf{X}$ = $\mathbf{x}$) + P(Y = 1 | $\mathbf{X}$ = $\mathbf{x}$) = 1:

$$P(Y=1|X=\mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

$$P(Y=0|X=\mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x})$$

Recall:
Sigmoid function

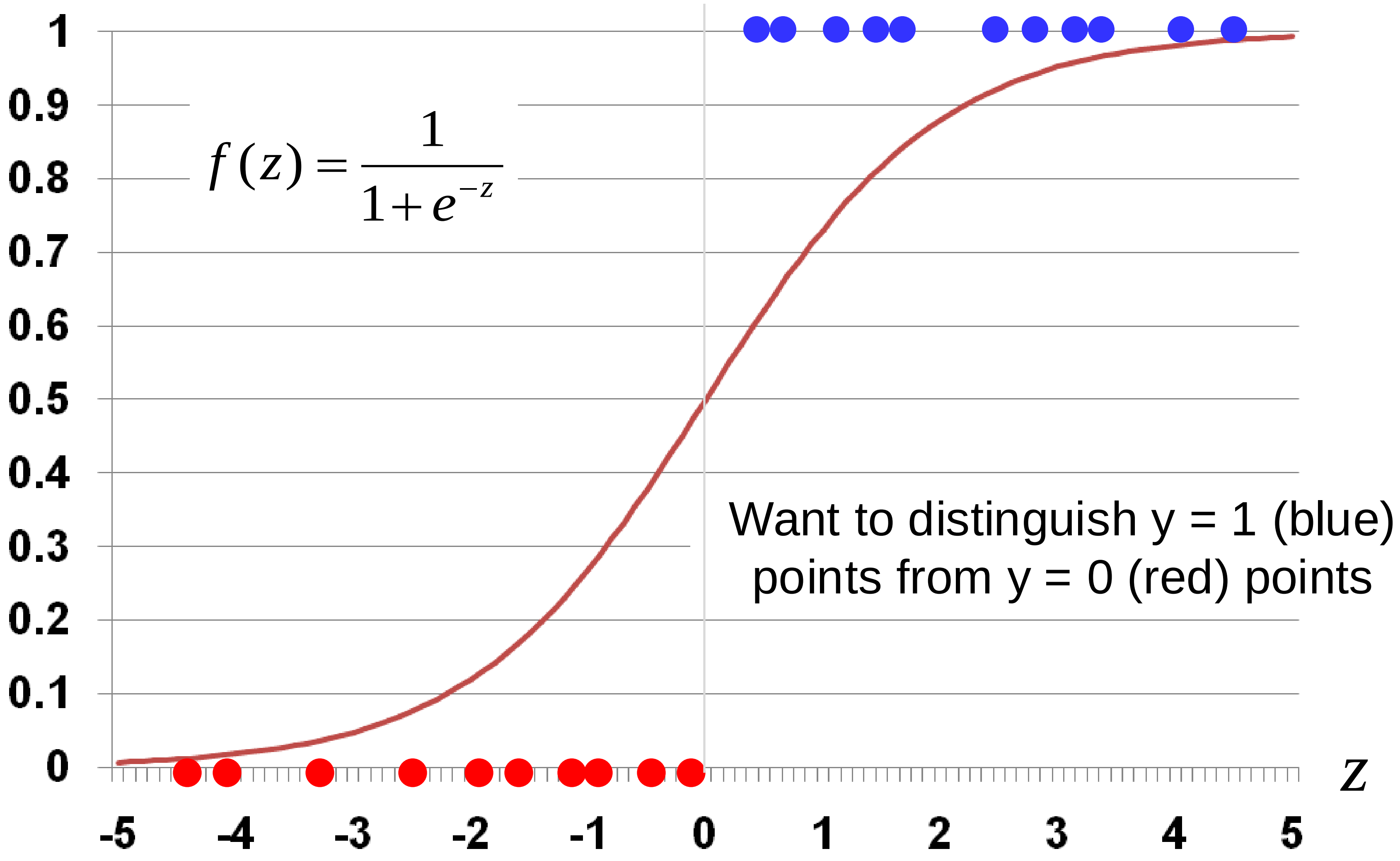
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Big Assumption

Logistic Regression Assumption:

$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

The Sigmoid Function



Note: inflection point at z = 0. $f(0) = 0.5$

What is in a Name

Regression Algorithms

Linear Regression 

Classification Algorithms

Decision Tree Classifier 

Logistic Regression 

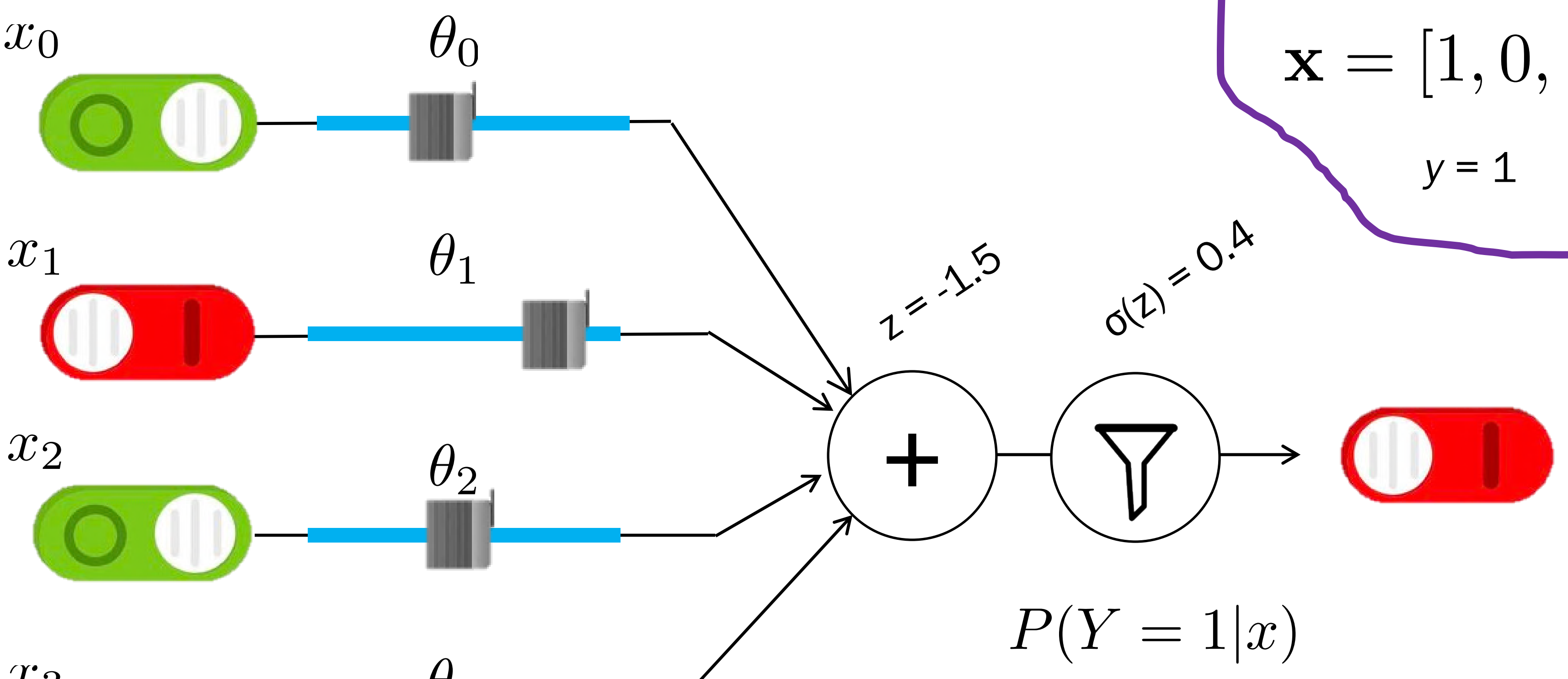
Awesome classifier, terrible name

If Chris could rename it he would call it: Sigmoidal Classification

What makes for a “smart”
logistic regression algorithm?

Logistic regression gets its ***intelligence*** from its thetas (aka its parameters)

How Do We Learn Parameters?



Let's say that:

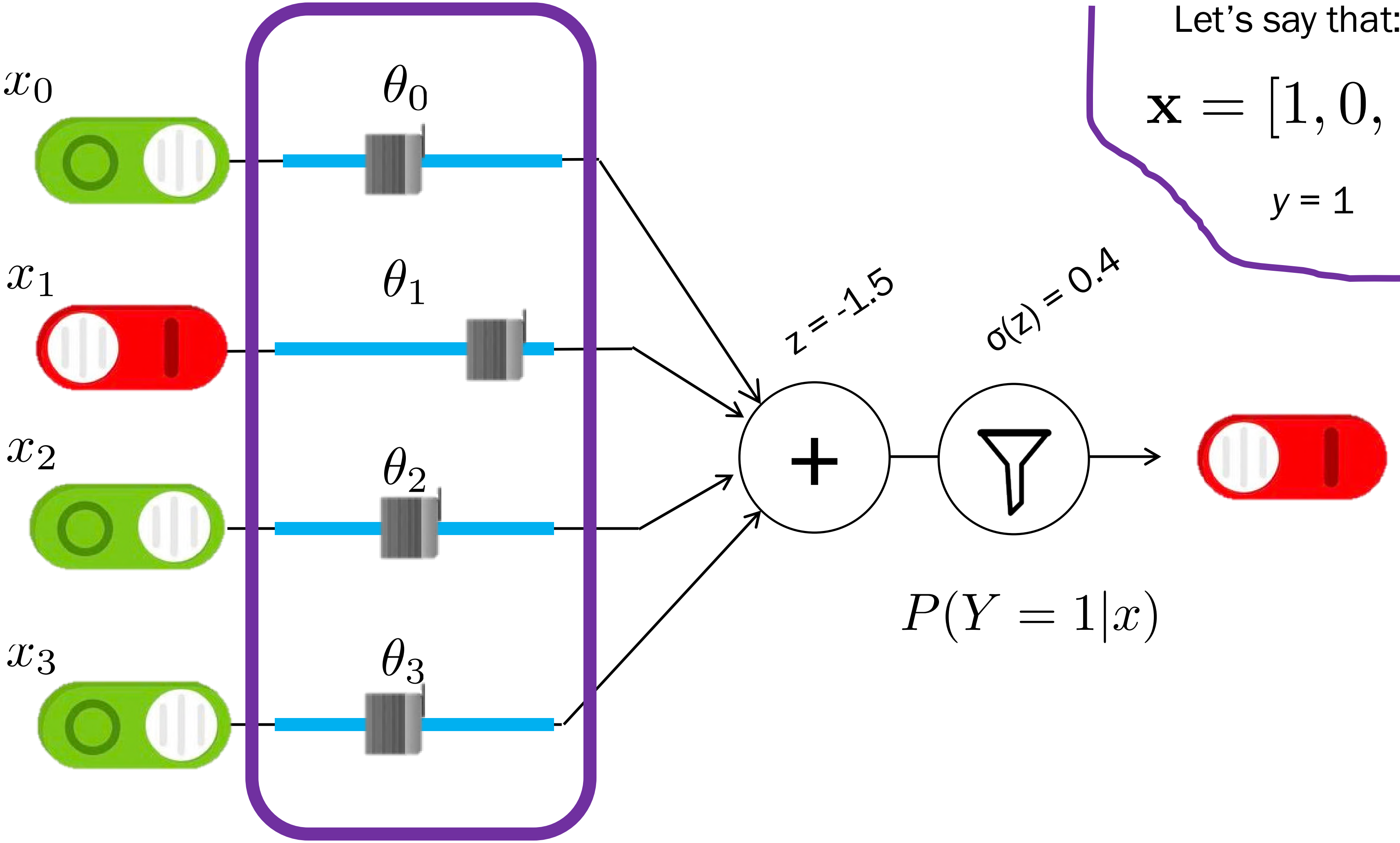
$$\mathbf{x} = [1, 0, 1, 1]$$

$y = 1$

$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right) = 0.4$$

Data looks unlikely

How Do We Learn Parameters?



Let's say that:

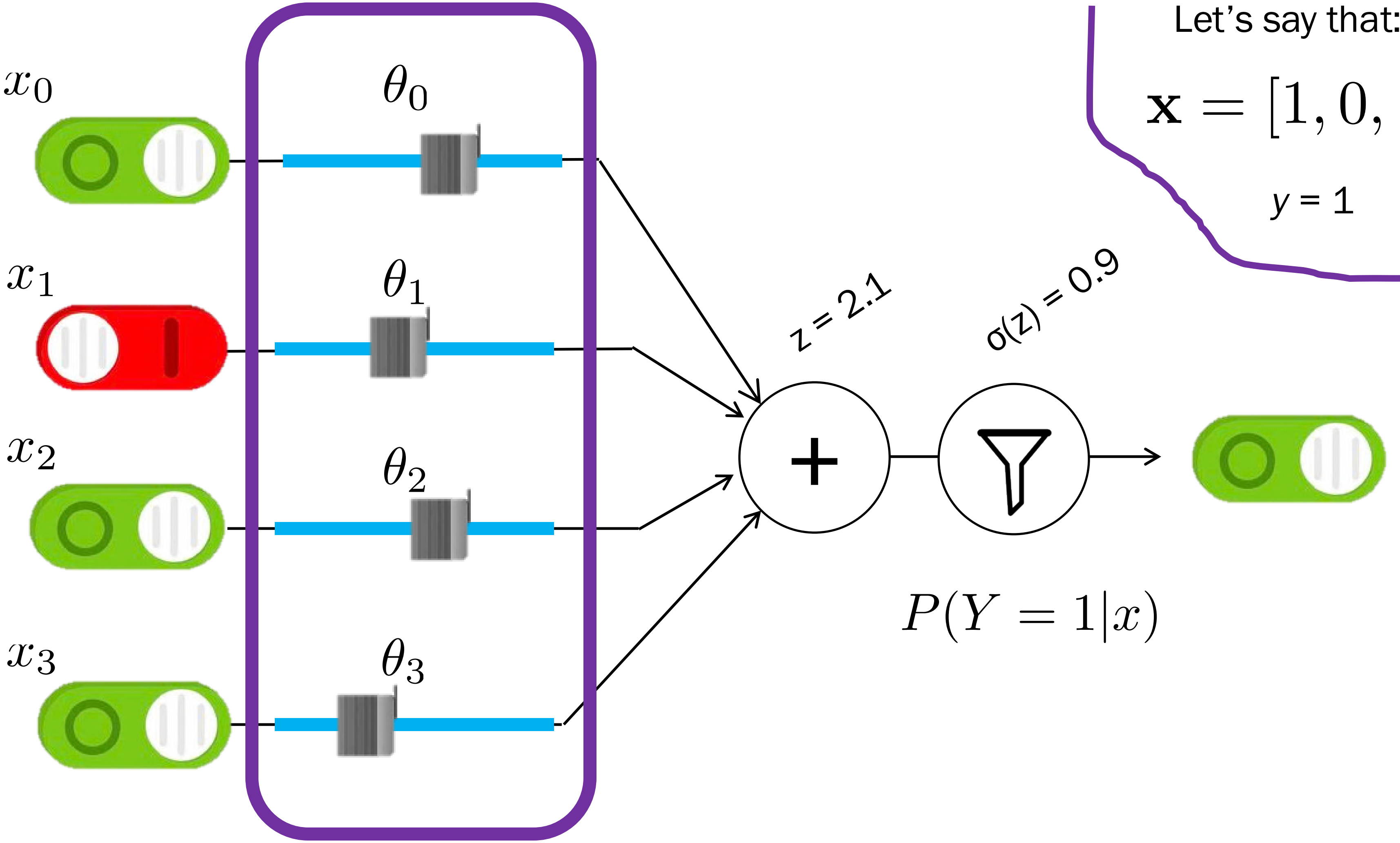
$$\mathbf{x} = [1, 0, 1, 1]$$

$y = 1$

$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\Big(\sum_i \theta_i x_i\Big) = 0.4$$

Data looks unlikely

How Do We Learn Parameters?



Let's say that:

$$\mathbf{x} = [1, 0, 1, 1]$$

y = 1

$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right) = 0.9$$

Data is much more likely!

Maximum Likelihood Estimation

Chose your parameter estimates

Parameter μ: 5 Parameter σ: 1.1

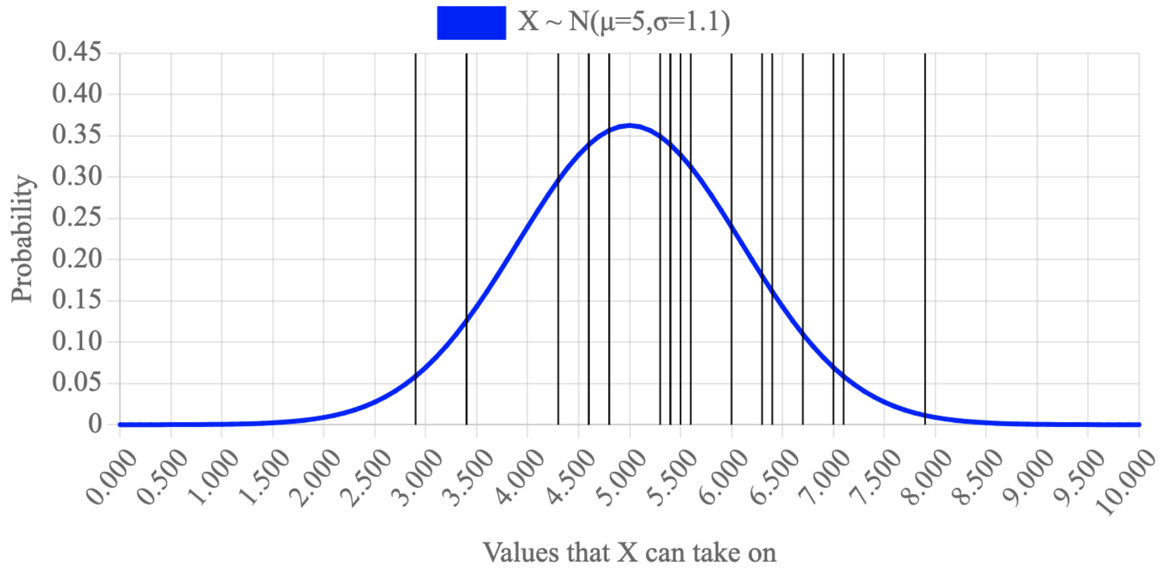
Likelihood of the data given your params

Likelihood: 5.204152095194613e-16

Log Likelihood: -314.1

Best Seen: -311.2

Your Gaussian



Pedagogy: show you the big picture,
then we can derive it!

Math for Logistic Regression

1. Make logistic regression assumption

$$P(Y=1|X=\mathbf{x}) = \sigma(\theta^T\mathbf{x})$$
$$P(Y=0|X=\mathbf{x}) = 1-\sigma(\theta^T\mathbf{x})$$

Often call this $\hat{y}$

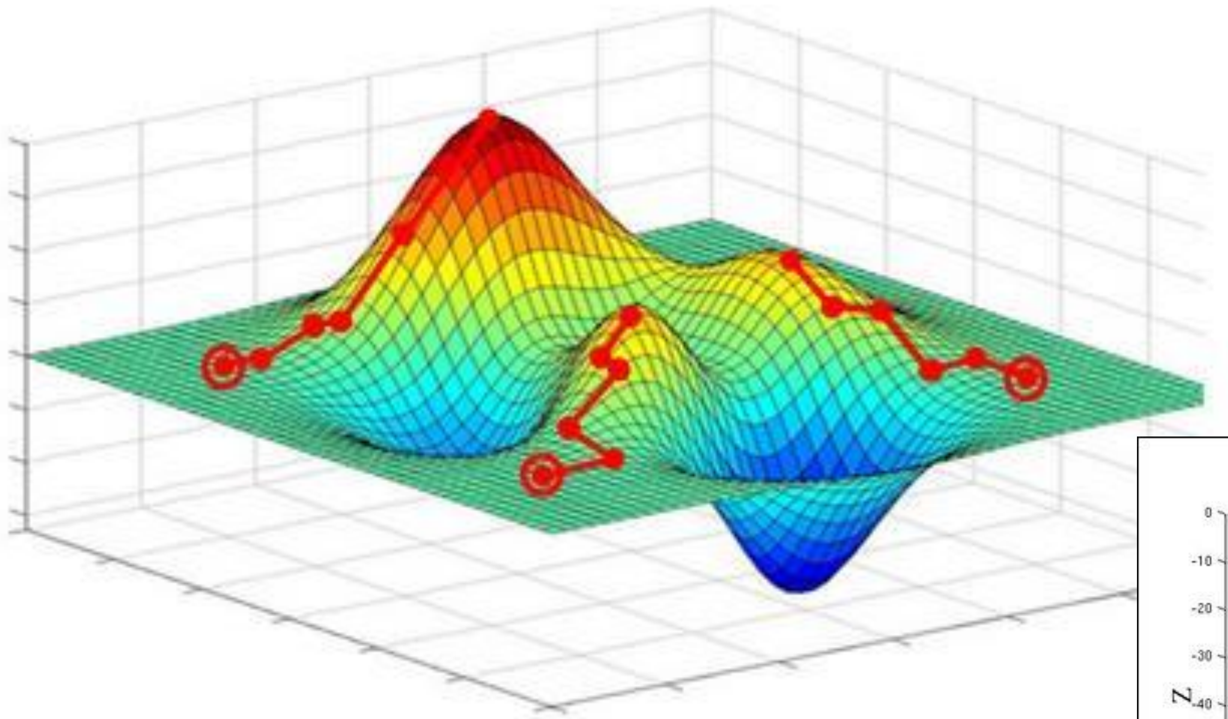
2. Calculate the log likelihood for all data

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T\mathbf{x}^{(i)}) + (1-y^{(i)}) \log[1-\sigma(\theta^T\mathbf{x}^{(i)})]$$

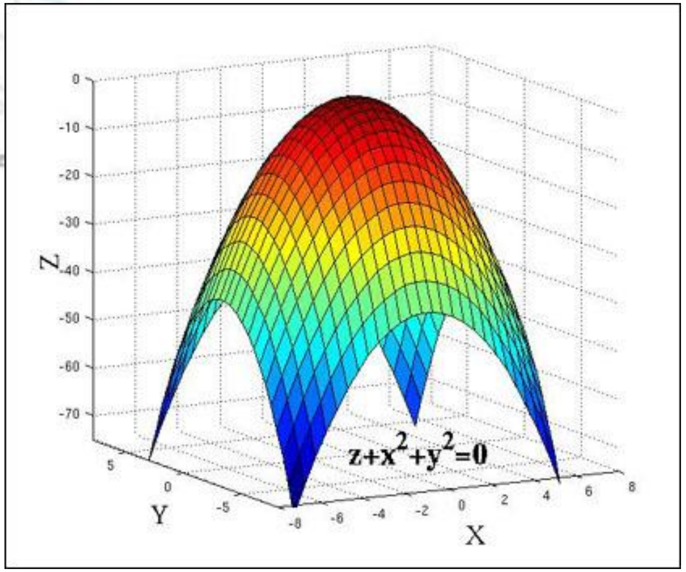
3. Get derivative of log likelihood with respect to thetas

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T\mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Gradient Ascent



Logistic regression LL function is convex



Walk uphill and you will find a local maxima
(if your step size is small enough)

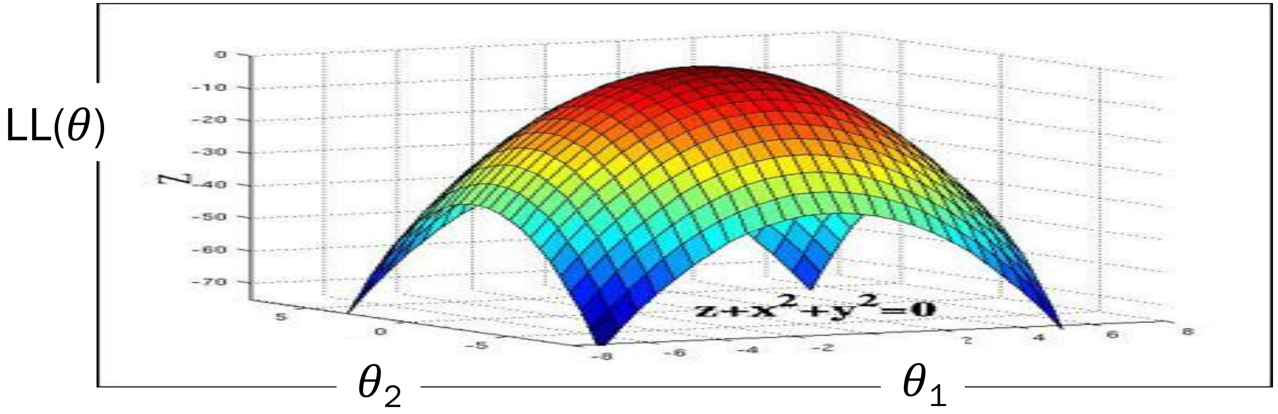
Gradient Ascent Step

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{ old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Do this for all thetas!



What does this look like in code?

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{ old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Real Code!!!

```
Initialize: θ_j = 0 for all 0 ≤ j ≤ m

Repeat many times:
    gradient[j] = 0 for all 0 ≤ j ≤ m

    For each training example (x, y):

        For each parameter j:

            gradient[j] += x_j (y - 1 / (1 + e^(-θ^T x)))

    θ_j += η * gradient[j] for all 0 ≤ j ≤ m
```

$$\texttt{gradient[j]} \mathrel{+}= x_j\left(y - \frac{1}{1+e^{-\theta^{\mathrm{T}}\mathbf{x}}}\right)$$

Logistic Regression Training

Initialize: $\theta_j = 0$ **for all** $0 \le j \le m$

Calculate all θ_j

Logistic Regression Training

Initialize: $\theta_j = 0$ **for all** $0 \le j \le m$

Repeat many times:

`gradient[j] = 0` **for all** $0 \le j \le m$

Calculate all **`gradient[j]`*****'s based on data***

θ_j `+=` η `* gradient[j]` **for all** $0 \le j \le m$

Logistic Regression Training

Initialize: $\theta_j = 0$ for all $0 \le j \le m$

Repeat many times:

gradient[j] = 0 for all $0 \le j \le m$

For each training example (x, y):

For each parameter j:

Update gradient[j] *for current training example (x, y)*

θ_j += η * gradient[j] for all $0 \le j \le m$

Logistic Regression Training

Initialize: $\theta_j = 0$ **for all** $0 \le j \le m$

Repeat many times:

gradient[j] = 0 for all $0 \le j \le m$

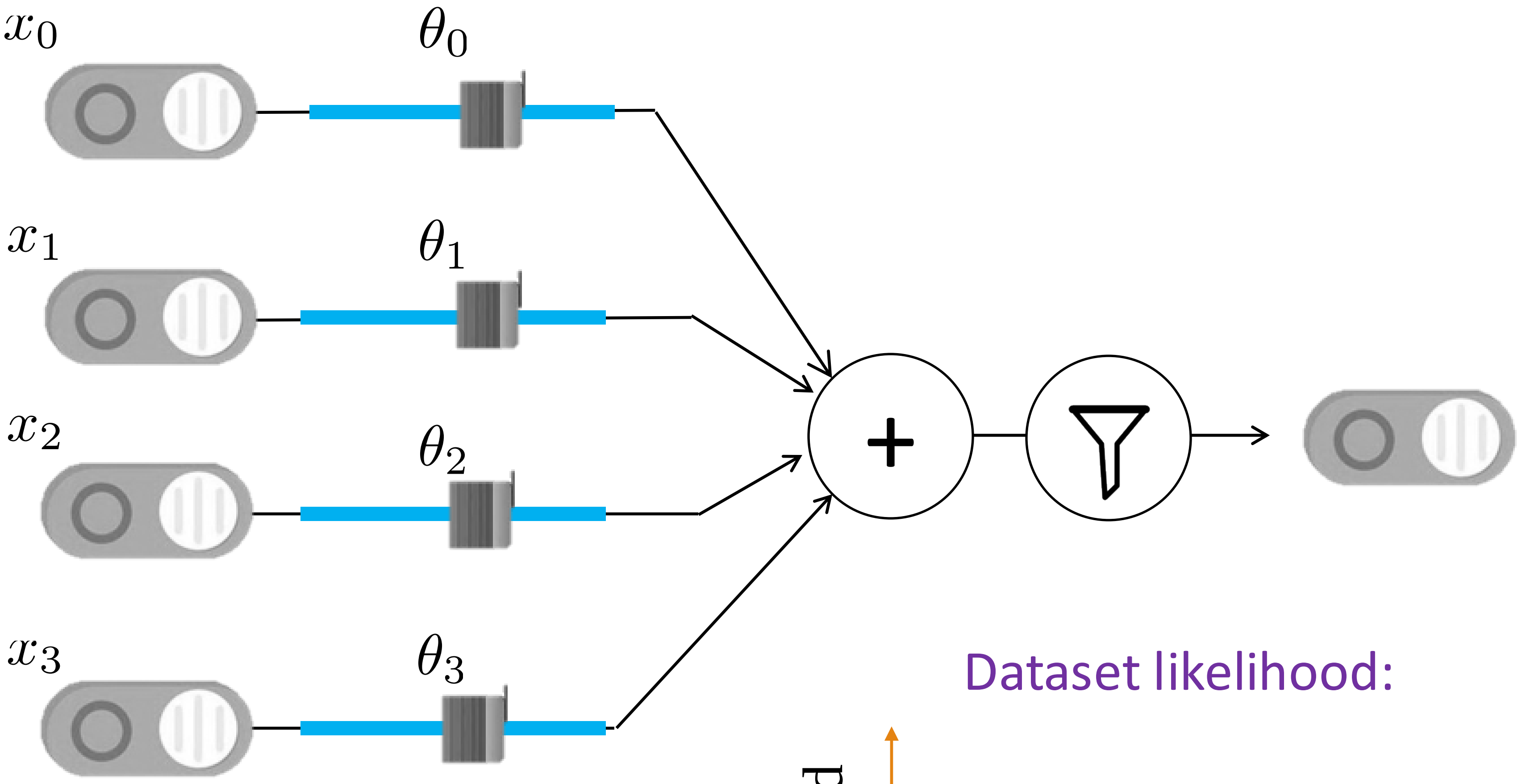
For each training example (x, y):

For each parameter j:

gradient[j] $\mathrel{+}= x_j\left(y - \dfrac{1}{1+e^{-\theta^{\mathrm{T}}\mathbf{x}}}\right)$

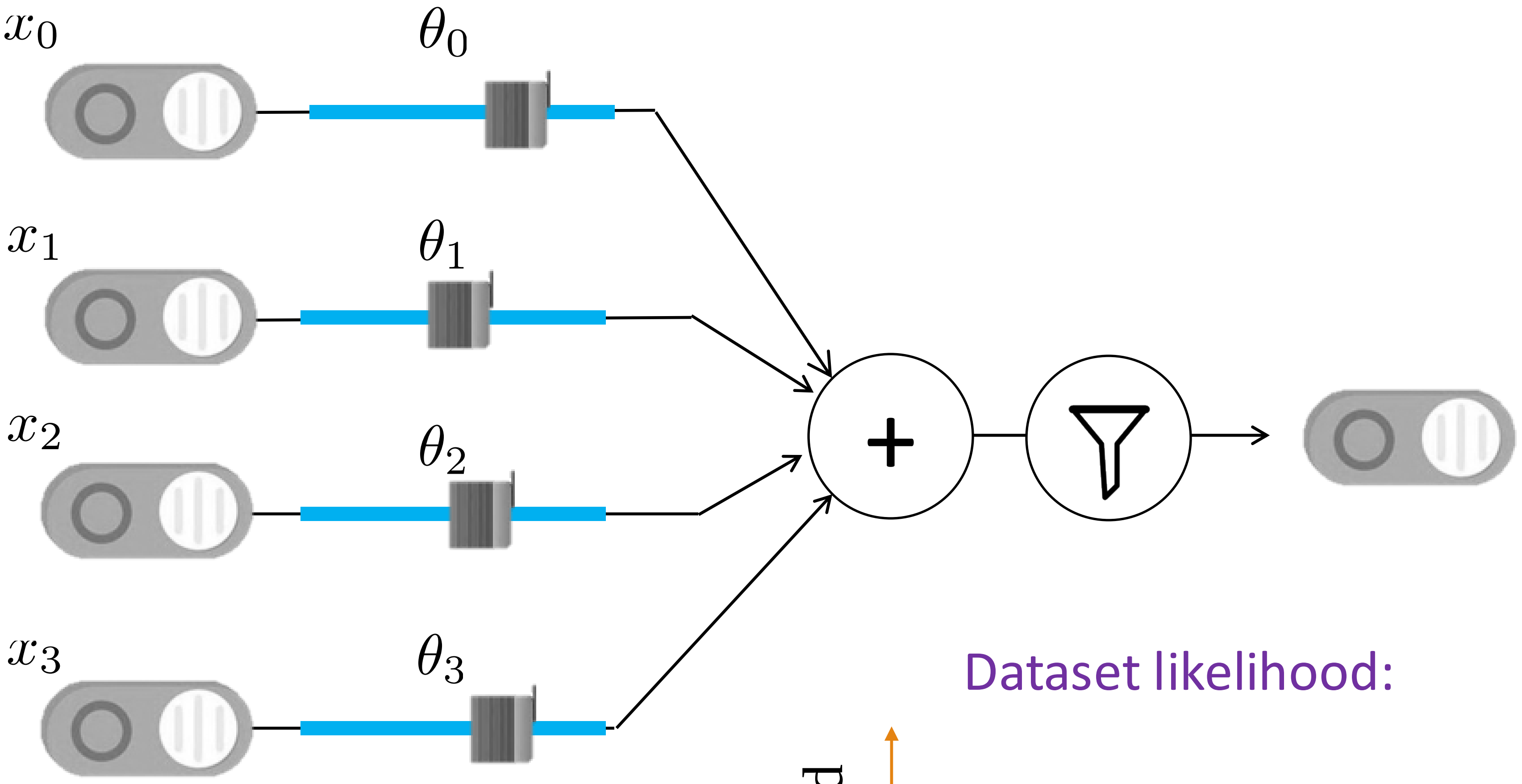
θ_j **+=** η *** gradient[j] for all** $0 \le j \le m$

Training

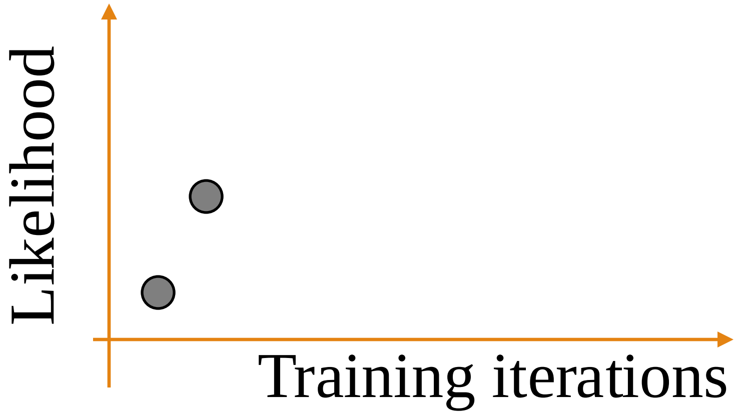


Dataset likelihood:

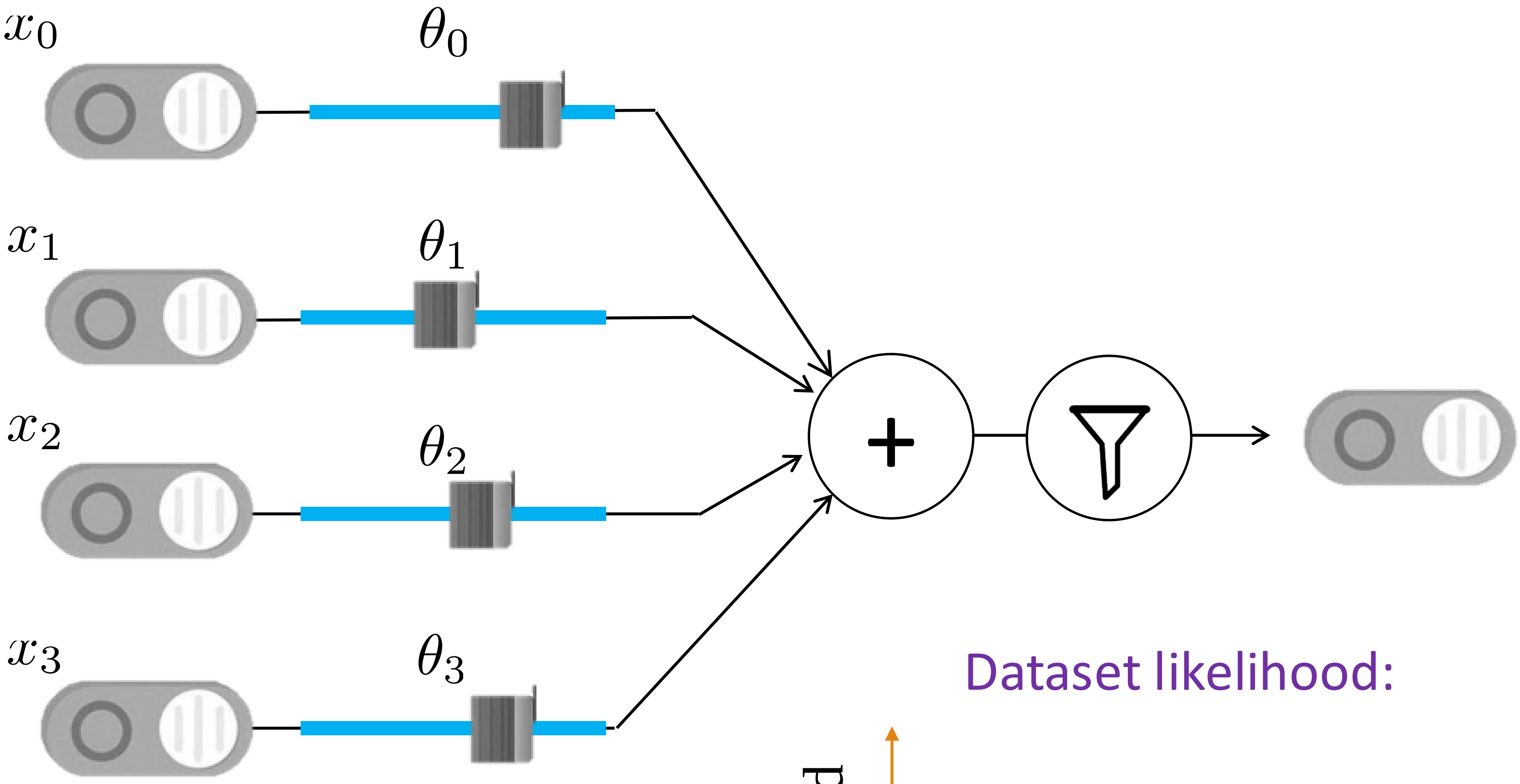
Training



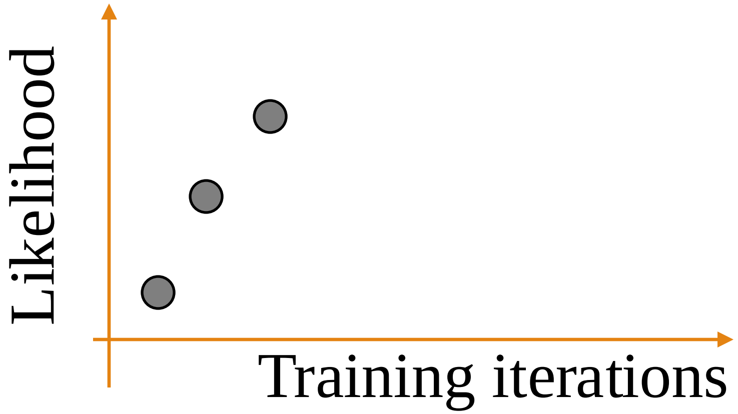
Dataset likelihood:



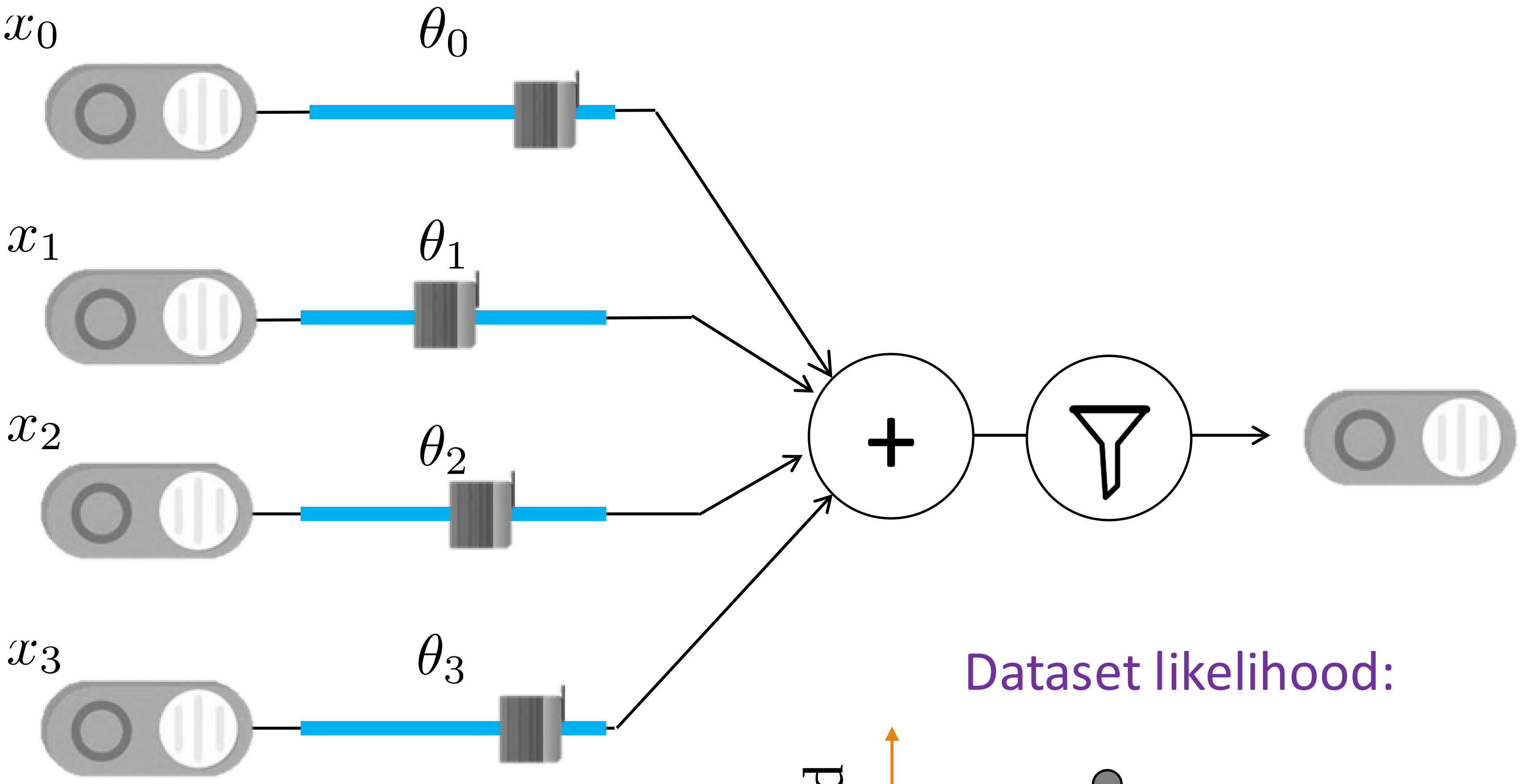
Training



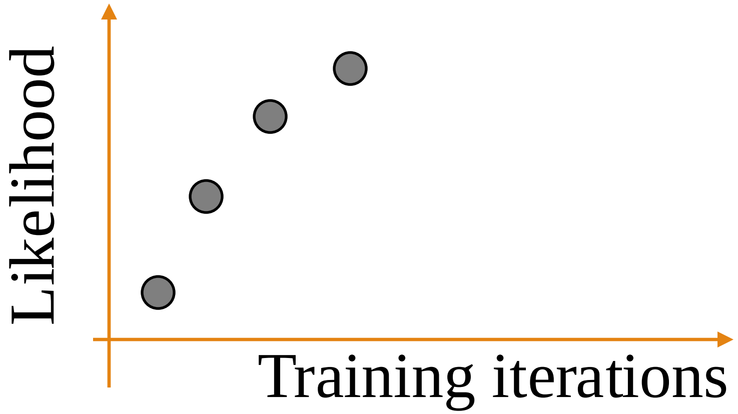
Dataset likelihood:



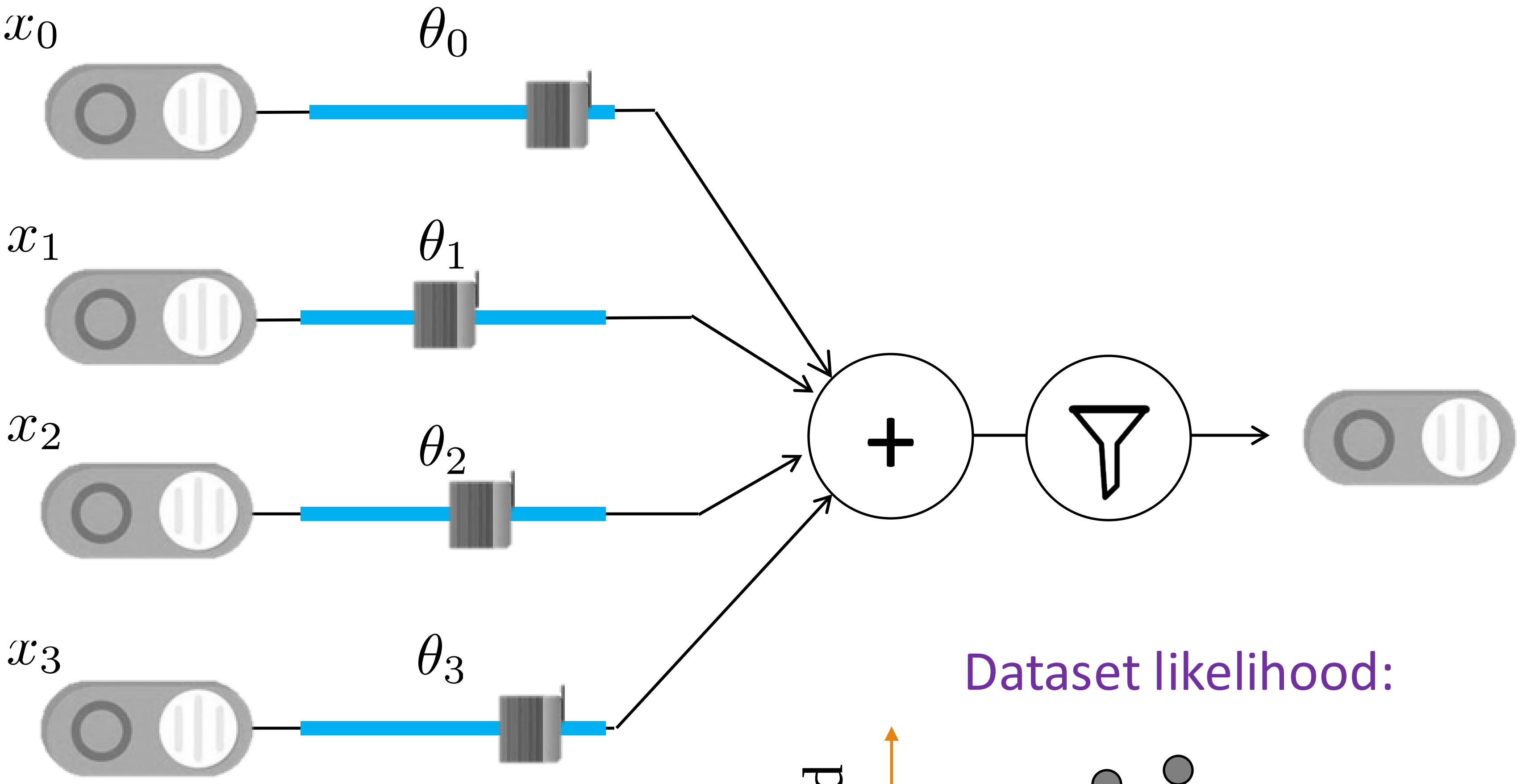
Training



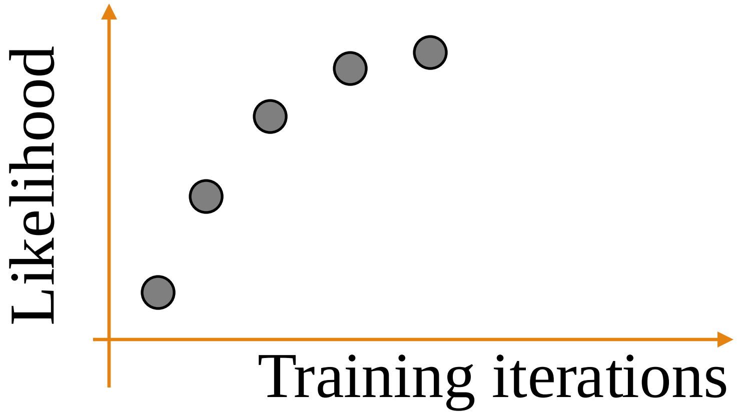
Dataset likelihood:



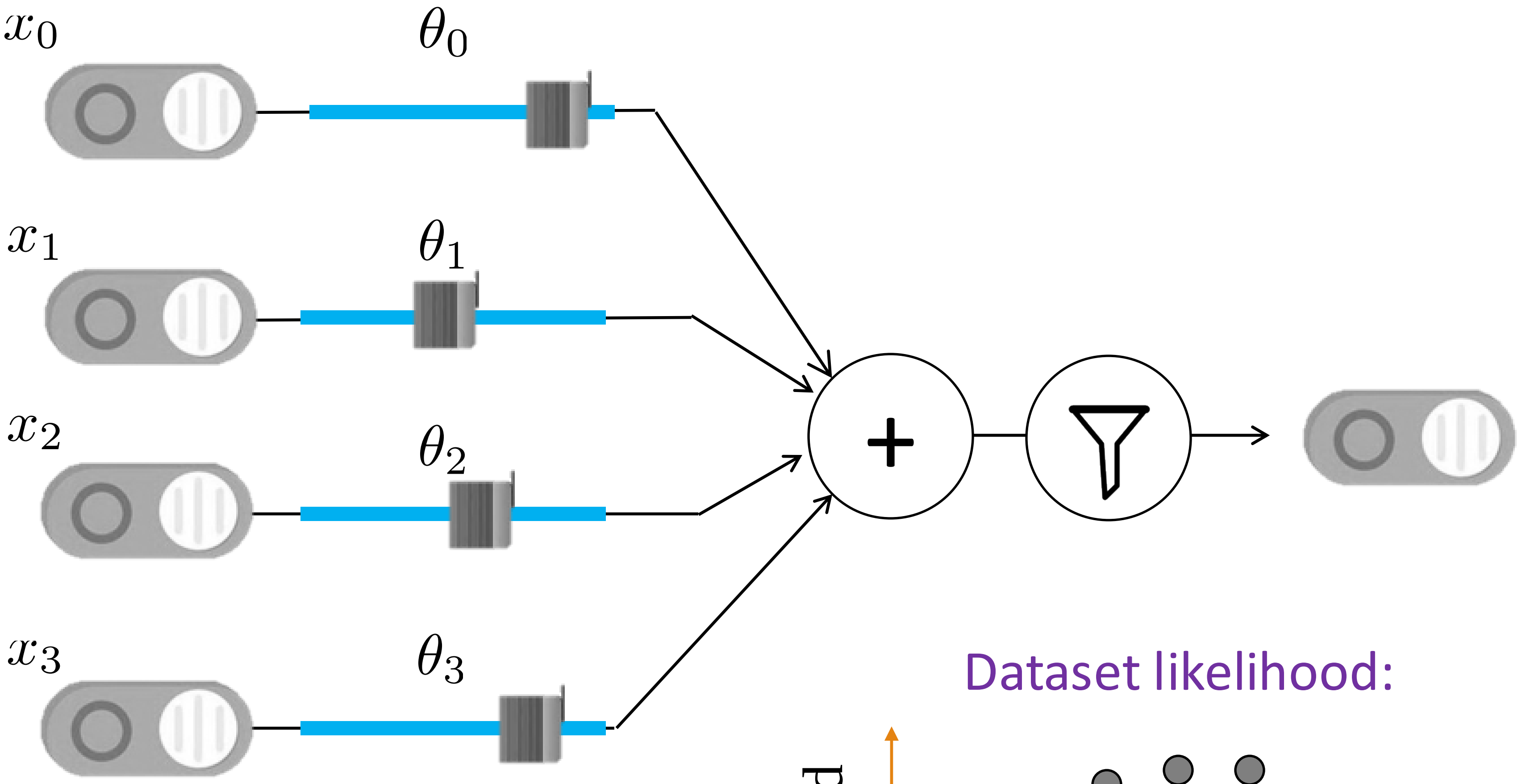
Training



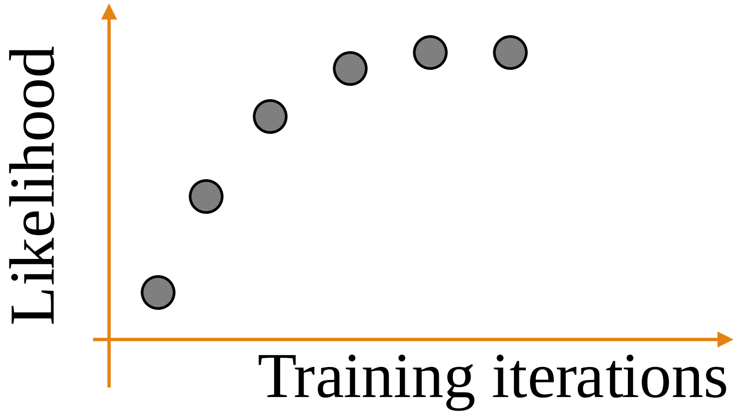
Dataset likelihood:



Training



Dataset likelihood:

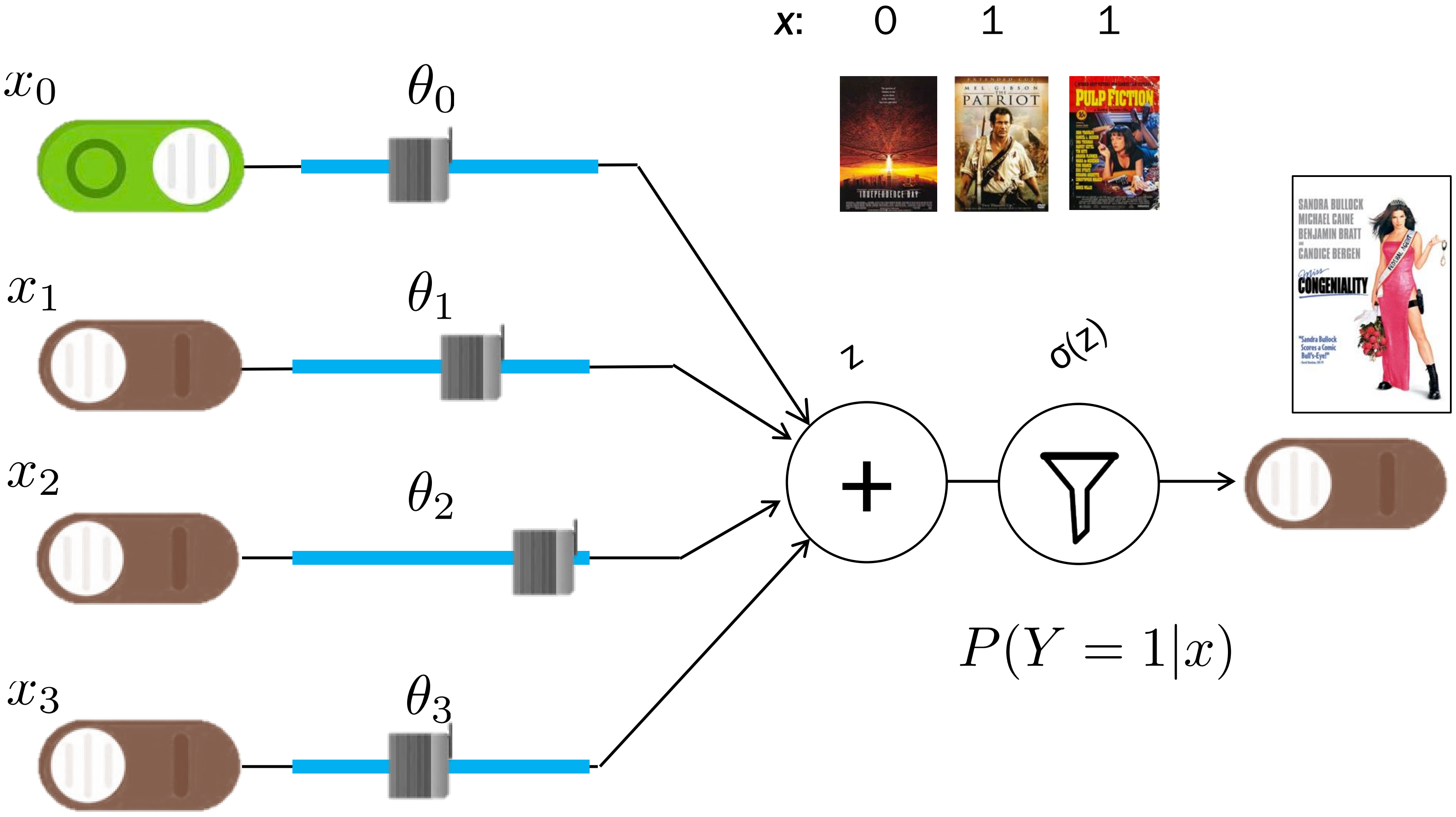


Don't forget:

x_j is j-th input variable and $x_0 = 1$.

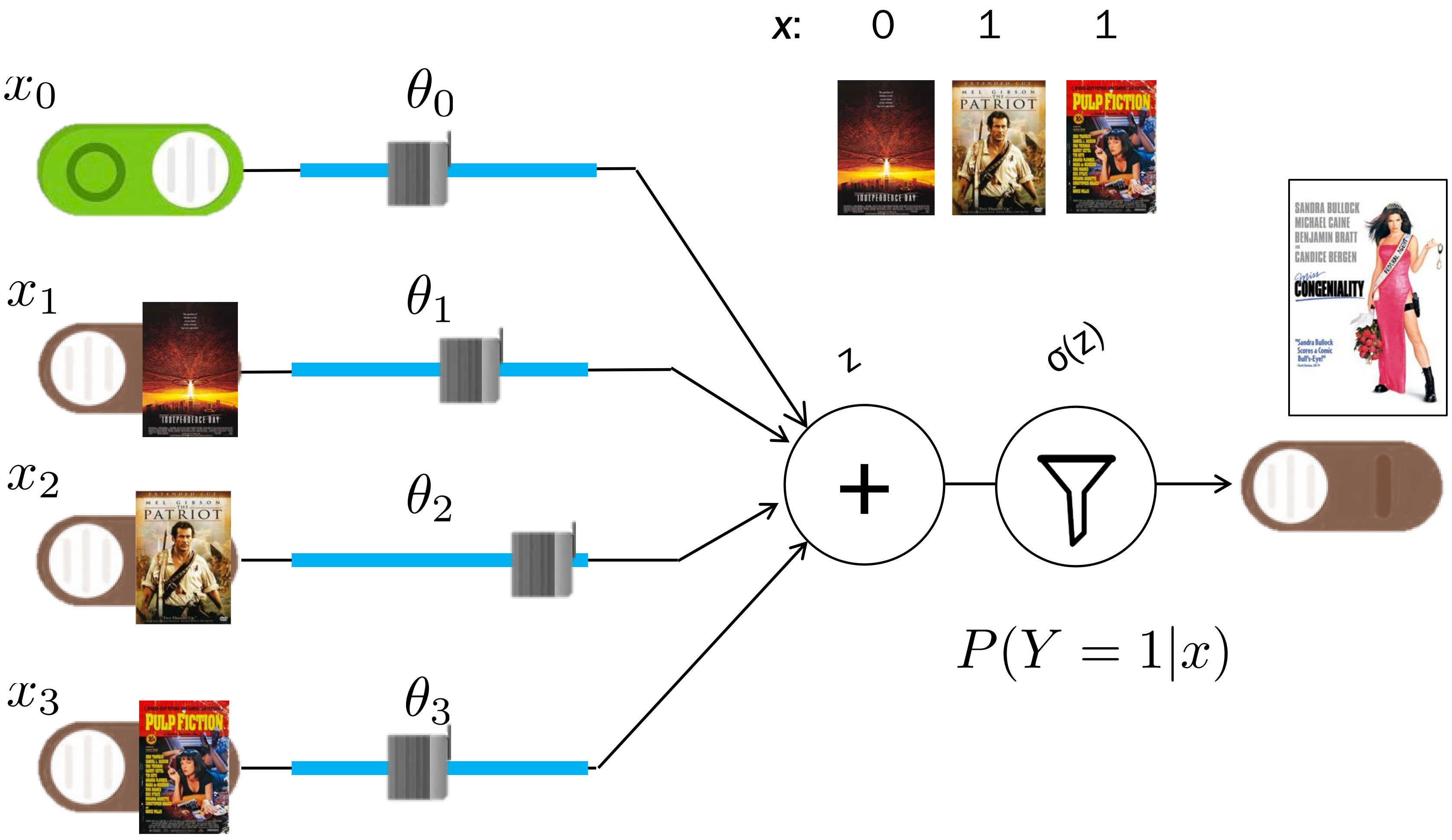
Allows for θ_0 to be an intercept.

Prediction



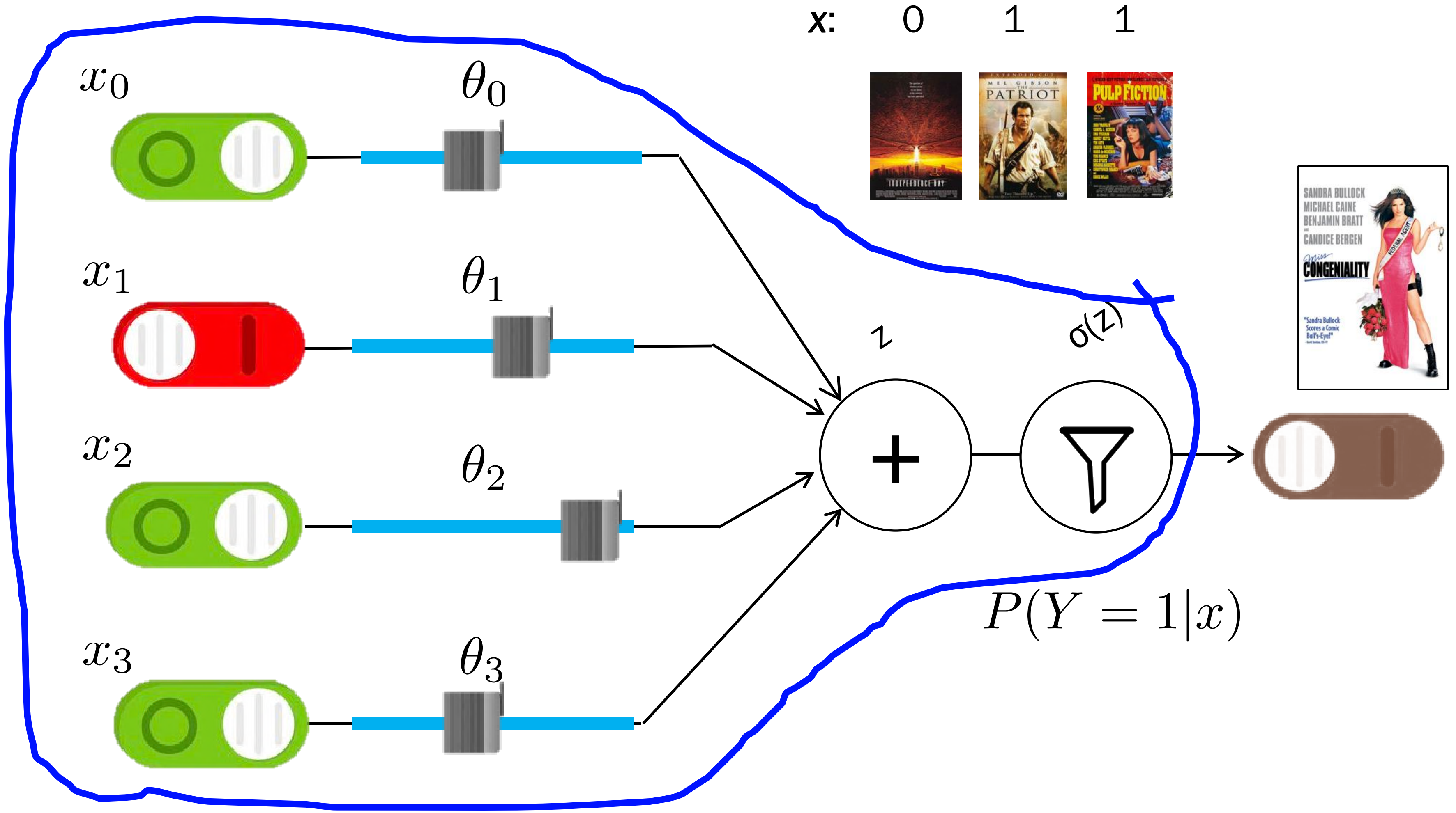
$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Prediction



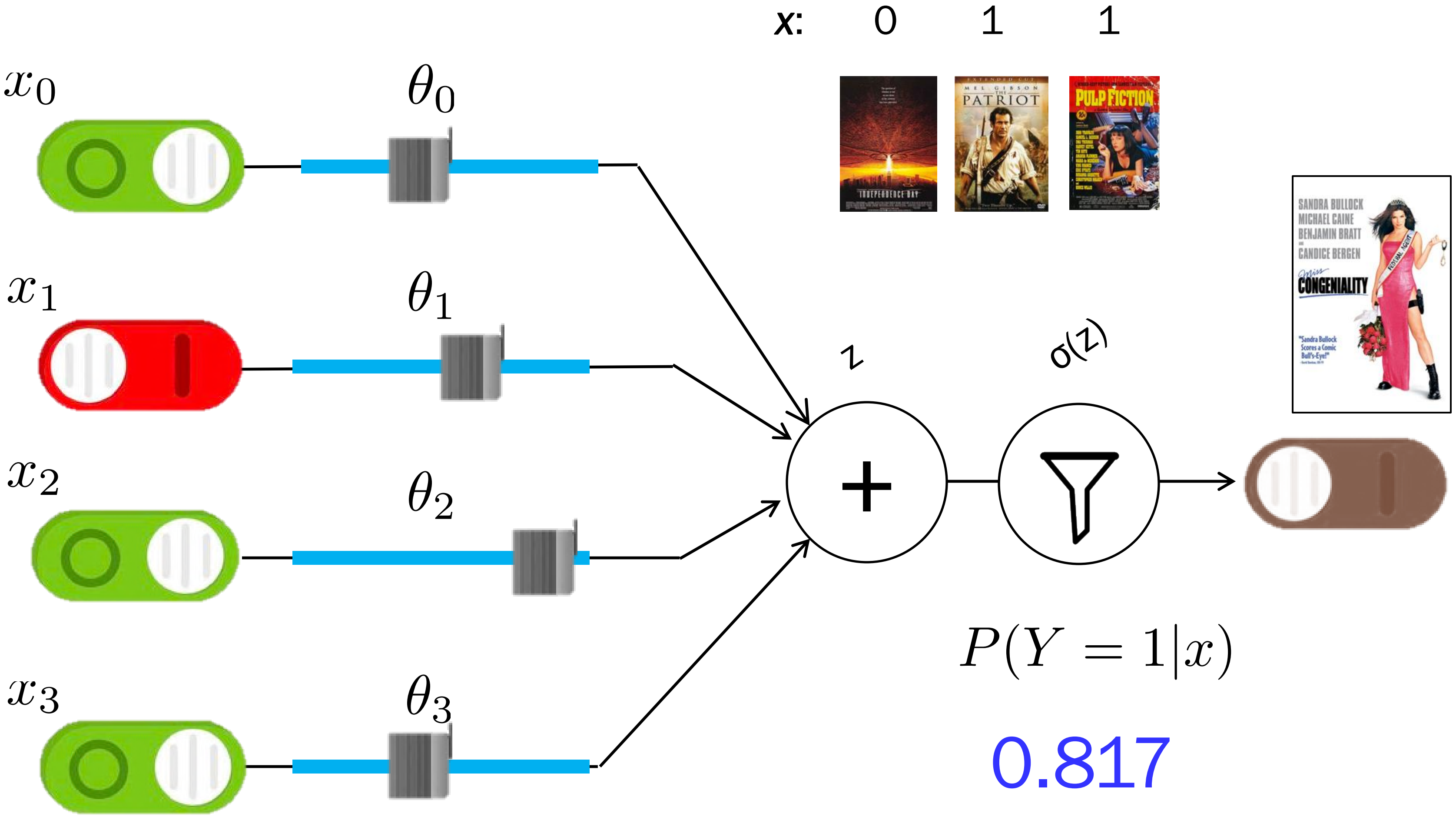
$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Prediction



$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Prediction

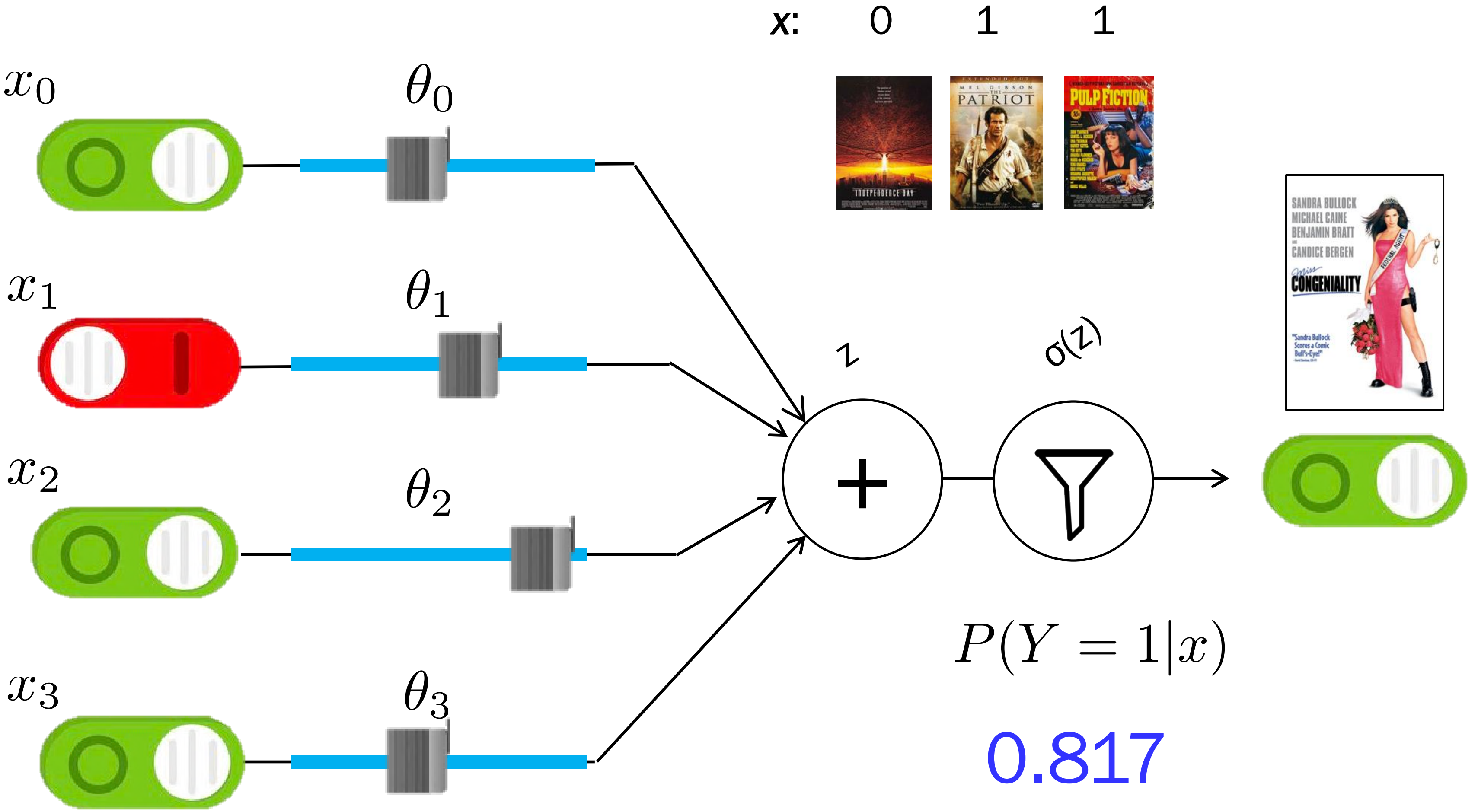


$P(Y = 1|x)$

0.817

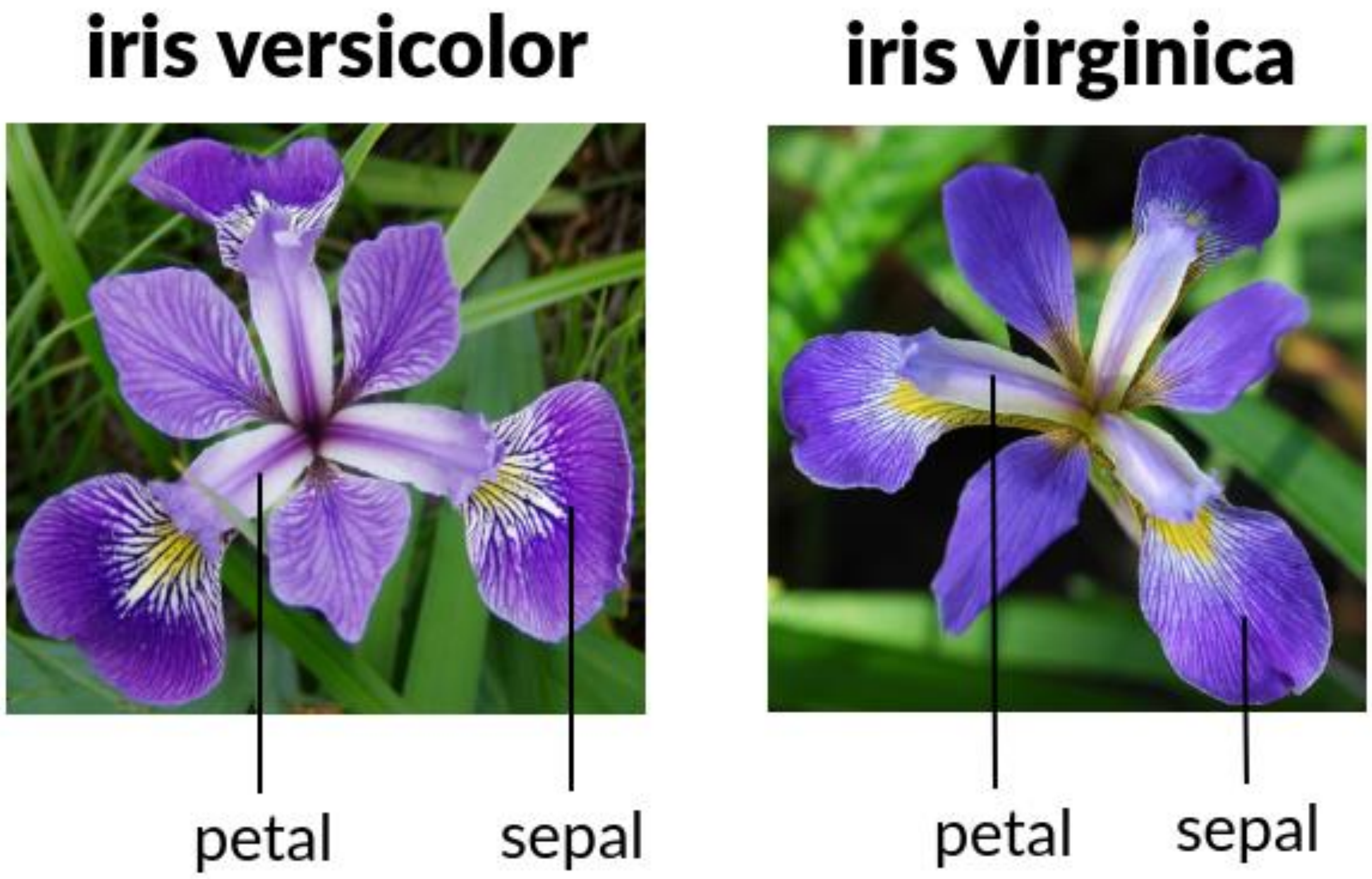
$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Prediction



$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Live Demo!!



Chapter 2: How Come?

Logistic Regression

1 Make logistic regression assumption

$$P(Y=1|X=\mathbf{x}) = \sigma(\theta^T\mathbf{x})$$

$$P(Y=0|X=\mathbf{x}) = 1-\sigma(\theta^T\mathbf{x})$$

Often call this $\hat{y}$

2 Calculate the log probability for all data

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T\mathbf{x}^{(i)}) + (1-y^{(i)}) \log[1-\sigma(\theta^T\mathbf{x}^{(i)})]$$

3 Get derivative of log probability with respect to thetas

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T\mathbf{x}^{(i)})\right] x_j^{(i)}$$

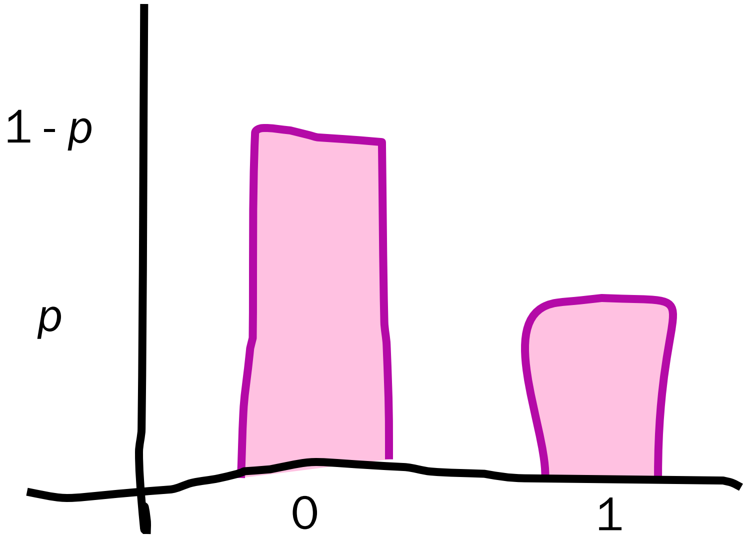
How did we get that LL function?

Recall: PMF of Bernoulli

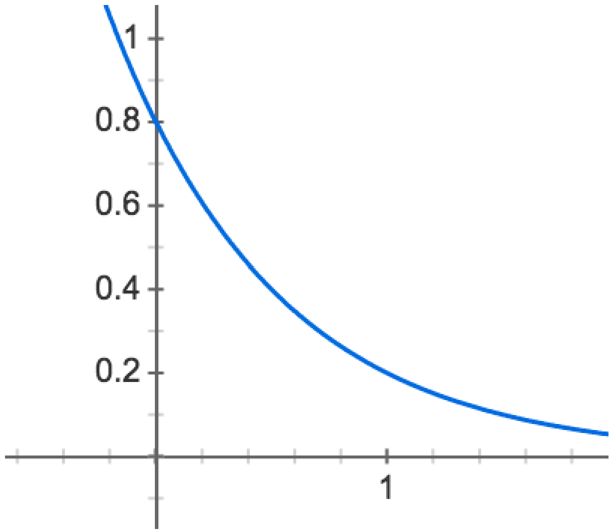
$$Y \sim \text{Bern}(p)$$

Probability mass function: $P(Y = y)$

PMF of Bernoulli



PMF of Bernoulli (p = 0.2)



$$P(Y = y) = p^y (1 - p)^{1-y}$$

$$P(Y = y) = 0.2^y (0.8)^{1-y}$$

Recall:

$Y \sim \text{Bern}(p)$

$P(Y = y) = p^y (1-p)^{1-y}$

$$P(Y = 1 | X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

$$P(Y = 0 | X = \mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x})$$

Implies

$$P(Y = y | X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})^y \cdot \left[1 - \sigma(\theta^T \mathbf{x})\right]^{(1-y)}$$

For IID data

$$L(\theta) = \prod_{i=1}^{n} P(Y = y^{(i)} | X = \mathbf{x}^{(i)})$$

$$= \prod_{i=1}^{n} \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} \cdot \left[1 - \sigma(\theta^T \mathbf{x}^{(i)})\right]^{(1-y^{(i)})}$$

Take the log

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Ask Chris:
Why not

$P(Y = y^{(i)}, X = x^{(i)})$

How did we get that gradient?

Sigmoid has a Beautiful Slope

True fact about sigmoid functions

$$\frac{\partial}{\partial z}\sigma(z) = \sigma(z)[1 - \sigma(z)]$$

Sigmoid has a Beautiful Slope

$$\frac{\partial}{\partial \theta_j} \sigma(\theta^T x)?$$

$$\frac{\partial}{\partial z} \sigma(z) = \sigma(z)[1 - \sigma(z)]$$

where $z = \theta^T x$

$$\frac{\partial}{\partial \theta_j} \sigma(\theta^T x) = \frac{\partial}{\partial z} \sigma(z) \cdot \frac{\partial z}{\partial \theta_j}$$

Chain rule!

$$\frac{\partial}{\partial \theta_j} \sigma(\theta^T x) = \sigma(\theta^T x)[1 - \sigma(\theta^T x)] x_j$$

Plug and chug

Sigmoid, you should be a ski hill

Sigmoid has a Beautiful Slope

$$\hat{y} = \sigma(\theta^T x)$$

$$\frac{\partial \hat{y}}{\partial \theta_j} = \sigma(\theta^T x)[1 - \sigma(\theta^T x)]x_j$$

$$= \hat{y}(1 - \hat{y})x_j$$

[pedagogical pause]

ARE YOU READY???

I think I'm Ready…

$$\frac{\partial LL(\theta)}{\partial \theta_j}$$

Where

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

This is Sparta!!!!!

This is ~~Sparta~~!!!!!

Stanford

Think About Only One Training Instance

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log[1 - \hat{y}^{(i)}]$$

We only need to calculate the gradient for one training example!

$$\frac{\partial}{\partial x} \sum_i f(x, i) = \sum_i \frac{\partial}{\partial x} f(x, i)$$

We will pretend we only have one example

$$LL(\theta) = y \log \hat{y} + (1 - y) \log[1 - \hat{y}]$$

We can sum up the gradients of each example to get the correct answer

First, imagine only one example

$$LL(\theta) = y \log \hat{y} + (1-y)\log[1-\hat{y}]$$

Where $\hat{y} = \sigma(\theta^T \mathbf{x})$

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \frac{\partial LL(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_j}$$

CHAIN RULZ!

$$= \frac{\partial LL(\theta)}{\partial \hat{y}} \hat{y}(1-\hat{y})x_j$$

Already did that one

$$= \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]\hat{y}(1-\hat{y})x_j$$

Derive this one

$$= (y-\hat{y})x_j$$

Simplify

Now, all the data

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log[1 - \hat{y}^{(i)}]$$

$$\hat{y}^{(i)} = \sigma(\theta^T \mathbf{x}^{(i)})$$

Derivative of sum…

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=0}^{n} \frac{\partial}{\partial \theta_j} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log[1 - \hat{y}^{(i)}] \right]$$

$$= \sum_{i=0}^{n} [y^{(i)} - \hat{y}^{(i)}] x_j^{(i)}$$

See last slide

$$= \sum_{i=0}^{n} [y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)})] x_j^{(i)}$$

Some people don't like hats…

Now, all the data

$$\frac{\partial LL(\theta)}{\partial \theta_j}$$

$$= \sum_{i=1}^{n} [y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)})] x_j^{(i)}$$

Logistic Regression

1 Make logistic regression assumption

$$P(Y=1|X=\mathbf{x}) = \sigma(\theta^T\mathbf{x})$$

$$P(Y=0|X=\mathbf{x}) = 1-\sigma(\theta^T\mathbf{x})$$

2 Calculate the log probability for all data

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma(\theta^T\mathbf{x}^{(i)}) + (1-y^{(i)}) \log[1-\sigma(\theta^T\mathbf{x}^{(i)})]$$

3 Get derivative of log probability with respect to thetas

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T\mathbf{x}^{(i)})\right] x_j^{(i)}$$

The Hard Way

$$LL(\theta) = y \log \sigma(\theta^T \mathbf{x}) + (1-y)\log[1-\sigma(\theta^T \mathbf{x})]$$

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} y \log \sigma(\theta^T \mathbf{x}) + \frac{\partial}{\partial \theta_j}(1-y)\log[1-\sigma(\theta^T \mathbf{x}]$$

$$= \left[\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)}\right]\frac{\partial}{\partial \theta_j}\sigma(\theta^T x)$$

$$= \left[\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)}\right]\frac{\partial}{\partial \theta_j}\sigma(\theta^T x)$$

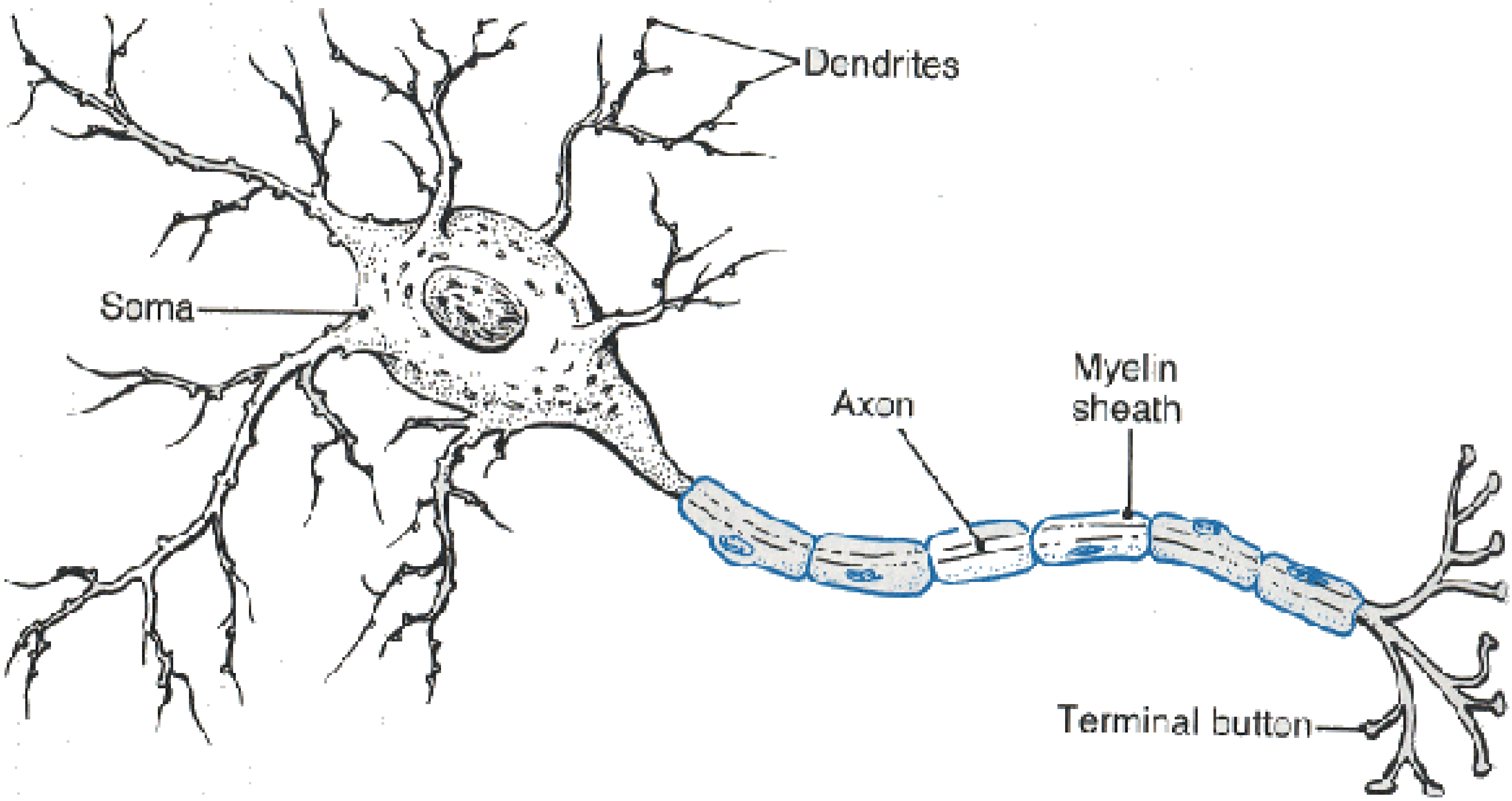
$$= \left[\frac{y-\sigma(\theta^T x)}{\sigma(\theta^T x)[1-\sigma(\theta^T x)]}\right]\sigma(\theta^T x)[1-\sigma(\theta^T x)]x_j$$

$$= \left[y - \sigma(\theta^T x)\right] x_j$$

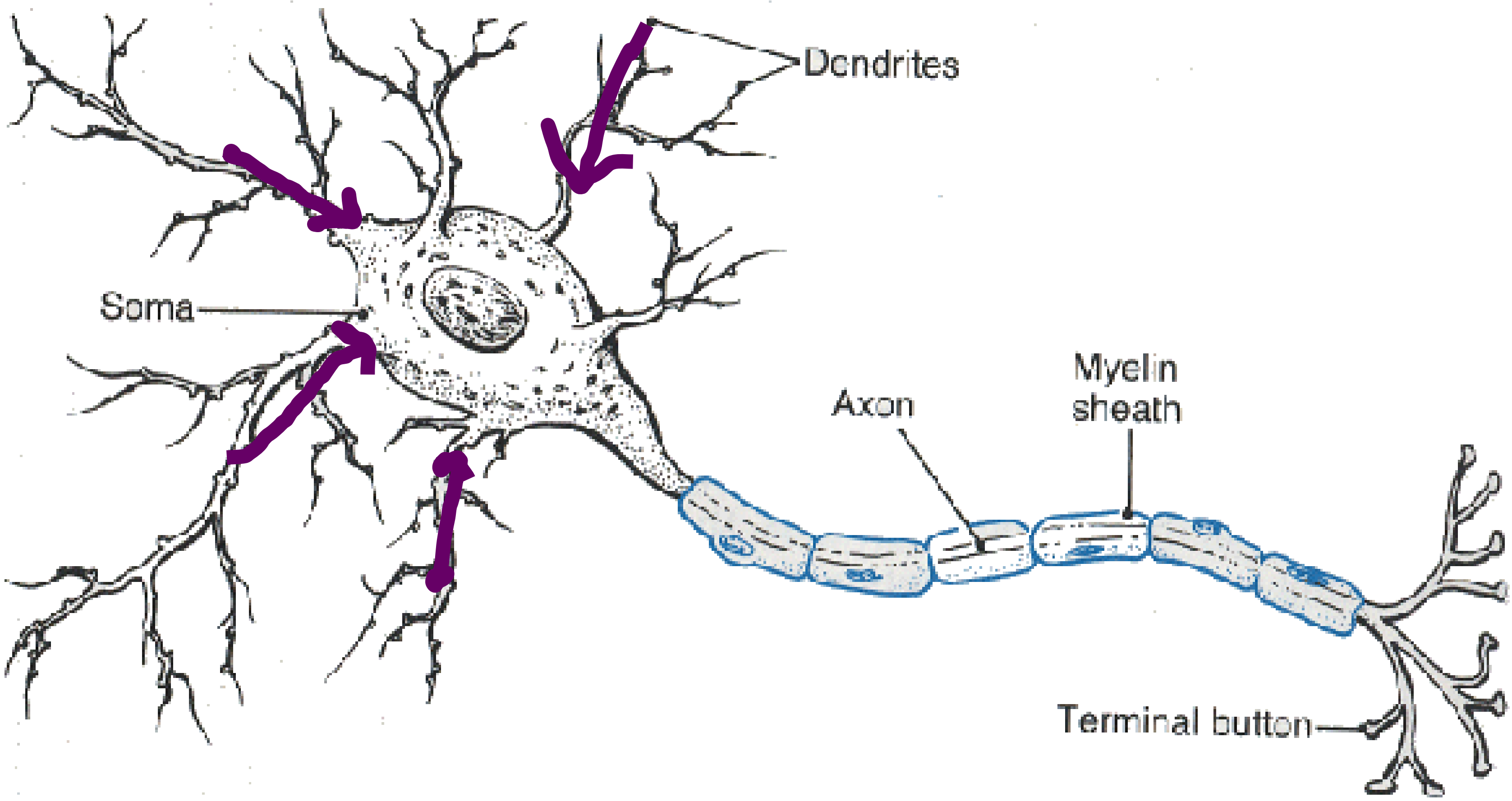
Phew!

Chapter 3: Philosophy
(if time)

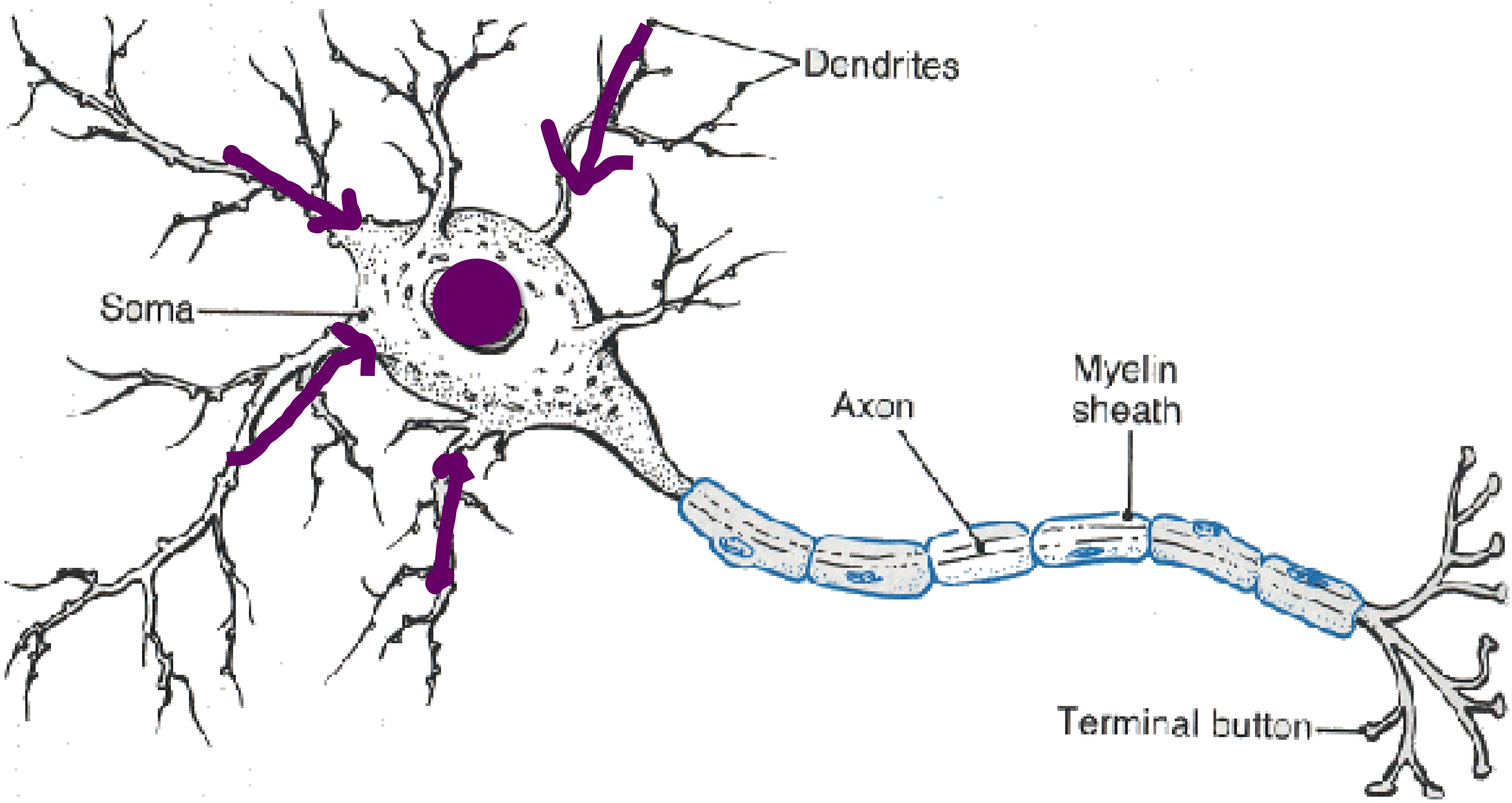
Neuron



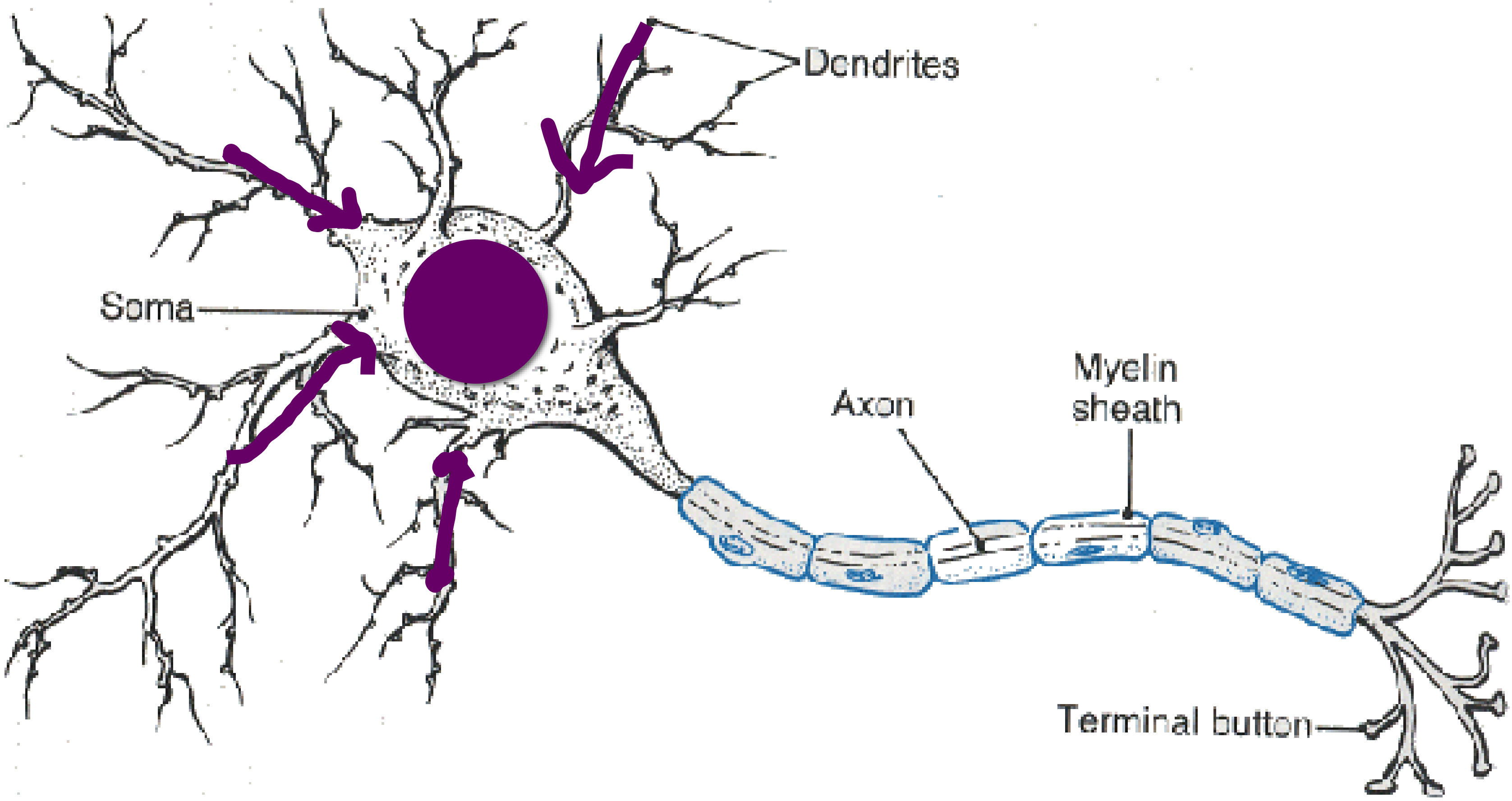
Neuron



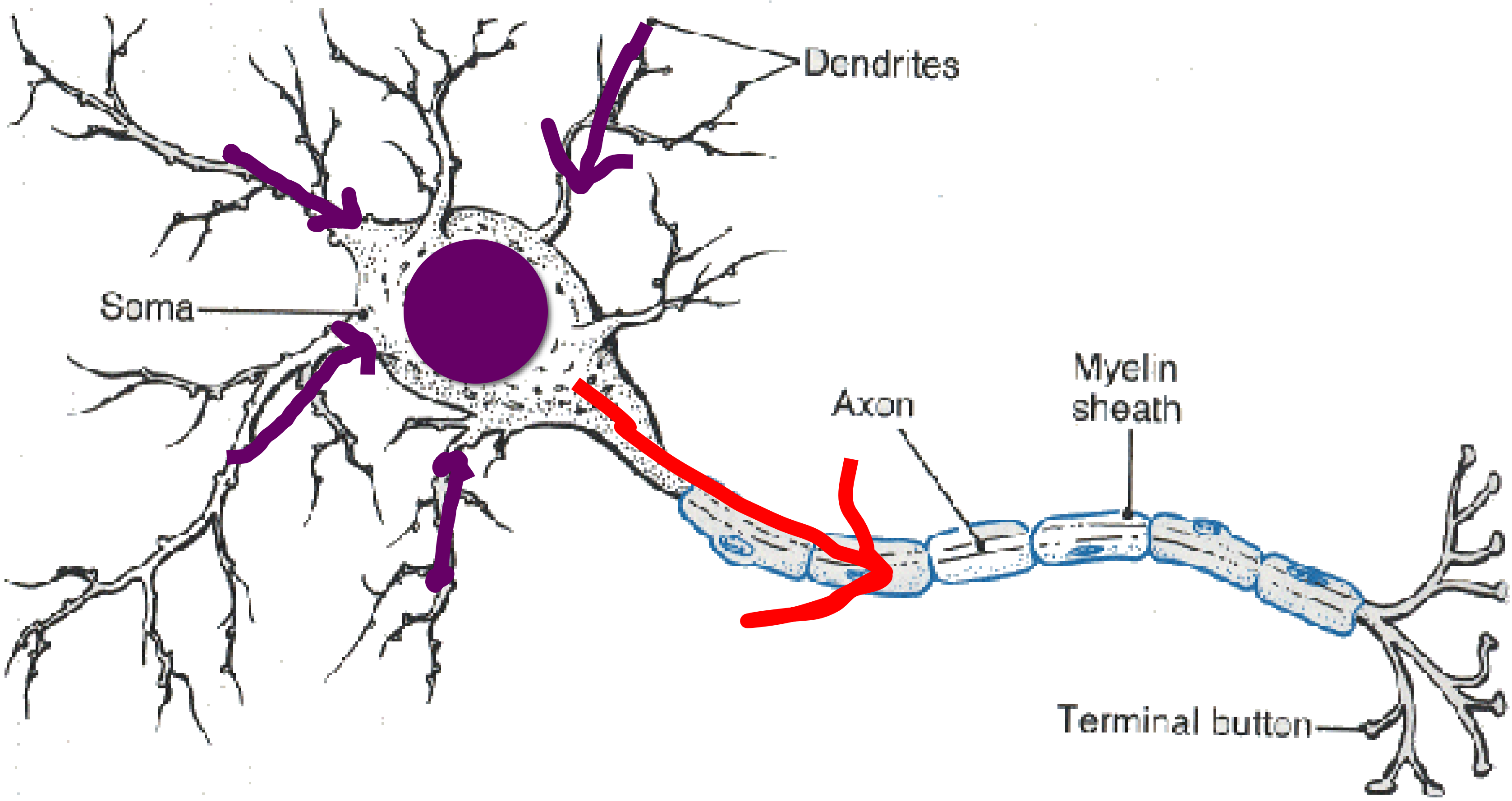
Neuron



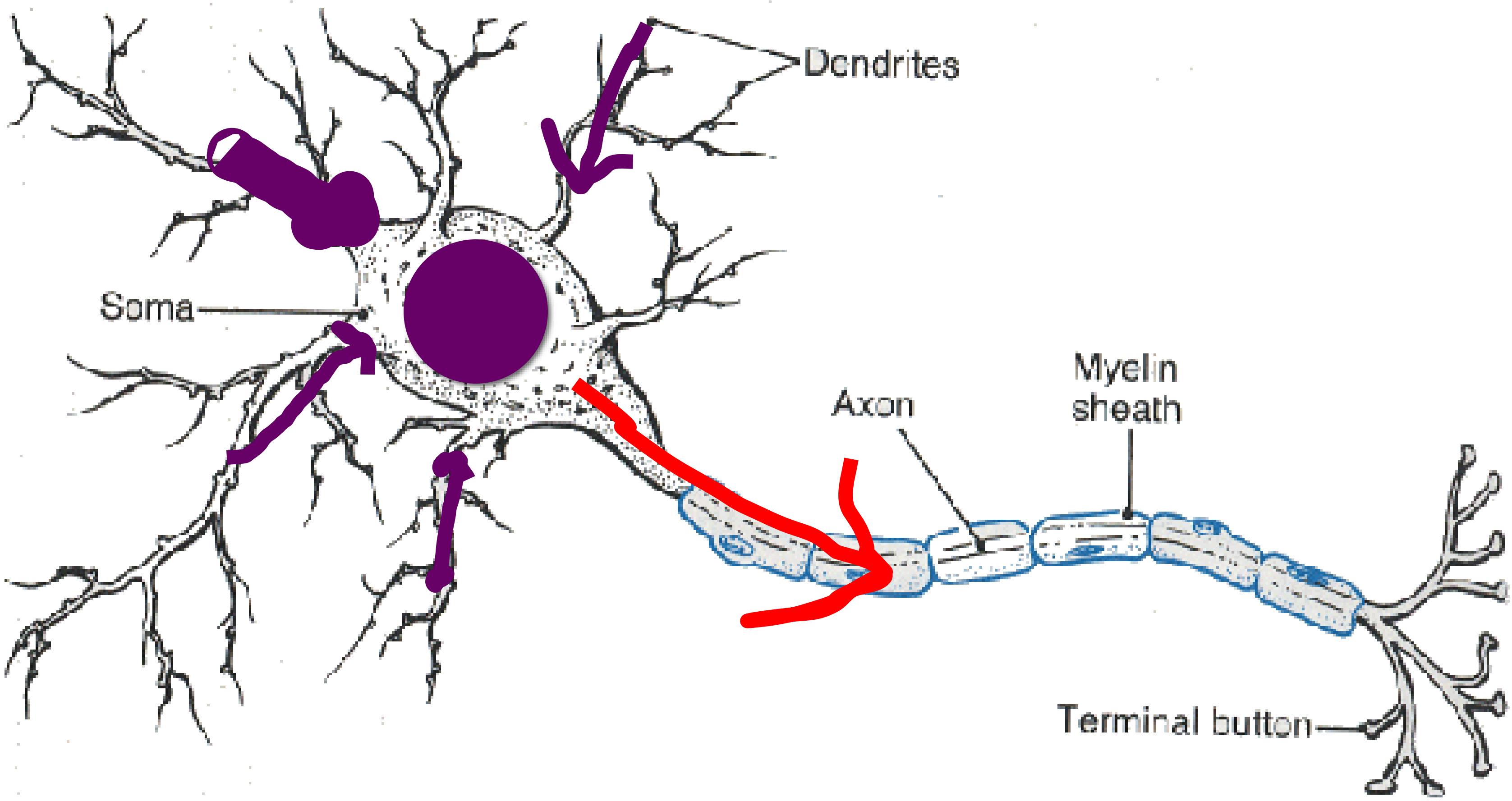
Neuron



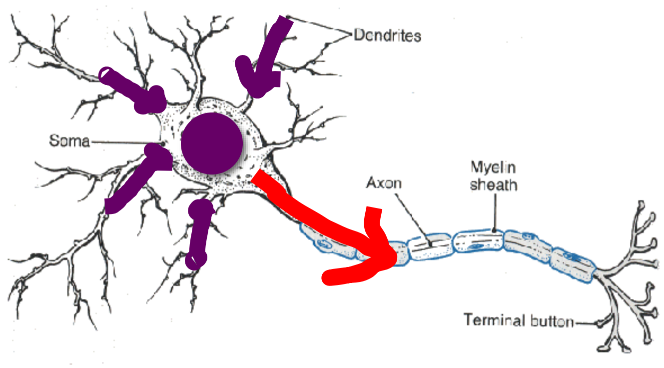
Neuron

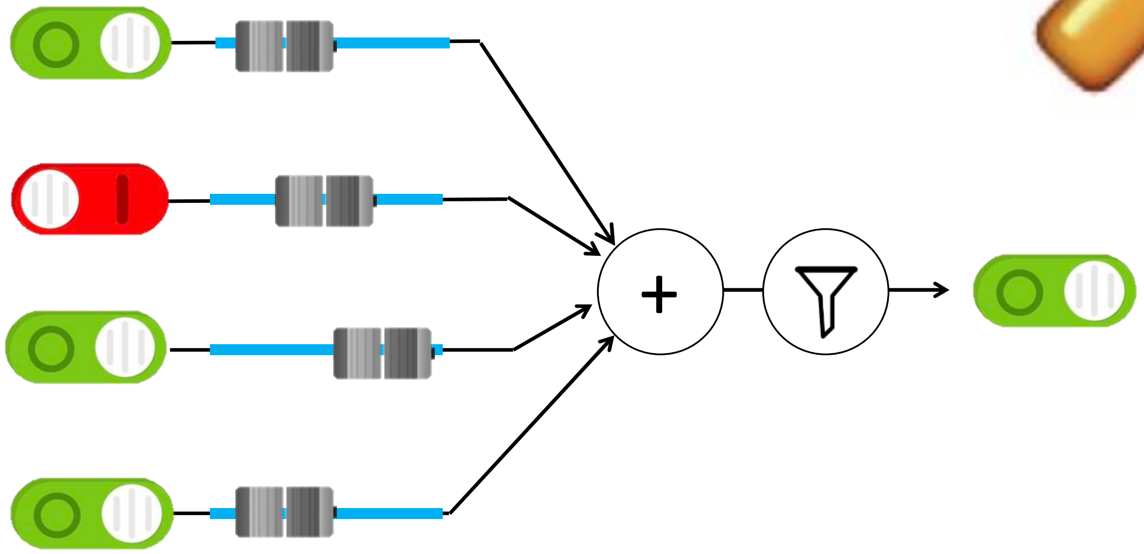


Some inputs are more important



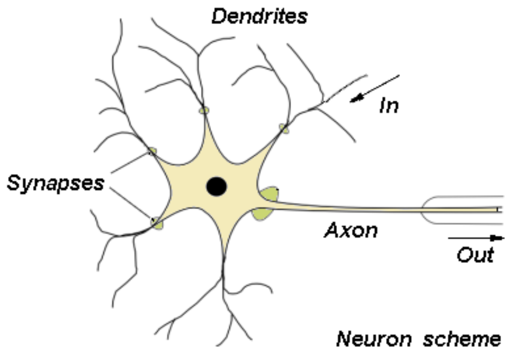
Artificial Neurons

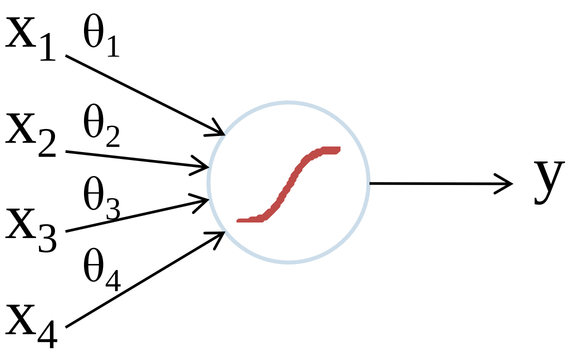




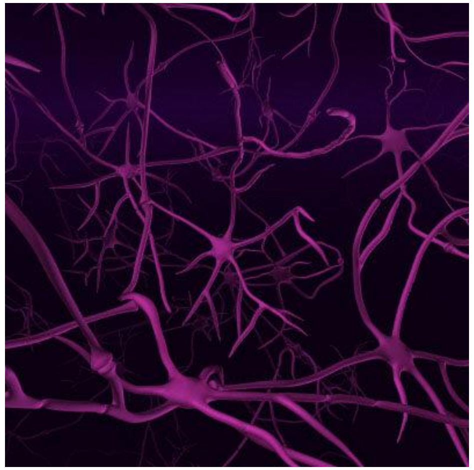
Biological Basis for Neural Networks

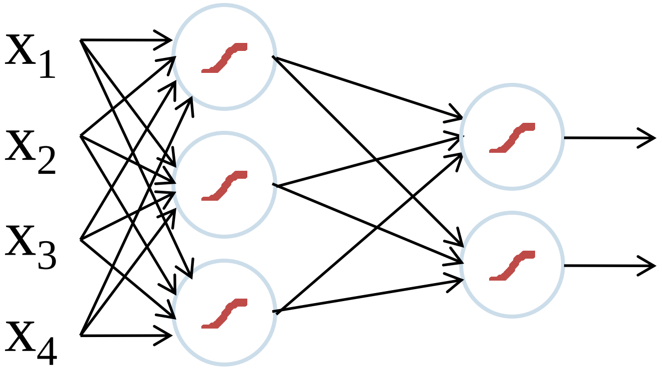
A neuron





Your brain





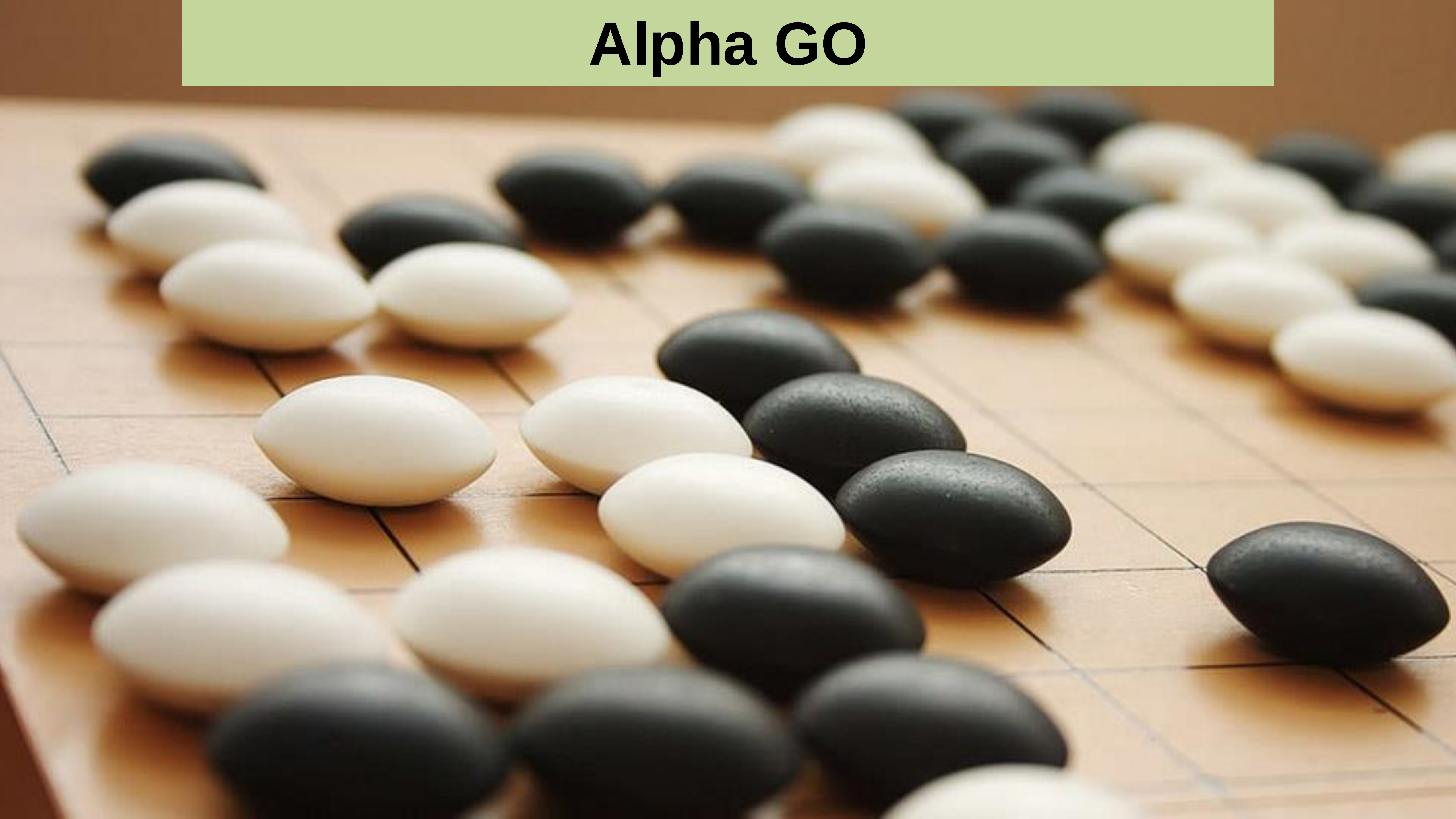
*** Actually, it's probably someone else's brain**

(aka Neural Networks)

Deep learning is (at its core) many logistic regression pieces stacked on top of each other.

Computer Vision

Alpha GO



Revolution in AI

Computers Making Art

Basically just many logistic regression cells
And lots of chain rule…