10-301/601: Introduction to Machine Learning Lecture 30: Course Recap & Large Language Models

Henry Chai

8/9/23

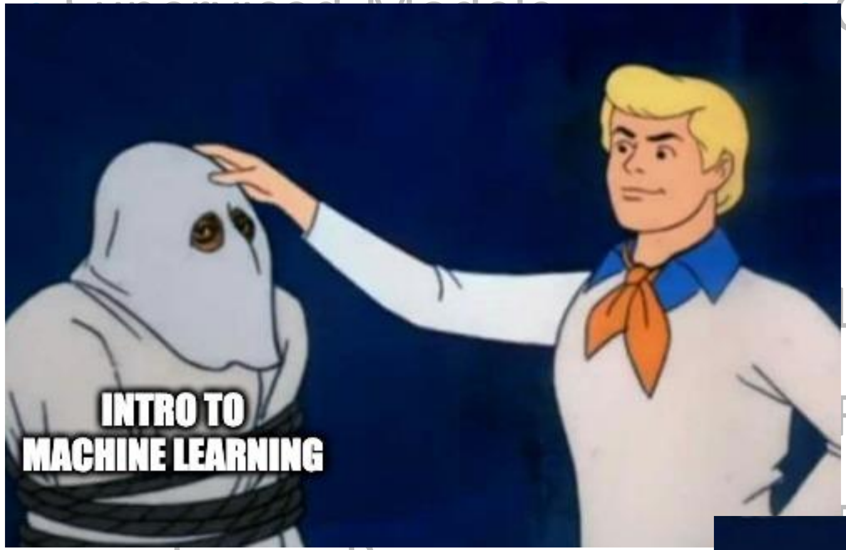
Front Matter

- Announcements
  - Final on 8/11, this Friday!
    - Today's lecture is out-of-scope for the Final
    - OH in lieu of recitation on 8/10 (tomorrow)
  - Please complete your course evals!
- Recommended Supplementary Material
  - Papers linked throughout the lecture slides

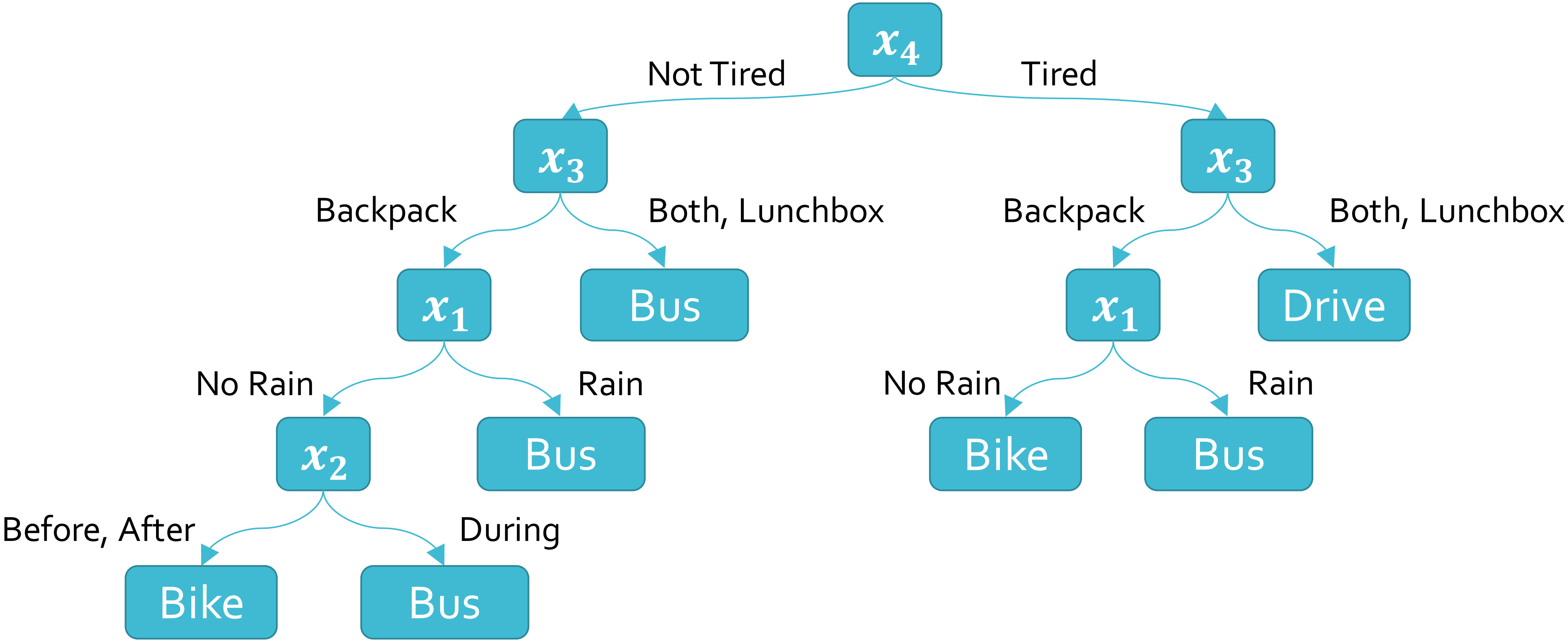
Recall: What is ~~Machine Learning~~ 10-301/601?

- Supervised Models
  - Decision Trees
  - KNN
  - Naïve Bayes
  - Perceptron
  - Logistic Regression
  - Linear Regression
  - Neural Networks
- Deep Learning
- Unsupervised Models
  - K-means
  - PCA
- Graphical Models
  - Bayesian Networks
  - HMMs
- Learning Theory
- Reinforcement Learning
- Ensemble Methods
- Important Concepts
  - Feature Engineering
  - Regularization and Overfitting
  - Experimental Design

It was all a ruse!



- Linear Regression
- Neural Networks
- Deep Learning
- Unsupervised Models
  - K-means
  - PCA
- Graphical Models
  - Bayesian Networks
  - HMMs
- Learning Theory
- Reinforcement Learning


x_4
Not Tired
Tired
x_3
x_3
Backpack
Both, Lunchbox
Backpack
Both, Lunchbox
x_1
Bus
x_1
Drive
No Rain
Rain
No Rain
Rain
x_2
Bus
Bike
Bus
Before, After
During
Bike
Bus

Decision Trees: Inductive Bias

- The **inductive bias** of a machine learning algorithm is the principal by which it generalizes to unseen examples
- What is the inductive bias of the ID3 algorithm i.e., decision tree learning with mutual information maximization as the splitting criterion?
  - Try to find the smallest tree that achieves a **training error rate of 0** with high mutual information features at the top
- Occam's razor: try to find the "simplest" (e.g., smallest decision tree) classifier that explains the training dataset

Overfitting in Decision Trees

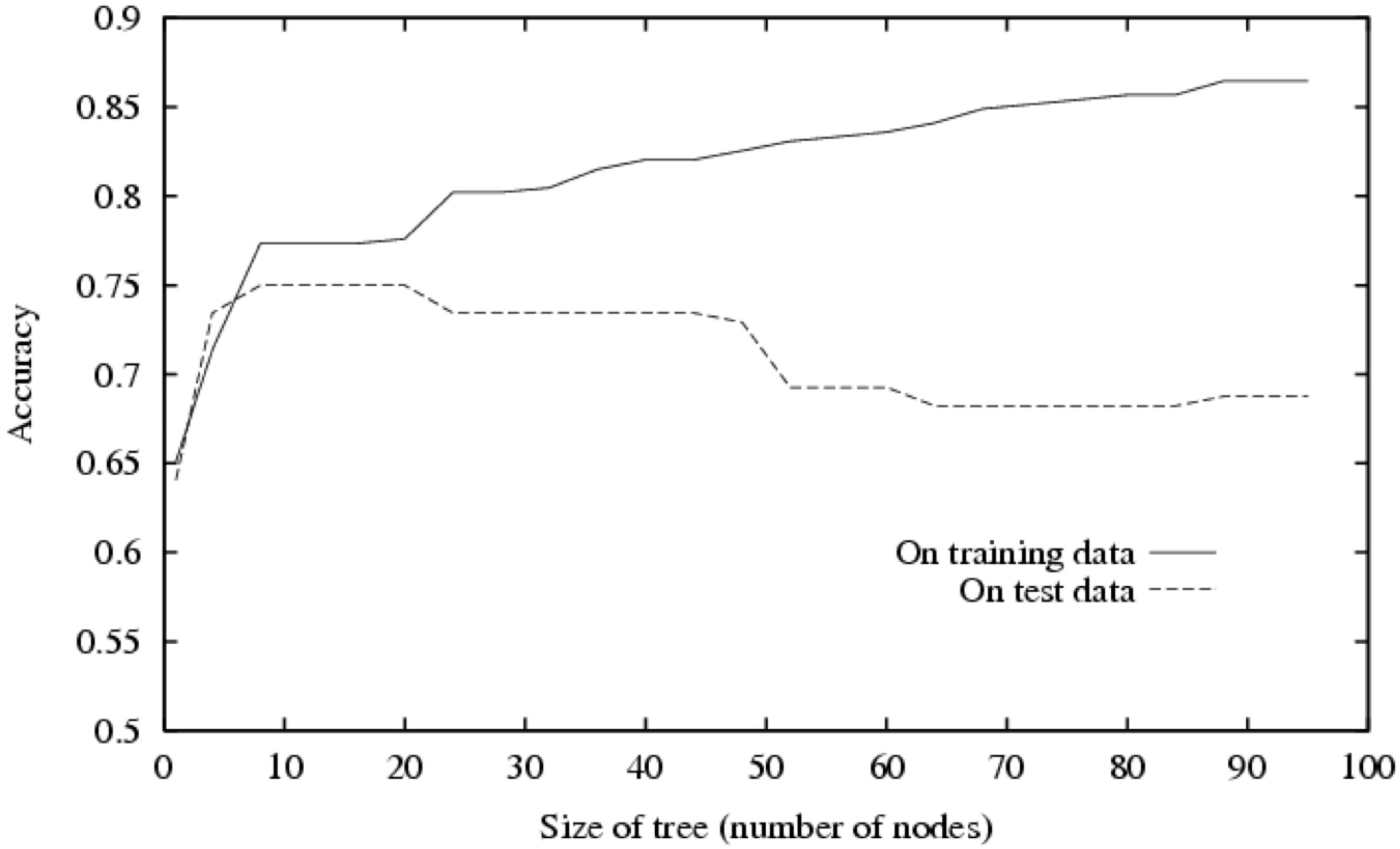
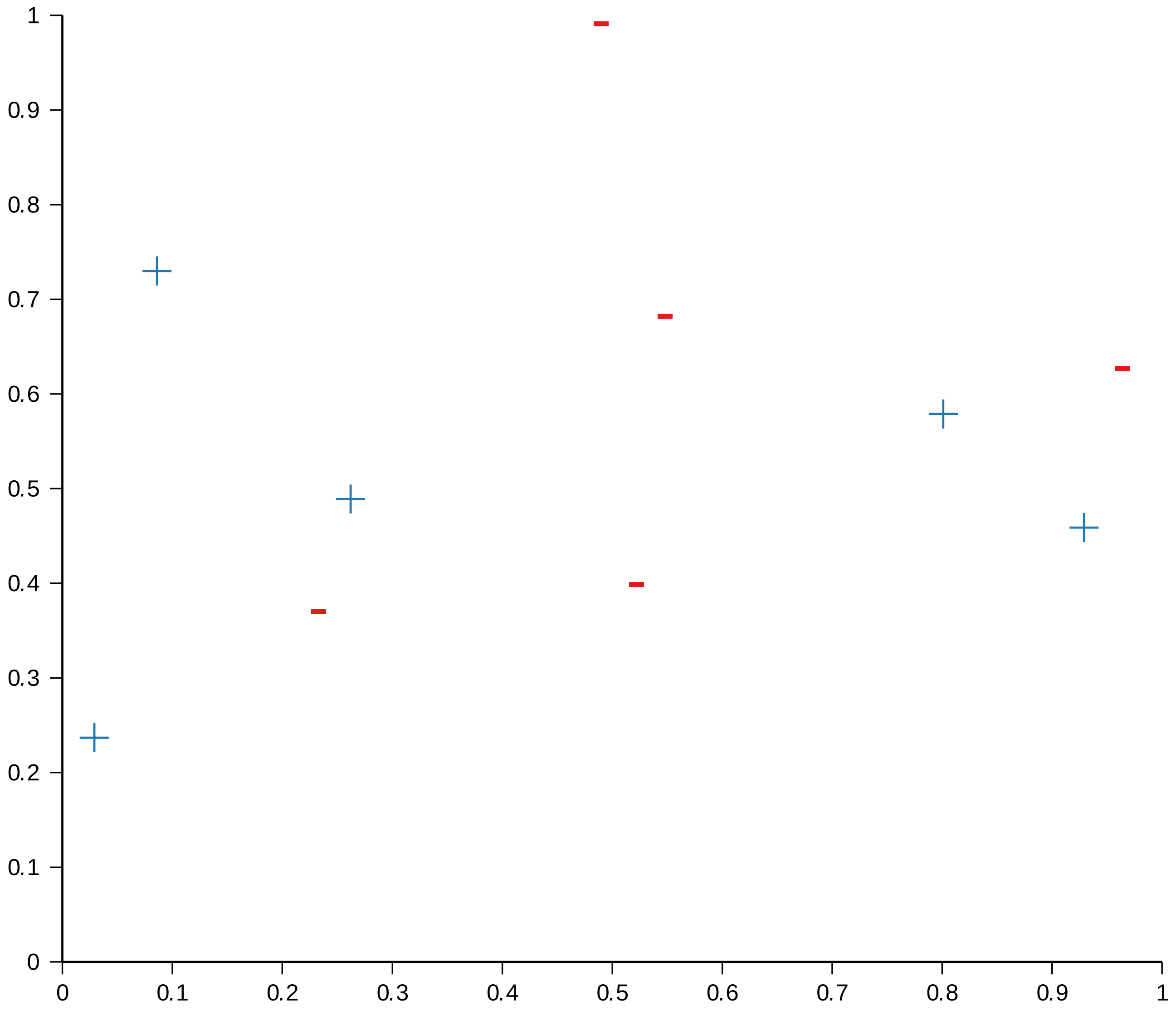


Figure courtesy of Tom Mitchell

Nearest Neighbor: Example



Setting k

- When $k = 1$:
  - many, complicated decision boundaries
  - may overfit
- When $k = N$:
  - no decision boundaries; always predicts the most common label in the training data
  - may underfit
- k controls the complexity of the hypothesis set $\Longrightarrow$ k affects how well the learned hypothesis will generalize

Setting k for kNN with Validation Sets

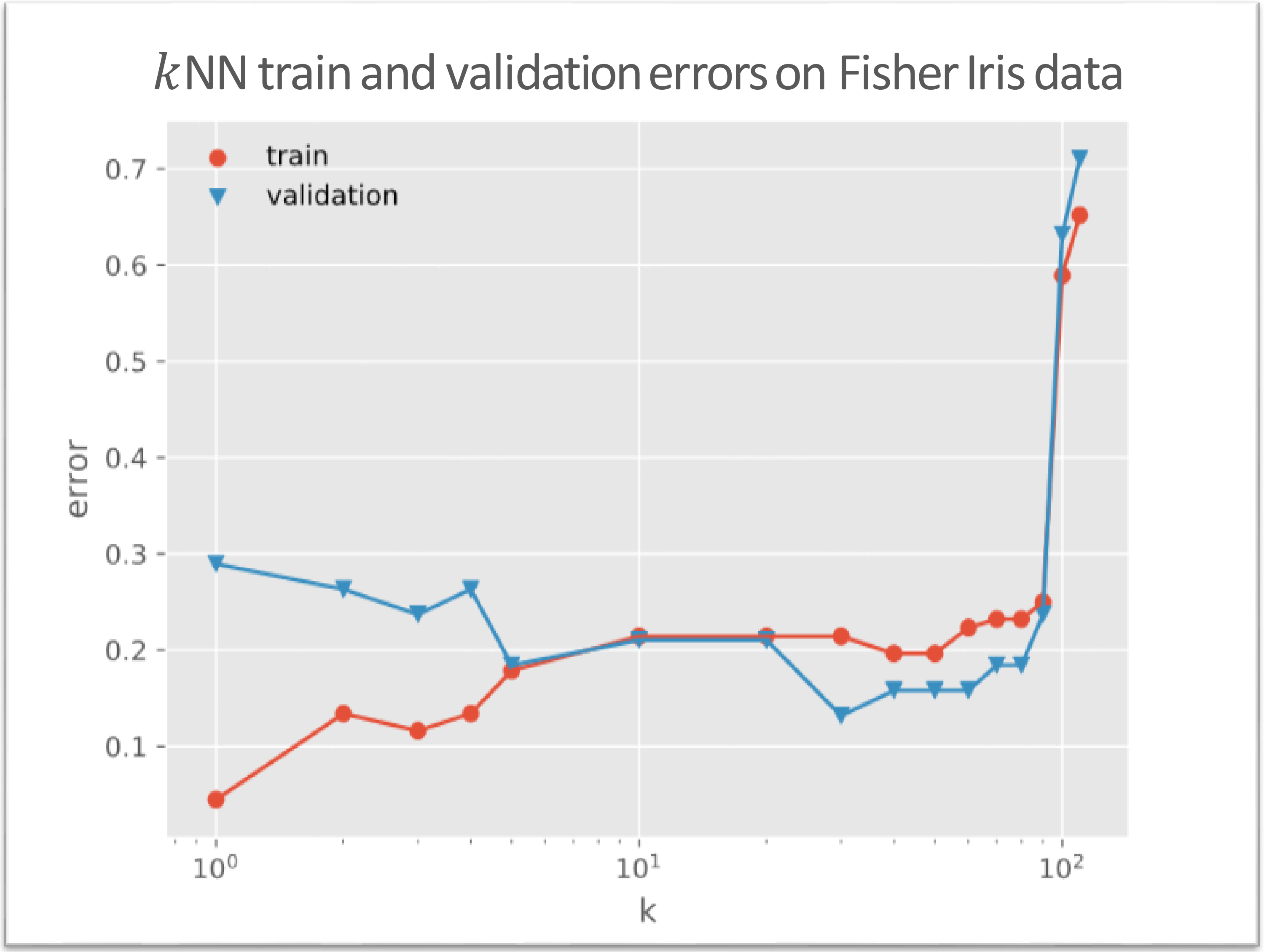


Figure courtesy of Matt Gormley

Recipe for Linear Regression

- Define a model and model parameters
  - Assume $y = \boldsymbol{w}^T \boldsymbol{x}$
  - Parameters: $\boldsymbol{w} = [w_0, w_1, \dots, w_D]$
- Write down an objective function
  - Minimize the squared error

$$\ell_{\mathcal{D}}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell^{(n)}(\boldsymbol{w}) = \sum_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}^{(n)} - y^{(n)}\right)^2$$

- Optimize the objective w.r.t. the model parameters
  - Solve in *closed form*: take partial derivatives, set to 0 and solve

Minimizing the Squared Error

$$\ell_{\mathcal{D}}(\boldsymbol{w}) = \sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{x}^{(n)} - y^{(n)}\right)^2 = \sum_{n=1}^{N}\left({\boldsymbol{x}^{(n)}}^T\boldsymbol{w} - y^{(n)}\right)^2$$

$$= \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 \text{ where } \|\boldsymbol{z}\|_2 = \sqrt{\sum_{d=1}^{D} z_d^2} = \sqrt{\boldsymbol{z}^T\boldsymbol{z}}$$

$$= (X\boldsymbol{w} - \boldsymbol{y})^T(X\boldsymbol{w} - \boldsymbol{y})$$

$$= (\boldsymbol{w}^T X^T X\boldsymbol{w} - 2\boldsymbol{w}^T X^T\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y})$$

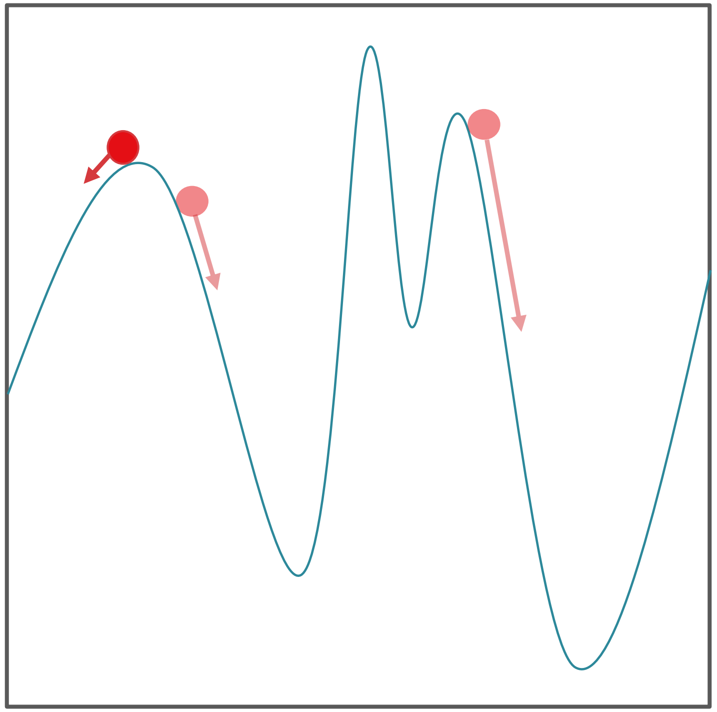
$$\nabla_{\boldsymbol{w}}\ell_{\mathcal{D}}(\widehat{\boldsymbol{w}}) = (2X^T X\widehat{\boldsymbol{w}} - 2X^T\boldsymbol{y}) = 0$$

$$\rightarrow X^T X\widehat{\boldsymbol{w}} = X^T\boldsymbol{y}$$

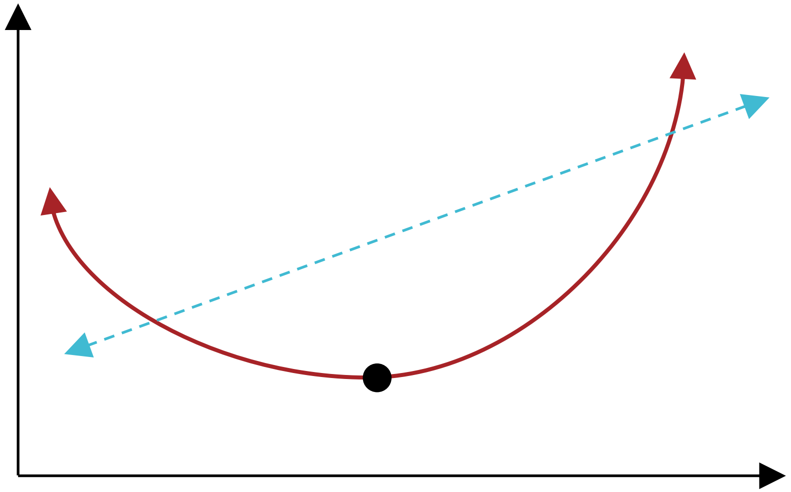
$$\rightarrow \widehat{\boldsymbol{w}} = (X^T X)^{-1}X^T\boldsymbol{y}$$

Gradient Descent: Intuition

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere

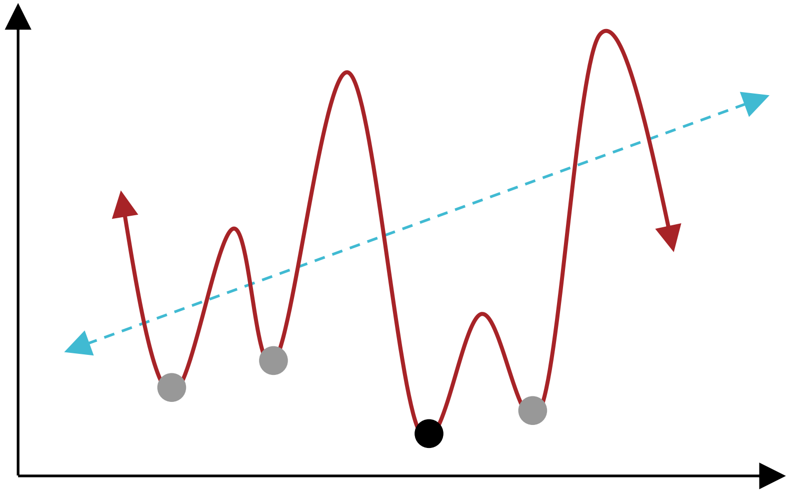


Convexity



Strictly convex functions:

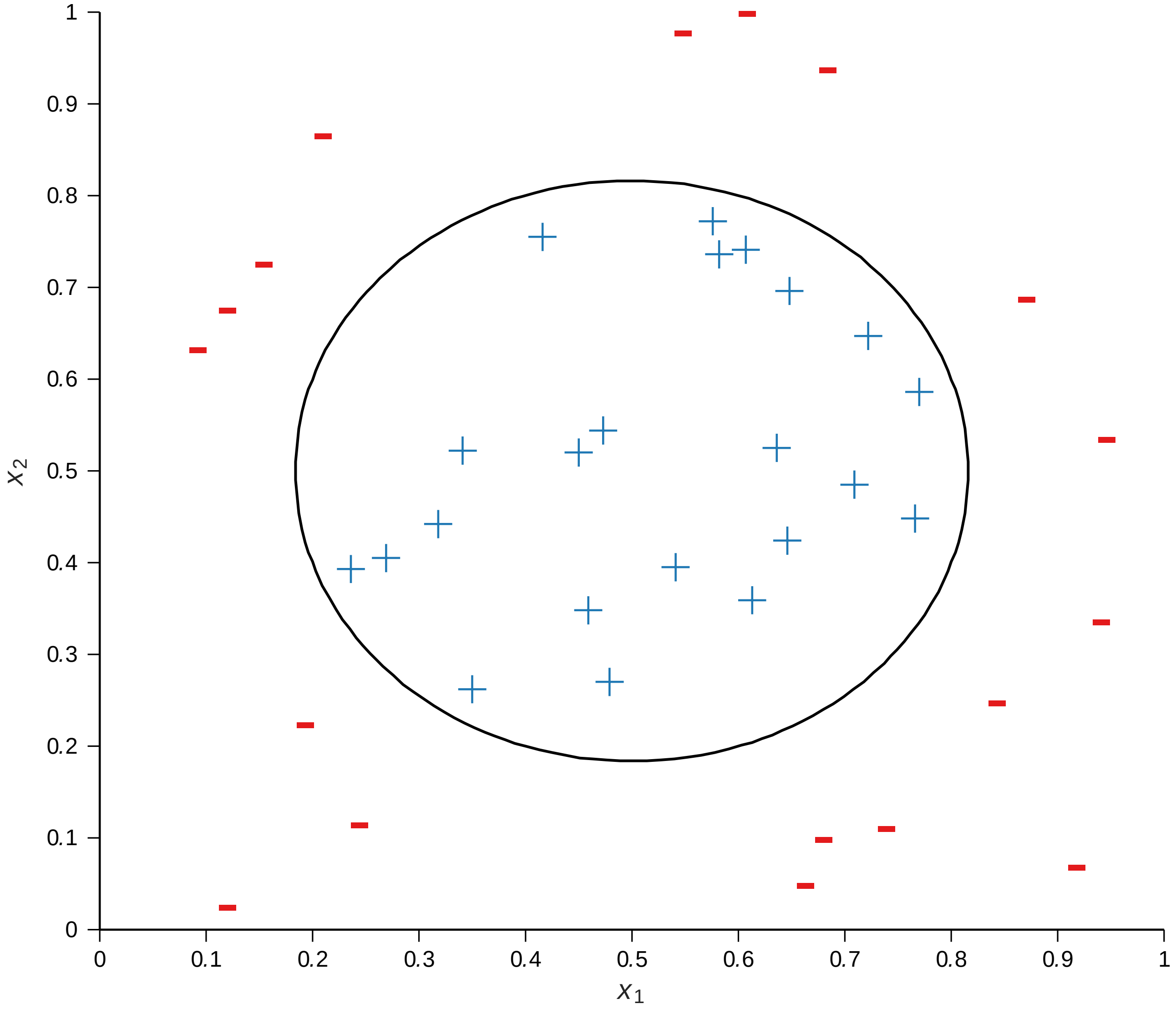
There exists a unique global minimum!



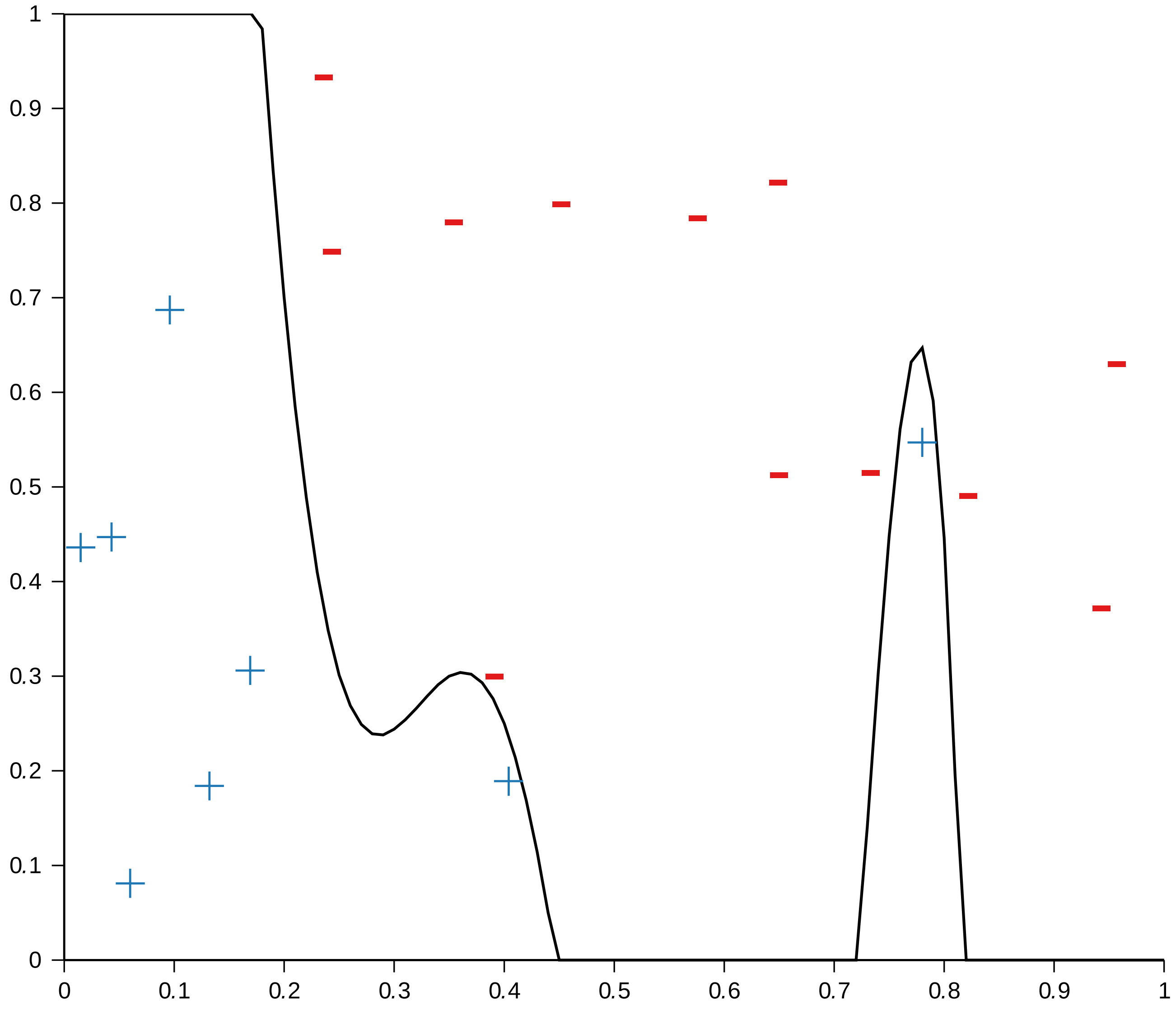
Non-convex functions:

A local minimum may or may not be a global minimum…

Nonlinear Models



Nonlinear Models?



Soft Constraints

minimize $\ell_{\mathcal{D}}(\boldsymbol{\omega}) = (\mathrm{X}\boldsymbol{\omega} - \boldsymbol{y})^T(\mathrm{X}\boldsymbol{\omega} - \boldsymbol{y})$

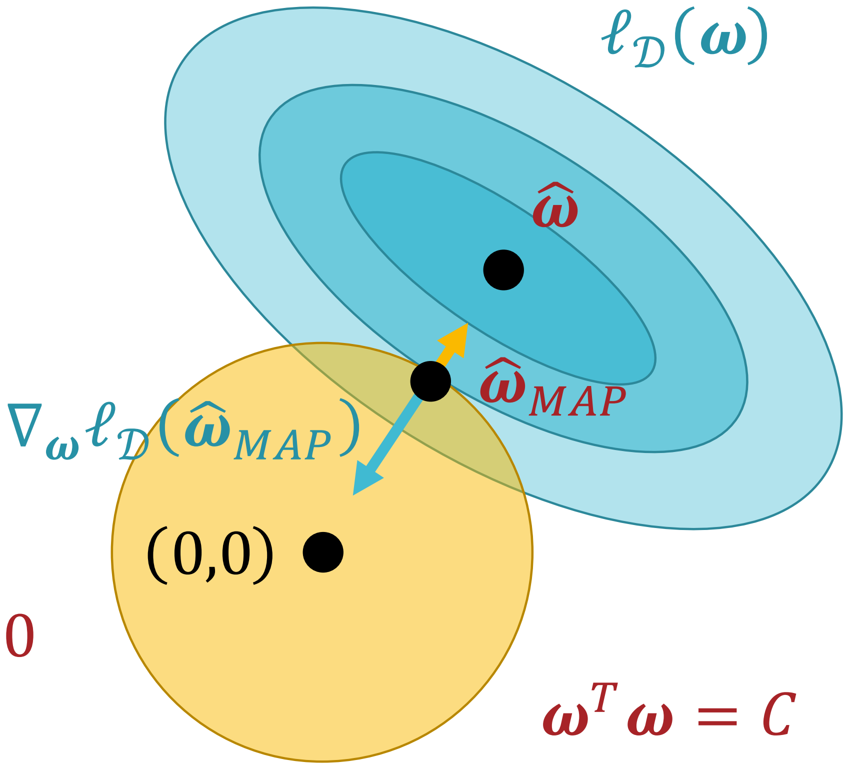
subject to $\boldsymbol{\omega}^T\boldsymbol{\omega} \leq C$

$$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) \propto -2\widehat{\boldsymbol{\omega}}_{MAP}$$

$$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) = -2\lambda_C\widehat{\boldsymbol{\omega}}_{MAP}$$

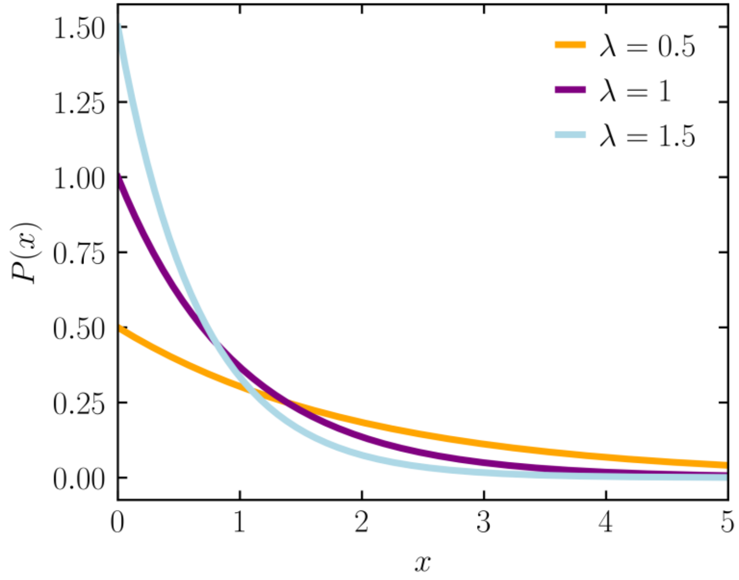
$$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) + 2\lambda_C\widehat{\boldsymbol{\omega}}_{MAP} = 0$$

$$\nabla_{\omega}(\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) + \lambda_C(\widehat{\boldsymbol{\omega}}_{MAP})^T\widehat{\boldsymbol{\omega}}_{MAP}) = 0$$



Maximum Likelihood Estimation (MLE)

- Insight: every valid probability distribution has a finite amount of probability mass as it must sum/integrate to 1
- Idea: set the parameter(s) so that the likelihood of the samples is maximized
- Intuition: assign as much of the (finite) probability mass to the observed data *at the expense of unobserved data*
- Example: the exponential distribution



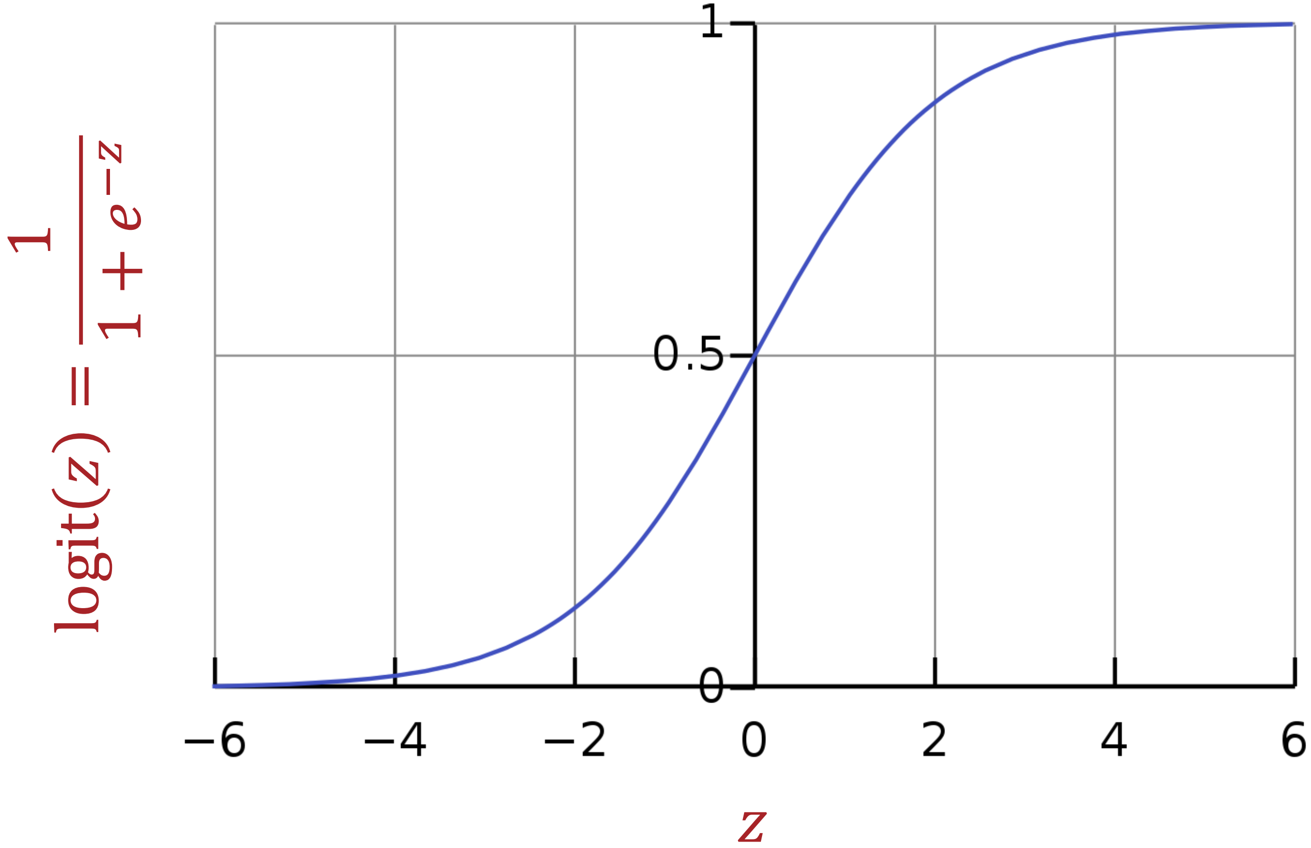
Source: https://en.wikipedia.org/wiki/Exponential_distribution#/media/File:Exponential_probability_density.svg

Building a Probabilistic Classifier

- Define a decision rule
  - Given a test data point $\boldsymbol{x}'$, predict its label $\hat{y}$ using the *posterior distribution* $P(Y = y|X = \boldsymbol{x}')$
  - Common choice: $\hat{y} = \underset{y}{\operatorname{argmax}}\, P(Y = y|X = \boldsymbol{x}')$
- Model the posterior distribution
  - Option 1 - Model $P(Y|X)$ directly as some function of X (today!)
  - Option 2 - Use Bayes' rule (later):

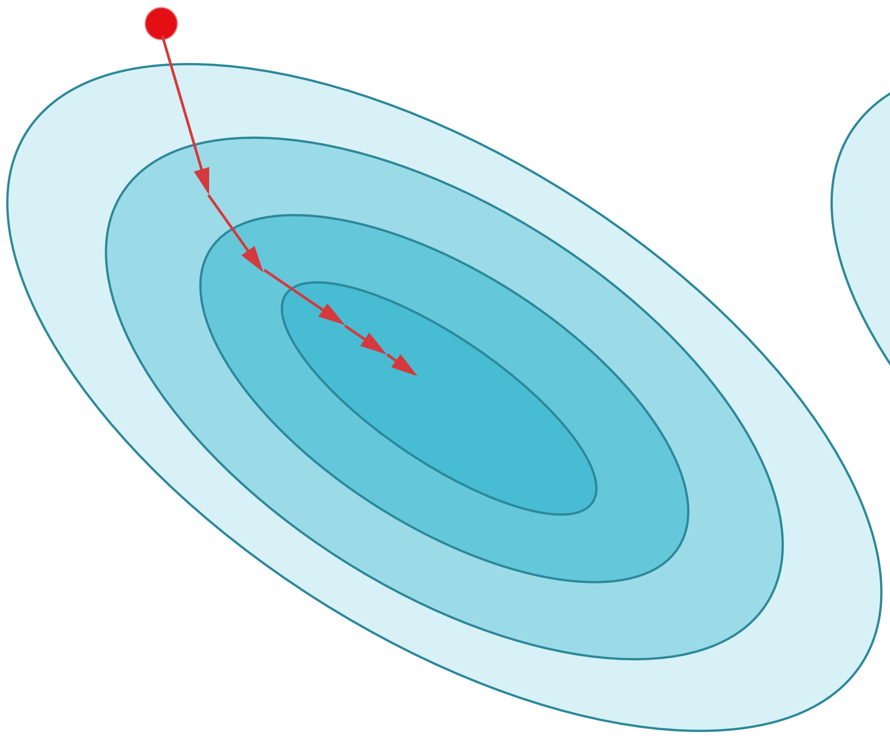
$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)} \propto P(X|Y)\,P(Y)$$

Logistic Function

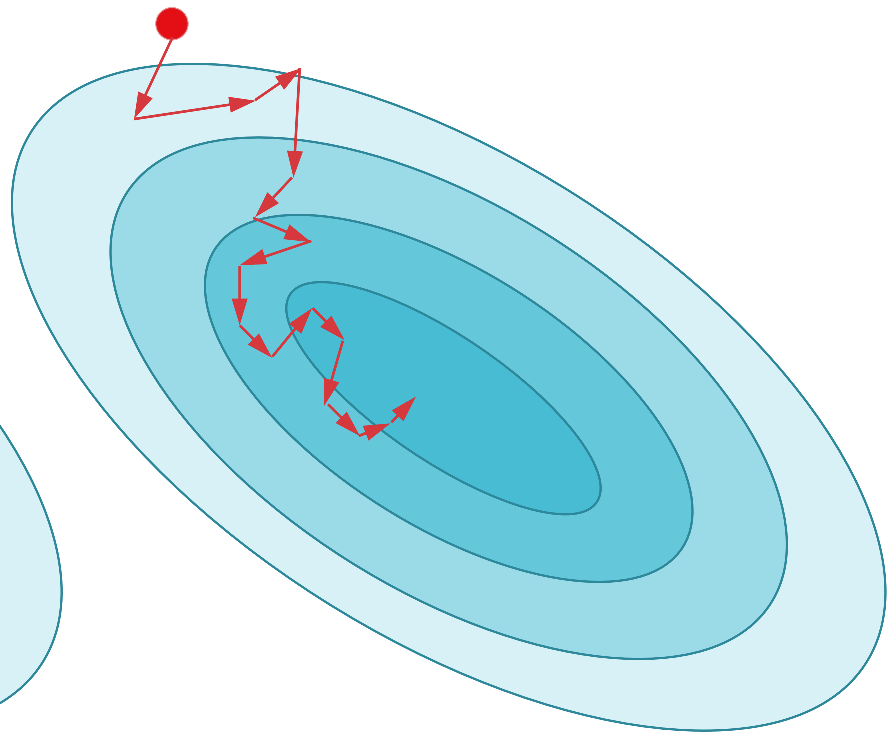


Source: https://en.wikipedia.org/wiki/Logistic_function#/media/File:Logistic-curve.svg

Stochastic Gradient Descent vs. Gradient Descent

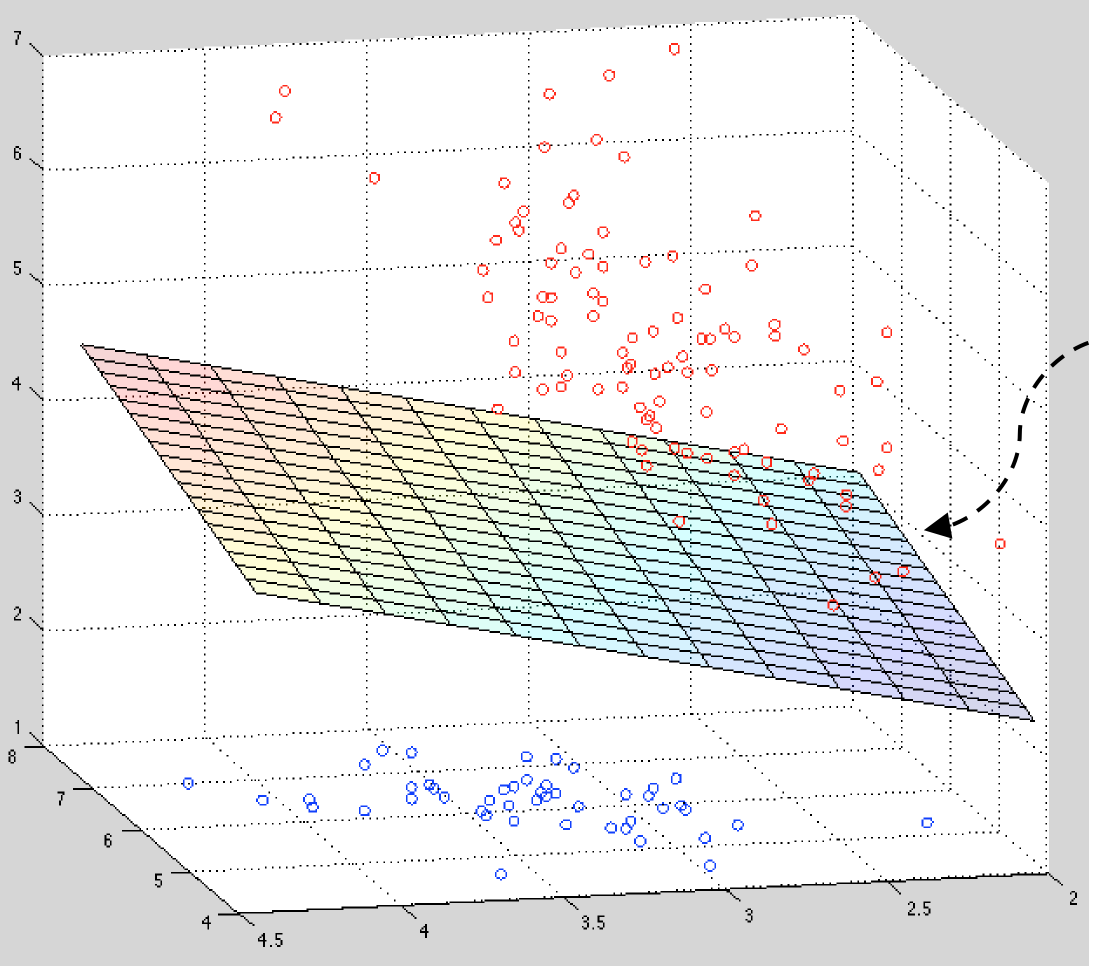


Gradient Descent



Stochastic Gradient Descent

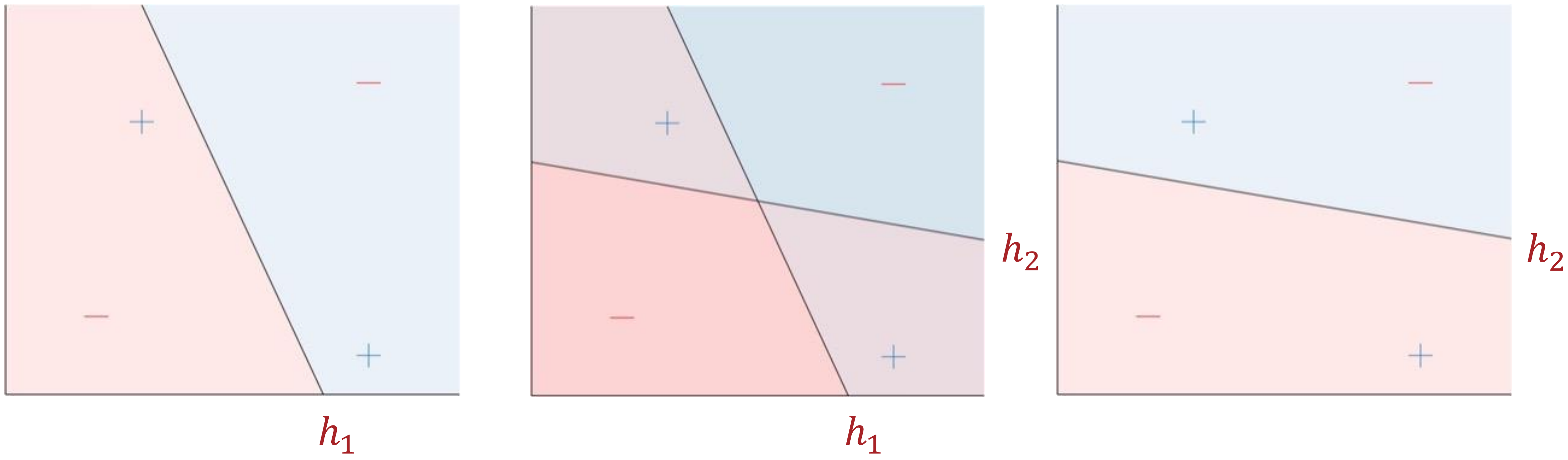
Linear Decision Boundaries: Example



Goal: learn classifiers of the form $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$ (assuming $y \in \{-1, +1\}$)

Key question: how do we learn the *parameters*, $\boldsymbol{w}$?

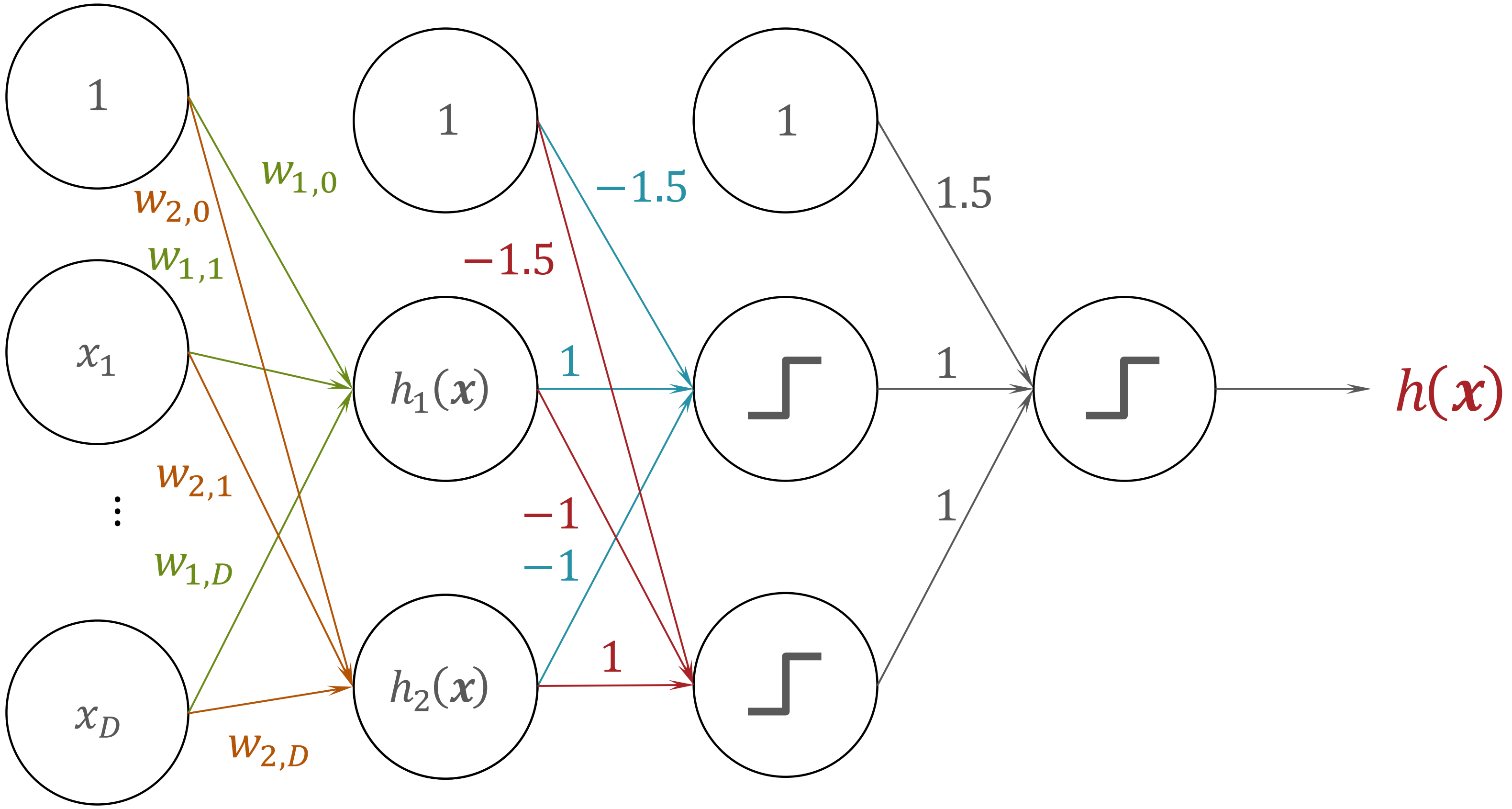
Figure courtesy of Matt Gormley



Combining Perceptrons

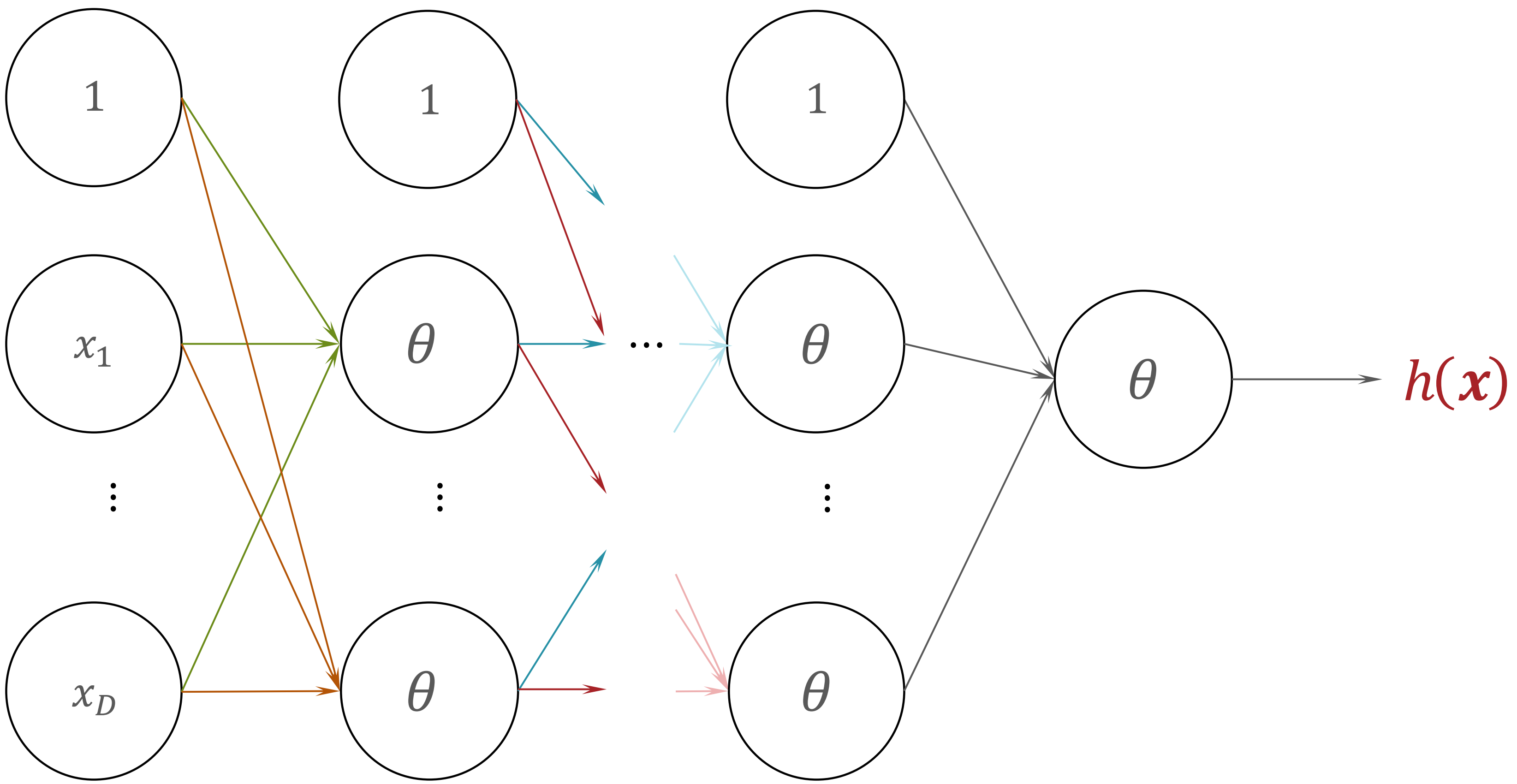
Building a Network

$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



$$h(\boldsymbol{x}) = \text{sign}(\text{sign}(\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) - \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) + \text{sign}(-\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) + \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) + 1.5)$$

(Fully-Connected) Feed Forward Neural Network



Back-propagation

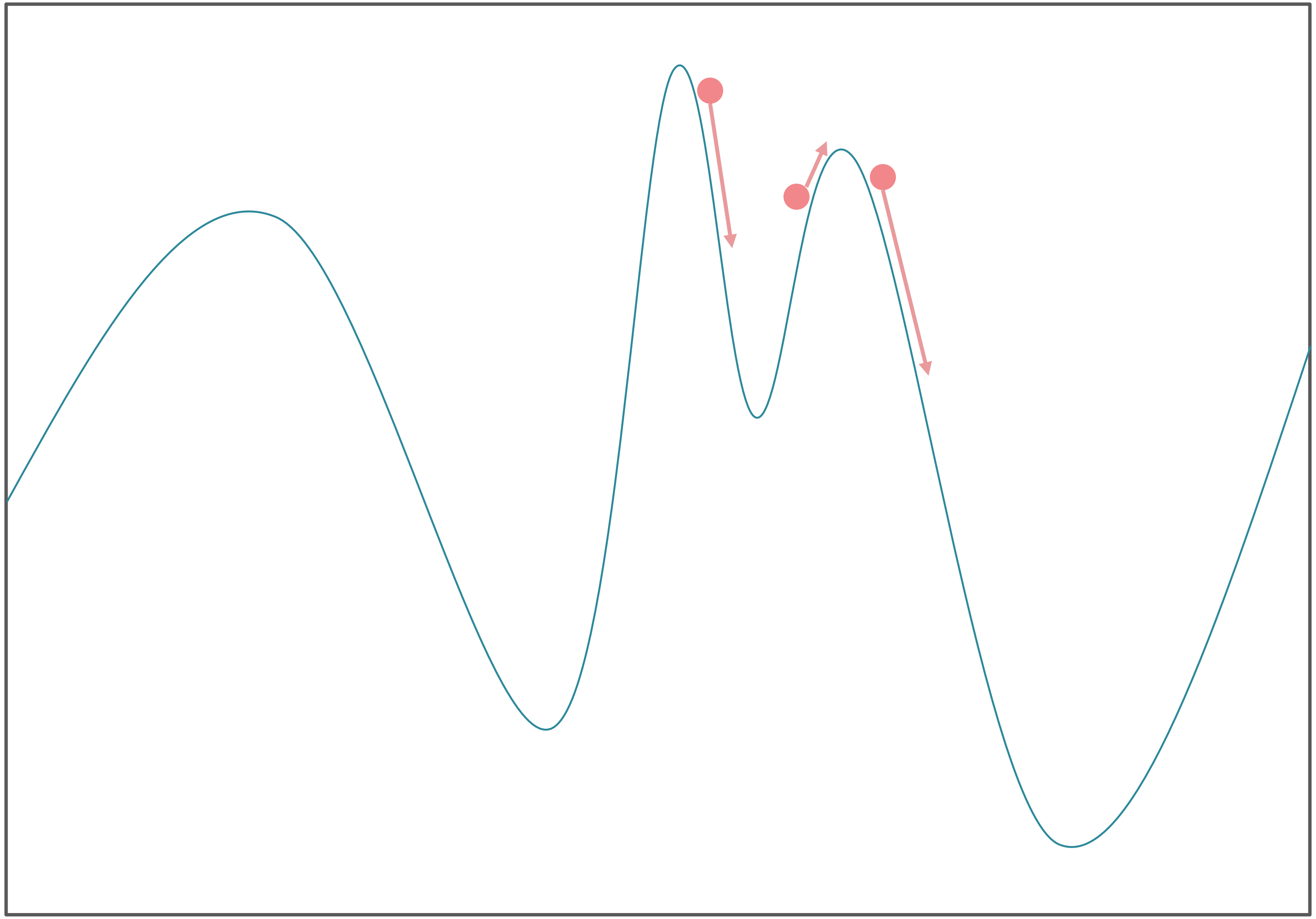
- Input: $W^{(1)}, \dots, W^{(L)}$ and $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
- Initialize: $\ell_{\mathcal{D}} = 0$ and $G^{(l)} = 0 \odot W^{(l)}\ \forall\, l = 1, \dots, L$
- For $n = 1, \dots, N$
  - Run forward propagation with $\boldsymbol{x}^{(n)}$ to get $\boldsymbol{o}^{(1)}, \dots, \boldsymbol{o}^{(L)}$
  - (Optional) Increment $\ell_{\mathcal{D}}$: $\ell_{\mathcal{D}} = \ell_{\mathcal{D}} + \left(o^{(L)} - y^{(n)}\right)^2$
  - Initialize: $\boldsymbol{\delta}^{(L)} = 2\left(o_1^{(L)} - y^{(n)}\right)\left(1 - \left(o_1^{(L)}\right)^2\right)$
  - For $l = L - 1, \dots, 1$
    - Compute $\boldsymbol{\delta}^{(l)} = W^{(l+1)^T} \boldsymbol{\delta}^{(l+1)} \odot \left(1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)}\right)$
    - Increment $G^{(l)}$: $G^{(l)} = G^{(l)} + \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$
- Output: $G^{(1)}, \dots, G^{(L)}$, the gradients of $\ell_{\mathcal{D}}$ w.r.t $W^{(1)}, \dots, W^{(L)}$

Three Approaches to Differentiation

- Given $f: \mathbb{R}^D \to \mathbb{R}$, compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = {}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$

1. Finite difference method
   - Requires the ability to call $f(\boldsymbol{x})$
   - Great for checking accuracy of implementations of more complex differentiation methods
   - Computationally expensive for high-dimensional inputs
2. Symbolic differentiation
   - Requires systematic knowledge of derivatives
   - Can be computationally expensive if poorly implemented
3. Automatic differentiation (reverse mode)
   - Requires systematic knowledge of derivatives *and* an algorithm for computing $f(\boldsymbol{x})$
   - Computational cost of computing ${}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$ is proportional to the cost of computing $f(\boldsymbol{x})$

Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks



Mini-batch Stochastic Gradient Descent with Adaptive Gradients for Neural Networks

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B, \epsilon$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $S_{-1}^{(l)} = 0 \odot W^{(l)} \ \forall\, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,
   $$G_t^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} e\left(\boldsymbol{o}^{(L)}, y^{(b)}\right) \ \forall\, l$$
   c. Update $S^{(l)}$: $S_t^{(l)} = S_{t-1}^{(l)} + G_t^{(l)} \odot G_t^{(l)} \ \forall\, l$
   d. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\eta_{MB}^{(0)}}{\sqrt{S_t^{(l)} + \epsilon}} \odot G_t^{(l)} \ \forall\, l$
   e. Increment t: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \ldots, W_t^{(L)}$

What is ~~Machine Learning~~ 10-301/601?

- Supervised Models
  - Decision Trees
  - KNN
  - Naïve Bayes
  - Perceptron
  - Logistic Regression
  - Linear Regression
  - Neural Networks
- Deep Learning
- Unsupervised Models
  - K-means
  - PCA
- Graphical Models
  - Bayesian Networks
  - HMMs
- Learning Theory
- Reinforcement Learning
- Ensemble Methods
- Important Concepts
  - Feature Engineering and Kernels
  - Regularization and Overfitting
  - Experimental Design

Q: Why did we cover so many unrelated topics in the second half of the semester?

A: You never know where the next big thing in machine learning is going to come from!

- Supervised Models
  - Decision Trees
  - KNN
  - Naïve Bayes
  - Perceptron
  - Logistic Regression
  - Linear Regression
  - Neural Networks
- Deep Learning
- Unsupervised Models
  - K-means
  - PCA
- Graphical Models
  - Bayesian Networks
  - HMMs
- Learning Theory
- Reinforcement Learning
- Ensemble Methods
- Important Concepts
  - Feature Engineering and Kernels
  - Regularization and Overfitting
  - Experimental Design

What is ChatGPT?

- Chatbot built on GPT 3.5 (or 4)

What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model

What is ~~ChatGPT~~ ~~GPT~~ a language model?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a **probability distribution** over **sequences of words** (e.g., sentences)

Recall: 3 Inference Questions for Hidden Markov Models

1. Marginal Computation: $P\left(Y_t = s_j \middle| \boldsymbol{x}^{(n)}\right)$ (or $P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$)

$$P\left(Y \middle| \boldsymbol{x}^{(n)}\right) = \frac{P\left(\boldsymbol{x}^{(n)} | Y\right) P(Y)}{P\left(\boldsymbol{x}^{(n)}\right)} = \frac{\prod_{t=1}^{T} P\left(\boldsymbol{x}_t^{(n)} \middle| Y_t\right) P(Y_t | Y_{t-1})}{P\left(\boldsymbol{x}^{(n)}\right)}$$

2. Decoding: $\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$

3. Evaluation: $P\left(\boldsymbol{x}^{(n)}\right)$

$$P\left(\boldsymbol{x}^{(n)}\right) = \sum_{\mathcal{Y} \in \{\text{all possible sequences}\}} P\left(\boldsymbol{x}^{(n)} \middle| \mathcal{Y}\right) P(\mathcal{Y})$$

What is ~~ChatGPT~~ ~~GPT~~ a *large* language model?

Name	Release date[a]	Developer	Number of parameters[b]	Corpus size
BERT	2018	Google	340 million[105]	3.3 billion words[105]
XLNet	2019	Google	~340 million[109]	33 billion words
GPT-2	2019	OpenAI	1.5 billion[112]	40GB[113] (~10 billion tokens)[114]
GPT-3	2020	OpenAI	175 billion[37]	300 billion tokens[114]
GPT-4	March 2023	OpenAI	Exact number unknown[f]	Unknown
PaLM 2 (Pathways Language Model 2)	May 2023	Google	340 billion[162]	3.6 trillion tokens[162]

Source: https://en.wikipedia.org/wiki/Large_language_model

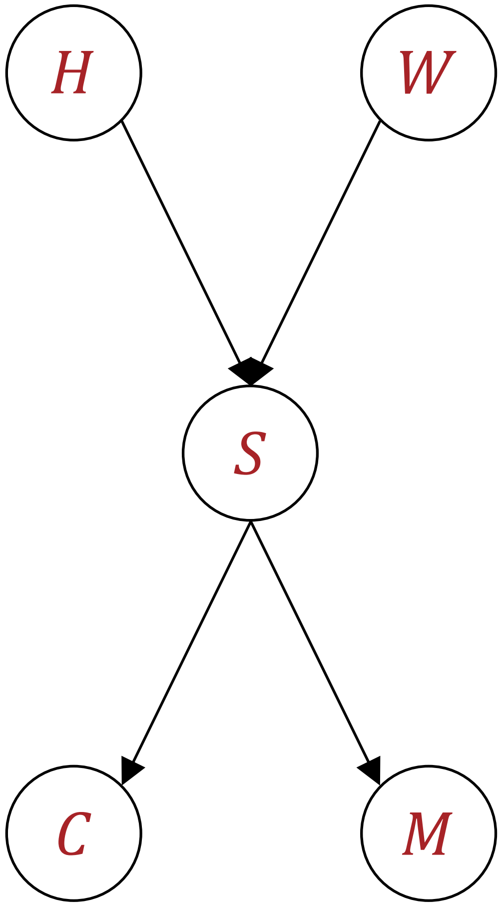
What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a probability distribution over sequences of words (e.g., sentences)
    - GPT is short for generative pre-trained transformer

What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a probability distribution over sequences of words (e.g., sentences)
    - GPT is short for *generative* pre-trained transformer
      - Generative means the model can create new sequences by **sampling** from the distribution

Sampling for Bayesian Networks



- Sampling from a Bayesian network is easy!
  1. Sample all free variables (H and W)
  2. Sample any variable whose parents have already been sampled
  3. Stop once all variables have been sampled

$$P(S = 1) \approx \frac{\text{\# of samples w/ } S = 1}{\text{\# of samples}}$$

What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a probability distribution over sequences of words (e.g., sentences)
    - GPT is short for generative *pre-trained* transformer
      - Pre-training is the process of initializing some or all model parameters using a dataset or objective function other than the actual task
      - Pre-trained parameters are then *fine-tuned* to the actual task

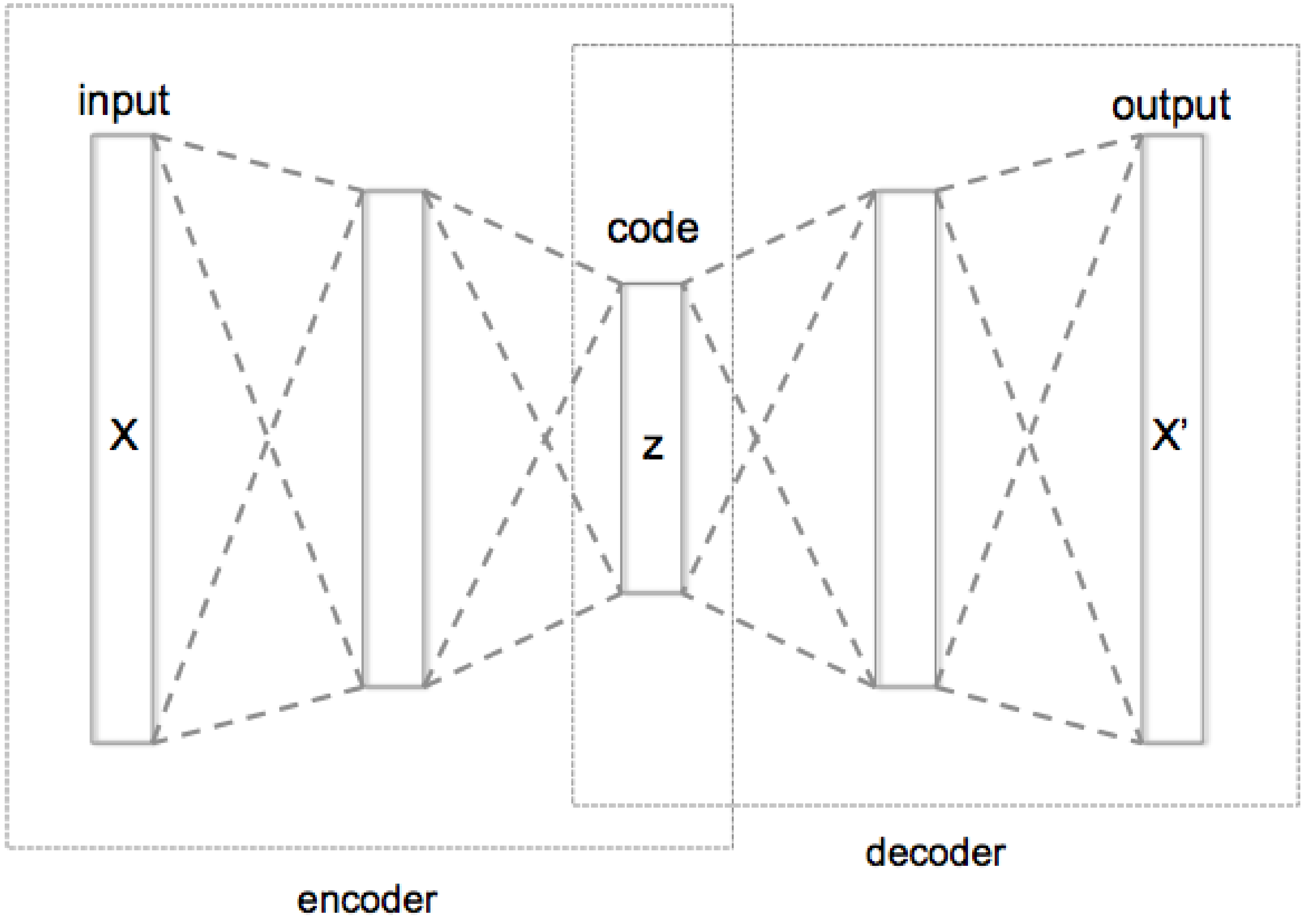
Pre-training (Bengio et al., 2006)

	Train	Val.	Test
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.7%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

- Error rates on MNIST
- Primary finding: pre-training is crucial to unlock the benefits of deep learning!
- Auto-associator is another word for **autoencoder**

Source: https://proceedings.neurips.cc/paper/2006/file/5da713a690c067105aeb2fae32403405-Paper.pdf

Deep Autoencoders



Source: https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png

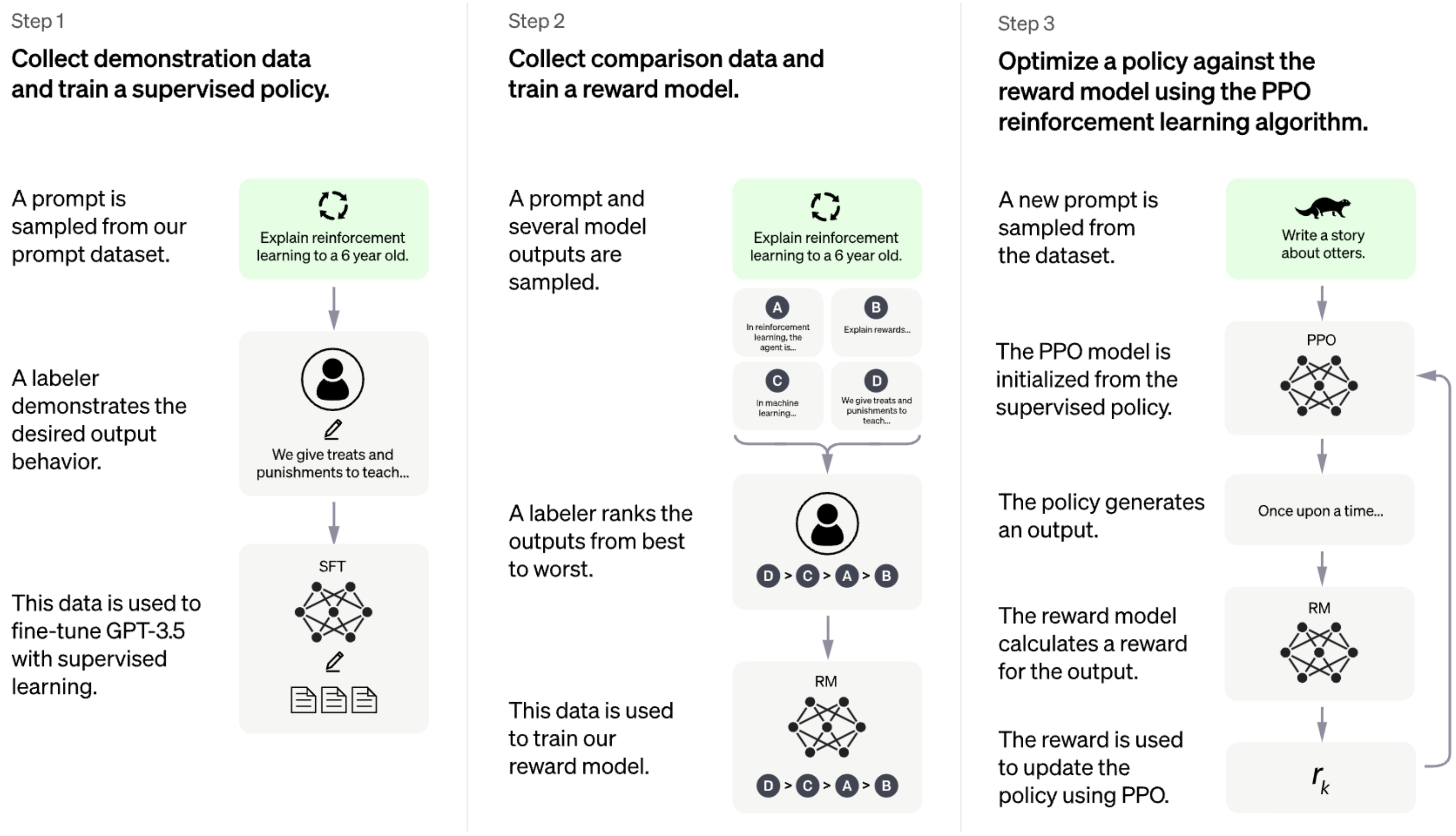
What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a probability distribution over sequences of words (e.g., sentences)
    - GPT is short for generative *pre-trained* transformer
      - GPT parameters are fine-tuned in part using ***reinforcement learning*** *with human feedback*

Reinforcement Learning with Human Feedback (RLHF)

- Insight: for many machine learning tasks, there is no universal ground truth, e.g., there are lots of possible ways to respond to a question or prompt.
- Idea: solve the problem using reinforcement learning and use human feedback as the reward function by having people determine how good or bad some action is.
- Issue: if the state/action space is huge, in order to train a good model, we would need tons and tons of feedback and human annotation is expensive…
- Idea: use a small number of annotations to learn a reward function!

Reinforcement Learning with Human Feedback (RLHF)



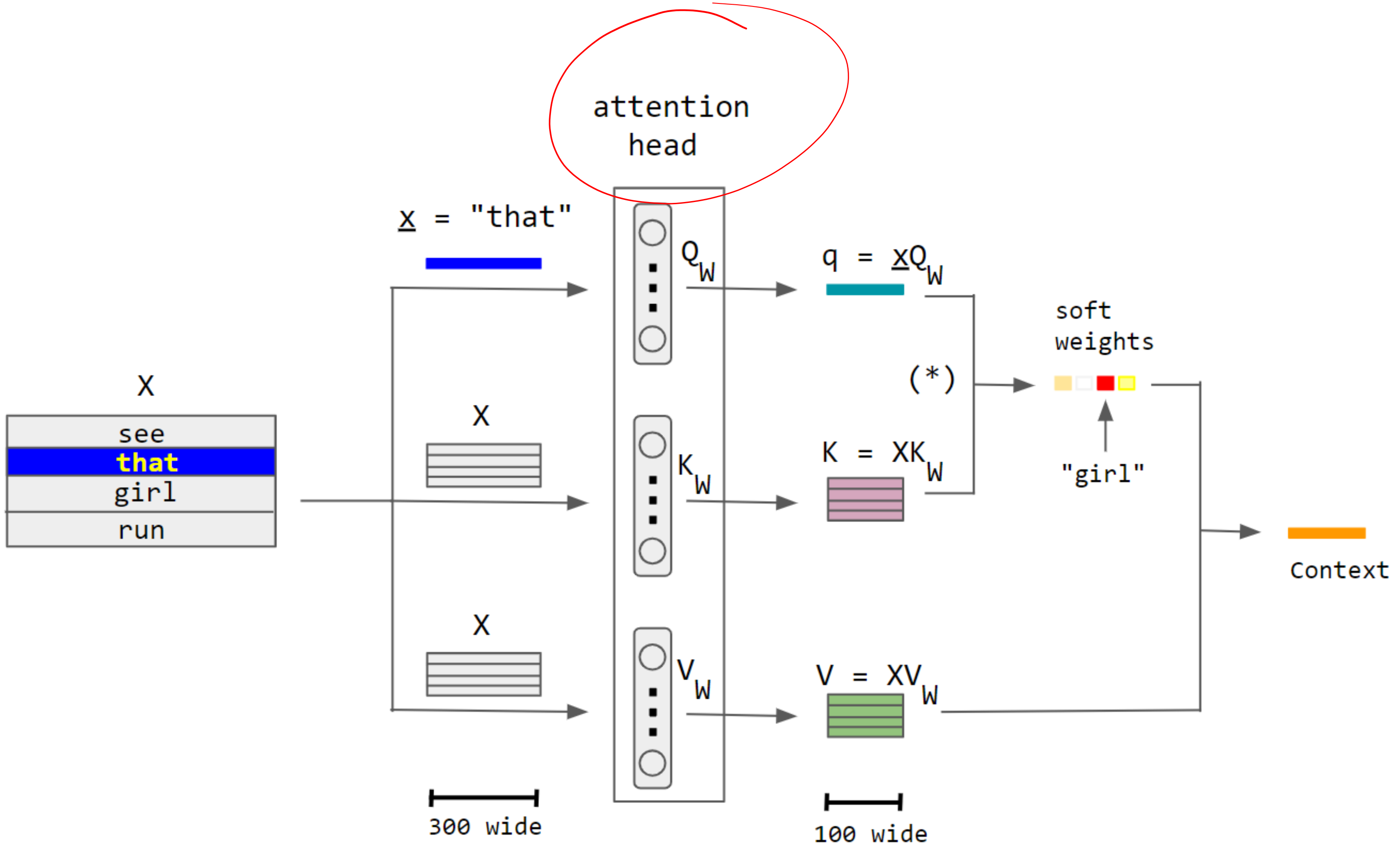
Source: https://openai.com/blog/chatgpt

What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a probability distribution over sequences of words (e.g., sentences)
    - GPT is short for generative pre-trained *transformer*
      - A transformer is a **neural network architecture** that uses just *attention* mechanisms to model sequences ("attention is all you need").

Source: https://arxiv.org/pdf/1706.03762.pdf

Attention

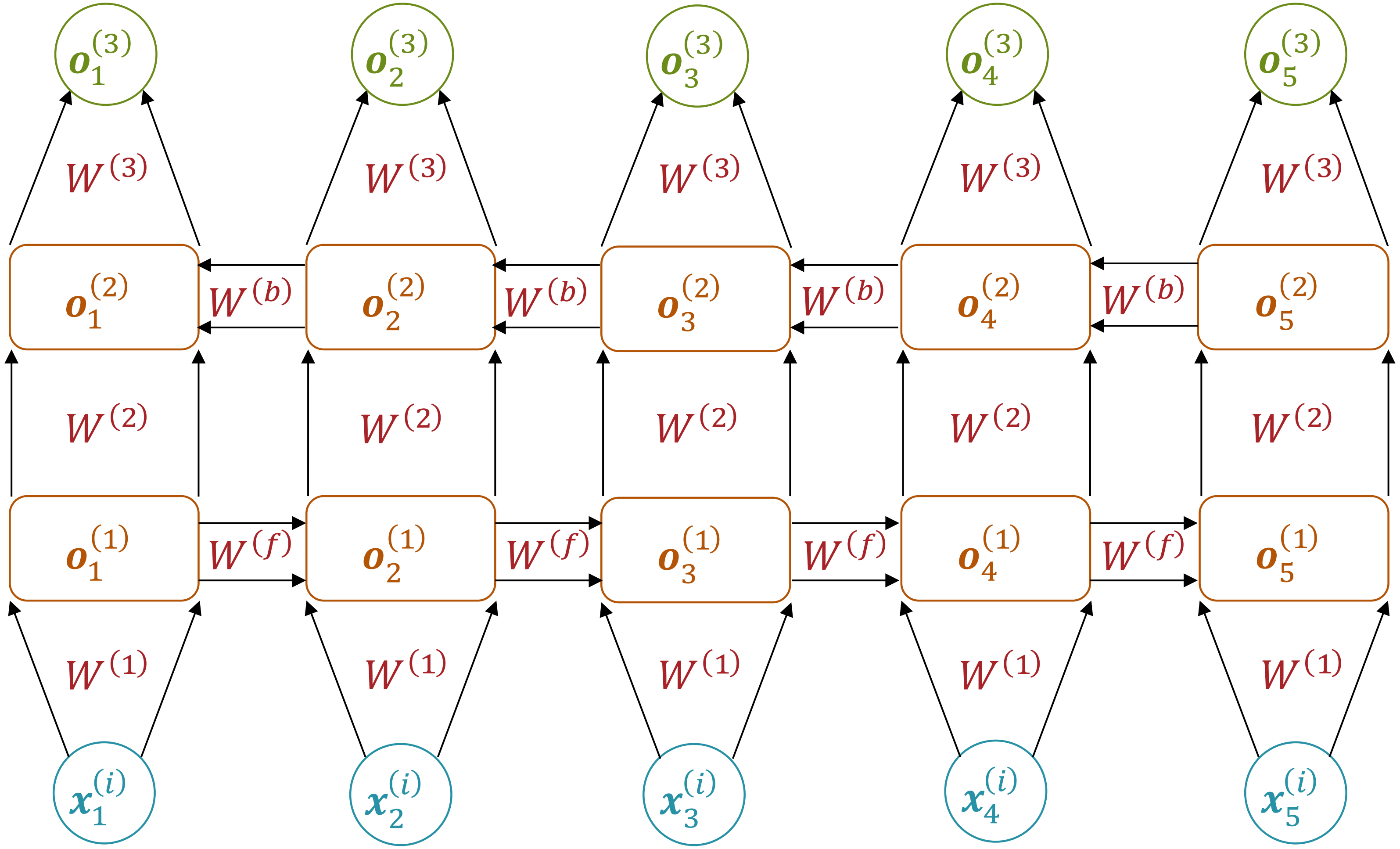


$$\text{Context} = softmax\left(\frac{xQ_W * XK_W^T}{\sqrt{100}}\right) * XV_W$$

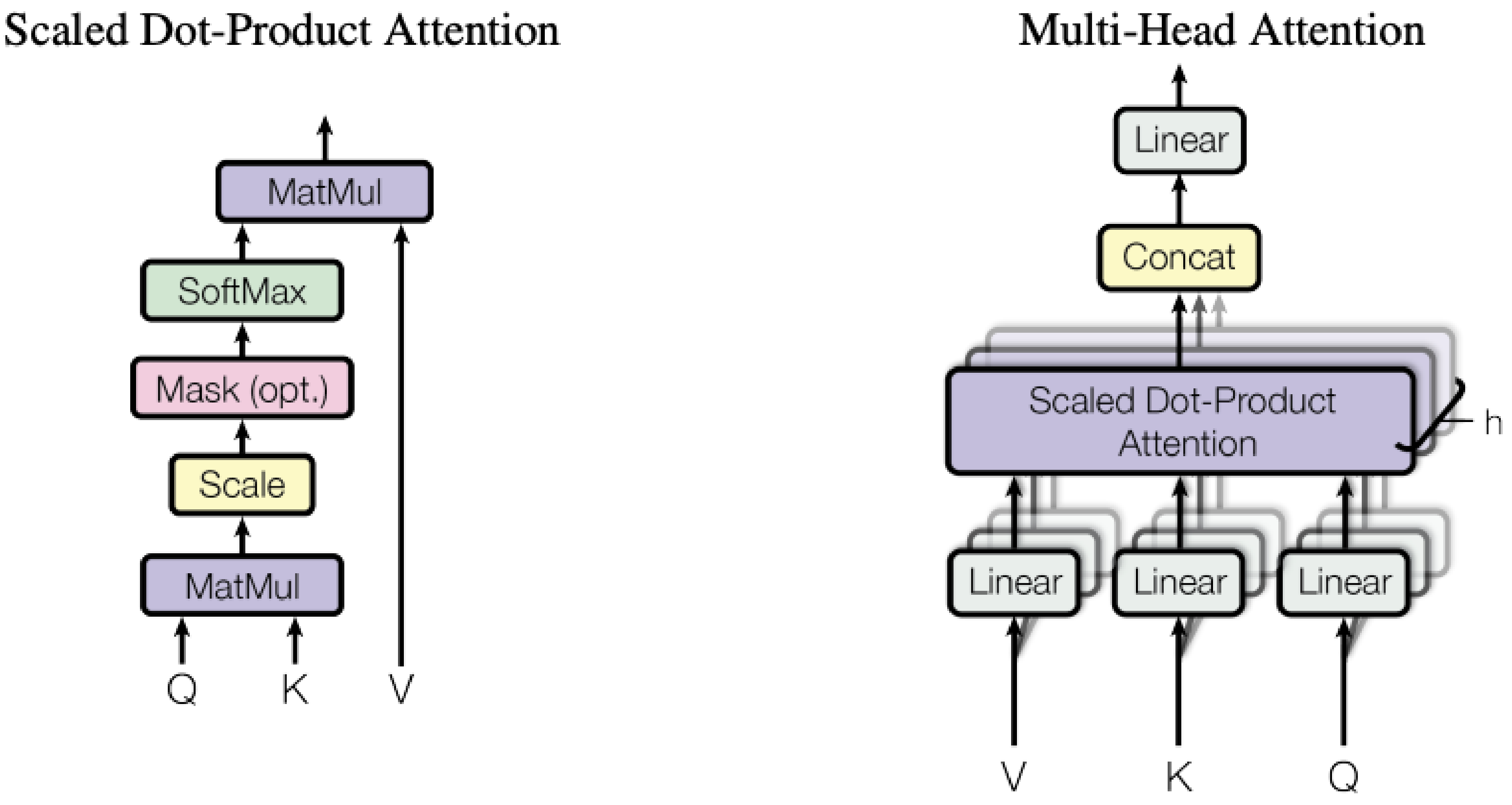
Source: https://en.wikipedia.org/wiki/Attention_(machine_learning)

Bidirectional Recurrent Neural Networks

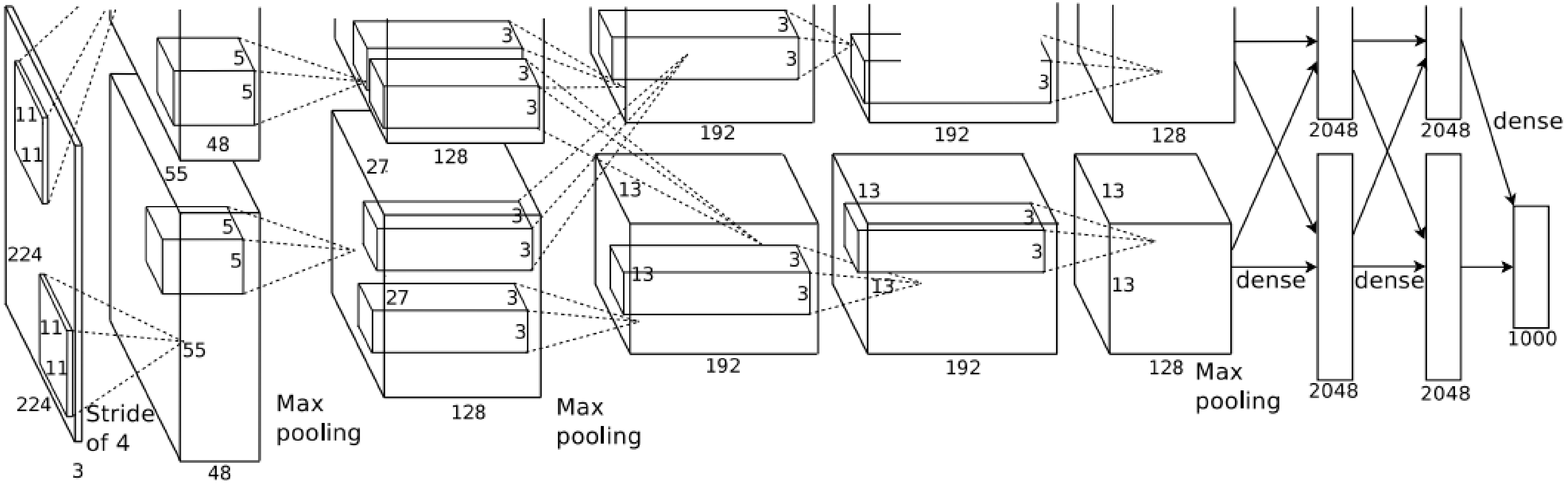
$\boldsymbol{o}_t^{(1)} = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W^{(f)}\boldsymbol{o}_{t-1}^{(1)}\right)\right]^T$ and $\boldsymbol{o}_t^{(2)} = \left[1, \theta\left(W^{(2)}\boldsymbol{o}_t^{(1)} + W^{(b)}\boldsymbol{o}_{t+1}^{(2)}\right)\right]^T$



Multi-headed Attention



Source: https://arxiv.org/pdf/1706.03762.pdf



AlexNet (Krizhevsky et al., 2012)

Source: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Transformers

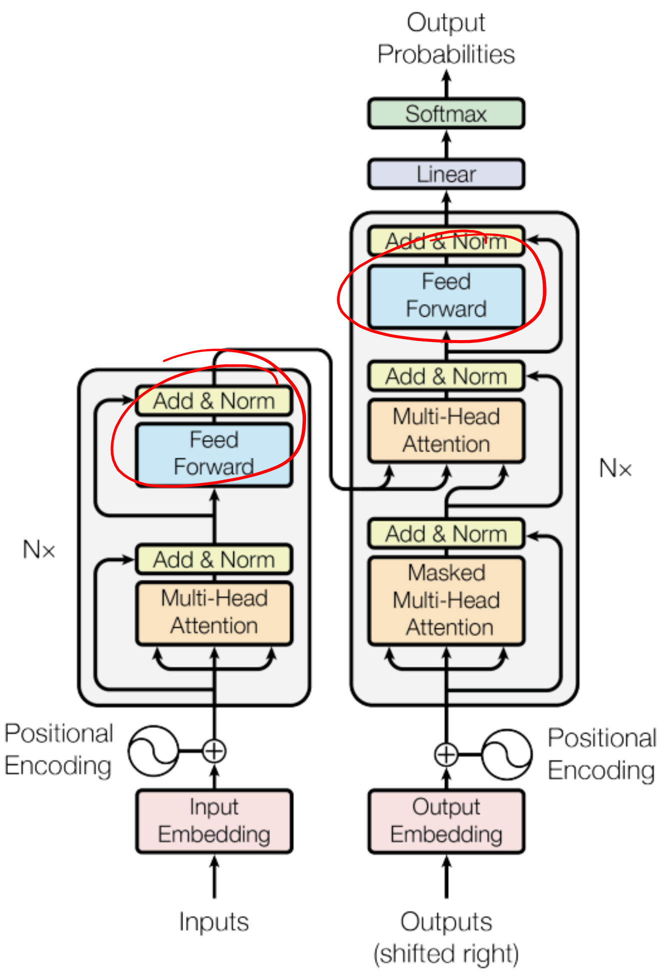


Figure 1: The Transformer - model architecture.

Source: https://arxiv.org/pdf/1706.03762.pdf

What is ~~ChatGPT~~ GPT?

- Chatbot built on GPT 3.5 (or 4)
  - GPT 3.5 is a large language model
    - A language model is just a probability distribution over sequences of words (e.g., sentences)
    - GPT is short for generative pre-trained *transformer*
      - Lots of other relevant implementation details:
        - Optimizer: Adam = **SGD** with **Momentum** + RMSprop (variant of **AdaGrad**)
        - Regularization: Normalized weight decay (variant of **L2 regularization**)
        - Hyperparameter tuning, bias mitigation, etc…

Source: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

Key Takeaways

- You are ready (at least in theory) to go out and learn about the latest machine learning models/concepts
    - You're also equipped to succeed in subsequent machine learning courses you might take
- You all have been a great class, thanks for an amazing summer!