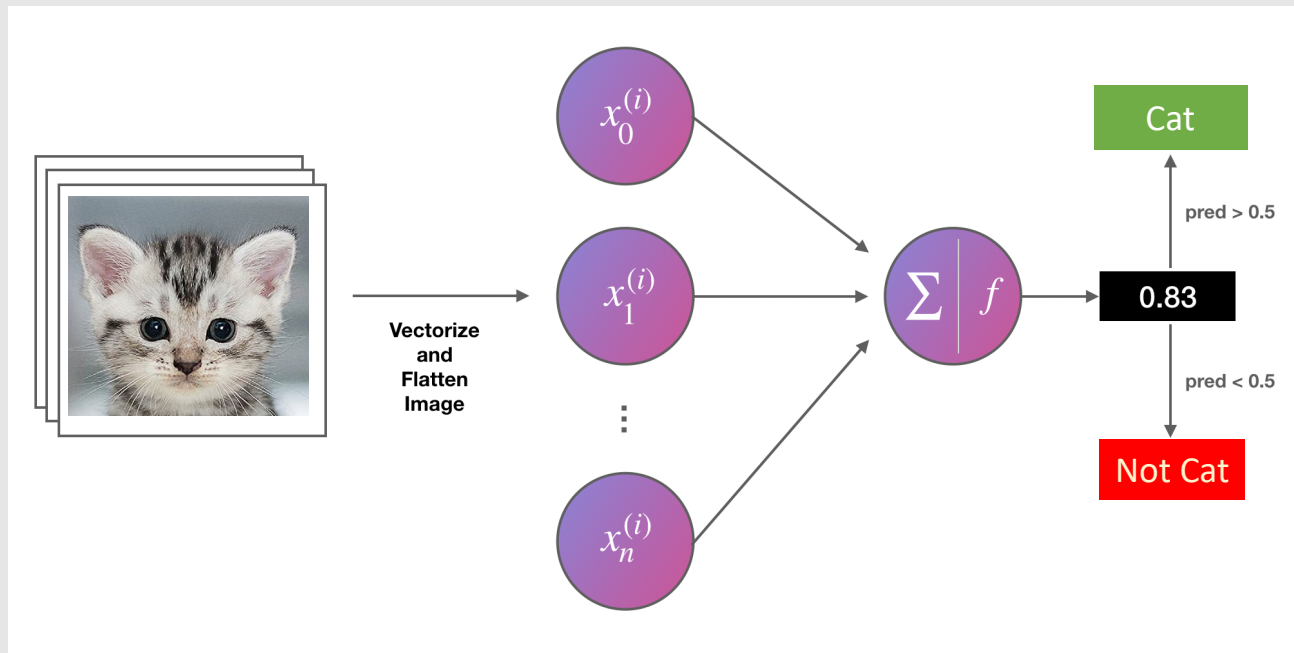


Lecture 6: Logistic Regression & PyTorch for Deep Learning



[Haiping Lu](#)

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/playlists>

[COM4059/6059: MLAI21@The University of Sheffield](#)

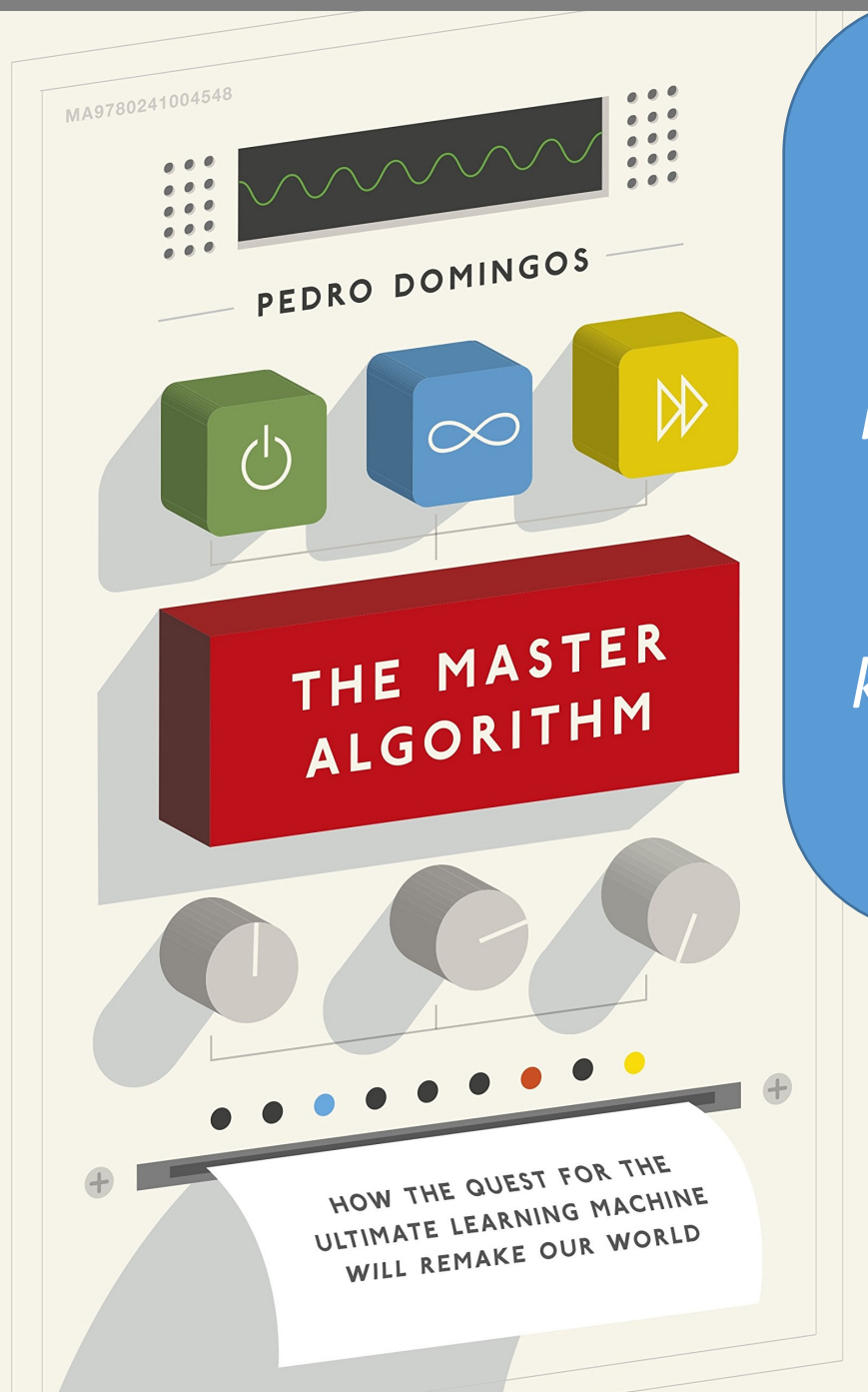
Week 6 Contents / Objectives

- Machine Learning Recap
- Motivation for Logistic Regression
- Logistic Regression
- Computational Graph
- PyTorch: A Deep Learning Library

Week 6 Contents / Objectives

- **Machine Learning Recap**
- Motivation for Logistic Regression
- Logistic Regression
- Computational Graph
- PyTorch: A Deep Learning Library

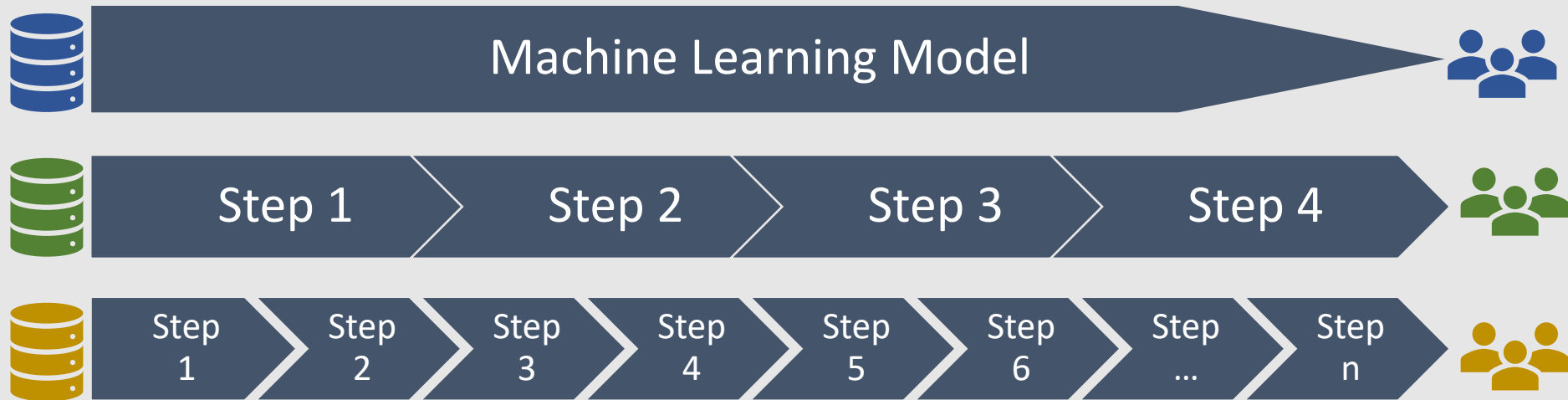
Learning algorithms are the seeds, data is the soil, and the learned programs are the grown plants. The machine-learning expert is like a farmer, sowing the seeds, irrigating and fertilizing the soil, and keeping an eye on the health of the crop but otherwise staying out of the way.



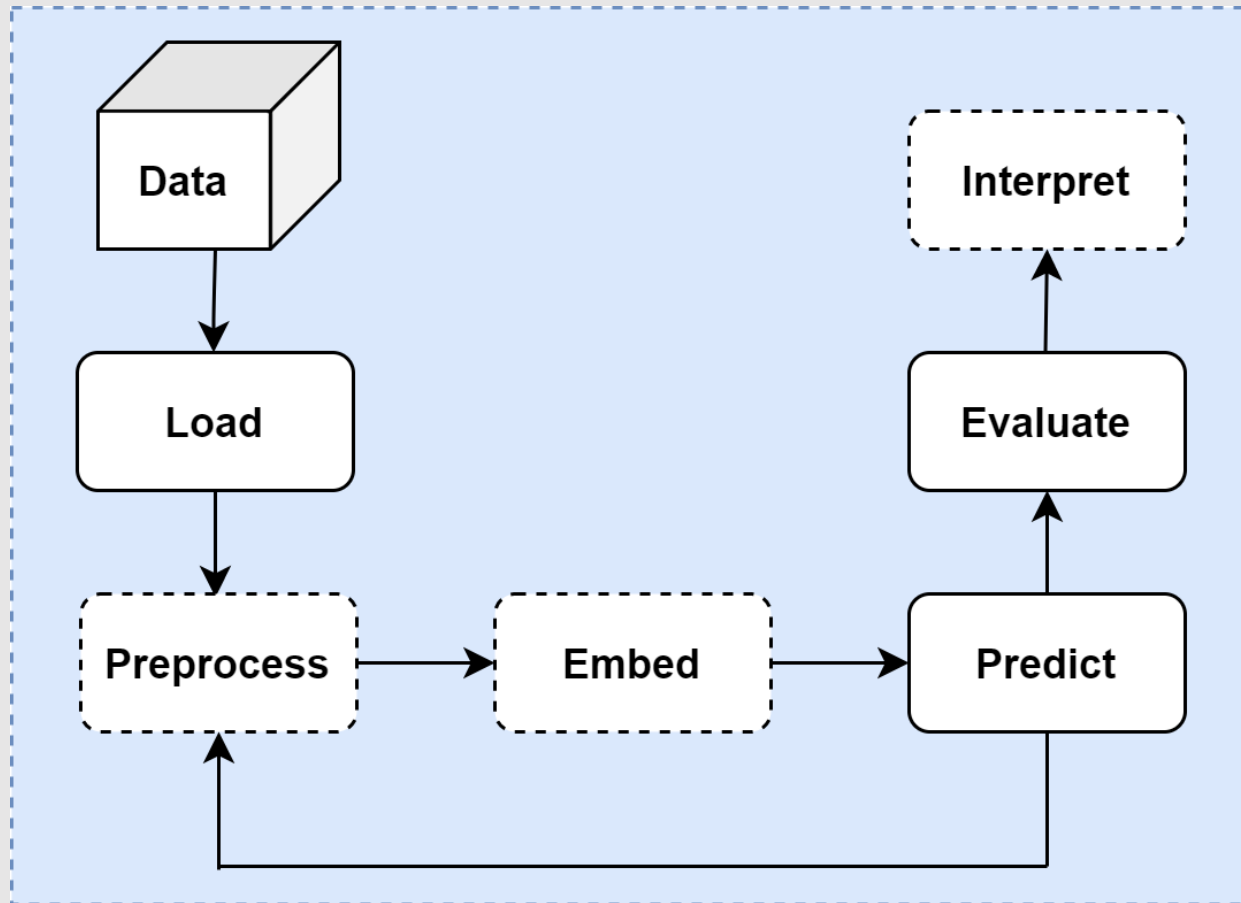
Machine Learning Ingredients

- **Data:** + pre-processing (& visualisation), e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure \sim Architecture \leftarrow expert knowledge
 - Must **specify** before ML, can optimise via cross validation (CV)
 - **Hyper-parameter**, e.g., prior, #degree, layer \leftarrow knowledge
 - Must **specify** (choices) and can optimise via CV (**tuning**)
 - Parameters (theta)
 - Compute/learn parameter, e.g., **weights**, bias \leftarrow optimisation alg.
- Evaluation metric (what's best): loss/error function
- Optimisation: (how to find the best) learnable parameters

ML API without Standardization



Machine Learning Pipeline

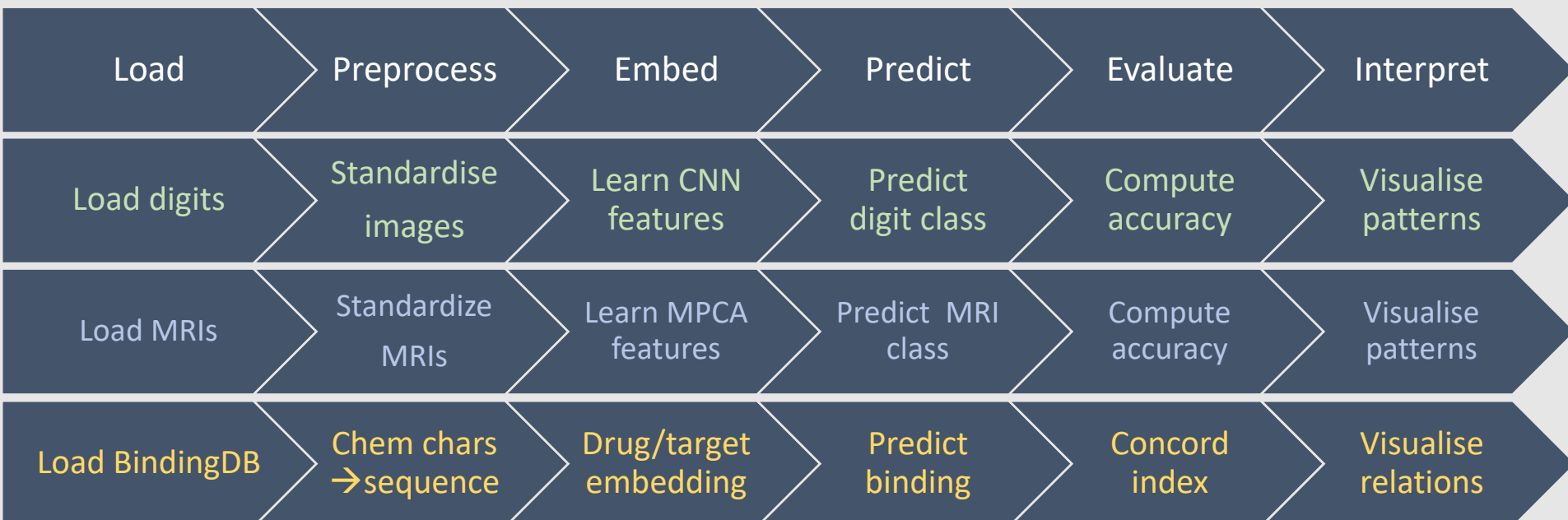


- PyKale: a library defined in this pipeline

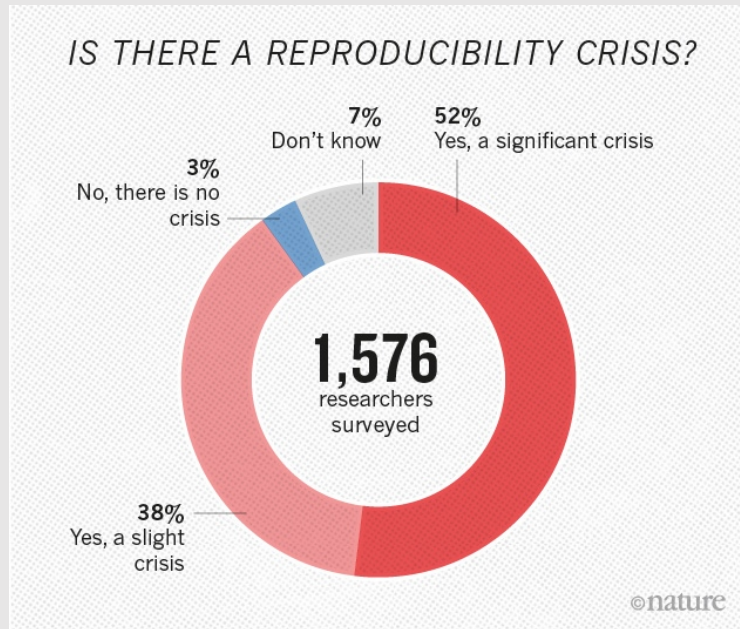


Pipeline-based API in PyKale

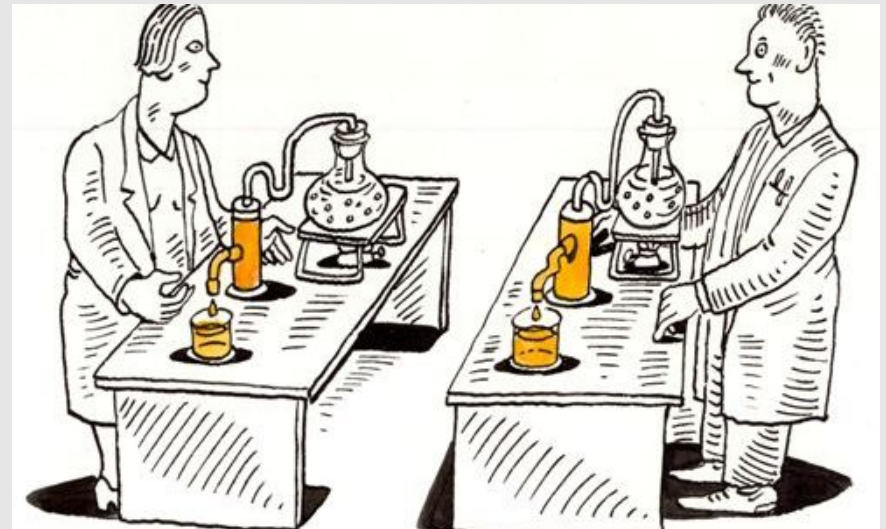
- PyTorch library built by us (week 10):
<https://github.com/pykale/pykale>



Reproducibility → Trustworthy



<https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970>

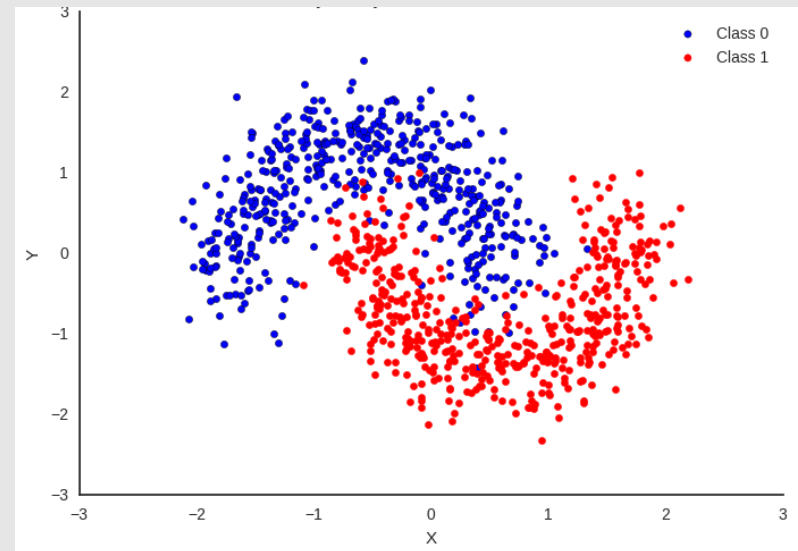
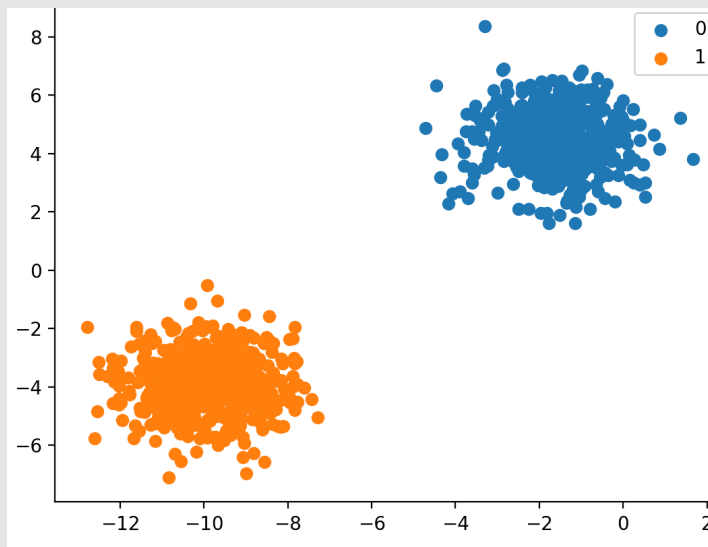


<https://www.ucl.ac.uk/pals/research/experimental-psychology/wp-content/uploads/2016/03/reproducibility-small-496x300.jpg>

- Reproducible machine learning
 - **Make it modular** to help understanding & tracing
 - **Keep a record** of all assumptions and settings
 - **Set a seed** when there is randomness

Start Simple & Small

- The simplest prediction task: binary classification
 - Input (to predict from): feature vectors
 - Output (to predict): 0 or 1
 - Difficulty determined by the distribution of the input



Week 6 Contents / Objectives

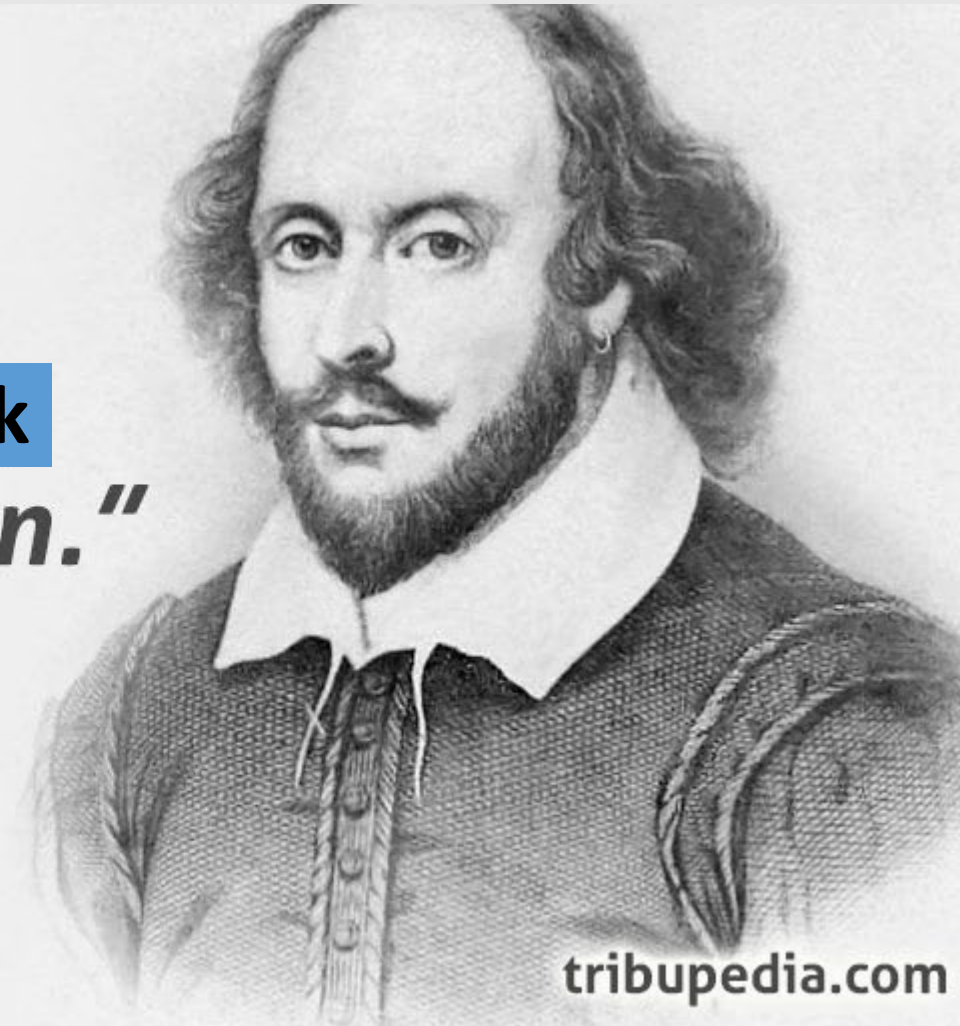
- Machine Learning Recap
- **Motivation for Logistic Regression**
- Logistic Regression
- Computational Graph
- PyTorch: A Deep Learning Library

The Question



***"To click or not to click
that is the question."***

William Shakespeare



tribupedia.com

Click-Through Rate (CTR) Prediction

- Estimating click probabilities: What is the probability that user i **will click** on ad j
 - Not important just for ads:
 - Optimize search results
 - Suggest news articles
 - Recommend products
- Logistic regression is used by many internet companies, making lots of money for them
 - E.g., [Facebook \(*Meta*\) ad matching](#)

A Binary Classification Problem

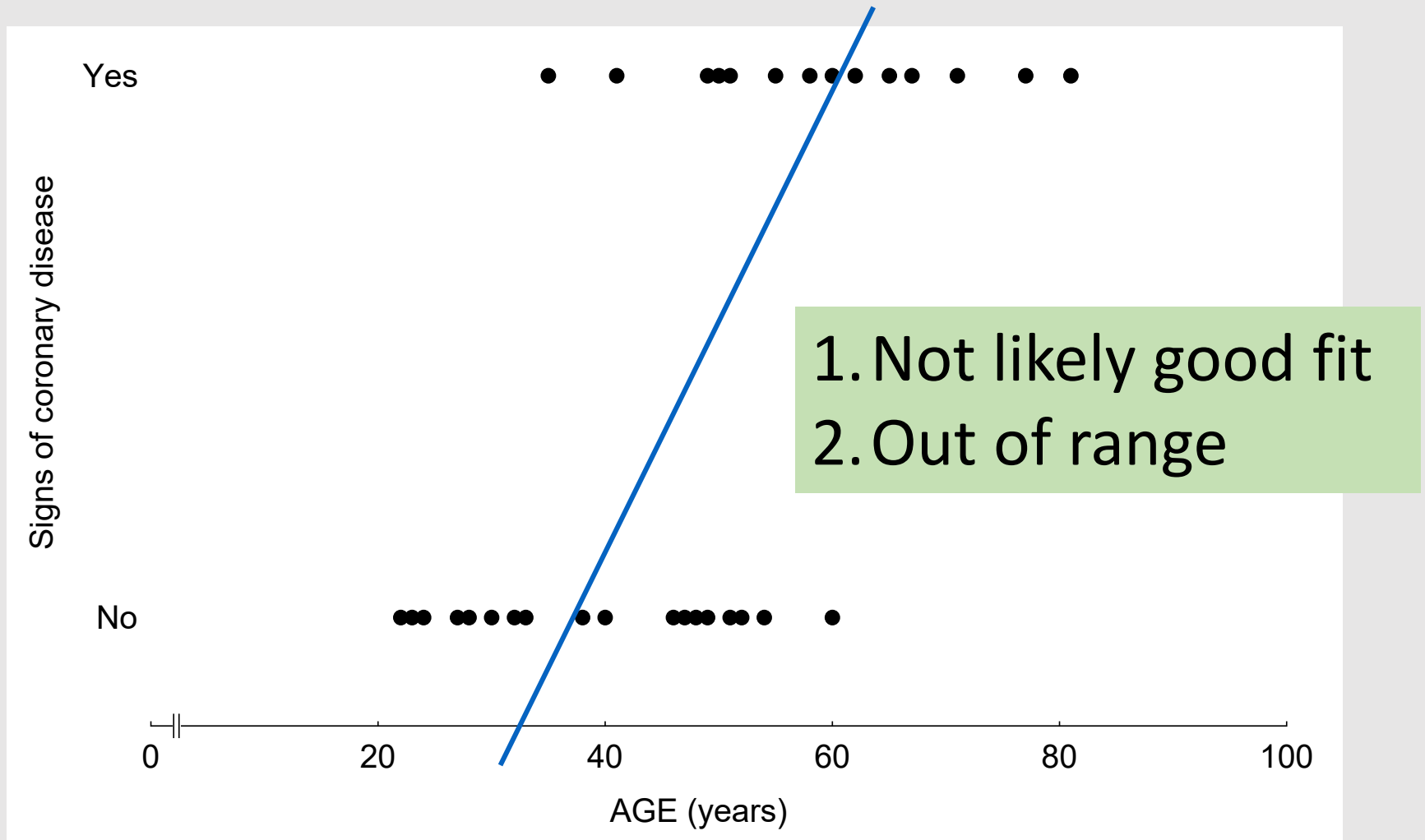
Table 1: Age and signs of coronary heart disease (CD)

Age	CD	Age	CD	Age	CD
22	0	40	0	54	0
23	0	41	1	55	1
24	0	46	0	58	1
27	0	47	0	60	1
28	0	48	0	60	0
30	0	49	1	62	1
30	0	49	0	65	1
32	0	50	1	67	1
33	0	51	0	71	1
35	1	51	1	77	1
38	0	52	0	81	1

Prediction question: a particular age \rightarrow CD

Linear regression?

Dot-plot: Data from Table 1

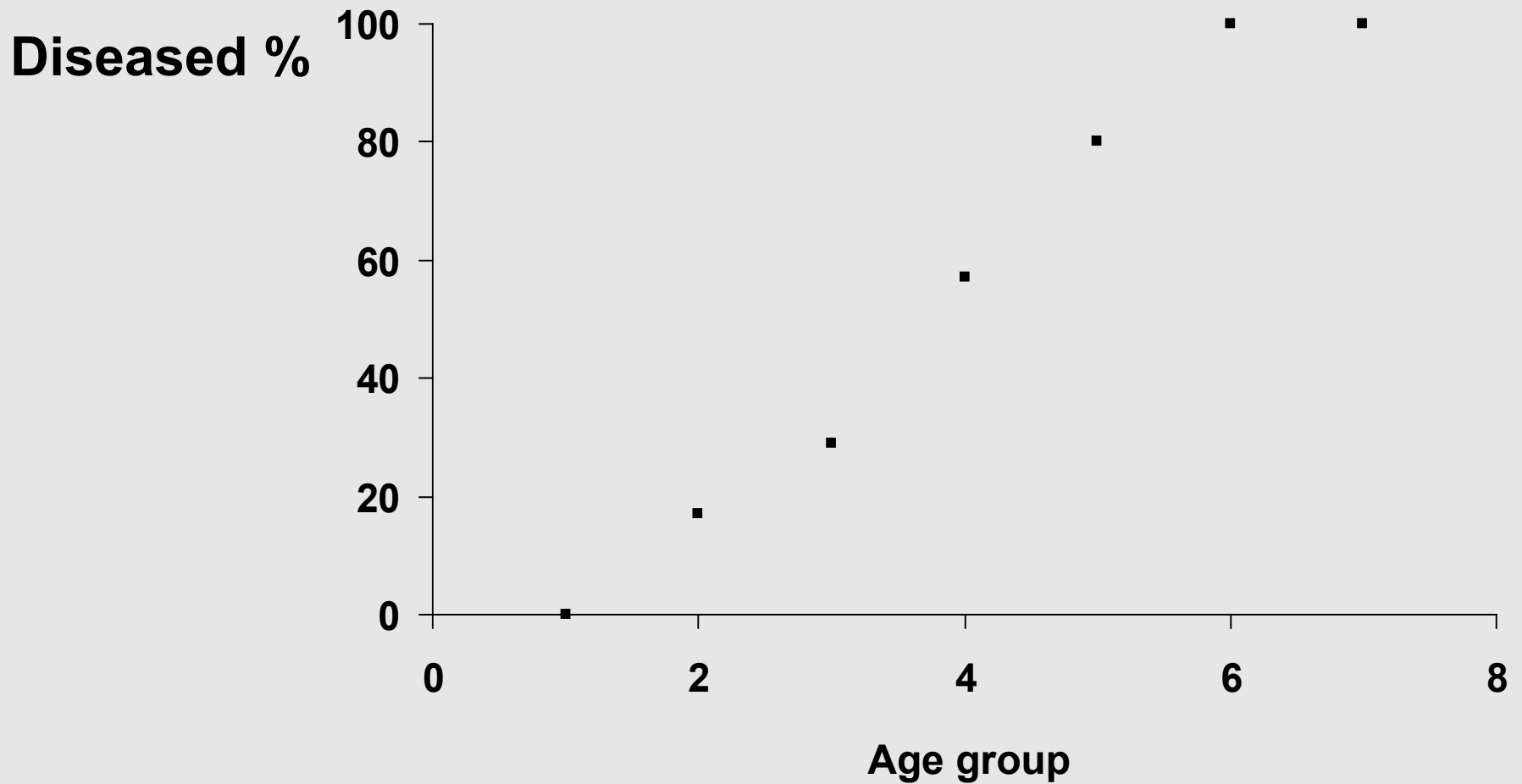


Transform the Data → Probability

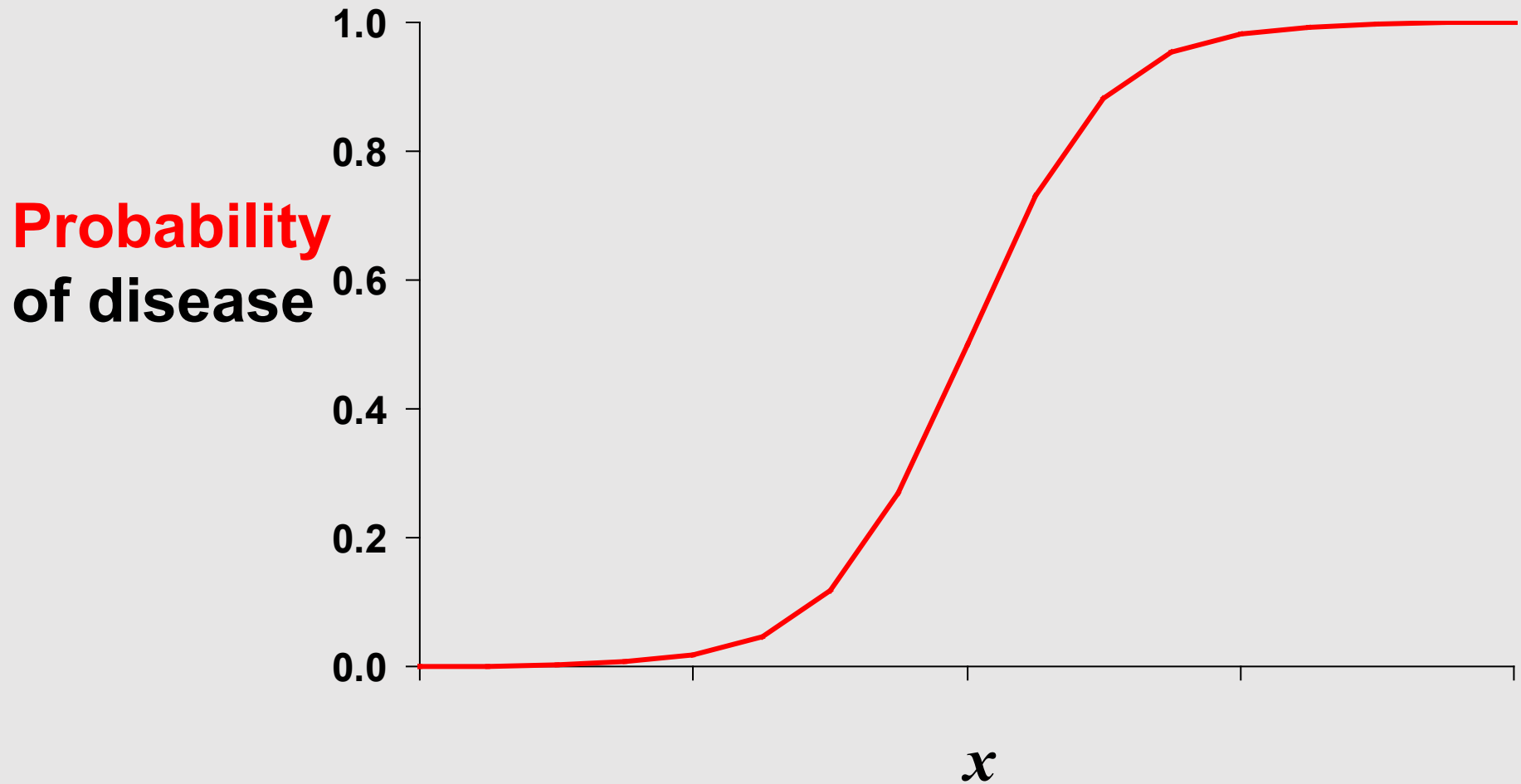
Table 2 Prevalence (%) of signs of CD according to age group

Age group	# in group	Diseased	
		#	%
20 - 29	5	0	0
30 - 39	6	1	17
40 - 49	7	2	29
50 - 59	7	4	57
60 - 69	5	4	80
70 - 79	2	2	100
80 - 89	1	1	100

Dot-plot: Data from Table 2



Logistic Function



Week 6 Contents / Objectives

- Machine Learning Recap
- Motivation for Logistic Regression
- **Logistic Regression**
- Computational Graph
- PyTorch: A Deep Learning Library

Probabilistic Classification

- Training classifiers: estimating $f: X \rightarrow Y$, or $P(Y|X)$
- **Discriminative** classifiers
 - Assume some functional form for $P(Y|X)$
 - Estimate parameters of $P(Y|X)$ directly from training data
- **Generative** classifiers
 - Assume some functional form for $P(X|Y)$, $P(X)$
 - Estimate parameters of $P(X|Y)$, $P(X)$ directly from training data
 - Use Bayes rule to calculate $P(Y|X = x_i)$

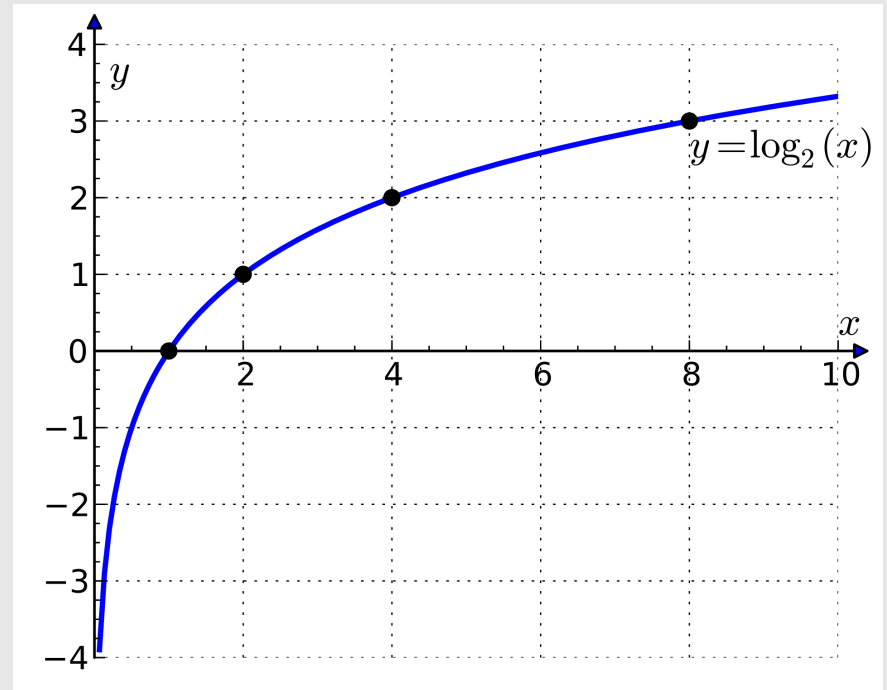
Log Odds

- **Odds**: the ratio of π , the probability of a positive outcome $P(y=1/\mathbf{x})$, to $(1-\pi)$, the probability of a negative outcome $P(y=0/\mathbf{x})$.

$$\frac{\pi}{1 - \pi}$$

- Probability: $[0, 1]$
- \rightarrow Odds: $[0, \infty]$
- \rightarrow Log odds (**logit**): $[-\infty, \infty]$

$$\text{logit}(\pi) = \log \frac{\pi}{1 - \pi}$$



Logit Function → Logistic Function

- Linear **regression** on **logit** function = logistic *regression*

$$\text{logit}(\pi) = \log \frac{\pi}{1 - \pi} = \mathbf{w}^\top \mathbf{x} = w_0 + w_1 x_1 + \dots$$

- More generally, we can use **basis function** as

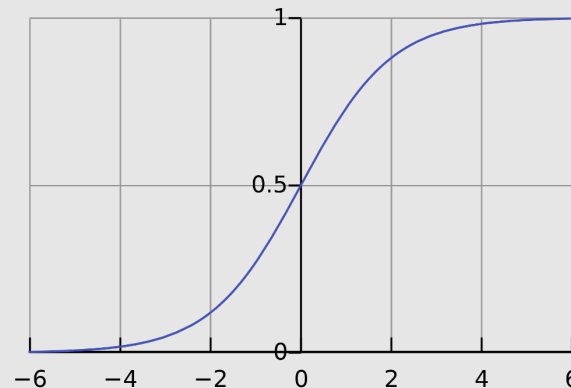
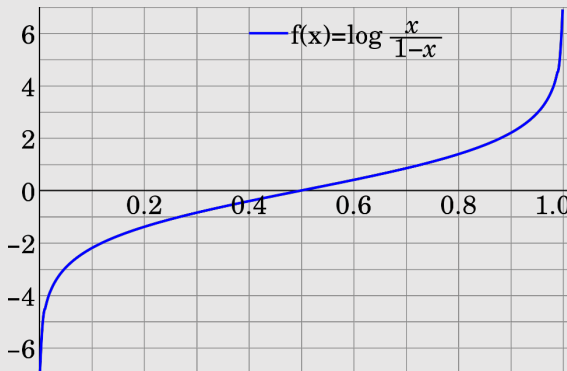
$$\text{logit}(\pi) = \log \frac{\pi}{1 - \pi} = \mathbf{w}^\top \phi(\mathbf{x}) = w_0 + w_1 \phi(x_1) + \dots$$

In the following, we use the simpler first form above

- Logistic function (**sigmoid**)= inverse of logit

$$P(y = 1 | \mathbf{x}) = \text{logit}^{-1}(\mathbf{w}^\top \mathbf{x}) = \text{logistic}(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

- Exercise:** verify the odds using the above equation



How to Estimate \mathbf{w} ? (Learning algo)

- Assumption: Conditional independence of data

- \rightarrow Likelihood:
$$P(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^n P(y_i|\mathbf{x}_i)$$

- Bernoulli distribution for binary classification

- $P(y=1) = \pi$; $P(y=0) = 1 - \pi$ (coin flipping)
- Write the above as a single equation: using y as a switch

$$P(y) = \pi^y (1 - \pi)^{(1-y)} \quad \pi_i = P(y_i = 1|\mathbf{x}_i)$$

- Log likelihood (negative log likelihood \rightarrow cross entropy)

$$\log P(\mathbf{y}|\mathbf{X}) = \sum_{i=1}^n \log P(y_i|\mathbf{x}_i) = \sum_{i=1}^n y_i \log \pi_i + \sum_{i=1}^n (1 - y_i) \log(1 - \pi_i)$$

- MLE: no closed form solution

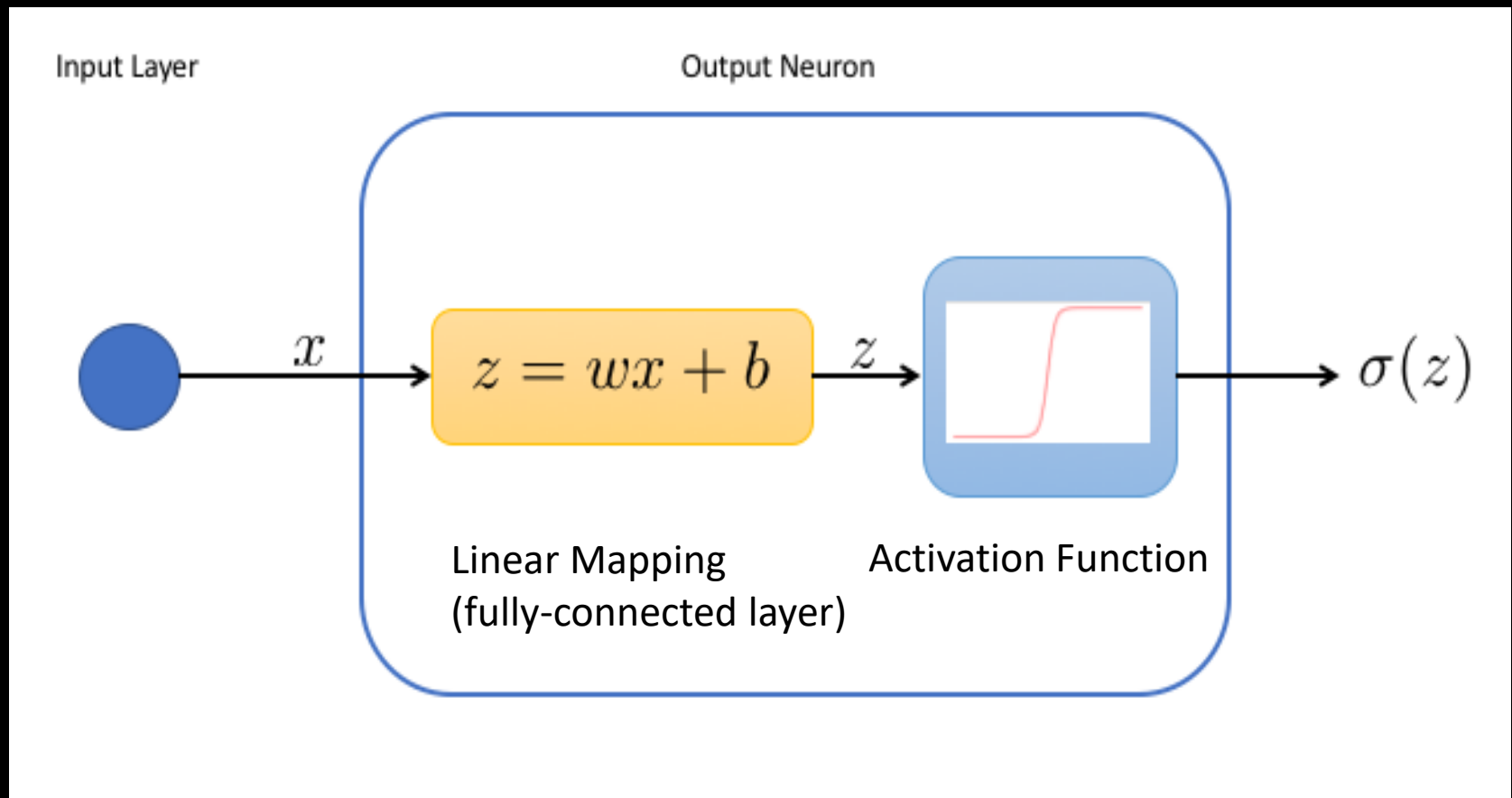
\rightarrow SGD on negative log likelihood (minimisation)

Machine Learning Ingredients

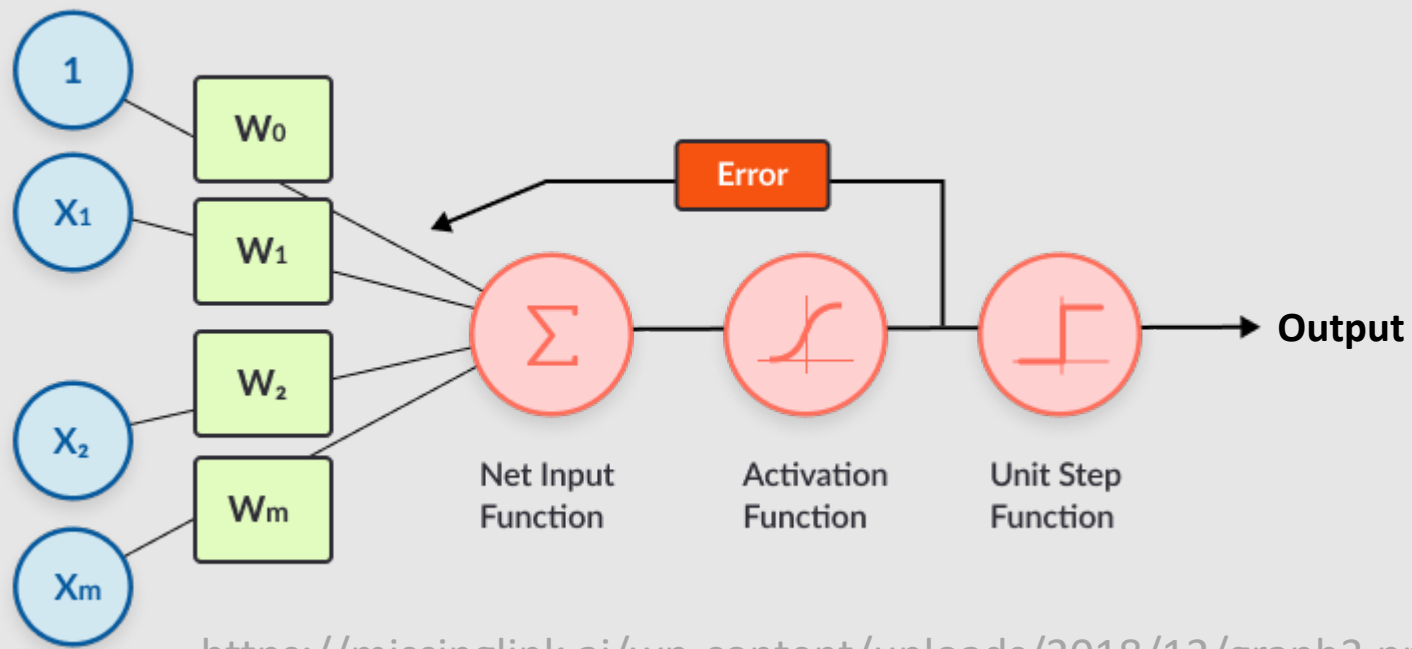
- **Data:** + pre-processing (& visualisation), e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure \sim Architecture \leftarrow expert knowledge
 - Must **specify** before ML, can optimise via cross validation (CV)
 - **Hyper-parameter**, e.g., prior, #degree, layer \leftarrow knowledge
 - Must **specify** (choices) and can optimise via CV (**tuning**)
 - Parameters (theta)
 - Compute/learn parameter, e.g., **weights**, bias \leftarrow optimisation alg.
- Evaluation metric (what's best): loss/error function
- Optimisation: (how to find the best) learnable parameters

Logistic Regression Ingredients

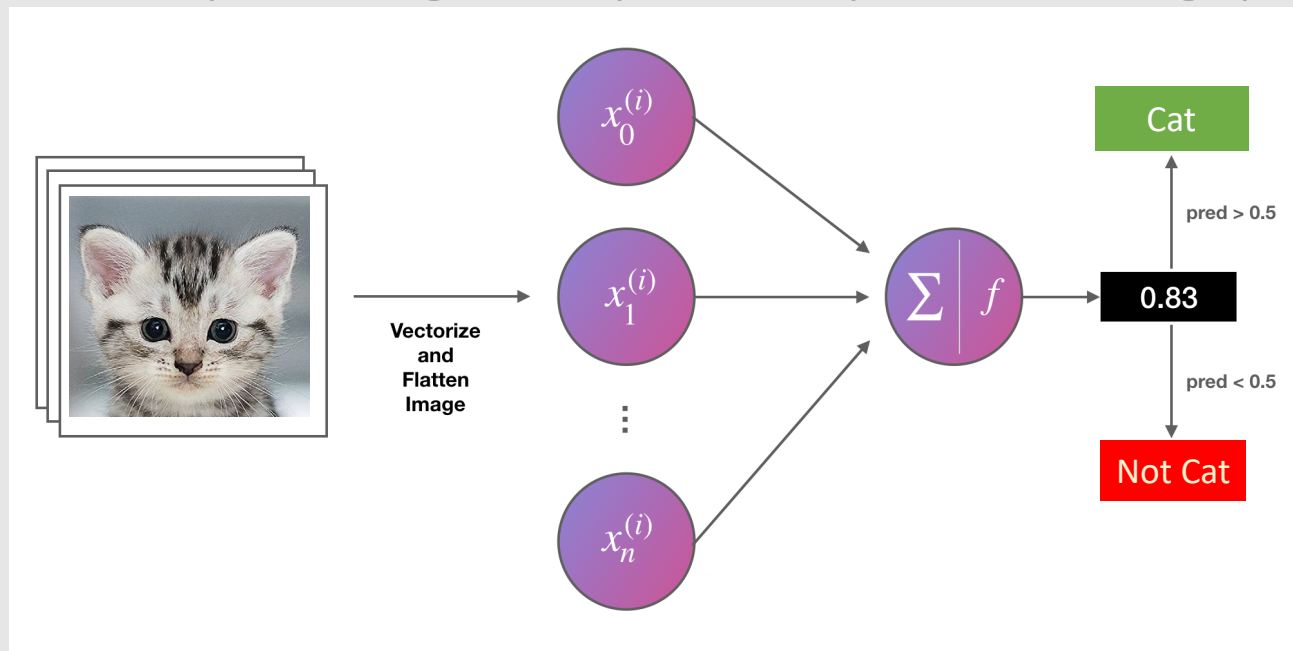
- **Data:** + pre-processing, e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure/Architecture: linear relationship
$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$
 - **Hyper-parameter:** no (unless + regularisation)
 - Parameters (theta): weights and bias
- Evaluation metric: max likelihood (min NLL)
- Optimisation: SGD or the like



Logistic Regression – The Simplest Neural Network



<https://missinglink.ai/wp-content/uploads/2018/12/graph3.png>



https://miro.medium.com/max/4112/1*5NV4NEtgR4rDpSVZkWh3oA.png

Multiclass Classification

- A simple way: one-vs-rest logistic regression
 - Run binary classification for all possible classes
 - Pick the one with the highest value
- More mathematical: multinomial logistic regression, also known as **softmax**
 - Generalise logistic regression to multiple classes
 - Binomial \rightarrow multinomial distribution
 - Sigmoid function \rightarrow softmax function
 - A linear classifier for multiple classes

Summary on Logistic Regression (LR)

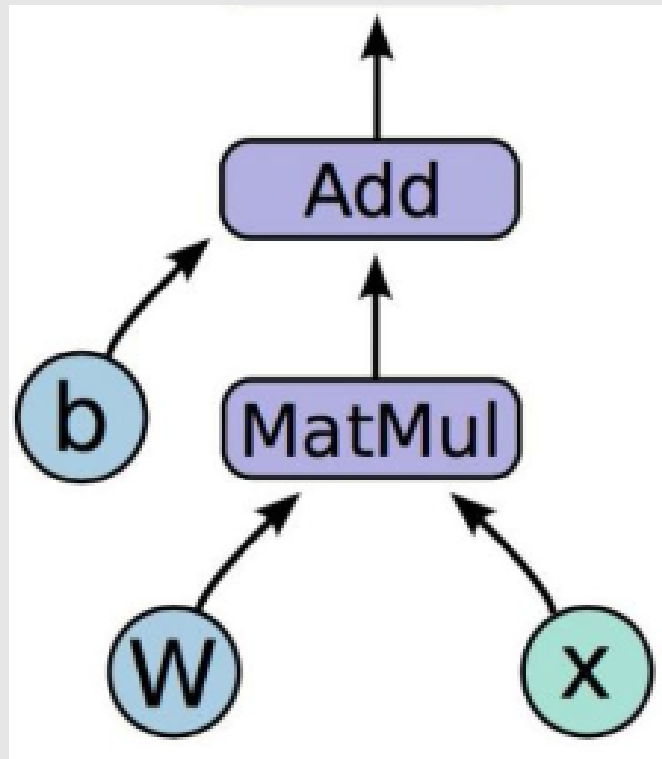
- **Discriminative** classifiers directly model the likelihood $P(Y/X)$
- A simple **linear** classifier that retains a **probabilistic** semantics (see lab)
- Parameters in LR are learned by **iterative** optimization (e.g. SGD), no closed-form solution
- The simplest neural network

Week 6 Contents / Objectives

- Machine Learning Recap
- Motivation for Logistic Regression
- Logistic Regression
- **Computational Graph**
- PyTorch: A Deep Learning Library

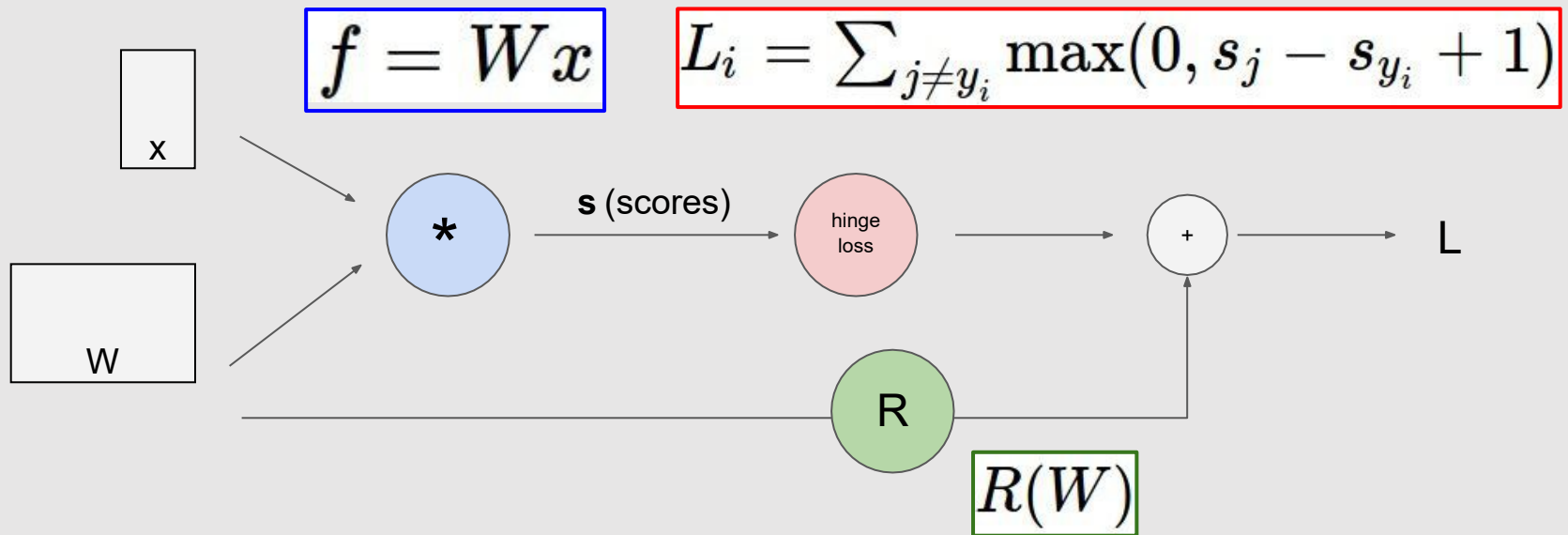
Computational Graph

- Linear regression $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$



Source: Nelson Liu: <https://colab.research.google.com/drive/1iLtGFDpnIuHj5B0rQDGG5lqq6BQ8FRh>

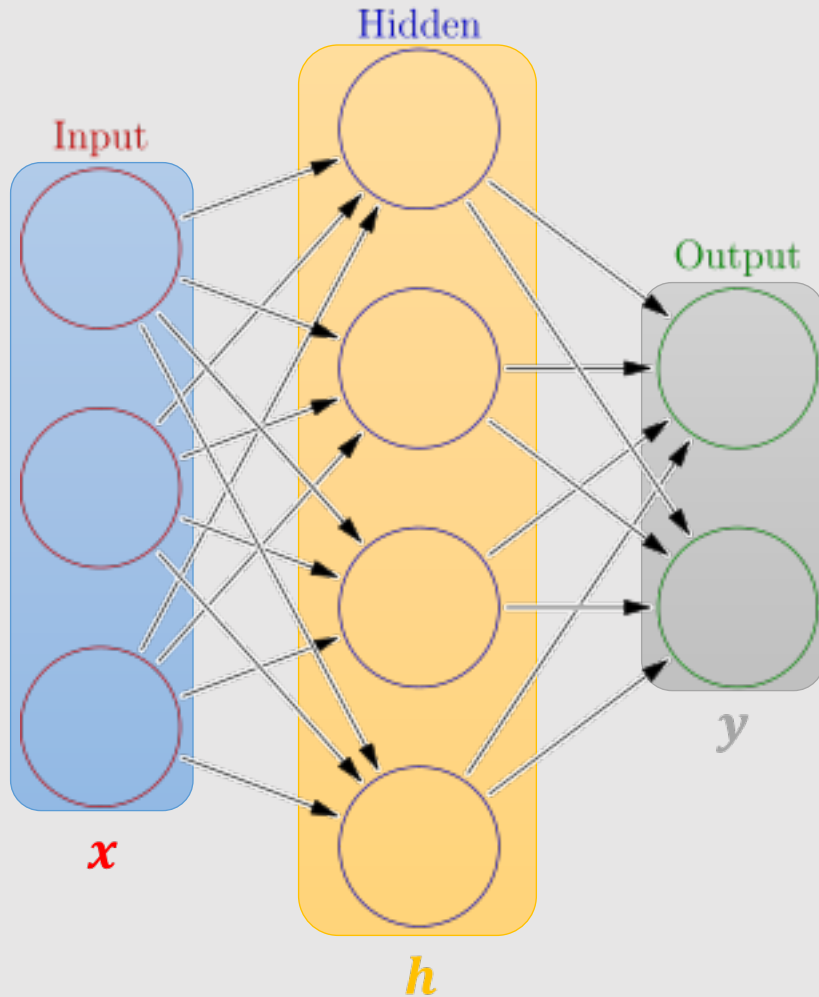
Computational Graph: w/t Reg.



Fei-Fei Li & Justin Johnson & Serena Yeung

2017

Multilayer Perceptron (NN) vs LR



$$h = \sigma(W_1 x + b_1)$$
$$y = \sigma(W_2 h + b_2)$$

Weights

Activation functions



Question:
How many model
parameters?

$$[3 \times 4] + [4 \times 2] = 20 \text{ weights}$$

$$4 + 2 = 6 \text{ biases}$$

26 learnable parameters

4 + 2 = 6 neurons (not counting inputs)

THAT'S NOT ENOUGH



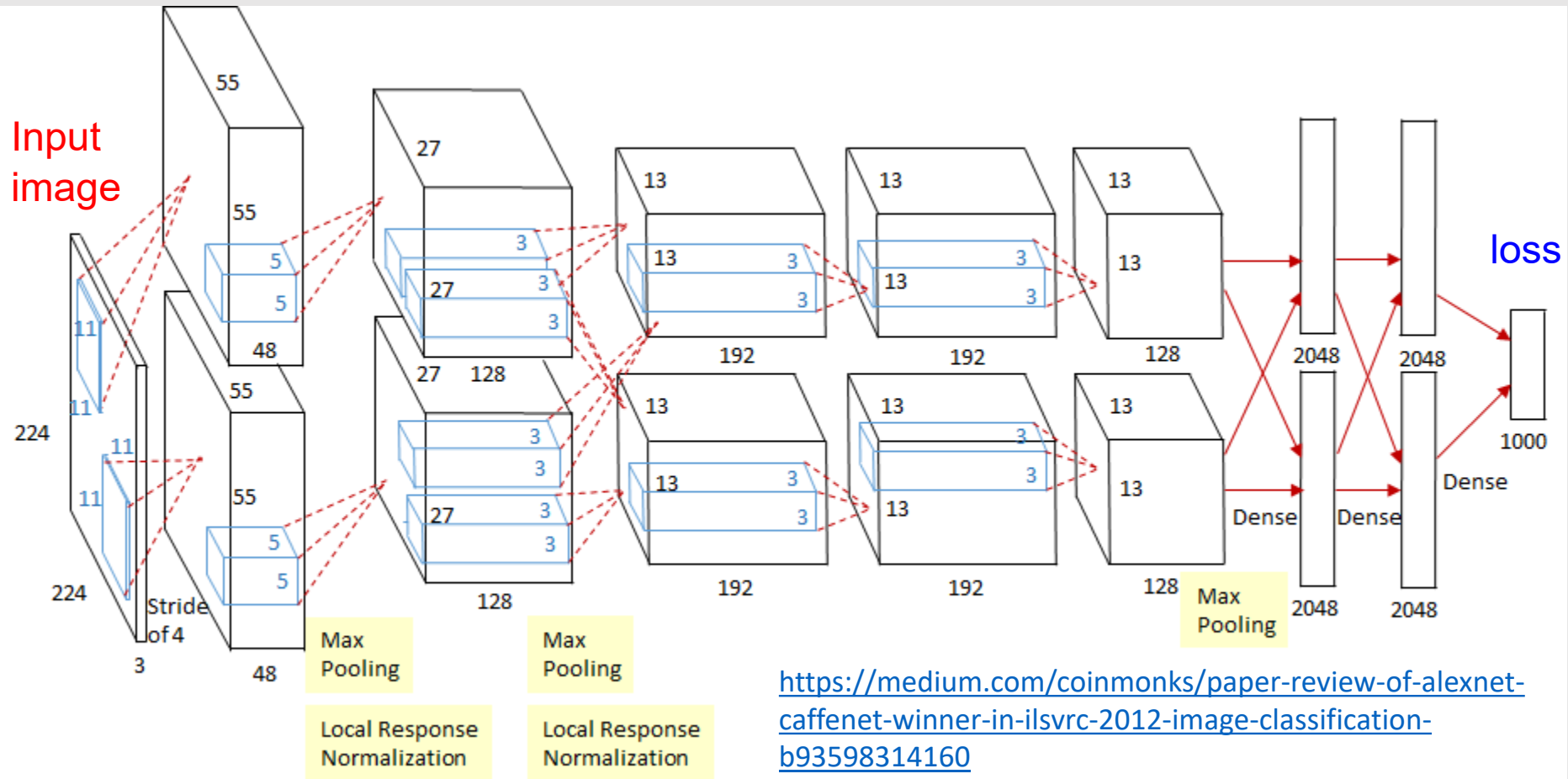
WE HAVE TO GO DEEPER

Source: https://miro.medium.com/max/570/0*y8AuUHSoTGRqX40h.jpeg

quickmeme.com

Computational Graph: DL

Weights (60 million parameters)



ImageNet I

Fancy feature
extraction

010

Logistic
Regression!

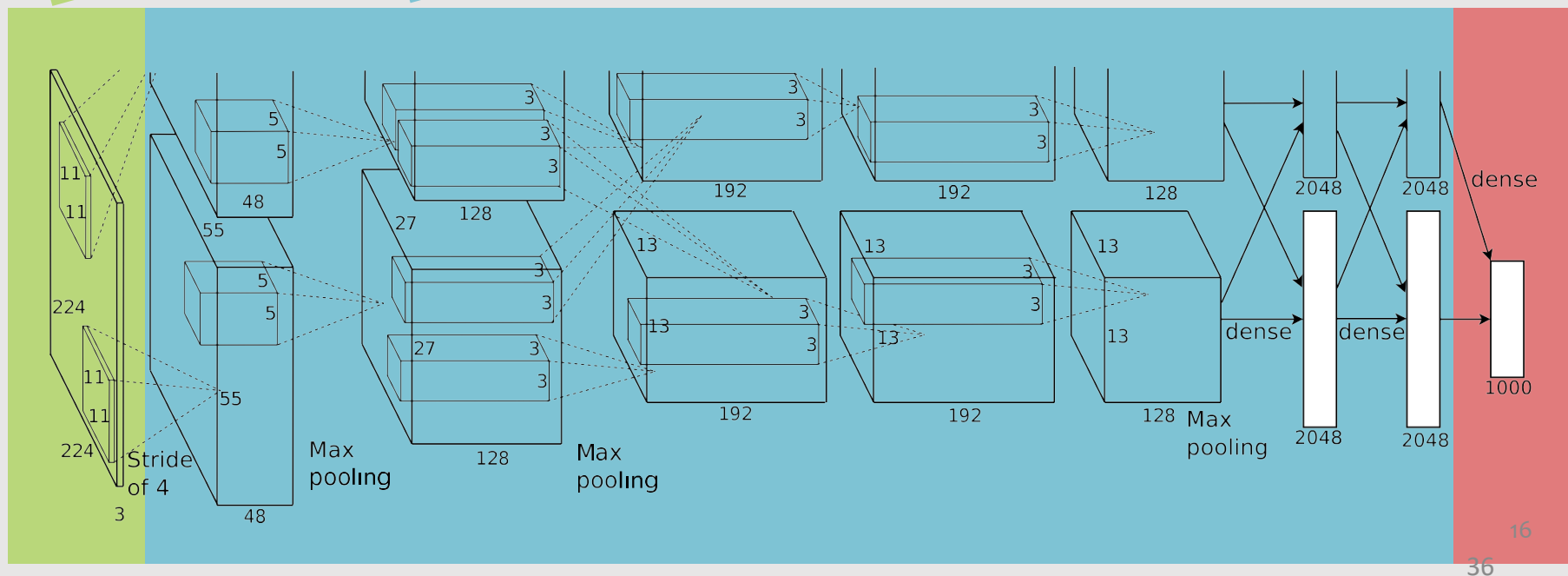
Dataset: 1.2 million /representation 1000 cl
CNN for Image Clas evsky, S
Hinton, 2011) → 17.5% error

Softmax: sigmoid
for multiclass

Input
image
(pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way
softmax



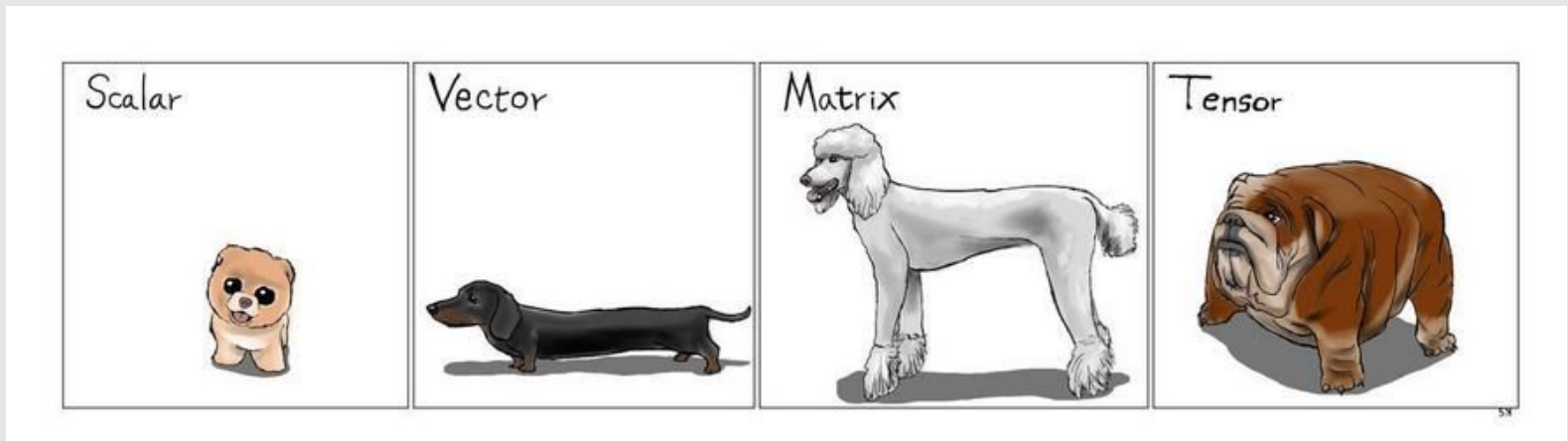
Week 6 Contents / Objectives

- Machine Learning Recap
- Motivation for Logistic Regression
- Logistic Regression
- Computational Graph
- **PyTorch: A Deep Learning Library**

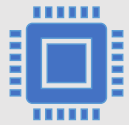


PyTorch

- An open source deep learning library by Facebook
 - **Tensor** computing with GPU acceleration
 - Deep neural networks built on autodiff
- **torch.Tensor**
 - multidimensional data structures/arrays for programming
 - Scalar: 0-D tensor; Vector: 1-D tensor; Matrix: 2-D tensor



<https://pbs.twimg.com/media/D2xTDMJWwAADHQt.jpg>



Key Modules in PyTorch

- **torch.autograd**
 - Automatic differentiation. A recorder records what operations have performed, and then it replays it backward to compute the gradients.
- **torch.optim**
 - Implementation of various optimization algorithms used for building neural networks (and other ML algorithms).
- **torch.nn**
 - High-level definition of the **computational graphs (architecture)** of complex neural networks (and other ML algorithms)

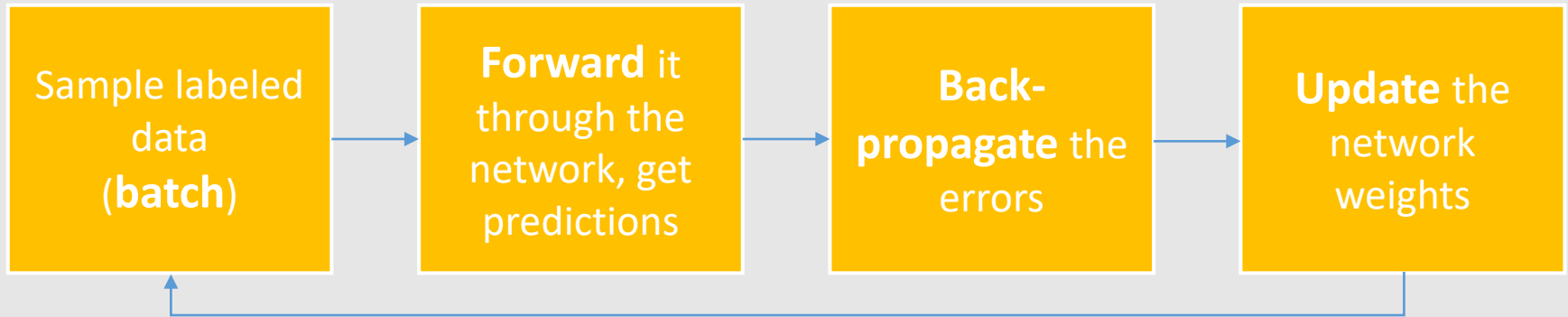
Dynamic Computational Graph

A graph is created on the fly

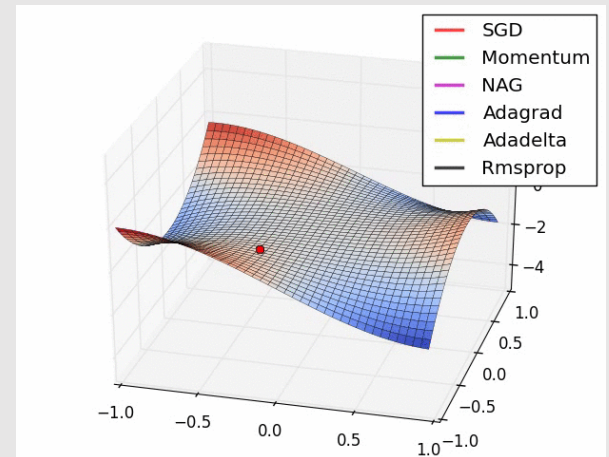
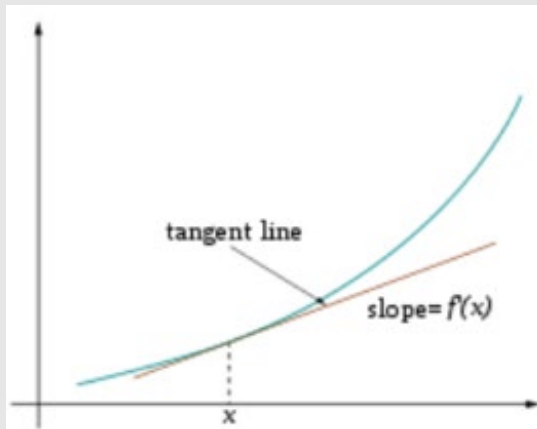
```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```



Training



Optimize (min. or max.) **objective/cost function $J(\theta)$**
Generate **error signal** that measures difference between predictions and target values



Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

<https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39>

Acknowledgement

- The slides used materials from:
Colin Bernet, Ismini Lourentzou, Fei-Fei Li & Justin Johnson & Serena Yeung, Rui Zhang, Nelson Liu, Matt Gormley, Rachid Salmi, Jean-Claude Desenclos, Thomas Grein, Alain Moren, Christophe Giraud-Carrier, Bart Selman, Sham Kakade, Raymond J. Mooney, Neil Lawrence, and Andrew Ng



Recommended Reading

- [Notes Logistic Regression by Andrew Ng](#)
- Wikipedia entries on topics, e.g. multiclass classification, softmax, multinomial logistic regression,
- PyTorch documentations
- The lab notebook and references