

Programming with Data Bootcamp: Lecture 6

Slides courtesy of Sam Madden
/ Tim Kraska (6.S079)

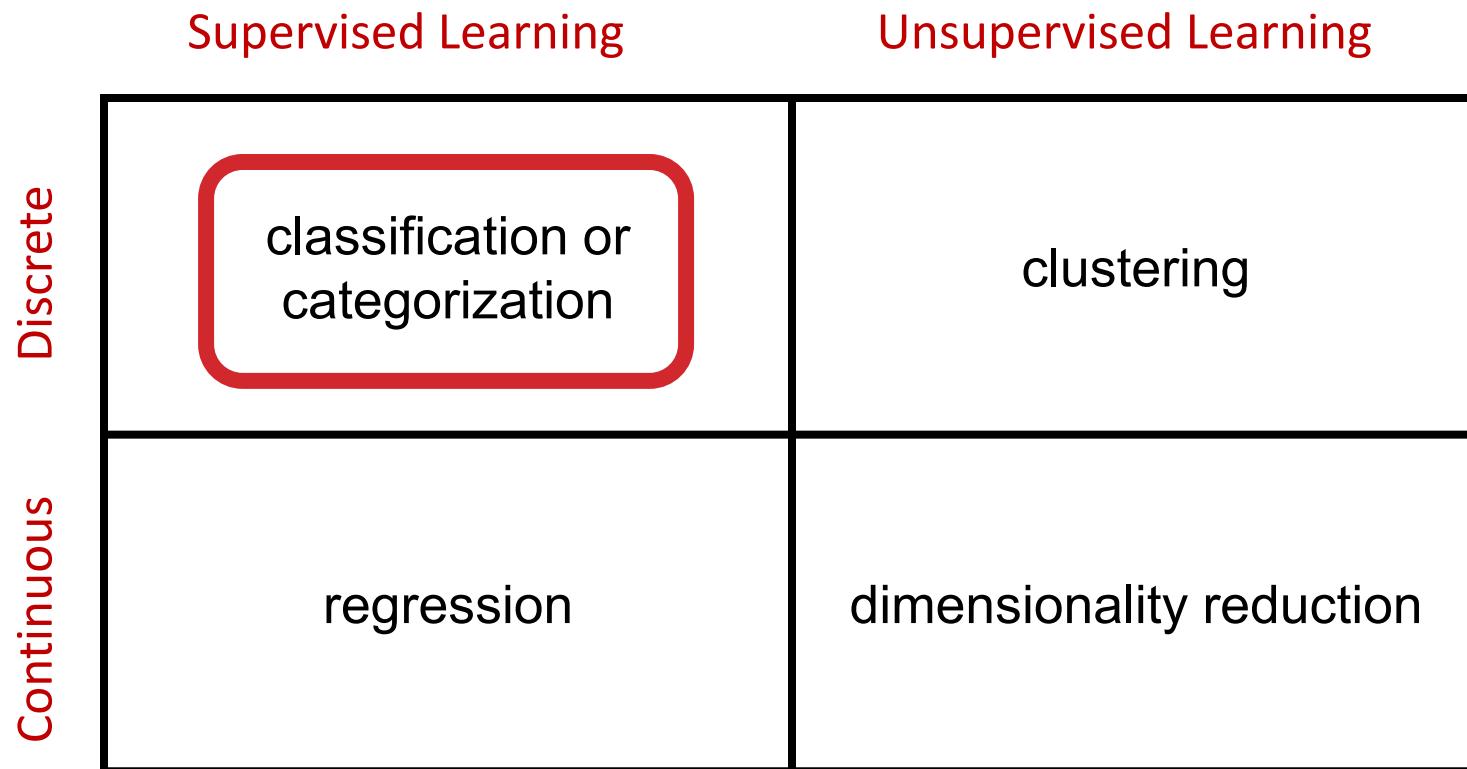
Key ideas:

- Evaluation
- Bias/Variance Tradeoff
- Perceptron
- Multi-layer Perceptron
- CNNs
- Transformers
- LLMs

<http://dsg.csail.mit.edu/6.S079/>



MACHINE LEARNING PROBLEMS



Machine Learning

Nightmare
series

What if your model has a high error?

- Try getting more training examples
- Try smaller sets of features
- Try getting additional features
- Try creating features from existing features (kernels)
- Try decrease regularization
- Try increase regularization

What Error/Quality Metric to use?

Classification:

- Accuracy
- F-score
- F1-micro
- F1-macro
- ROC AUC (micro, macro)
- ...

Regression

- Mean-Squared Error
- Root-Mean Squared Error
- Mean absolute Error
- R²
- Cohen Kappa
- ..

Precision, Recall, Accuracy

	True	False
True	tp	fp
False	fn	tn

- **Precision:** correctly identified positive cases
Precision $P = tp/(tp + fp)$
- **Recall:** correctly identified positive cases from all the actual positive cases.
Recall $R = tp/(tp + fn)$
- **Accuracy:** measure of all the correctly identified cases
Accuracy $R = (tp+tn)/(tp + fp + fn + tn)$

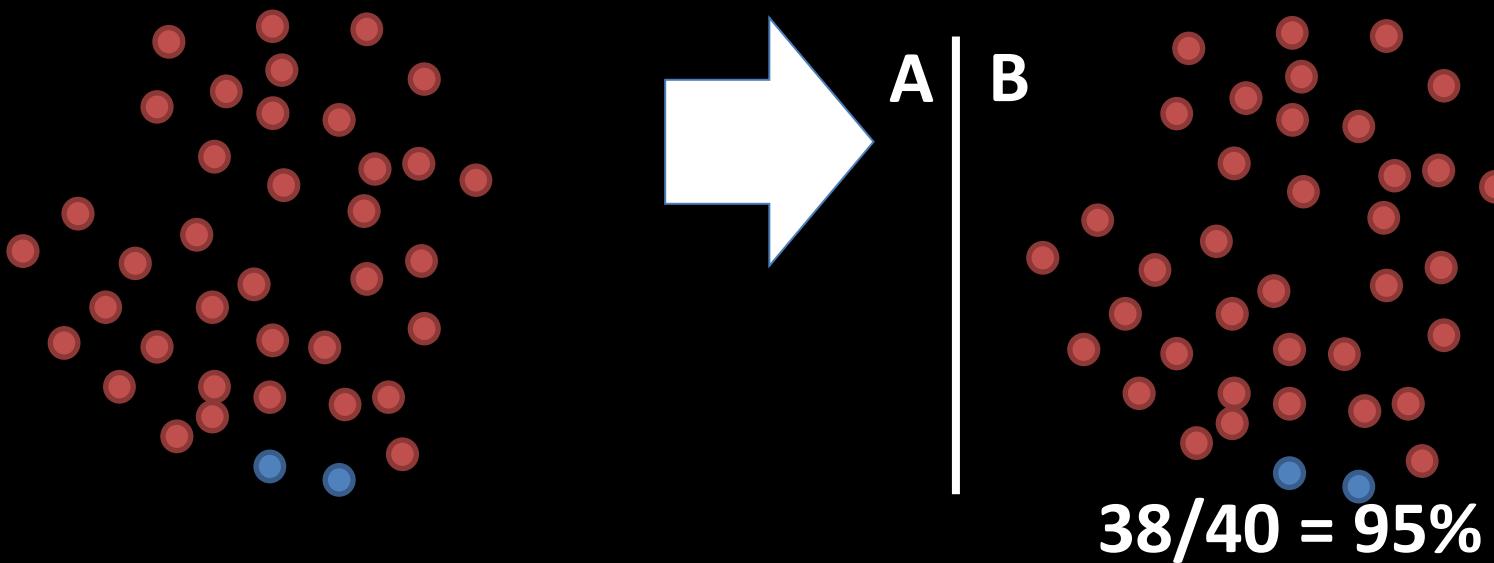
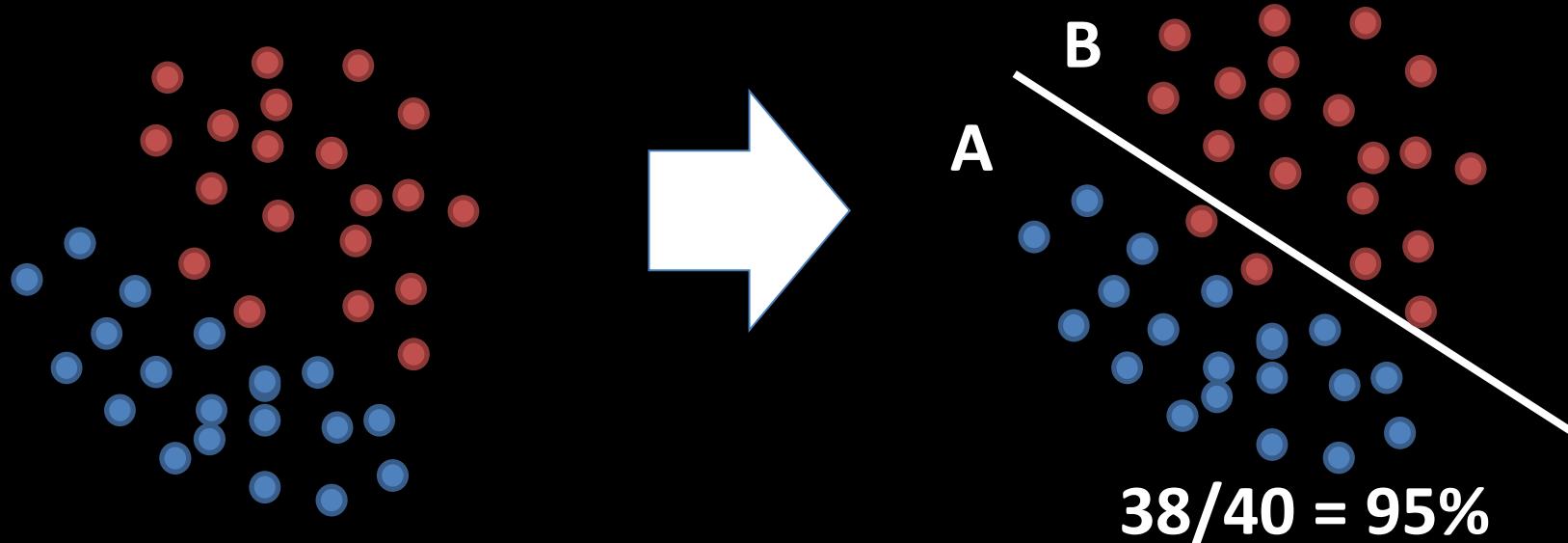
Evaluation: Accuracy isn't always enough

- How do you interpret 90% accuracy?

Evaluation: Accuracy isn't always enough

- How do you interpret 90% accuracy?
 - You can't; it depends on the problem
- Need a baseline:
 - Base Rate
 - Accuracy of trivially predicting the most-frequent class
 - Random Rate
 - Accuracy of making a random class assignment
 - Might apply prior knowledge to assign random distribution
 - Naïve Rate
 - Accuracy of some simple default or pre-existing model
 - Ex: "All females survived"

Why Optimize? Pitfalls



What Error/Quality Metric to use?

Classification:

- Accuracy
- **F-score**
- F1-micro
- F1-macro
- ROC AUC (micro, macro)
- ...

Regression

- Mean-Squared Error
- Root-Mean Squared Error
- Mean absolute Error
- R^2
- Cohen Kappa
- ..

Precision, Recall, Accuracy

		True Label	
		True	False
Predicted Label	True	tp	fp
	False	fn	tn

- **Precision:** correctly identified positive cases
Precision P = $tp/(tp + fp)$
- **Recall:** correctly identified positive cases from all the actual positive cases.
Recall R = $tp/(tp + fn)$
- **F-Score:** is the harmonic mean of precision and recall

$$F = \frac{2}{\frac{1}{R} + \frac{1}{P}} = \frac{tp}{\frac{1}{2}(fp+fn)}$$

F1 Micro

		True Label		
		L1	L2	L3
Predicted Label	L1	7	1	4
	L2	0	1	12
	L3	1	6	6

Precision micro: true positives for all the classes divided by the all positive predictions

$$\text{Precision Score Micro} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{TP} = (7 + 1 + 6)$$

$$\text{FP} = 1 + 4 + 0 + 12 + 1 + 6$$

Recall micro: Sum of **true positives for all the classes** divided by the actual positives.

$$\text{Recall Score Micro: TP} / (\text{TP} + \text{FN})$$

$$\text{F1 Score: } \frac{tp}{tp + \frac{1}{2}(fp + fn)}$$

F1 Macro

		True Label		
		L1	L2	L3
Predicted Label	L1	7	1	4
	L2	0	1	12
	L3	1	6	6

Precision micro: **arithmetic mean of all the precision scores** of different classes

$$\text{Precision Score Macro} = ((7/8) + (1/8) + (6/22))/3$$

Recall micro: **arithmetic mean of all the recall scores** .

When to use F1 Micro and when to use F1 Macro?

F1 Macro

		True Label		
		L1	L2	L3
Predicted Label	L1	7	1	4
	L2	0	1	12
	L3	1	6	6

Precision micro: **arithmetic mean of all the precision scores** of different classes

$$\text{Precision Score Macro} = ((7/8) + (1/8) + (6/22))/3$$

Recall micro: **arithmetic mean of all the recall scores** .

When to use F1 Micro and when to use F1 Macro?

- Micro weights each instance or prediction equally.
- Macro weights each class equally (better for imbalance of labels)
- Use weighted macro-averaging score in case of class imbalances (different number of instances related to different class labels).

ROC AUC

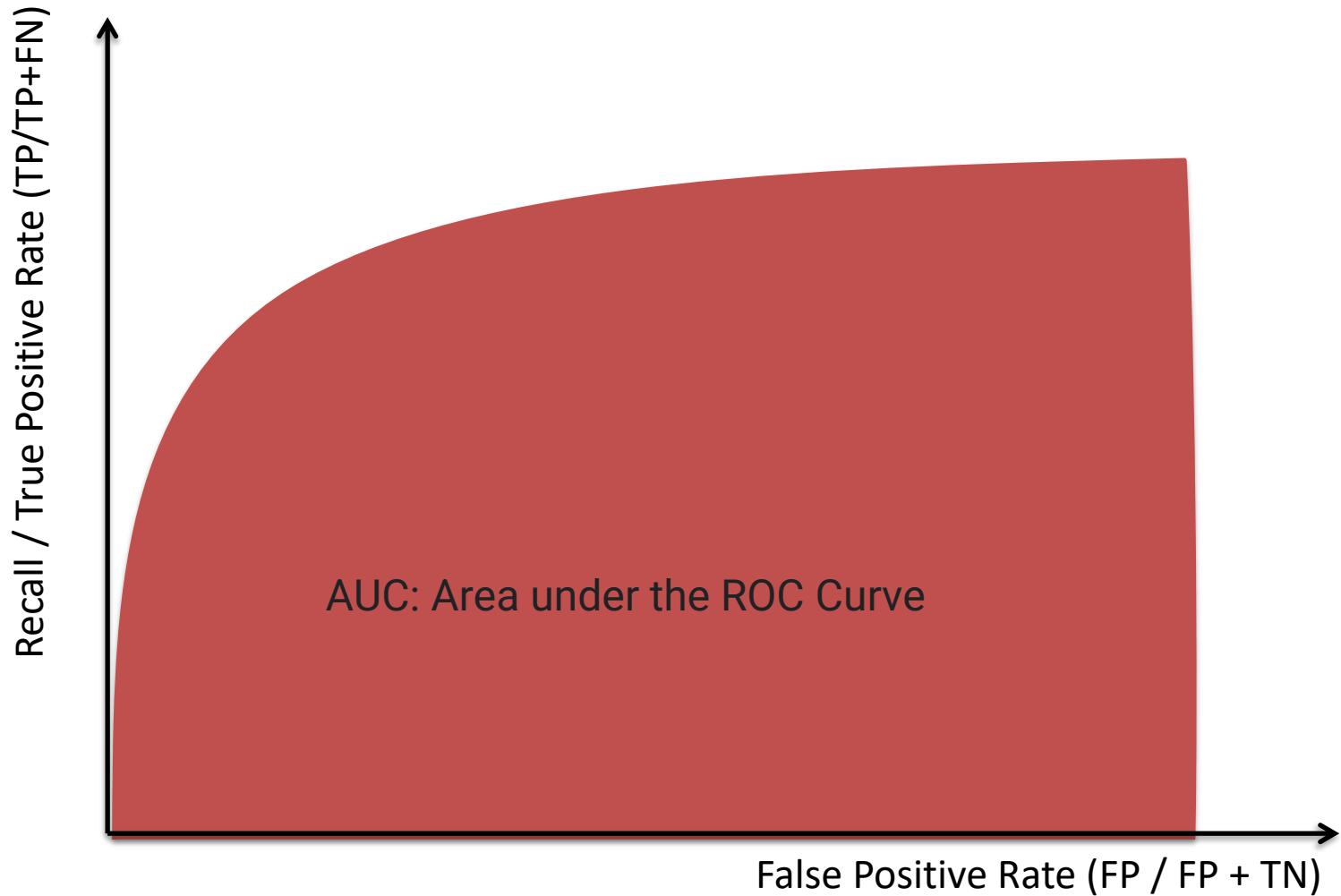
(usually used for models with a threshold)



What would be the ideal ROC curve?
How would a random guess look like

ROC AUC

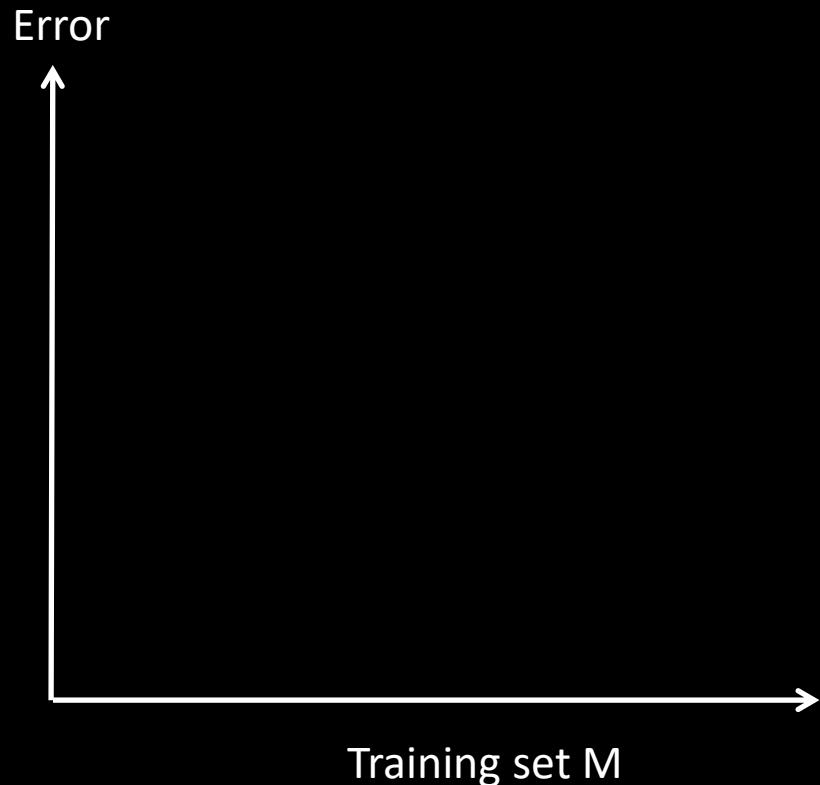
(usually used for models with a threshold)



What if your model has a high error?

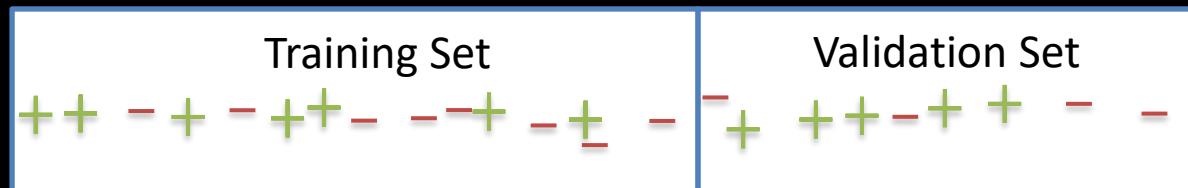
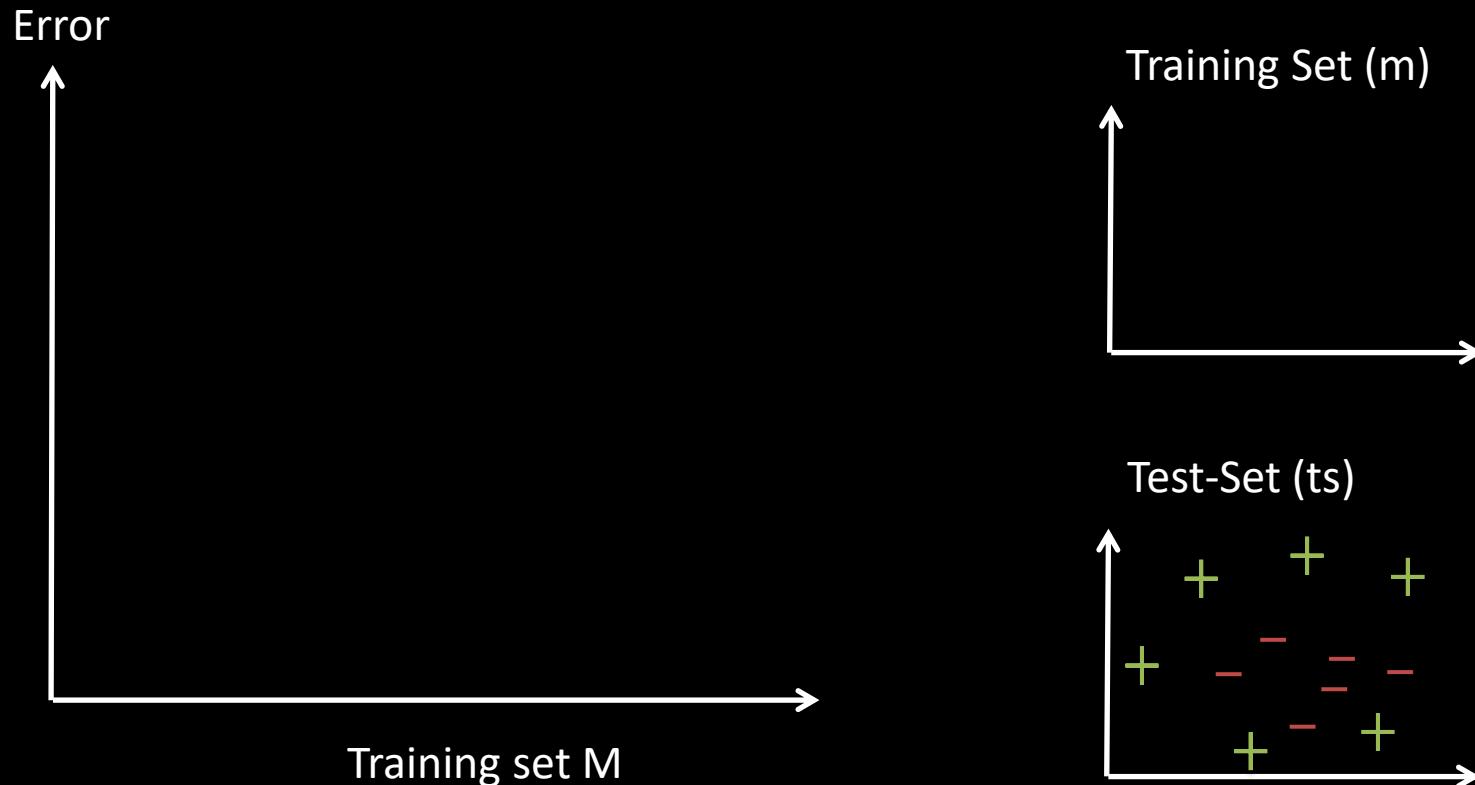
- Try getting more training examples
- Try smaller sets of features
- Try getting additional features
- Try creating features from existing features (kernels)
- Try decrease regularization
- Try increase regularization

Bias and Variance

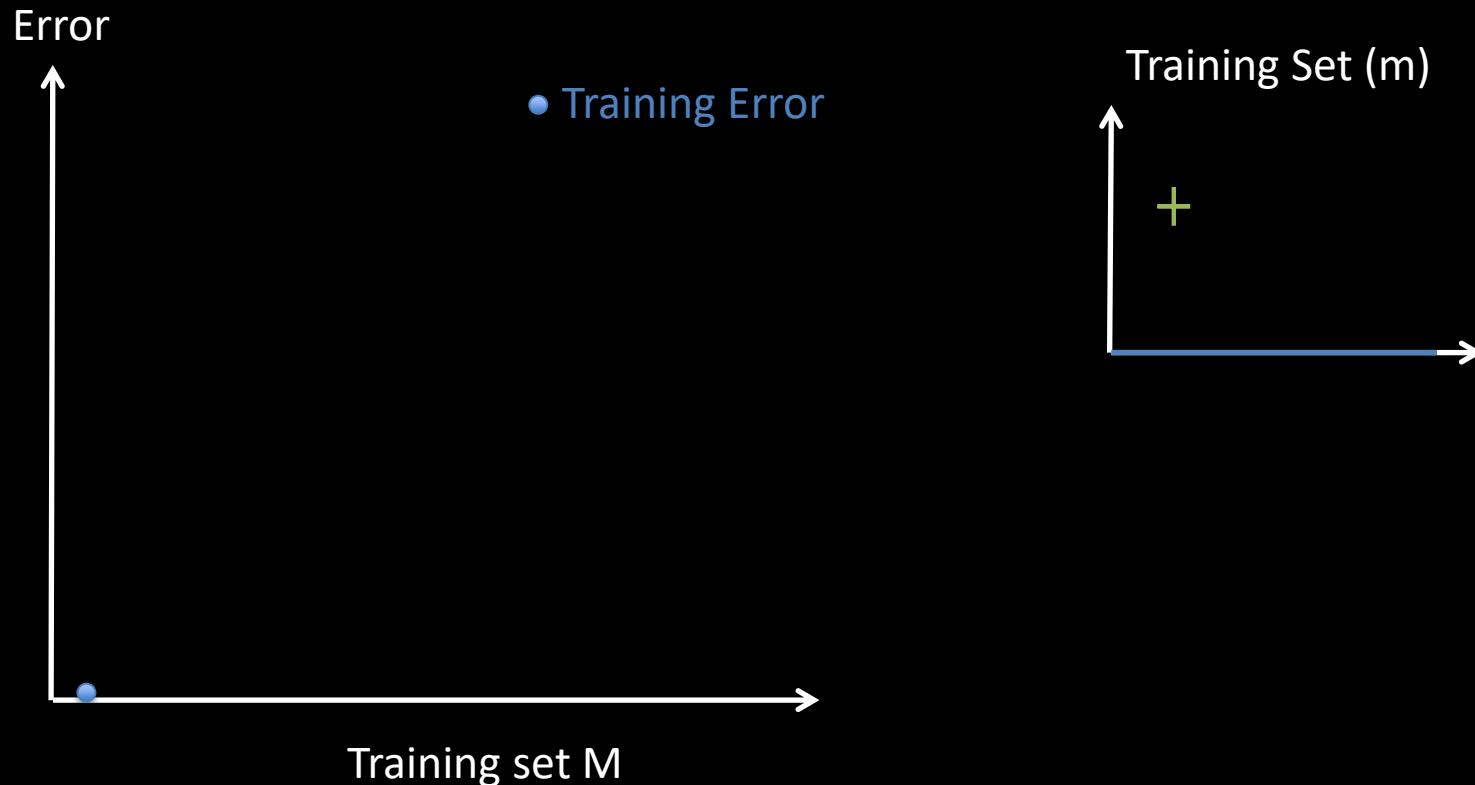


++ - + - ++ - - - + - + - - + + + - - -

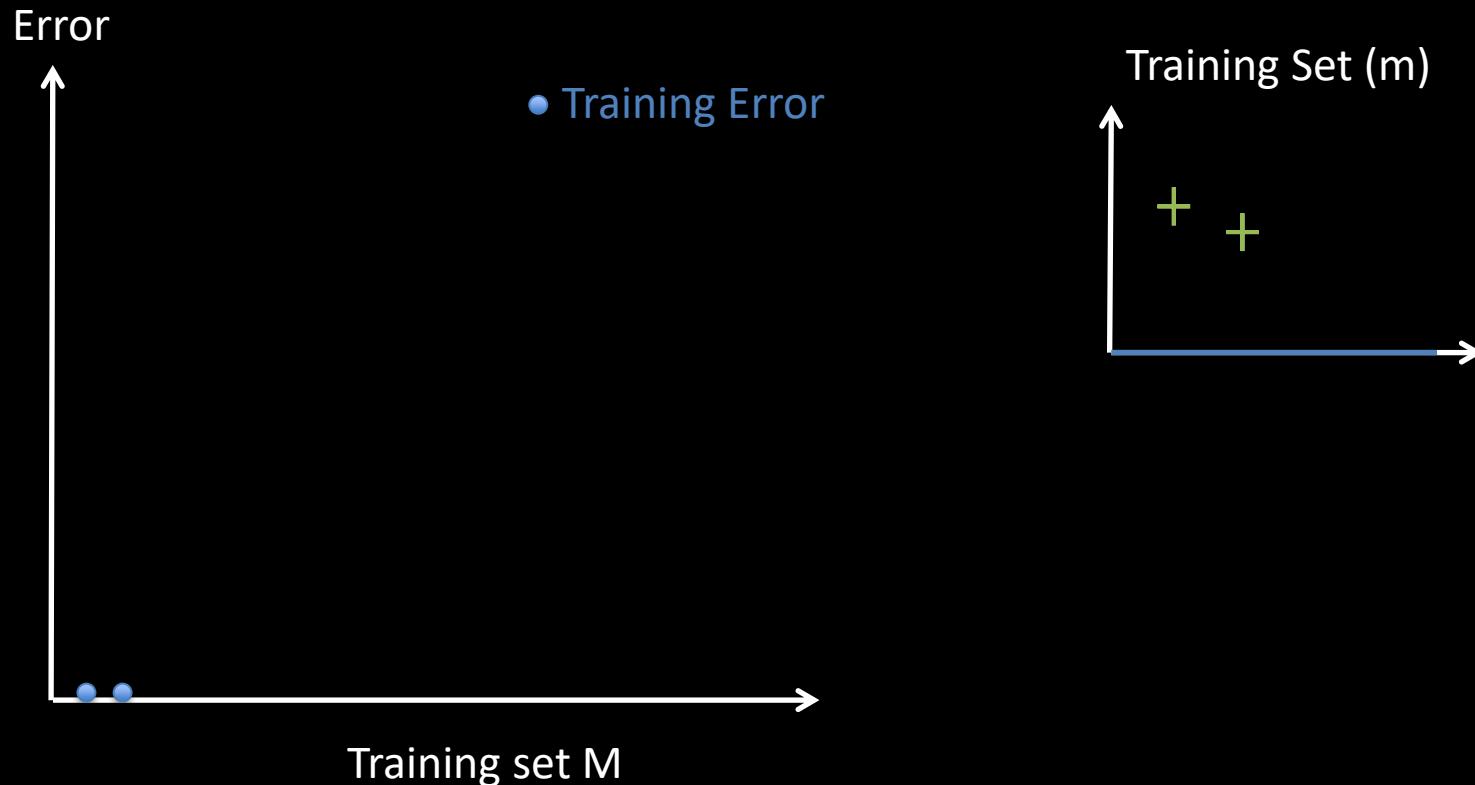
Bias and Variance



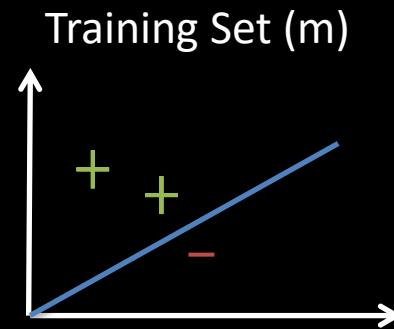
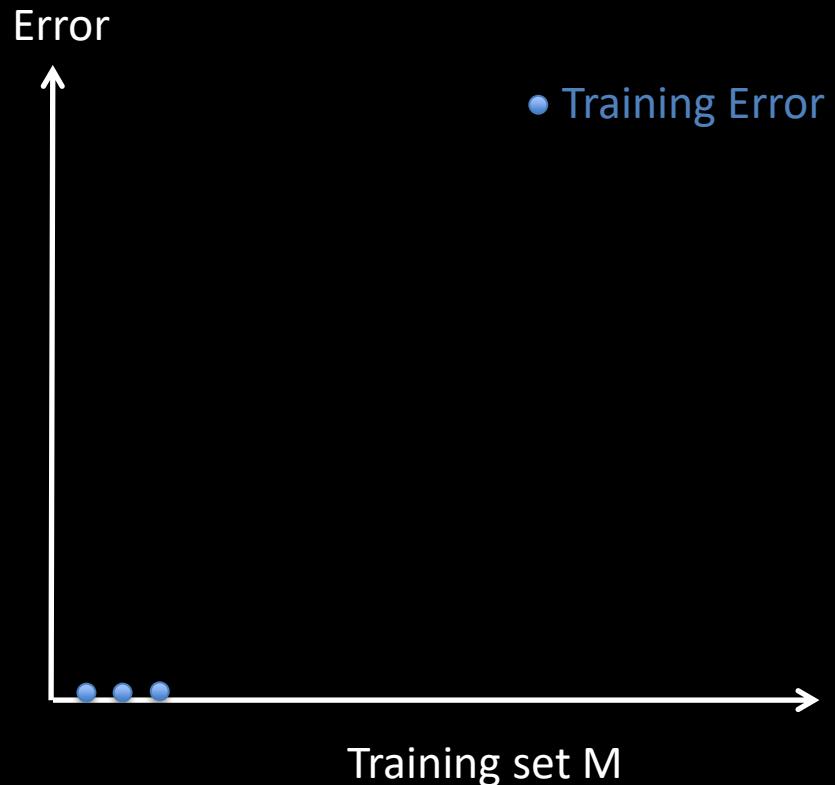
Bias and Variance



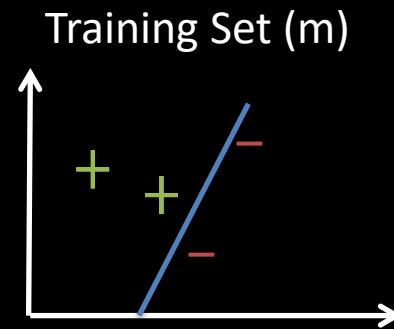
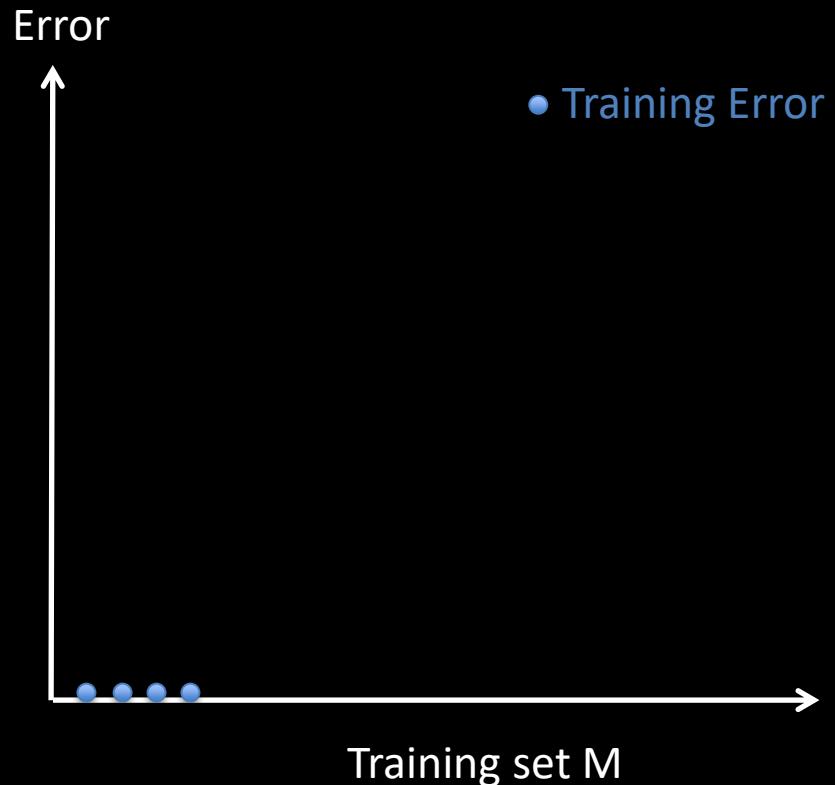
Bias and Variance



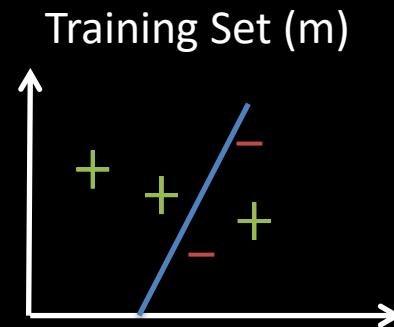
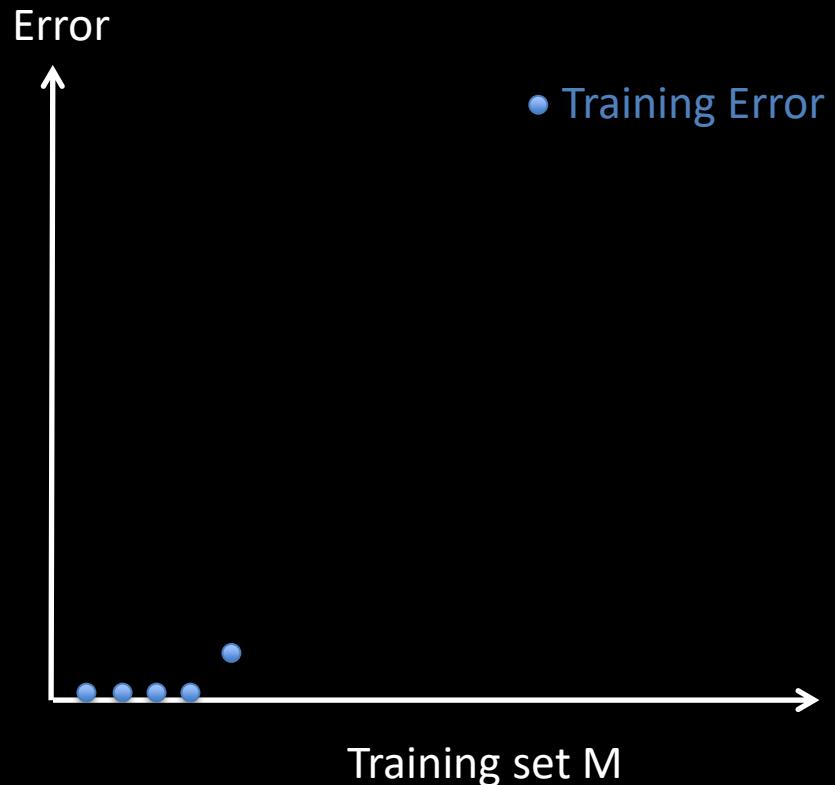
Bias and Variance



Bias and Variance



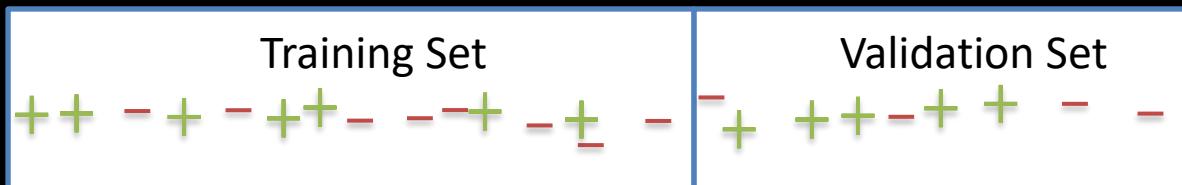
Bias and Variance



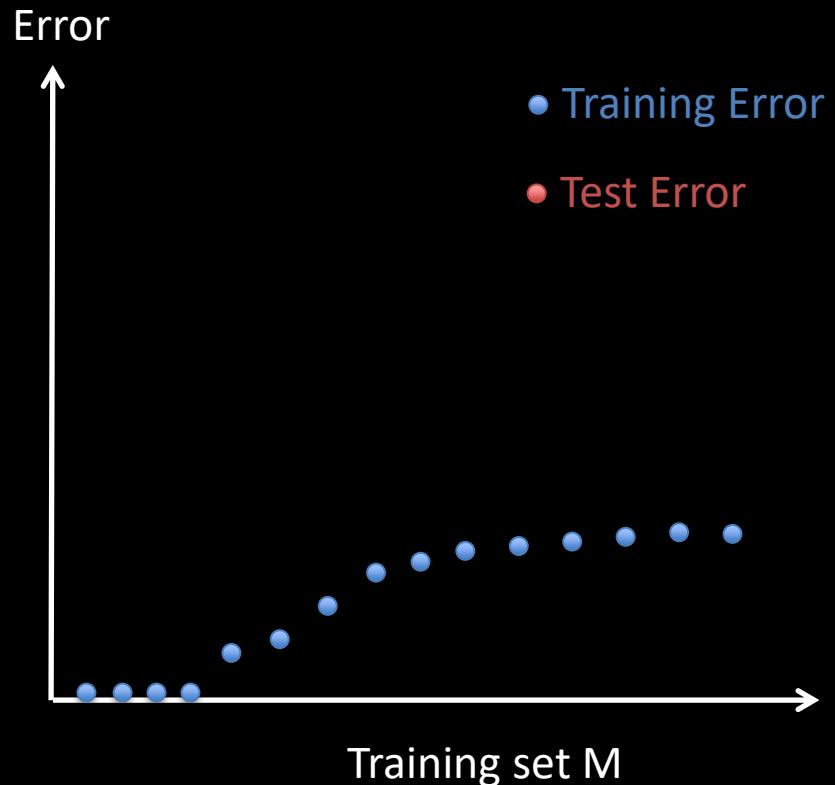
Bias and Variance



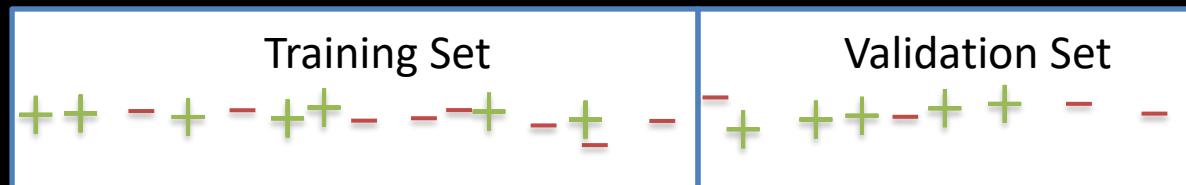
Bias and Variance



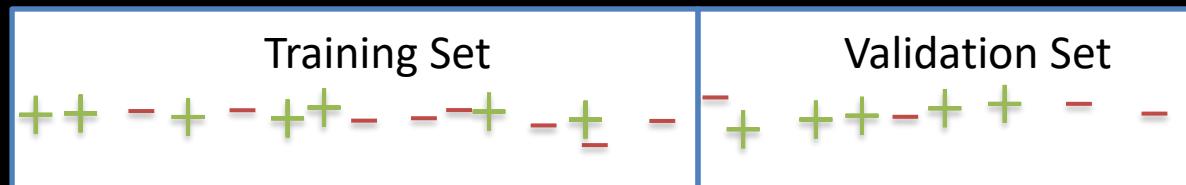
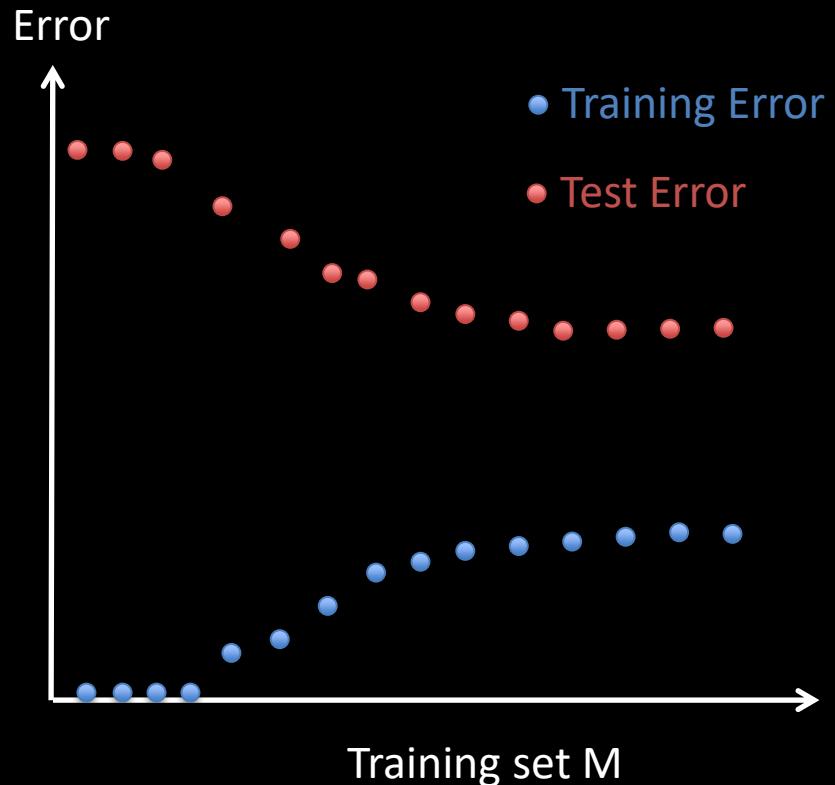
Bias and Variance



Clicker:
Test error
a) decreases with M
b) increases with M
c) stays constant



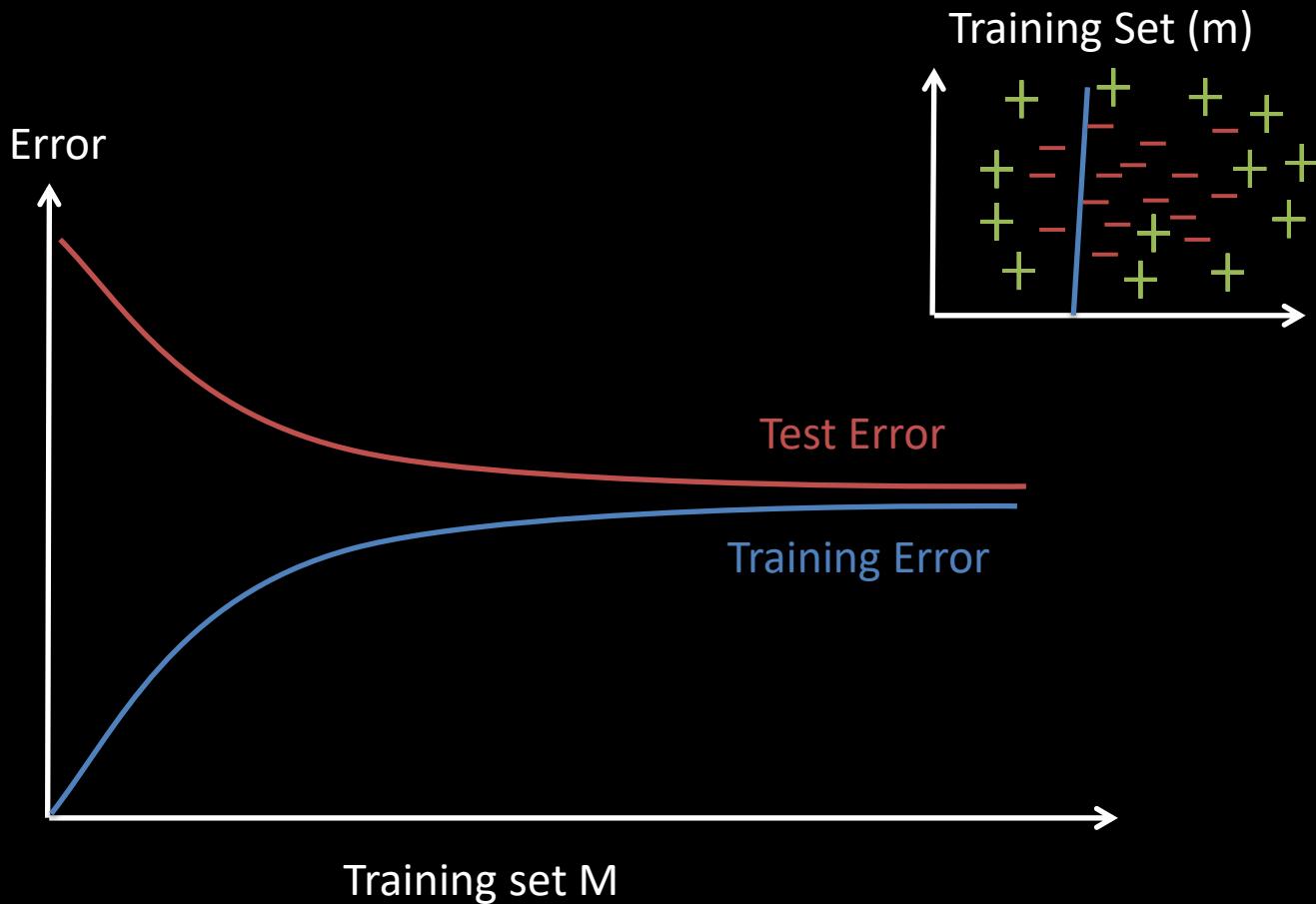
Bias and Variance



High Bias



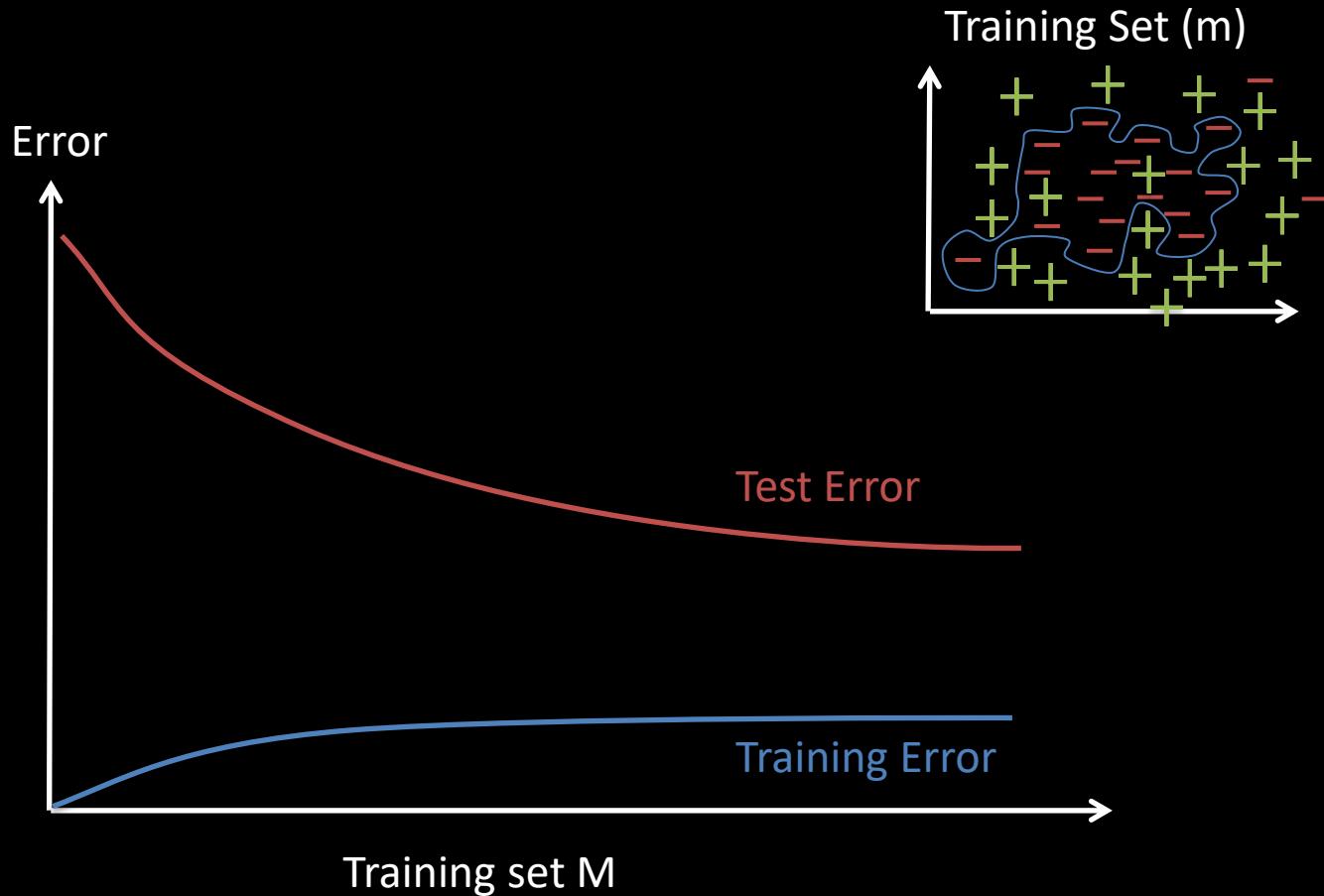
High Bias



Clicker: If you have high-bias, does more data help?

- a) No
- b) Yes

High Variance



Clicker: If you have high-variance, does more data help?

- a) No
- b) Yes

Clicker

1. Get more training examples
2. Try smaller sets of features
3. Try getting additional features
4. Try adding polynomial features (kernels)
5. Try increase regularization
6. Try decrease regularization

Helps with

- A. High Variance
- B. High Bias
- C. Both
- D. None

Clicker

1. Get more training examples
2. **Try smaller sets of features**
3. Try getting additional features
4. Try adding polynomial features (kernels)
5. Try increase regularization
6. Try decrease regularization

Helps with

- A. High Variance
- B. High Bias
- C. Both
- D. None

Clicker

1. Get more training examples
2. Try smaller sets of features
- 3. Try getting additional features**
4. Try adding polynomial features (kernels)
5. Try increase regularization
6. Try decrease regularization

Helps with

- A. High Variance
- B. High Bias
- C. Both
- D. None

Clicker

1. Get more training examples
2. Try smaller sets of features
3. Try getting additional features
4. **Try adding polynomial features (kernels)**
5. Try increase regularization
6. Try decrease regularization

Helps with

- A. High Variance
- B. High Bias
- C. Both
- D. None

Clicker

1. Get more training examples
2. Try smaller sets of features
3. Try getting additional features
4. Try adding polynomial features (kernels)
5. Try increase regularization
6. Try decrease regularization

Helps with

- A. High Variance
- B. High Bias
- C. Both
- D. None

Clicker

1. Get more training examples
2. Try smaller sets of features
3. Try getting additional features
4. Try adding polynomial features (kernels)
5. Try increase regularization
6. **Try decrease regularization**

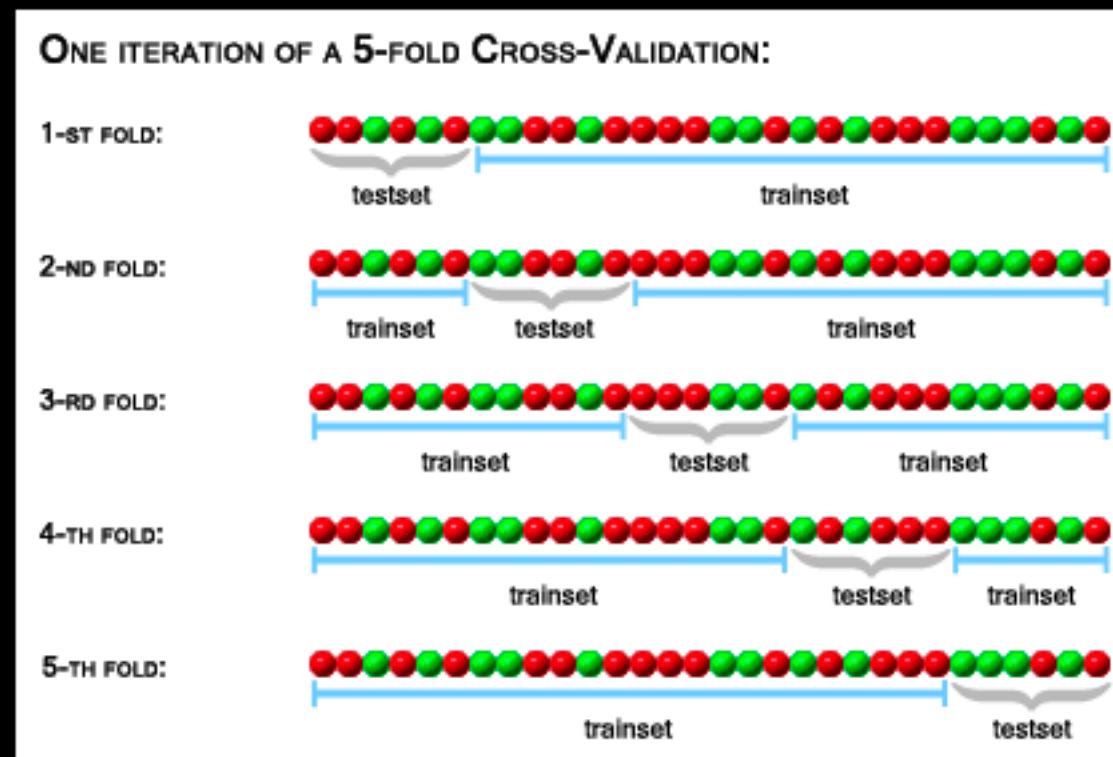
Helps with

- A. High Variance
- B. High Bias
- C. Both
- D. None

Cross-validation

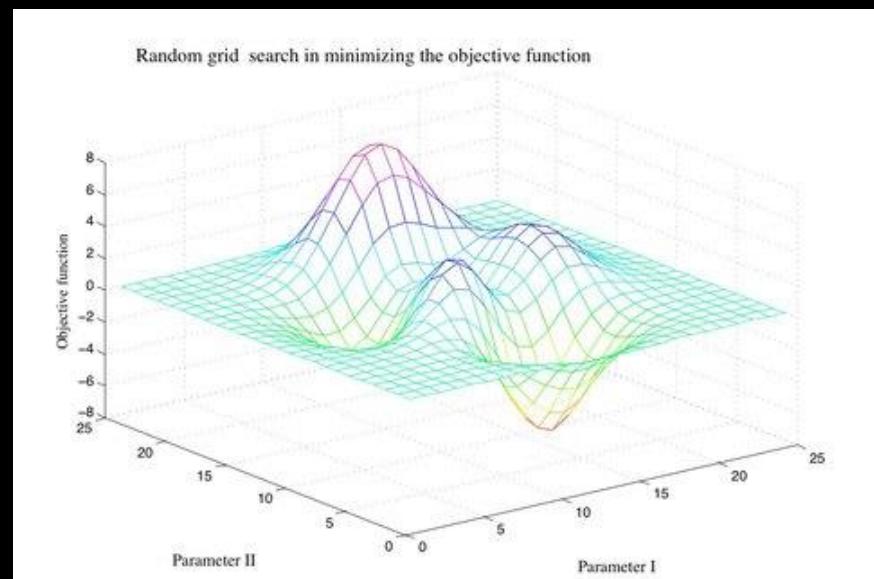
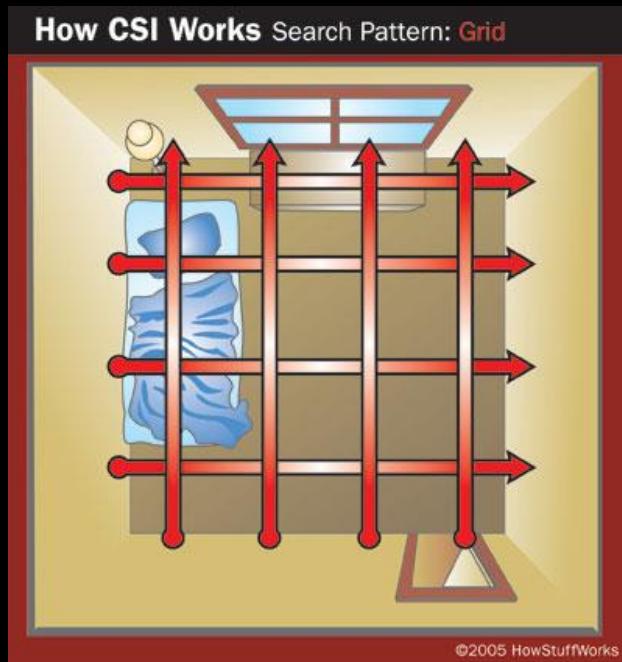
k-fold: split the data into k groups, train on every group except for one, which you test on.

Repeat for all groups



Parameter Tuning

Grid Search



How to speed-up tuning?

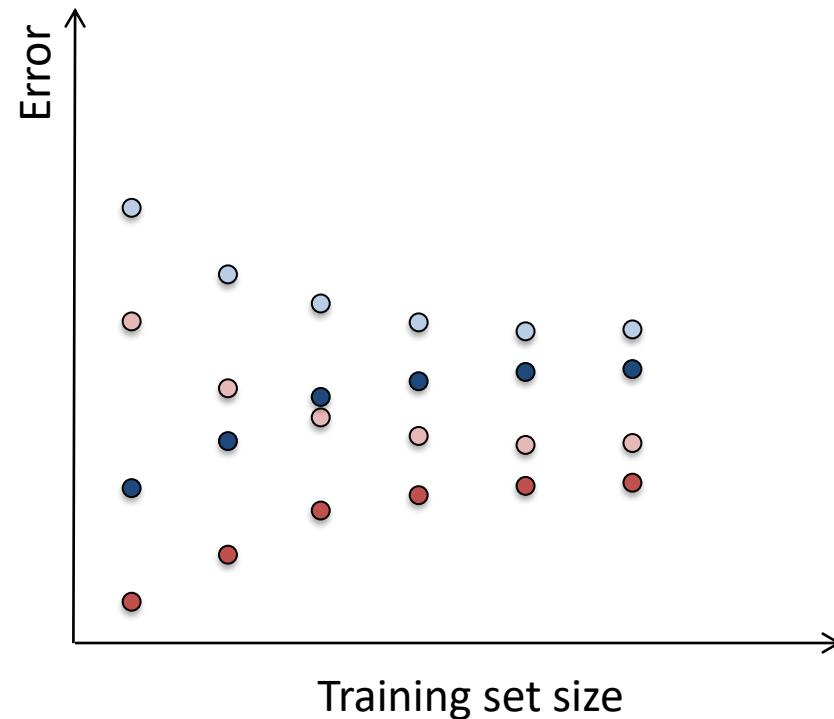
Can we use sampling?

Algorithm 1:

● Training ○ Validation

Algorithm 2:

● Training ○ Validation



How to speed-up tuning?

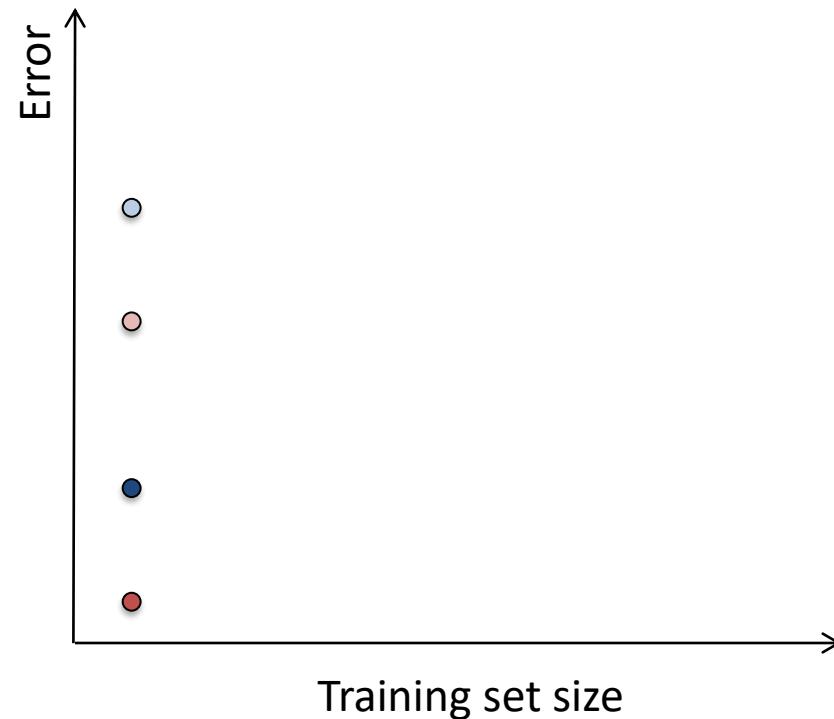
Can we use sampling?

Algorithm 1:

- Training
- Validation

Algorithm 2:

- Training
- Validation



How to speed-up tuning?

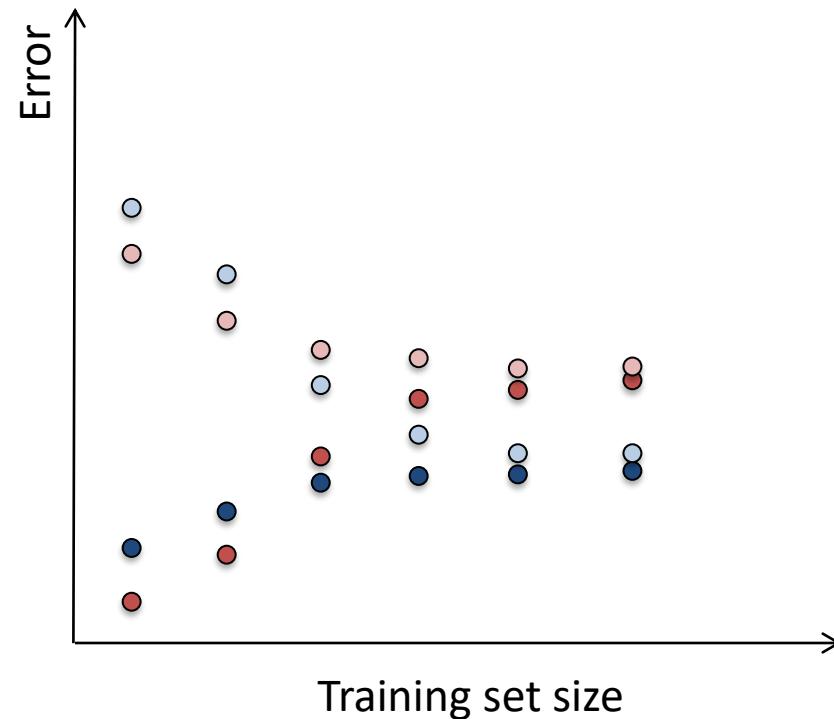
Can we use sampling?

Algorithm 1:

- Training
- Validation

Algorithm 2:

- Training
- Validation



How to speed-up tuning?

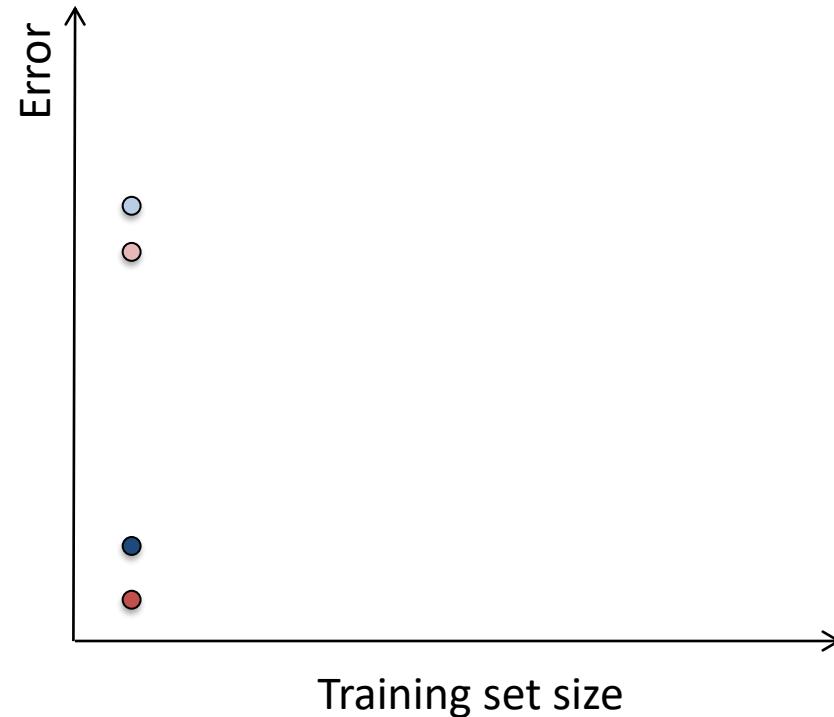
Can we use sampling?

Algorithm 1:

- Training
- Validation

Algorithm 2:

- Training
- Validation



How to speed-up tuning?

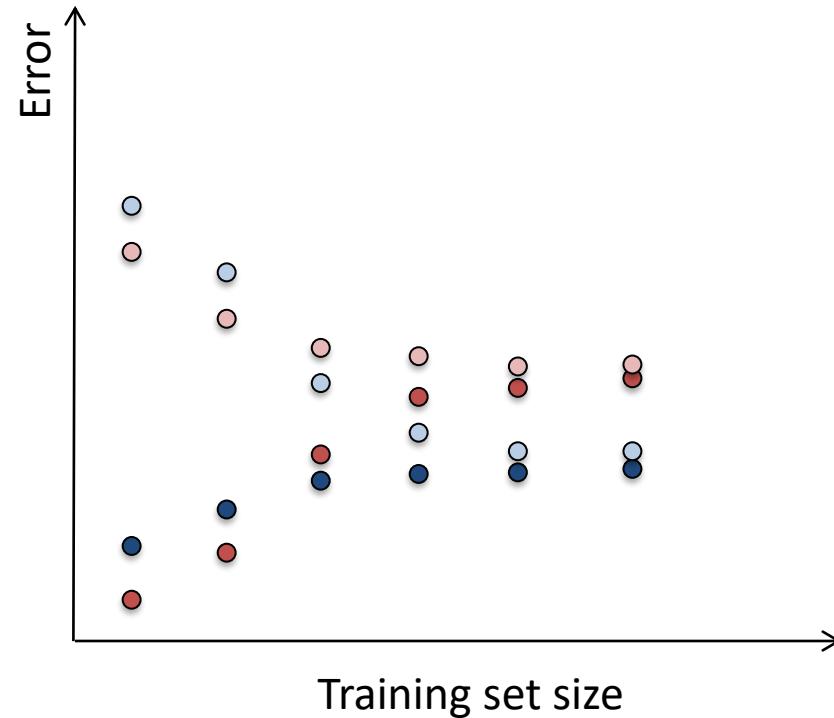
Can we use sampling?

Algorithm 1:

- Training
- Validation

Algorithm 2:

- Training
- Validation



How to speed-up tuning?

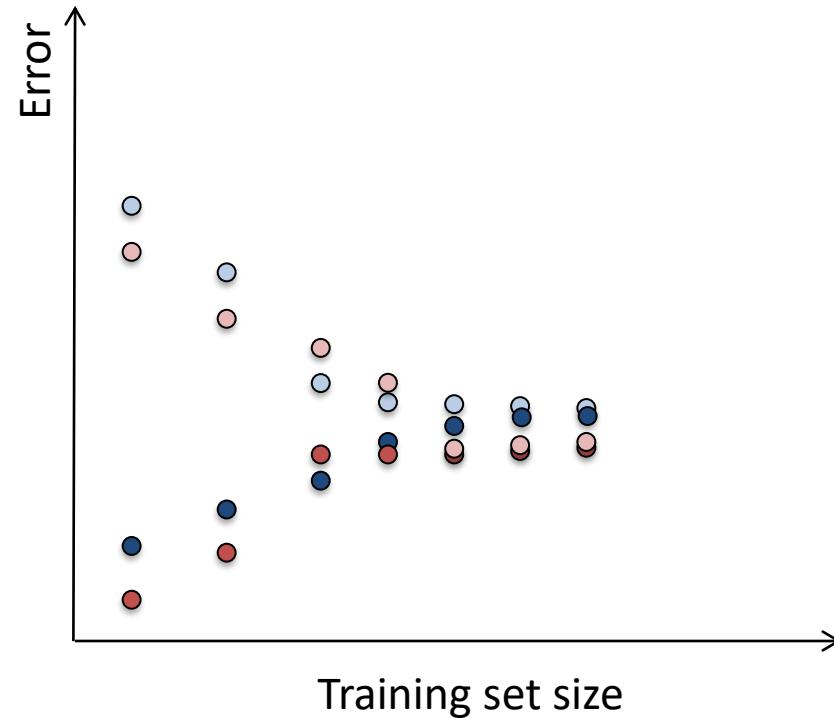
Can we use sampling?

Algorithm 1:

● Training ○ Validation

Algorithm 2:

● Training ○ Validation



Can we prune now?

How to speed-up tuning?

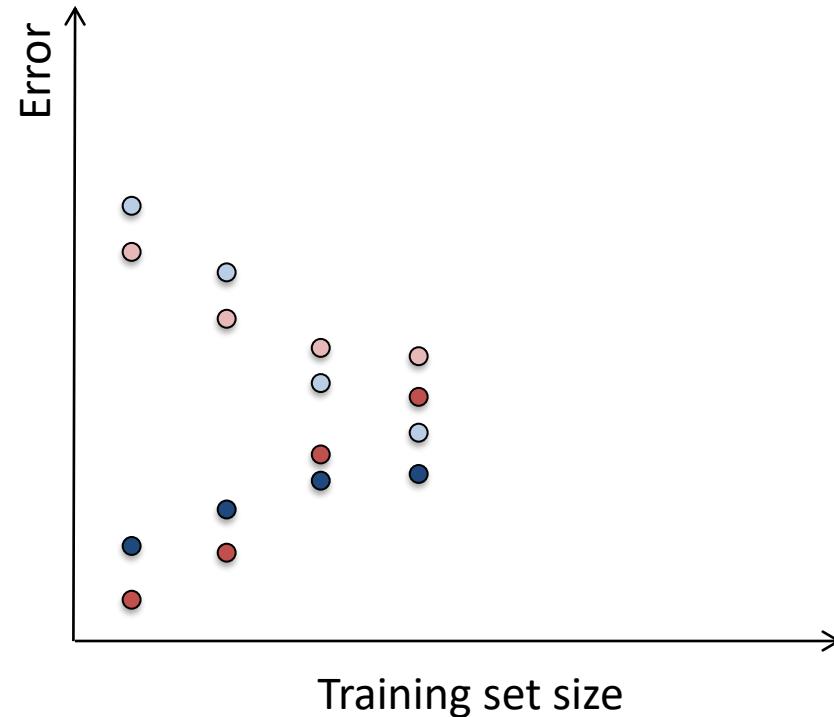
Can we use sampling?

Algorithm 1:

● Training ○ Validation

Algorithm 2:

● Training ○ Validation



Algorithm 1 training error > Algorithm 2 validation error

Credits

- Kilian Weinberger's Notes for [Cornell's CS4780](#)
- Paul's Online Math [Notes](#)
- Andrej Karpathy's State of GPT Slides
- CMU's 11-785 Introduction to Deep Learning

Outline

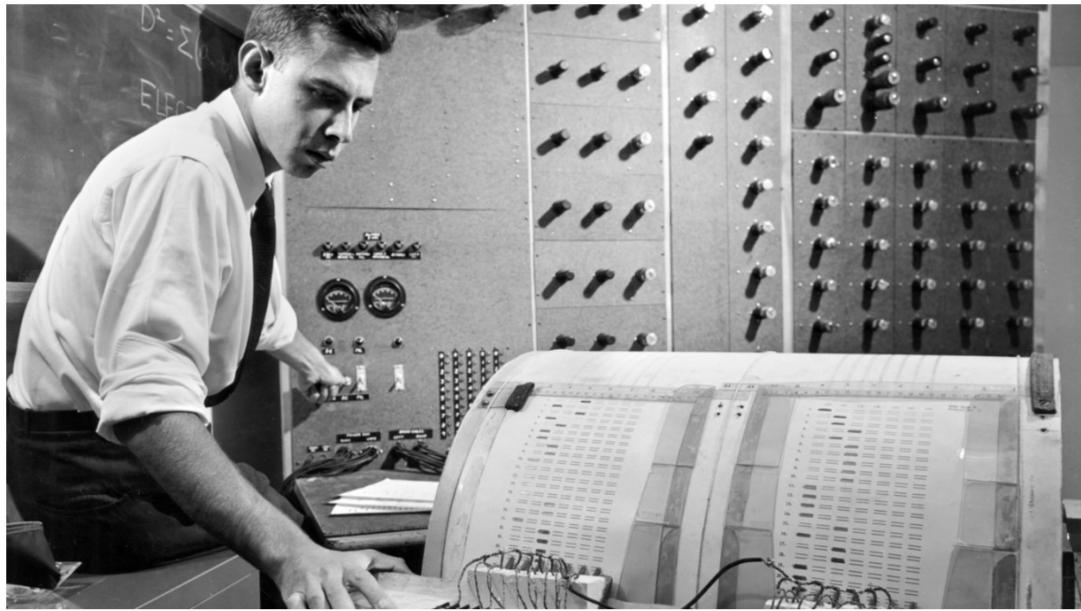
- The Perceptron
- Multi-Layer Perceptron (NN)
 - Stochastic Gradient Descent
 - Forward/Backward Pass
- Convolutional Neural Network (CNN)
- Transformers
- LLMs

Outline

- **The Perceptron**
- Multi-Layer Perceptron (NN)
 - Stochastic Gradient Descent
 - Forward/Backward Pass
- Convolutional Neural Network (CNN)
- Transformers
- LLMs

Perceptron

- Invented by Frank Rosenblatt in 1958
- Described it as “the first machine which is capable of having an original idea”
- Linear classifier which could learn to correctly classify linearly separable data

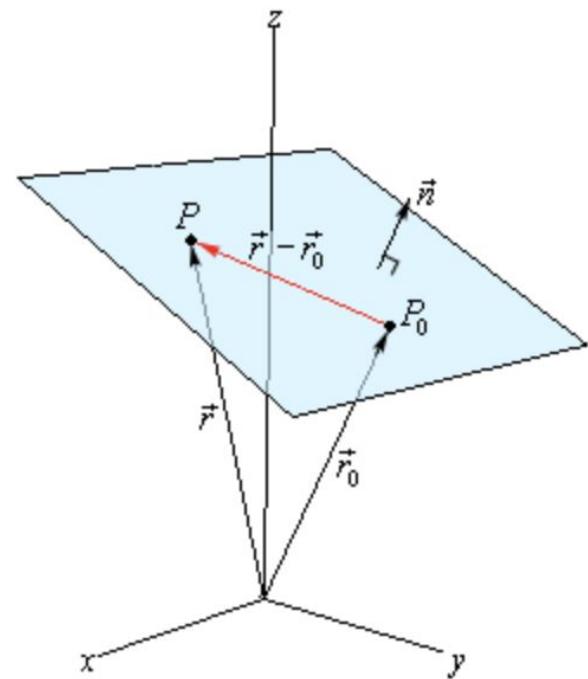


[1] Kilian Weinberger, [CS 4780 Spring 2023 Lecture 3 Notes](#). Cornell University.

[2] Melanie Lefkowitz, [Professor's Perceptron Paved the Way for AI – 60 Years Too Soon](#). Cornell Chronicle.

Recall from... Pre-Calc / (Multivariable) Calc. / Lin. Alg. / etc.

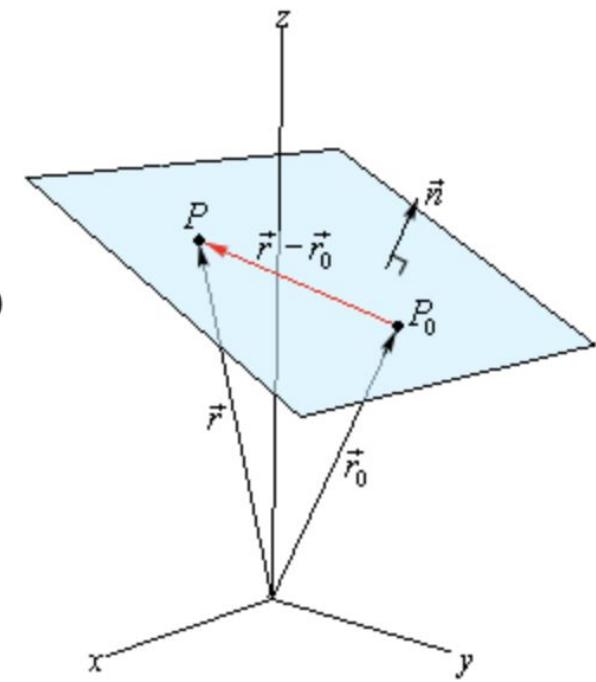
- A (hyper)plane is defined by the vector normal to its surface
- For any vector that lies:
 - In the plane \rightarrow dot product with normal vector = 0
 - On same side of plane \rightarrow " $" > 0$
 - On opp. side of plane \rightarrow " $" < 0$
- We will exploit this mathematical property to create a binary classifier



Recall from... Pre-Calc / (Multivariable) Calc. / Lin. Alg. / etc.

- More formally, the equation for a hyperplane:

$$\left. \begin{array}{l} \vec{n} = \langle a, b, c \rangle \\ \vec{r} = \langle x, y, z \rangle \\ \vec{r}_0 = \langle x_0, y_0, z_0 \rangle \end{array} \right\} \begin{array}{l} \vec{n} \cdot (\vec{r} - \vec{r}_0) = 0 \\ \langle a, b, c \rangle \cdot (\langle x, y, z \rangle - \langle x_0, y_0, z_0 \rangle) = 0 \\ \langle a, b, c \rangle \cdot \langle x - x_0, y - y_0, z - z_0 \rangle = 0 \\ a(x - x_0) + b(y - y_0) + c(z - z_0) = 0 \\ \Rightarrow ax + by + cz = d \end{array}$$



- Which is often re-written as:

$$\left. \begin{array}{l} \vec{w} = \langle a, b, c \rangle \\ \vec{x} = \langle x, y, z \rangle \end{array} \right\} \vec{w}^T \vec{x} + b = 0$$

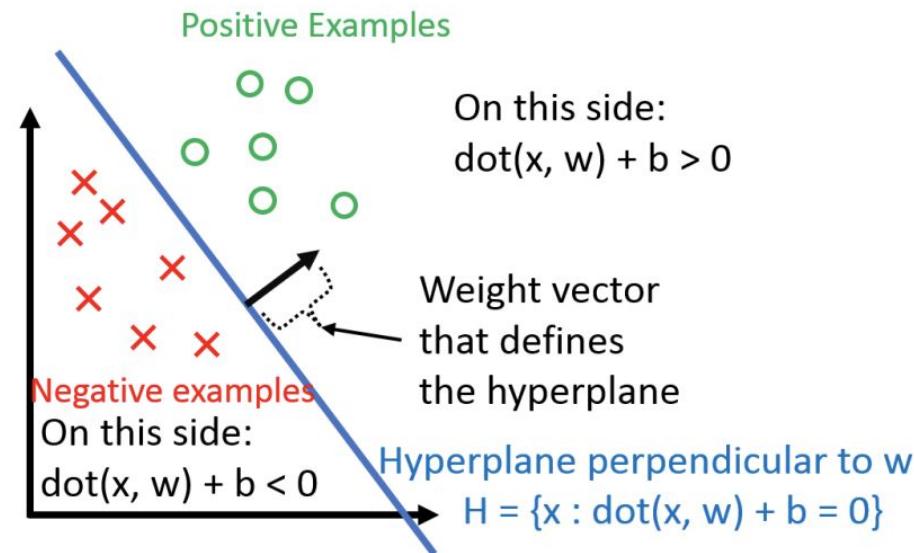
Key Idea (1)

- A hyperplane defines a linear classifier if we take the *sign* of its dot product w/a vector defining a data point

- i.e. we define the classifier:

$$h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b)$$

- Q: given a linearly separable dataset, how do we learn the weights **w** (and **b**) that separate the data?



Note: we can absorb the bias term b into our vectors **w** and **x** by adding a dim. => we usually write $h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$

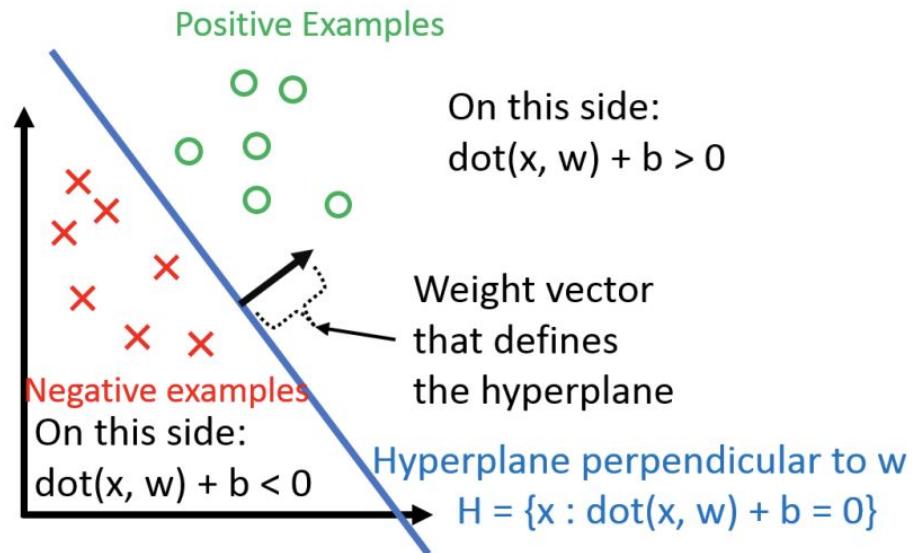
Key Idea (2)

- Assume $y_i \in \{-1, 1\}$; then a data point x_i is classified correctly iff:

$$y_i(\vec{w}^T \vec{x}_i) > 0$$

- If a data point is misclassified, then the following update rule is guaranteed to nudge the classifier closer to a correct prediction:

$$\vec{w}_{t+1} = \vec{w}_t + y_i(\vec{x}_i)$$



Perceptron Update

- If a data point is misclassified, then the following update rule is guaranteed to nudge the classifier closer to a correct prediction:

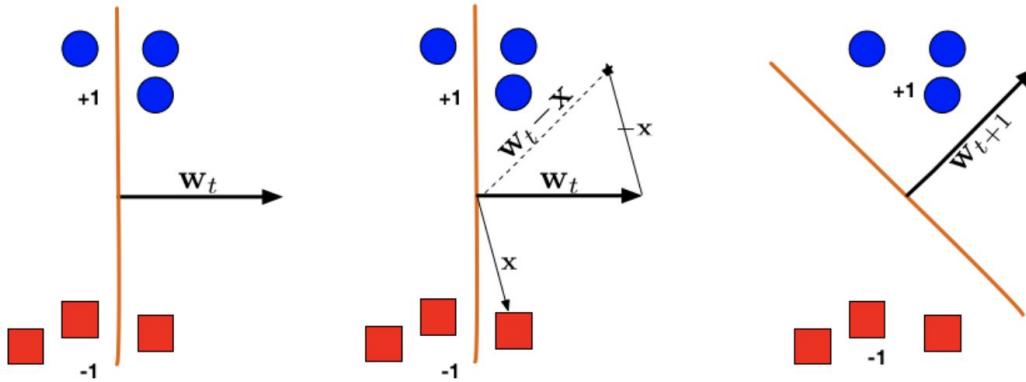
$$\vec{w}_{t+1} = \vec{w}_t + y_i(\vec{x}_i)$$

- To see why, consider the following:

$$\left. \begin{array}{l} y_i(\vec{w}_0^T \vec{x}_i) < 0 \\ \vec{w}_1 = \vec{w}_0 + y_i(\vec{x}_i) \end{array} \right\} \begin{aligned} & y_i(\vec{w}_1^T \vec{x}_i) \\ &= y_i((\vec{w}_0 + y_i(\vec{x}_i))^T \vec{x}_i) \\ &= y_i(\vec{w}_0^T \vec{x}_i) + \| \vec{x}_i \|^2 \\ &> y_i(\vec{w}_0^T \vec{x}_i) \end{aligned}$$

→ new prediction is closer to being correct (i.e. closer to being ≥ 0)

Perceptron Update



*Illustration of a Perceptron update. (Left:) The hyperplane defined by \mathbf{w}_t misclassifies one red (-1) and one blue ($+1$) point. (Middle:) The red point \mathbf{x} is chosen and used for an update. Because its label is -1 we need to **subtract** \mathbf{x} from \mathbf{w}_t . (Right:) The updated hyperplane $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ separates the two classes and the Perceptron algorithm has converged.*

Perceptron Algorithm

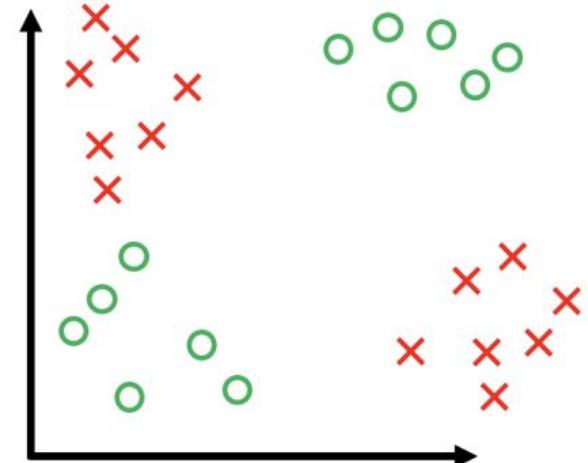
```
Initialize  $\vec{w} = \vec{0}$ 
while TRUE do
     $m = 0$ 
    for  $(x_i, y_i) \in D$  do
        if  $y_i(\vec{w}^T \cdot \vec{x}_i) \leq 0$  then
             $\vec{w} \leftarrow \vec{w} + y\vec{x}$ 
             $m \leftarrow m + 1$ 
        end if
    end for
    if  $m = 0$  then
        break
    end if
end while

// Initialize  $\vec{w}$ .  $\vec{w} = \vec{0}$  misclassifies everything.
// Keep looping
// Count the number of misclassifications,  $m$ 
// Loop over each (data, label) pair in the dataset,  $D$ 
// If the pair  $(\vec{x}_i, y_i)$  is misclassified
// Update the weight vector  $\vec{w}$ 
// Counter the number of misclassification

// If the most recent  $\vec{w}$  gave 0 misclassifications
// Break out of the while-loop
// Otherwise, keep looping!
```

Perceptron Legacy

- Breakthrough learning algorithm that could learn a linear classifier from linearly separable data with strong formal guarantees on convergence
- Initially, helped to generate interest/buzz around A.I.
- Unfortunately, algorithm was too primitive to correctly classify XOR data; demonstrations of this fact contributed to “A.I. winter”

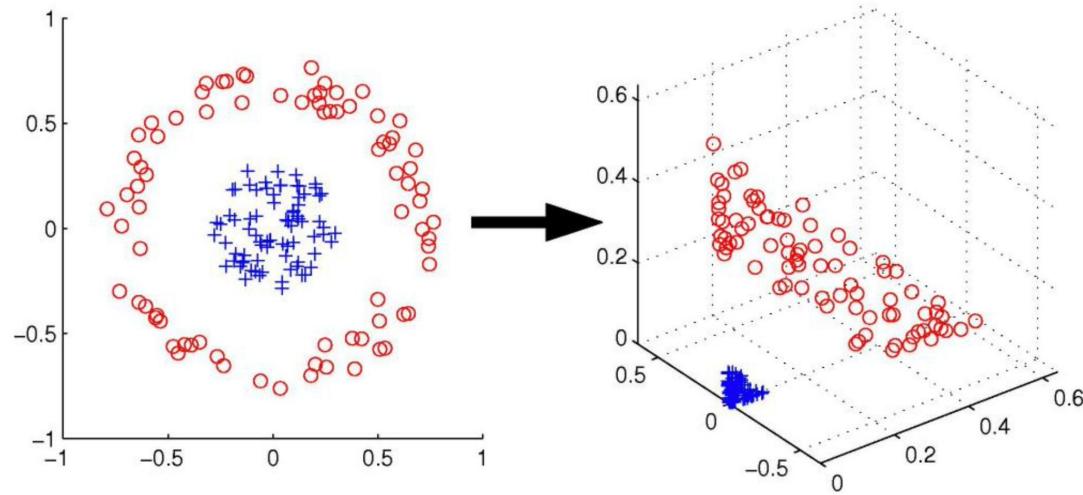


Outline

- The Perceptron
- **Multi-Layer Perceptron (NN)**
 - Stochastic Gradient Descent
 - Forward/Backward Pass
- Convolutional Neural Network (CNN)
- Transformers
- LLMs

Re-Visiting the Kernel Trick for the Perceptron

- Recall that we used the “Kernel Trick” to cast non-linearly separable data into a higher dim. where we could separate it
- **Idea:** introduce non-linearity to the perceptron to allow it to classify non-linearly separable data (e.g. XOR)



$$\begin{aligned}\phi : \quad \mathbb{R}^2 &\longrightarrow \mathbb{R}^3 \\ (x_1, x_2) &\longmapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)\end{aligned}$$

Note: unlike kernels, we will learn this mapping from our dataset

Multi-Layer Perceptron (Feed-Forward Neural Network)

- A single layer of an MLP is defined by the following computation:

$$\phi(x) = \sigma(W\vec{x})$$

- Here, σ is a non-linear activation function (or “transition fcn.”)
 - E.g. $\tanh(x)$, $\text{sigmoid}(x)$, $\max(x, 0) \rightarrow$ (this last one is called “ReLU”)
- Without σ , $\phi(x)$ would still be a linear function, even after composing multiple layers:

$$\phi_i(\vec{x}) = W_i \vec{x} \implies \phi_i(\phi_j(\vec{x})) = W_i(W_j \vec{x}) = \hat{W} \vec{x} \implies \text{linear}$$

Multi-Layer Perceptron (Feed-Forward Neural Network)

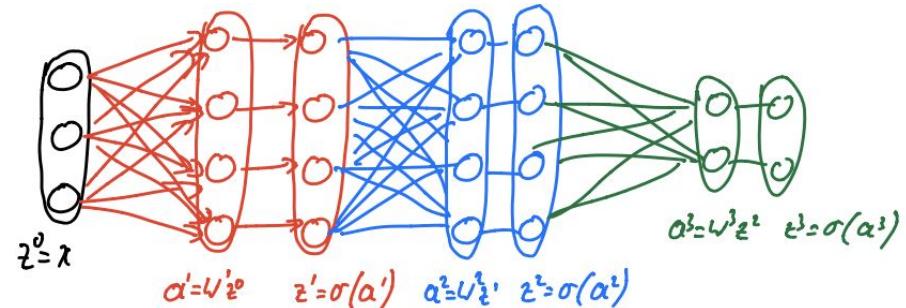
- Example: learning/fitting non-linear function for regression (w/ReLU)

$$\begin{aligned} h(\vec{x}) &= \vec{w}^T \max(W\vec{x} + \vec{b}, 0) + b \\ &= \sum_j w_j \max(\underbrace{\vec{w}_j^T \vec{x}_j + b_j, 0}_{\text{piecewise linear component}}) + b \end{aligned}$$

learnable
on/off switch piecewise linear component bias

$$W = \begin{bmatrix} \dots & \vec{w}_1^T & \dots \\ \dots & \vec{w}_2^T & \dots \\ \dots & \vec{w}_3^T & \dots \end{bmatrix}$$

- Here[^], $\phi(x) = \max(x, 0)$ is non-linear
- We can compose layers to make
Deep Networks →



Multi-Layer Perceptron (Feed-Forward Neural Network)

- **Q:** how are we going learn our model parameters $\theta = W_1, \dots, W_k, b_1, \dots, b_k$ from our dataset?
- **A:** Using stochastic gradient descent:
 - Define a **loss function** (i.e. measure of how far prediction is from groundtruth)
 - Compute gradient of loss w.r.t. model parameters
 - Make small update to model parameters in dir. of gradient
 - (Similar to perceptron update)
 - “Stochastic” b/c we only use sample from dataset (i.e. **minibatch**) to compute loss

Outline

- The Perceptron
- Multi-Layer Perceptron (NN)
 - **Stochastic Gradient Descent**
 - Forward/Backward Pass
- Convolutional Neural Network (CNN)
- Transformers
- LLMs

Stochastic Gradient Descent

- Recall: gradient of function w.r.t. some parameter(s) points in the direction of maximum rate of change of function w.r.t. those parameter(s)
- Gradient descent is an optimization algo. that minimizes some objective fcn. (the loss) w.r.t. its parameters (the model weights θ)

Loss Function: $\mathcal{L} = \sum_{i=1}^n L(h(x_i), y_i)$ E.g.: $L = \frac{1}{2} \sum_{i=1}^n (h(x_i) - y_i)^2$

Gradient: $\nabla_{\theta} \mathcal{L} = \sum_{i=1}^n \frac{\partial L(h(x_i), y_i)}{\partial \theta}$

- **Stochastic** gradient descent approximates the gradient with $m \ll n$ samples (where n is the size of the dataset)

Stochastic Gradient Descent

- Recall our earlier example $w/\theta = \vec{w}^T, W, \vec{b}, b$

$$\begin{aligned} h(\vec{x}) &= \vec{w}^T \max(W\vec{x} + \vec{b}, 0) + b \\ &= \sum_j w_j \max(\vec{w}_j^T x_j + b_j, 0) + b \end{aligned} \quad W = \begin{bmatrix} \dots & \vec{w}_1^T & \dots \\ \dots & \vec{w}_2^T & \dots \\ \dots & \vec{w}_3^T & \dots \end{bmatrix}$$

- SGD would update our parameters as follows (α is **learning rate**):

$$\vec{w}^T \leftarrow \vec{w}^T - \alpha \left(\frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial \vec{w}^T} \right) \quad \vec{b} \leftarrow \vec{b} - \alpha \left(\frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial \vec{b}} \right)$$

$$W \leftarrow W - \alpha \left(\frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial W} \right) \quad b \leftarrow b - \alpha \left(\frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial b} \right)$$

Outline

- The Perceptron
- Multi-Layer Perceptron (NN)
 - Stochastic Gradient Descent
 - **Forward/Backward Pass**
- Convolutional Neural Network (CNN)
- Transformers
- LLMs

Forward Pass

- **Forward Pass: (prediction)**
- $z_0 = x$ (\leftarrow input)
- For $l = 1, \dots, L$
 - $a_l = W^l(z_{\{l-1\}}) + b_l$
 - $z_l = \phi(a_l)$
- End
- Return z_L
- Note: typically we do not apply an activation function ϕ at the final layer

Backward Pass

- Backward Pass: (gradient update)

$$\vec{\delta}_L = \frac{\delta_L}{\partial z_L} \odot \sigma'(a_L)$$

for $l = L-1:-1:1$

$$w_l = w_l - \alpha \vec{\delta}_l z_{l-1}^T$$

$$b_l = b_l - \alpha \vec{\delta}_l$$

$$\vec{\delta}_{l-1} = \sigma'_{l-1}(a_{l-1}) \odot (w_l^T \vec{\delta}_l)$$

End

\odot is the element wise product

$$\begin{pmatrix} a \\ b \end{pmatrix} \odot \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a \cdot c \\ b \cdot d \end{pmatrix}$$

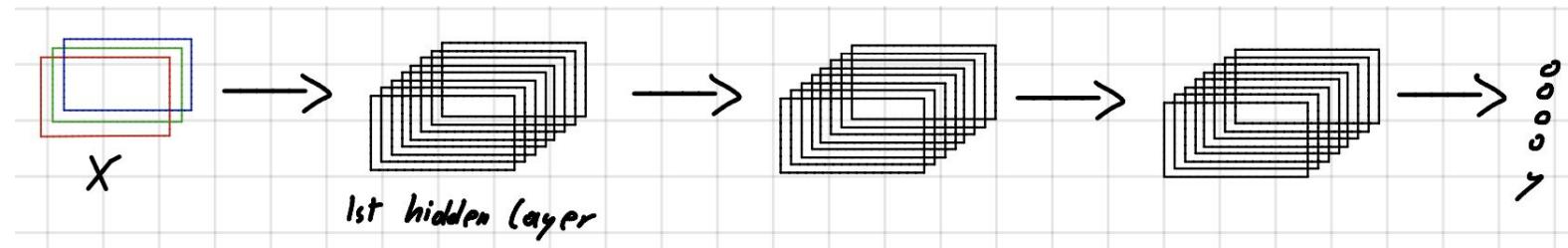
σ' is the gradient of the transition function σ

Outline

- The Perceptron
- Multi-Layer Perceptron (NN)
 - Stochastic Gradient Descent
 - Forward/Backward Pass
- **Convolutional Neural Network (CNN)**
- Transformers
- LLMs

Convolutional Neural Networks

- Typically used for training models on image/video inputs
- Images often have dimension: $W \times H \times 3$
- CNNs restrict linear transformations to be convolutions:
 - Shared weights
 - Ensures intermediate representations are also “images”



- Final layer(s) are usually a flatten operation followed by a MLP

Convolution operation:

convolution
kernel

input image				
1	3	2	5	7
6	0	1	1	3
2	5	2	7	2

$$\times \quad \begin{matrix} 1 & 2 \\ 0 & 1 \end{matrix} = \quad \begin{matrix} 7 & 7 & 13 & 21 \\ 11 & 4 & 9 & 9 \end{matrix}$$

These are the learned weights

Typically you have one kernel per input channel. The output is the sum of the convolved images.

$$\begin{matrix} 1 & 3 & 2 & 5 & 7 \\ 6 & 0 & 1 & 1 & 3 \\ 2 & 5 & 2 & 7 & 2 \end{matrix} \times \begin{matrix} 1 & 2 \\ 0 & 1 \end{matrix} = \begin{matrix} 7 \\ 11 & 4 & 9 & 9 \end{matrix}$$

Convolution can be interpreted as a pattern detector. Patches that look similar to the kernel will yield large activations. The conv. operator 'scans' the input for such patterns.

$$7 = 1 \cdot 1 + 3 \cdot 2 + 6 \cdot 0 + 0 \cdot 1$$

$$9 = 1 \cdot 1 + 3 \cdot 2 + 7 \cdot 0 + 2 \cdot 1$$

Convolutional Neural Networks

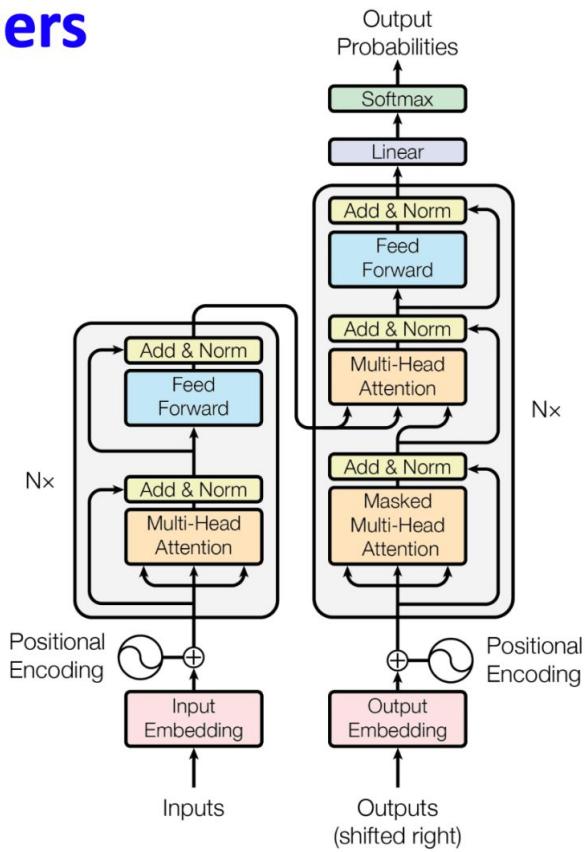
- Lots of research has gone into optimizing CNNs:
- (Max) Pooling
 - Helps w/translation invariance b/c small shifts in object yield same activation output
- Batch-Norm / Layer-Norm
 - Normalize outputs at each layer to better stabilize learning
- Residual Connections
 - Help to counteract vanishing gradients in deep networks
- And more...

Outline

- The Perceptron
- Multi-Layer Perceptron (NN)
 - Stochastic Gradient Descent
 - Forward/Backward Pass
- Convolutional Neural Network (CNN)
- **Transformers**
- LLMs

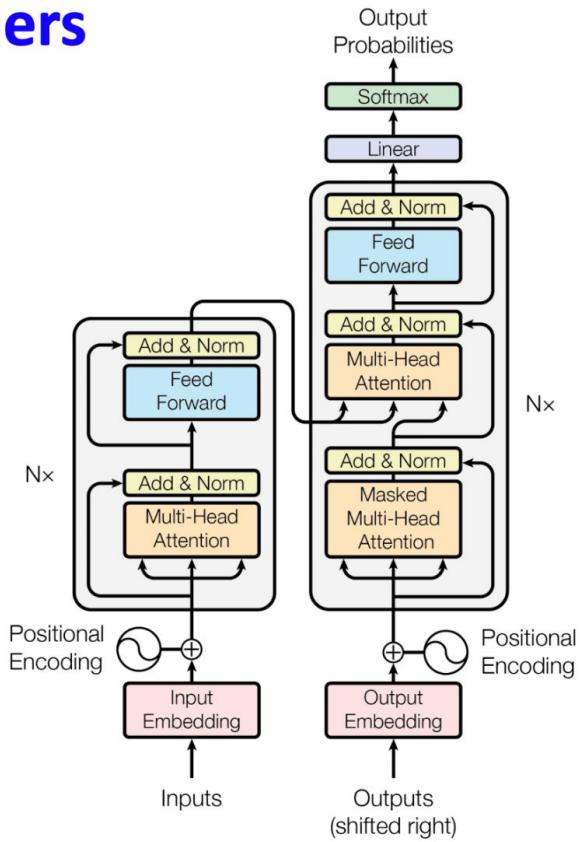
Transformers

- Tokenization
- Input Embeddings
- Position Encodings
- Residuals
- Query
- Key
- Value
- Add & Norm
- Encoder
- Decoder
- Attention
- Self Attention
- Multi Head Attention
- Masked Attention
- Encoder Decoder Attention
- Output Probabilities / Logits
- Softmax
- Encoder-Decoder models
- Decoder only models



Transformers

- Tokenization
 - Input Embeddings
 - Position Encodings
 - Residuals
 - Query
 - Key
 - Value
 - Add & Norm
 - Encoder
 - Decoder
- Attention
• Self Attention
• Multi Head Attention
• Masked Attention
• Encoder Attention
• Decoder Attention
• Probabilities / Logits
-
- Encoder
Decoder models
Encoder only models



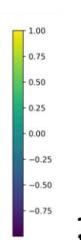
Input Embeddings

Initialization:

Input is a set (or sequence) of tokens from a finite dictionary.

E.g.

Text: She works in the white house.



$\downarrow z_1 \downarrow z_2 \downarrow z_3 \downarrow z_4 \downarrow z_5 \downarrow z_6 \downarrow z_7$ Add positional embedding

Map tokens to word vectors. $z_i \in \mathbb{R}^d$
These vectors are learned

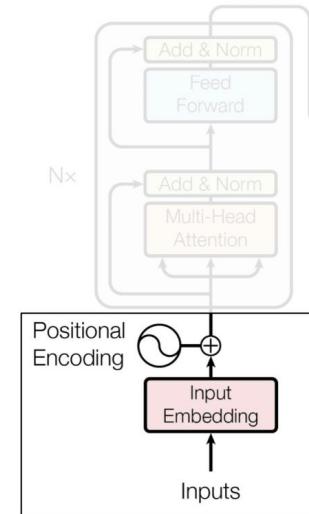
P

$$Z = [z_1 \dots z_n]$$

$$Z \leftarrow Z + P \quad P_{k,2i} = \sin\left(\frac{k}{n^{2i}}\right) \quad P_{k,2i+1} = \cos\left(\frac{k}{n^{2i}}\right)$$

The vector z_i disills the "meaning" of the i th word. Similar words have similar embeddings. e.g. "white" \approx "blue"

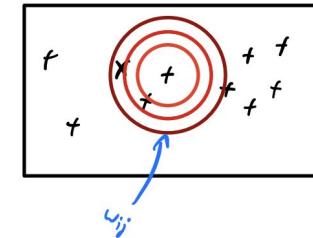
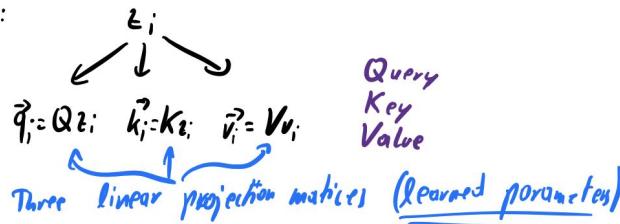
Issue: The vector z_5 of "white" should change as it is followed by "house". Words modify each other's meaning.



Self-Attention

(Self-)Attention: High level: Each vector becomes a weighted average of its nearest neighbors and itself.

Subtleties:



Similarity measure between \vec{z}_i & \vec{z}_j : $s_{ij} = \frac{\vec{q}_i^T \vec{k}_j}{\sqrt{d}}$

Weight of \vec{z}_j as neighbor for \vec{z}_i : $w_{ij} = \frac{e^{s_{ij}}}{\sum_{i'} e^{s_{ij}}} \leftarrow$ softmax function

Self-attention: $\vec{z}'_i \leftarrow \sum_{j=1}^n w_{ij} \vec{v}_j$ Skip connection

$$\vec{z}' \leftarrow V \underbrace{\text{softmax}(K^T Q^T \vec{z})}_{W^T G I R^{n \times n}} + \vec{z}$$

Self-Attention

Query	Key	Value
1. Search for info	1. Interacts directly with Queries 2. Distinguishes one object from another 3. Identify which object is the most relevant and by how much	1. Actual details of the object 2. More fine grained

Attention



I_1

|



I_2

ate



I_3

an



I_4

apple

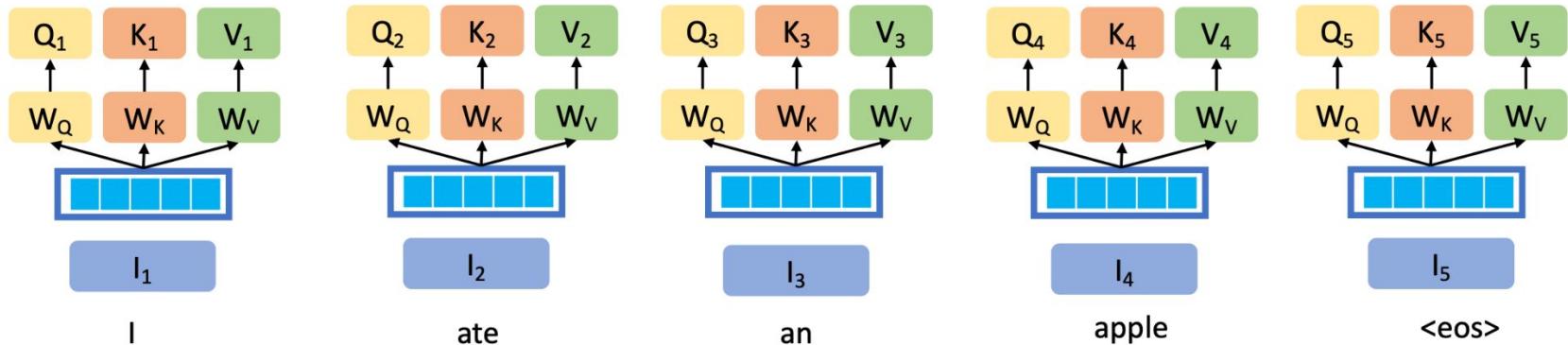


I_5

<eos>

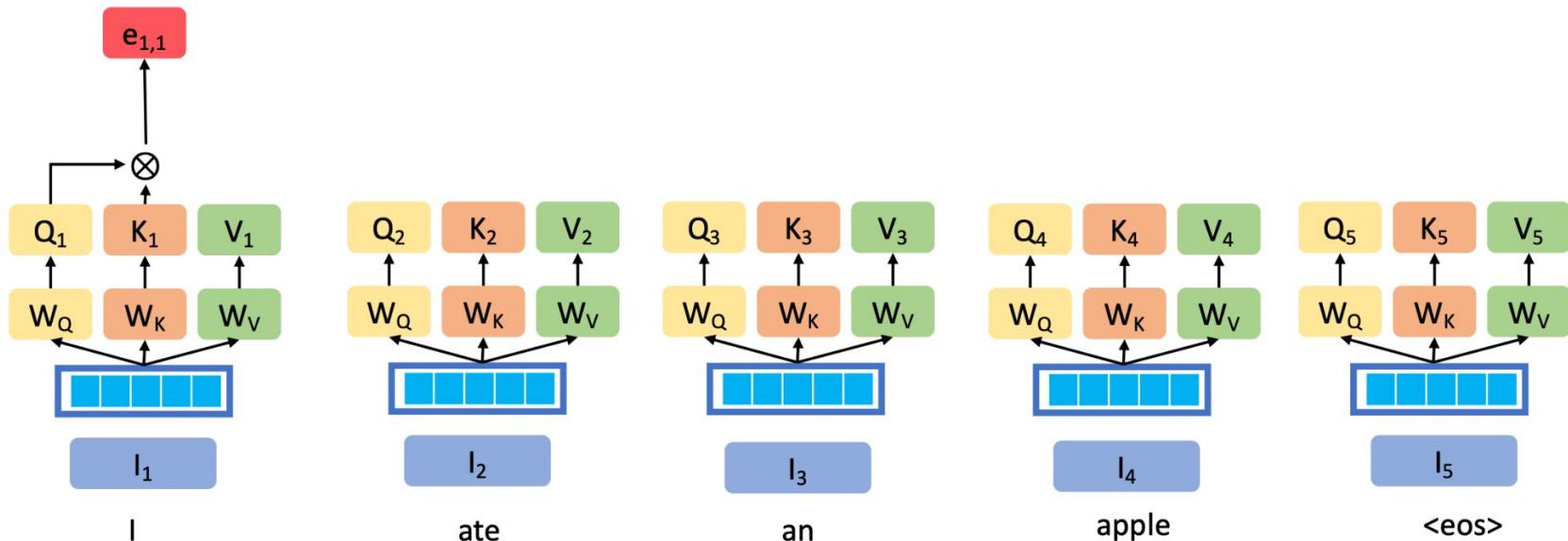
Attention

Dimensions across QKV have been dropped for brevity



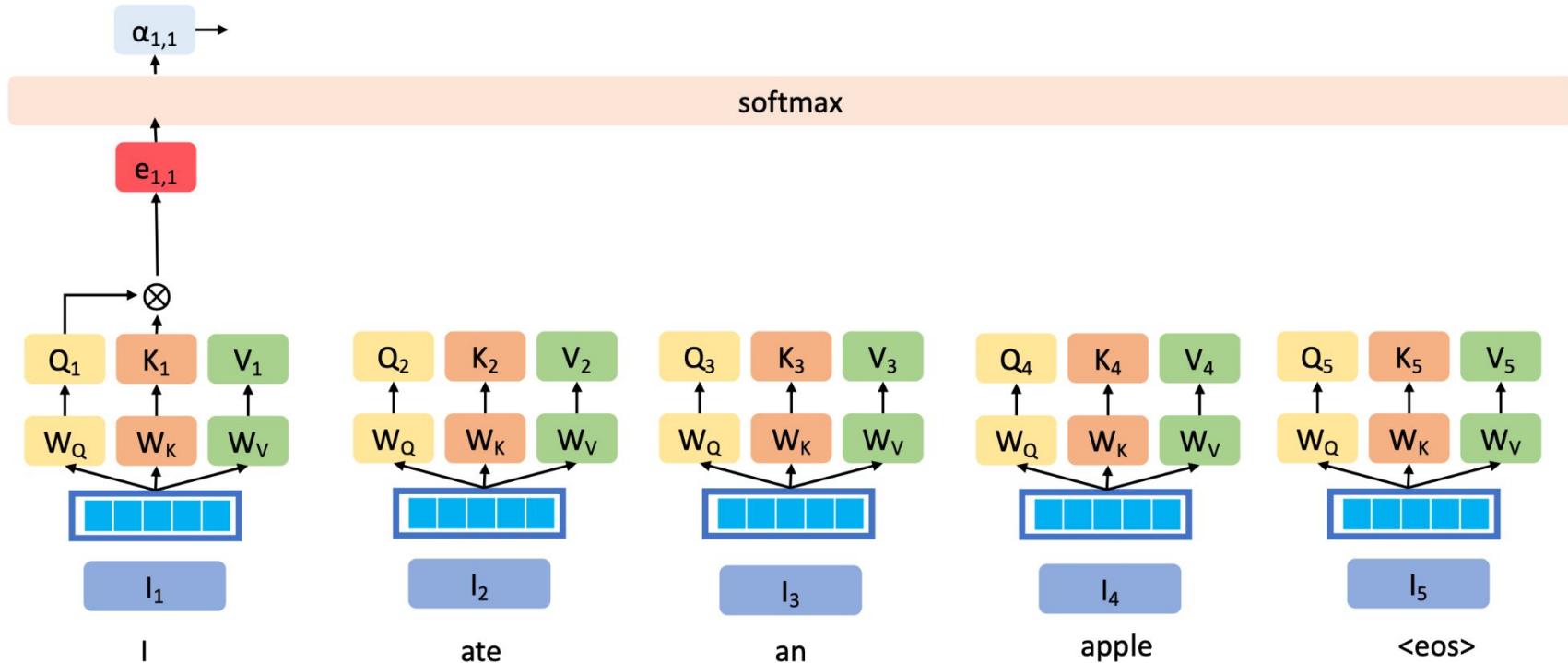
Attention

Dimensions across QKV have been dropped for brevity

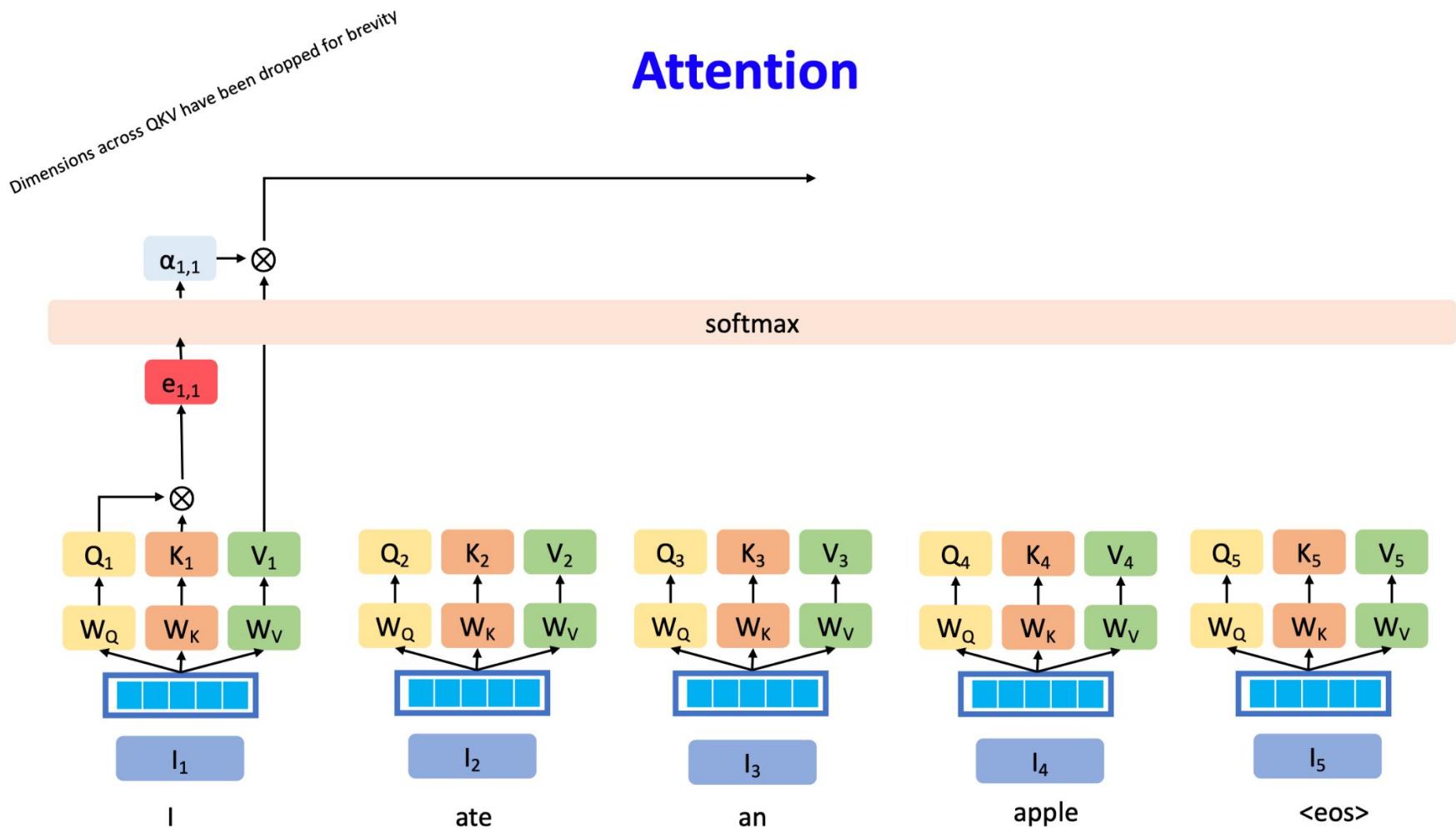


Attention

Dimensions across QKV have been dropped for brevity

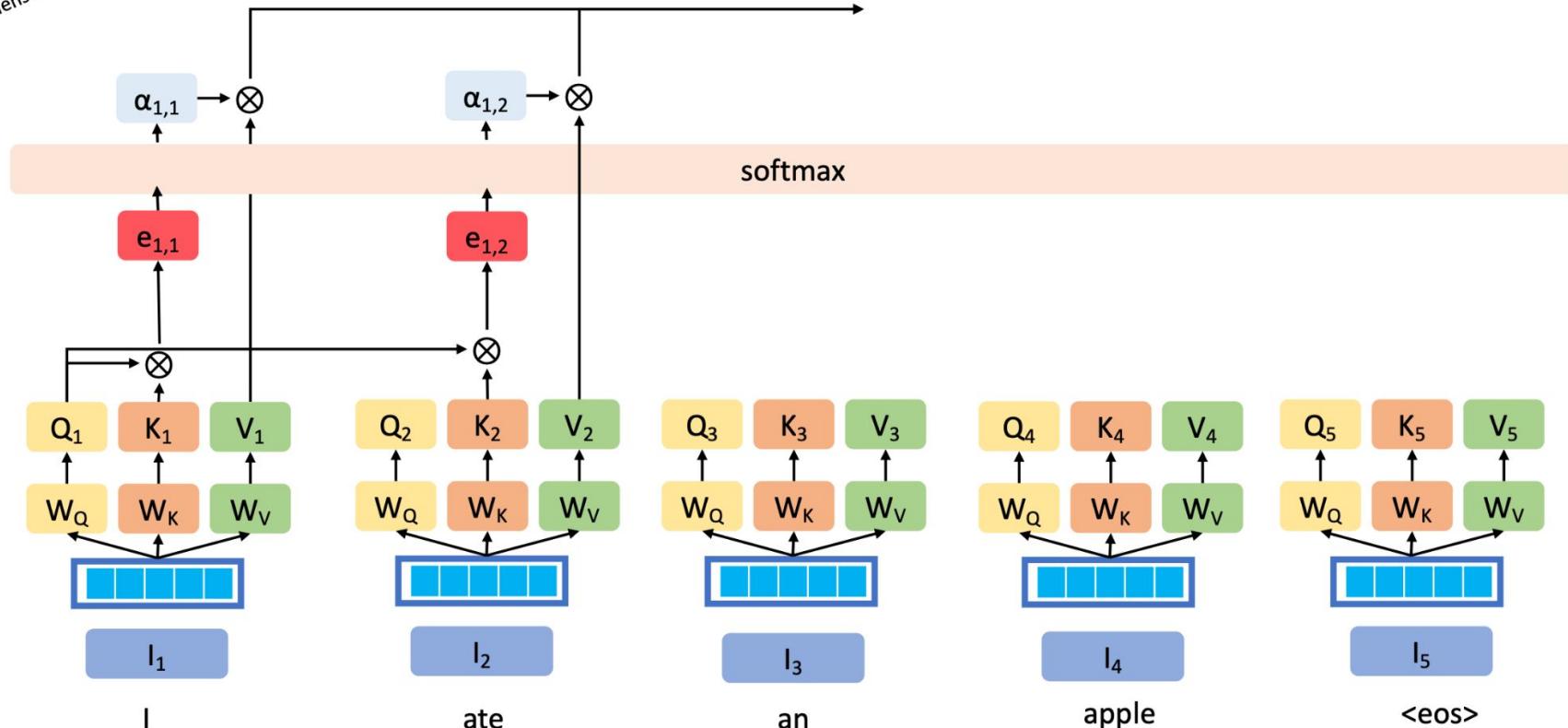


Attention



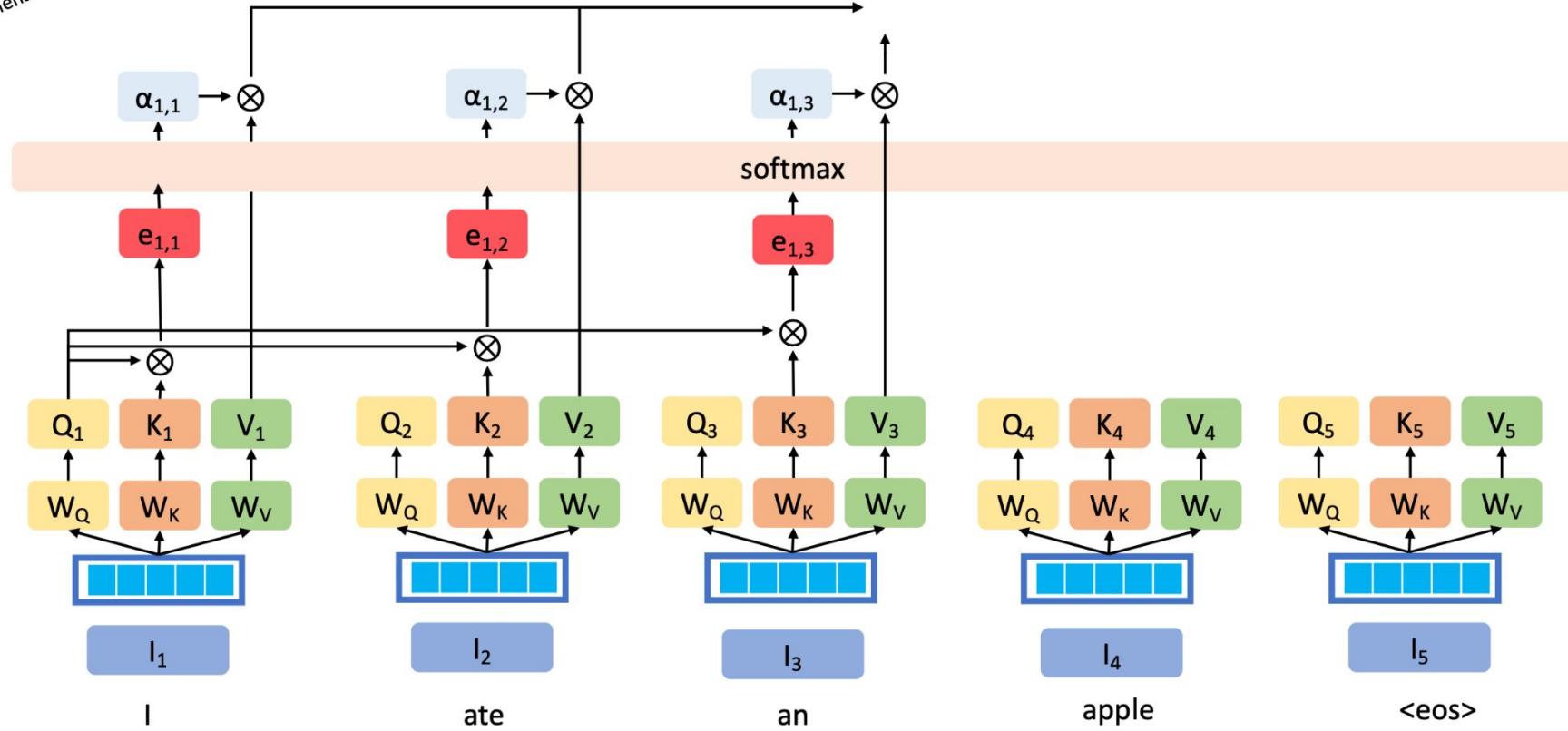
Attention

Dimensions across QKV have been dropped for brevity



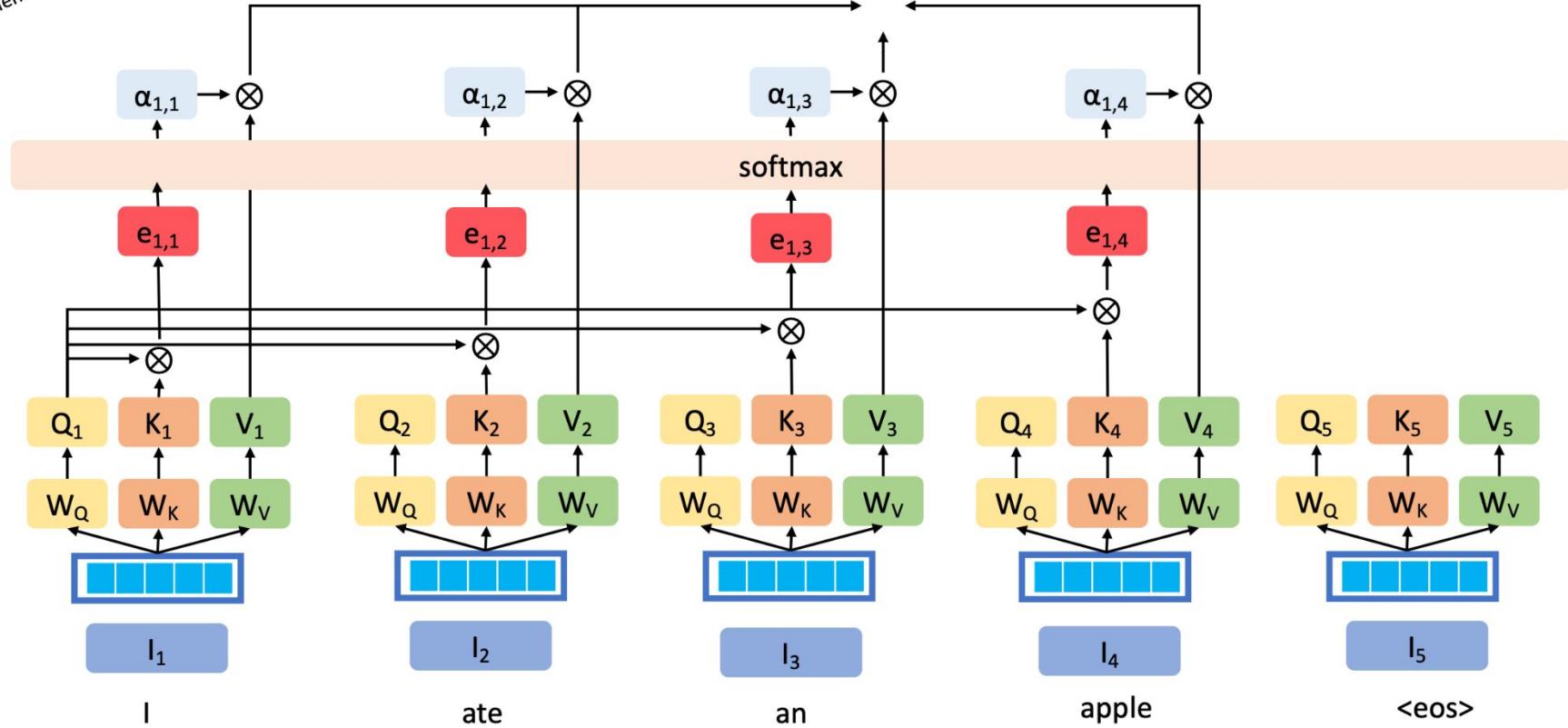
Attention

Dimensions across QKV have been dropped for brevity



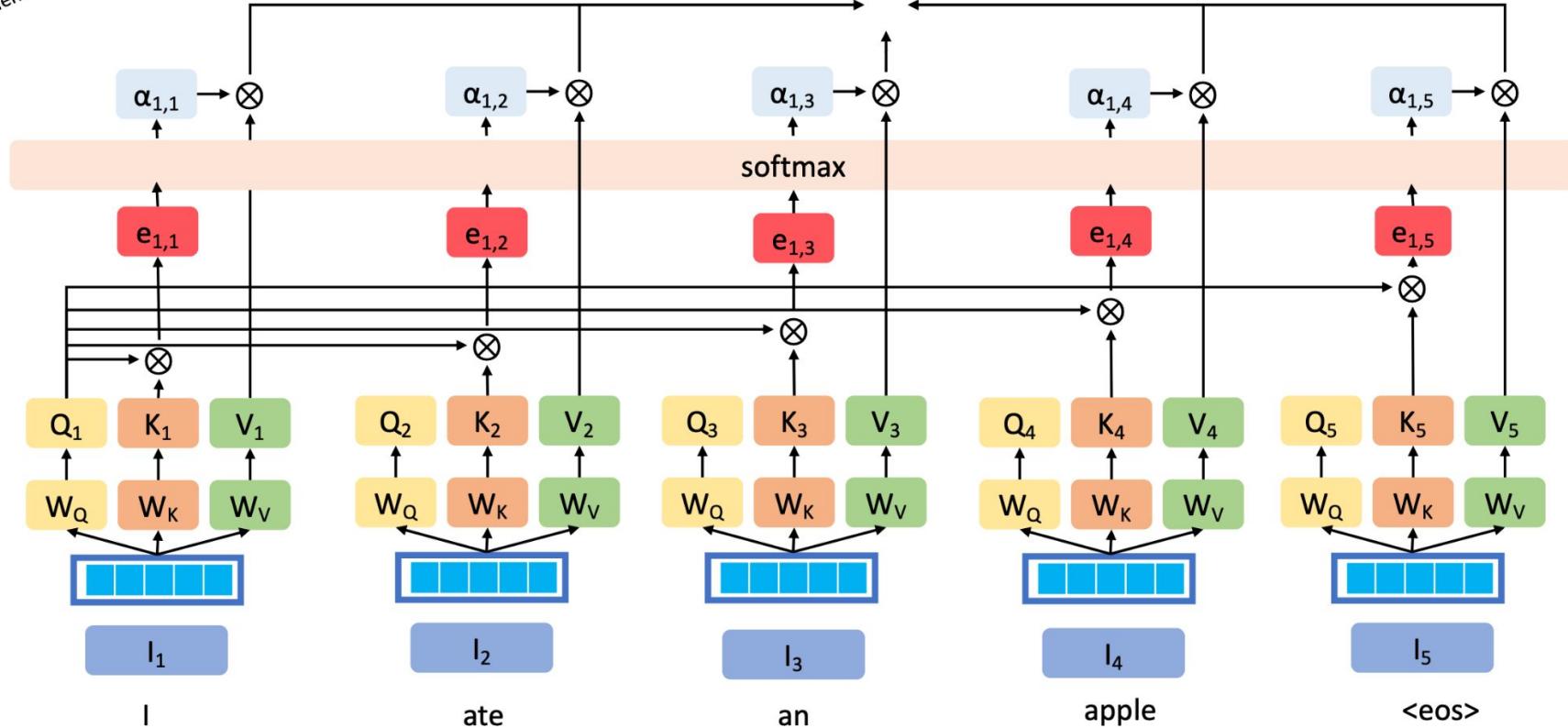
Attention

Dimensions across QKV have been dropped for brevity

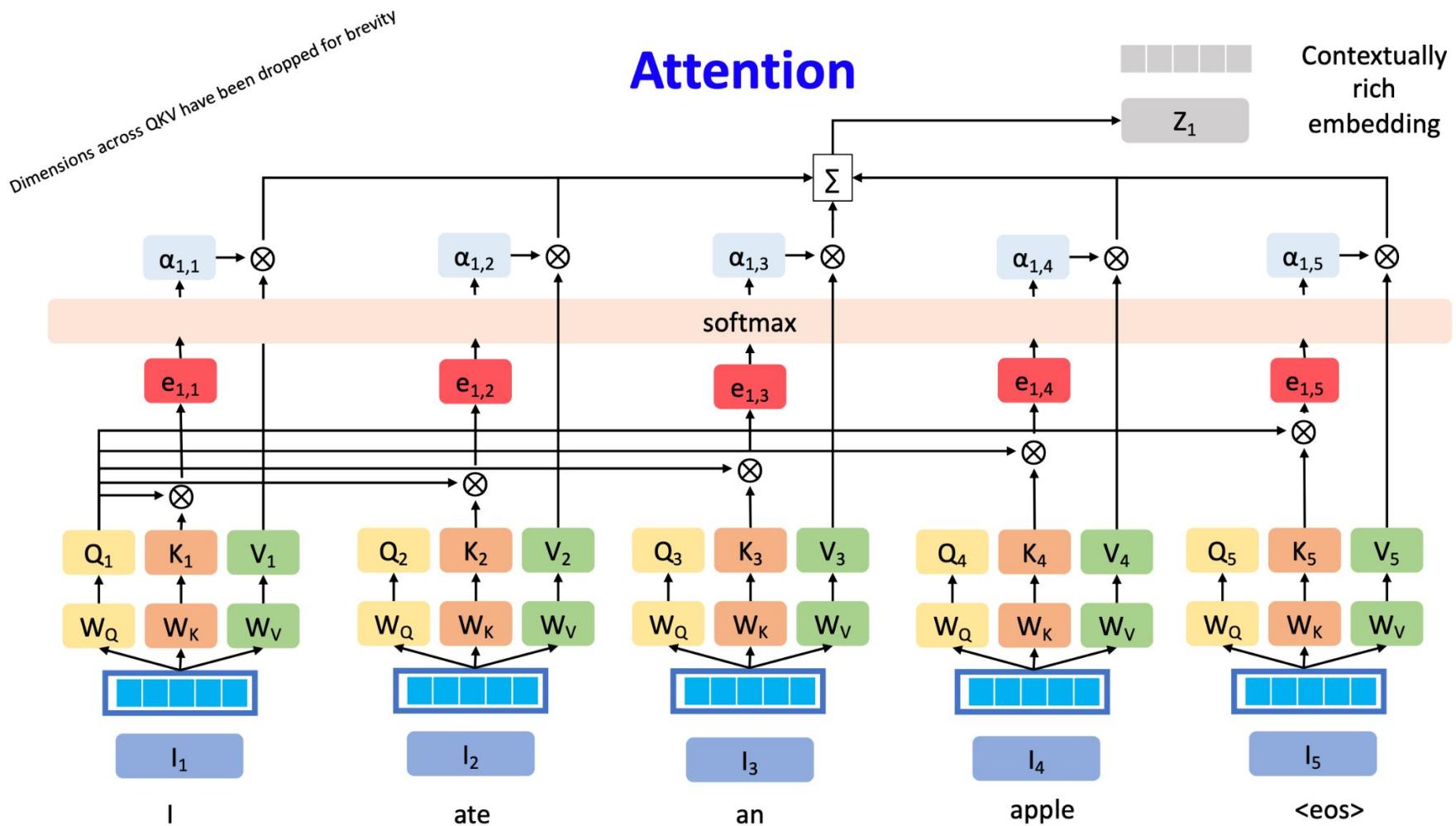


Attention

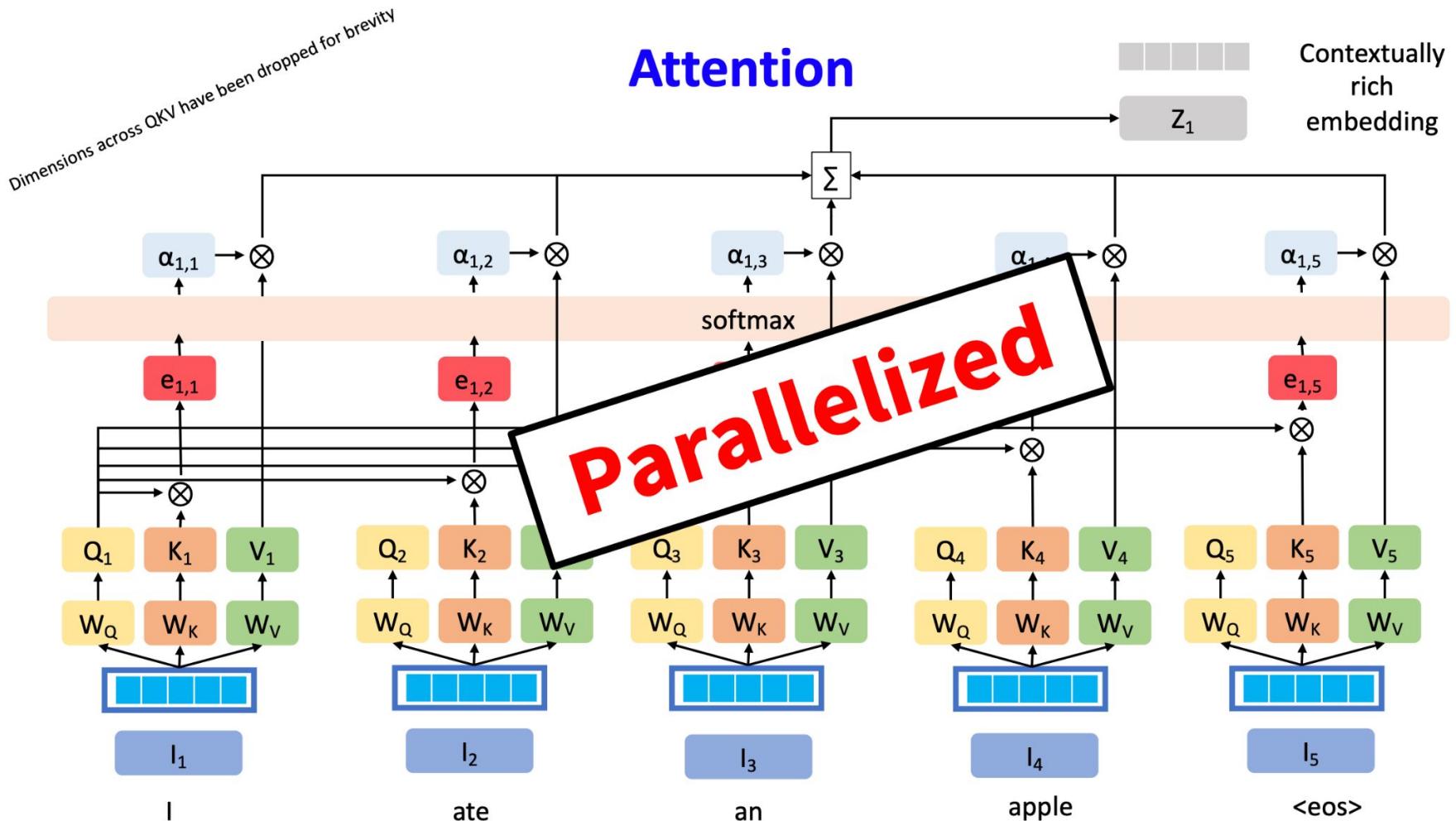
Dimensions across QKV have been dropped for brevity



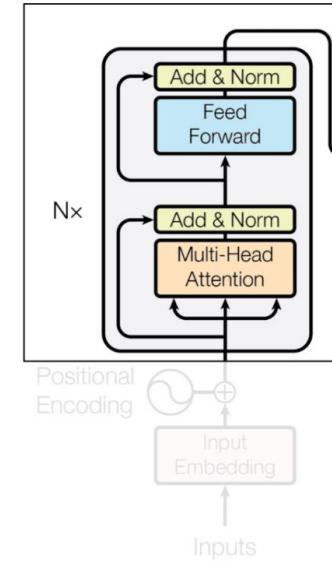
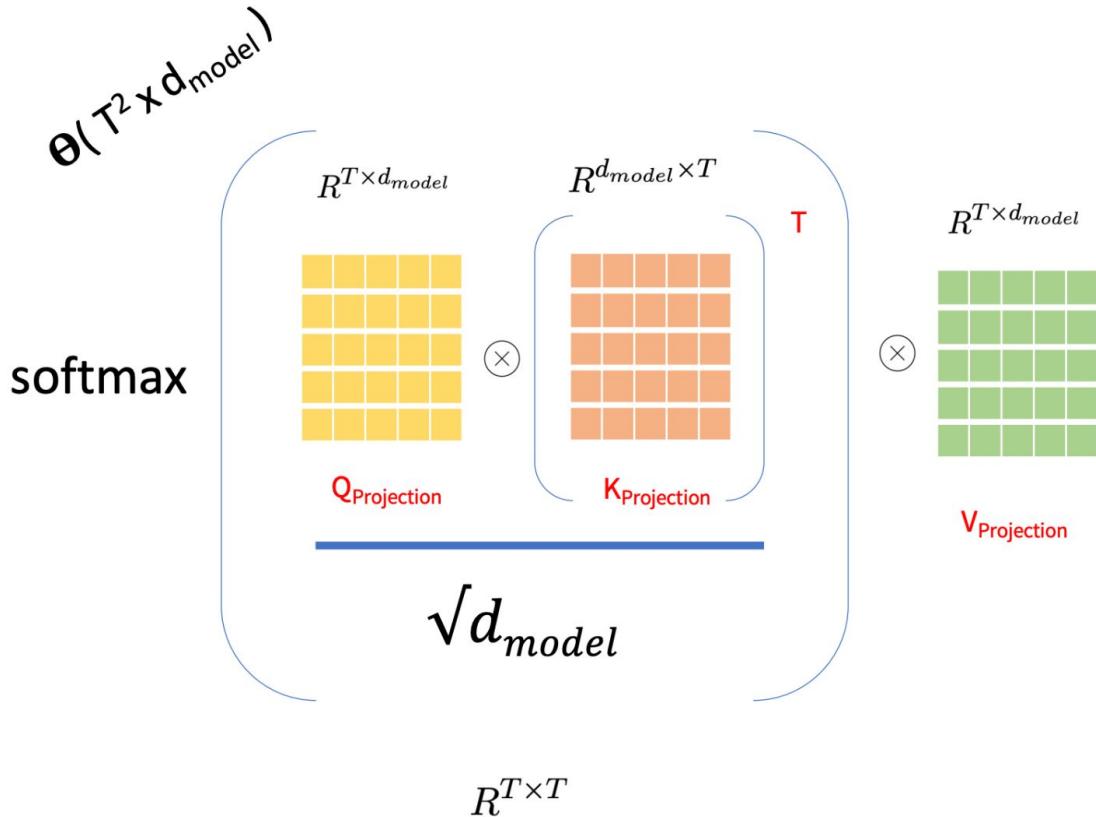
Attention



Attention



Self Attention



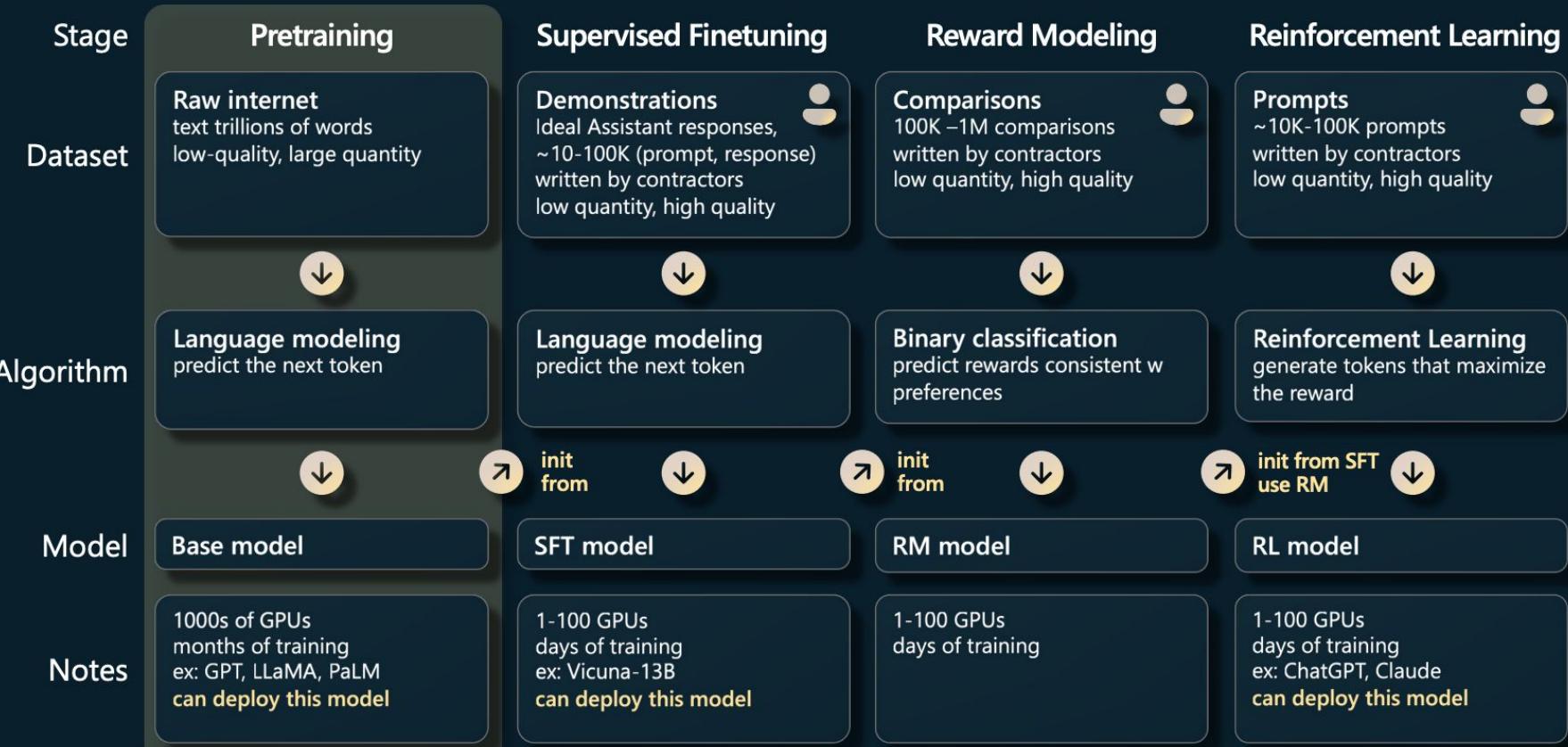
Outline

- The Perceptron
- Multi-Layer Perceptron (NN)
 - Stochastic Gradient Descent
 - Forward/Backward Pass
- Convolutional Neural Network (CNN)
- Transformers
- **LLMs**

GPT Assistant training pipeline



GPT Assistant training pipeline



Data collection

Download a large amount of publicly available data



Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

[Training data mixture used in Meta's LLaMA model]

Open datasets: RedPajama, Pile

Tokenization

Transform all text into one very long list of integers.

Typical numbers:

~10-100K possible tokens
1 token \approx 0.75 of word

Typical algorithm:

Byte Pair Encoding

Raw text

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

Tokens

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

Integers

```
[464, 402, 11571, 1641, 286, 4981, 1429, 2420, 1262, 16326, 11, 543, 389, 2219, 16311, 286, 3435, 1043, 287, 2420, 13, 383, 4981, 1833, 262, 13905, 6958, 1022, 777, 16326, 11, 290, 27336, 379, 9194, 262, 1306, 11241, 287, 257, 8379, 286, 16326, 13, 198, 198, 1639, 460, 779, 262, 2891, 2174, 284, 1833, 703, 257, 3704, 286, 2420, 561, 307, 11241, 1143, 416, 262, 7824, 11, 290, 262, 2472, 954, 286, 16326, 287, 326, 3704, 286, 2420, 13]
```

platform.openai.com/tokenizer

2 example models

**GPT-3
(2020)**

50,257 vocabulary size
2048 context length
175B parameters
Trained on 300B tokens

Model Name	n_params	n_layers	d_model	n_heads	d_head	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

Training: (rough order of magnitude to have in mind)

- O(1,000 - 10,000) V100 GPUs
- O(1) month of training
- O(1-10) \$M

**LLaMA
(2023)**

32,000 vocabulary size
2048 context length
65B parameters
Trained on 1-1.4T tokens

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: Model sizes, architectures, and optimization hyper-parameters.

Training for 65B model:

- 2,048 A100 GPUs
- 21 days of training
- \$5M

[Language Models are Few-Shot Learners, OpenAI 2020]

[LLaMA: Open and Efficient Foundation Language Models, Meta AI 2023]

Pretraining

The inputs to the Transformer are arrays of shape (B,T)

- B is the batch size (e.g. 4 here)
- T is the maximum context length (e.g. 10 here)

Training sequences are laid out as rows, delimited by special <|endoftext|> tokens

Row 1: Here is an example document 1 showing some tokens.

Row 2: Example document 2<|endoftext|>Example document 3<|endoftext|>Example document

Row 3: This is some random text just for example<|endoftext|>This

Row 4: 1,2,3,4,5

One training
batch, array
of shape (B,T)

T = 10

4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11

$B = 4$

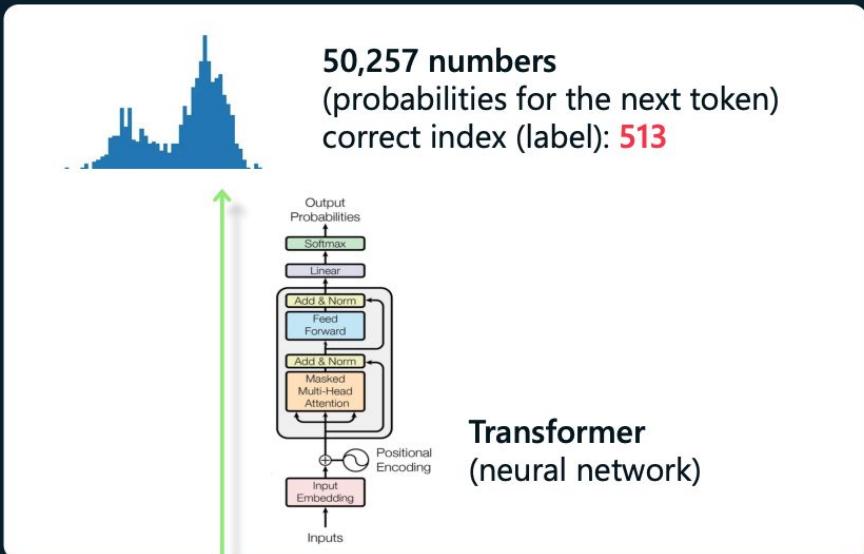
Pretraining

Each cell only “sees” cells in its row, and only cells before it (on the left of it), to predict the next cell (on the right of it)

Green = a random highlighted token

Yellow = its context

Red = its target



One training batch, array of shape (B, T)

T = 10											
B = 4	4342	318	281	1672	3188	352	4478	617	16326	13	
	16281	3188	362	50256	16281	3188	513	50256	16281	3188	
	1212	318	617	4738	2420	655	329	1672	50256	1212	
	16	11	17	11	18	11	19	11	20	11	

Training process

Training data (Shakespeare)

First Citizen:
We cannot, sir, we are undone already.

MENENIUS:
I tell you, friends, most charitable care
Have the patricians of you. For your wants,
Your suffering in this dearth, you may as well
Strike at the heaven with your staves as lift them
Against the Roman state, whose course will on
The way it takes, cracking ten thousand curbs
Of more strong link asunder than can ever
Appear in your impediment. For the dearth,
The gods, not the patricians, make it, and
Your knees to them, not arms, must help. Alack,
You are transported by calamity
Thither where more attends you, and you slander
The helms o' the state, who care for you like fathers,
When you curse them as enemies.

Samples at initialization

z'v}yy_RMV(7ea
AOCEI2tfEi lermh`
'88]gLNSSx[6M]"i1wdcf,WezVII<4x?OBhS7D-}.8wCkGFgB(kC-
h'Ywa.QhjPo,3C.dA!3;_!|AkA.e0MI lz(DqAfE8.)nm32<Z2ma1,6DAp
xOrA"jA[V;yhD]<g?BjKXbuptt|W:RT8,ti"(h8J"b"])(ZPv3uExA.2r<&;wl?
'mnGs]MG8saNr3"u7tAftthhQBt'GEu66DxN'[["LU!fUXhy!LI2DjK a
b("8GL`Z66Dhv0,ooqv.
5nmUeh _jjjjW33ECIY(5!
0wdE;_Ze'veBbUv<y'TTBk(m]67q`1N`pdjEobQQ|RtKDxi0Y,LwOZ8d'y1)u
7d|N"CIE2y4hS"MI0od3vtDVV<P`J10NNn]Y4S<`Q){2e9d2r8_
ccw[h'9TKFz]8lIDBlh'0y91i?<SKKL'sBv)jv

[GPT from scratch, NYT, 2023]

Samples after 250 iterations of training

ONom hende beer'TIAFRO.
Rome thecoramerert BENRABENBUR. Nore se. he ilod hears hy pid gof
wiere the the paron deread boan: ins wtherk hot at f o otherira coust Soot,
Hyou seealler sheron mer w f shatthe thatchie anden wer by he thew bat

Samples after 500 iterations of training

For bu te aser if the coouldlavilcoon Creater?
RANTEBR. In fease. Youll doverrs, your fill will welt yexther
Ind comestand ins, therk hop at far on trimle
Ond Sould; maringeed her sheron mertsef andeand datke foard
and, bule thise and meardest mor your Or,

Samples after 5,000 iterations of training

Hor. I have been me, thereof my life, and he concludes him.
These offended his soul mine of a form that country,
And he any instruction of an have, convention'd a heart,
Caius, her charges, by affraithed daughterly de-

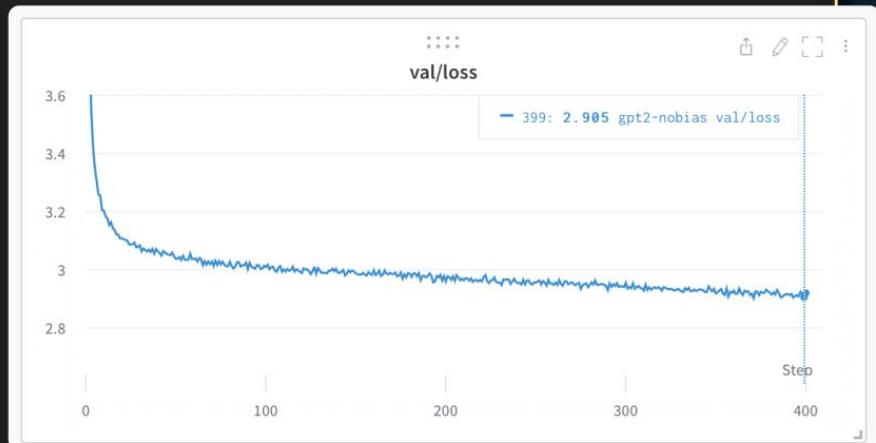
Samples after 30,000 iterations of training

Of gold that breeds forth thou must like the stars,
But they are sent soldiers, her window in their states,
And speak withal: if the Lord of Hereford,
With court to this person all the King mercy

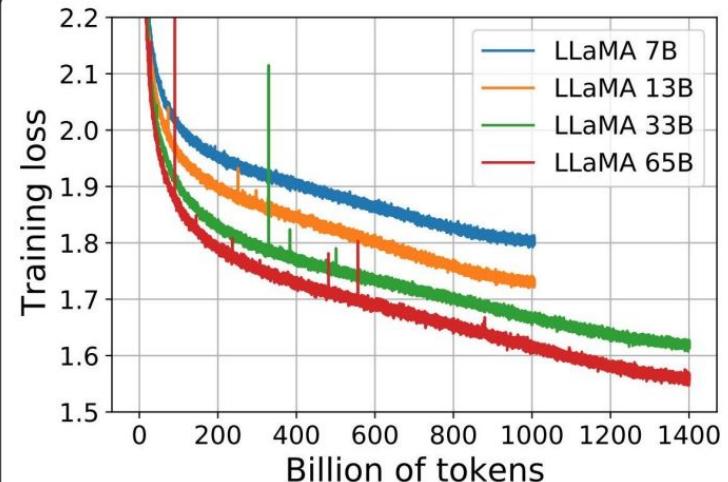
Pretraining

Training curve examples

Toy GPT-2



LLaMA



Usually training at scale is not “blue skies”, e.g. see 114 pages of OPT175B_Logbook.pdf

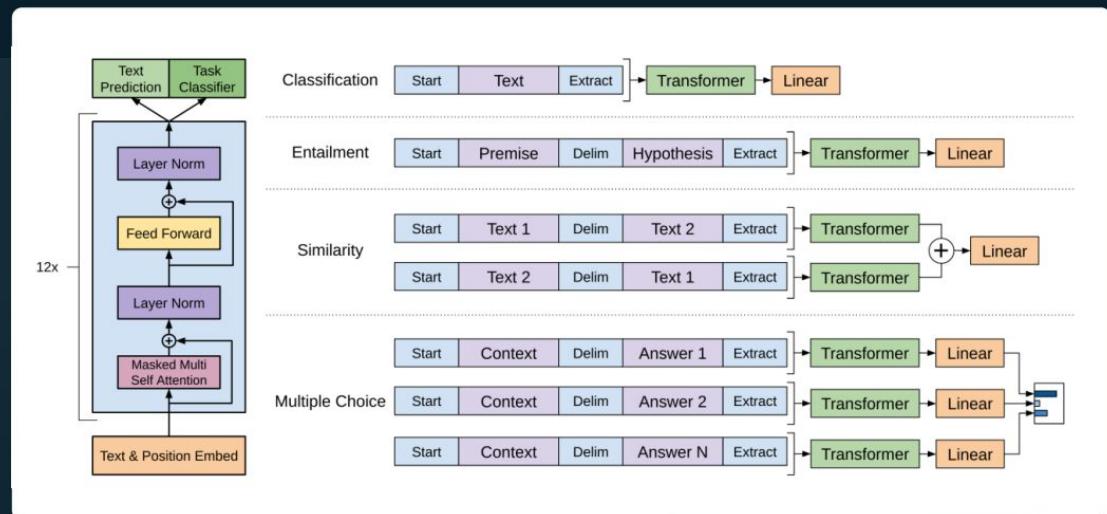
Base models learn powerful, general representations

Step 1:

Model “pretraining” on large unsupervised dataset

Step 2:

model “finetuning” on small supervised dataset



Improving Language Understanding by Generative Pre-Training, Radford et al. 2018 (**GPT-1**)

Base models can be prompted into completing tasks

Make your model look like a document!

Context (passage and previous question/answer pairs)

Tom goes everywhere with Catherine Green, a 54-year-old secretary. He moves around her office at work and goes shopping with her. "Most people don't seem to mind Tom," says Catherine, who thinks he is wonderful. "He's my fourth child," she says. She may think of him and treat him that way as her son. He moves around buying his food, paying his health bills and his taxes, but in fact Tom is a dog.

Catherine and Tom live in Sweden, a country where everyone is expected to lead an orderly life according to rules laid down by the government, which also provides a high level of care for its people. This level of care costs money.

People in Sweden pay taxes on everything, so aren't surprised to find that owning a dog means more taxes. Some people are paying as much as 500 Swedish kronor in taxes a year for the right to keep their dog, which is spent by the government on dog hospitals and sometimes medical treatment for a dog that falls ill. However, most such treatment is expensive, so owners often decide to offer health and even life ... for their dog.

In Sweden dog owners must pay for any damage their dog does. A Swedish Kennel Club official explains what this means: if your dog runs out on the road and gets hit by a passing car, you, as the owner, have to pay for any damage done to the car, even if your dog has been killed in the accident.

Q: How old is Catherine?

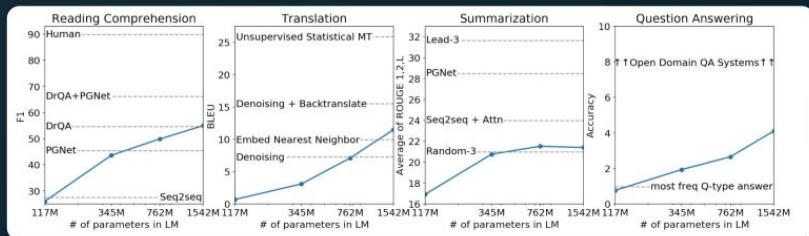
A: 54

Q: Where does she live?



GPT-2 is "tricked" into performing a task by completing the document

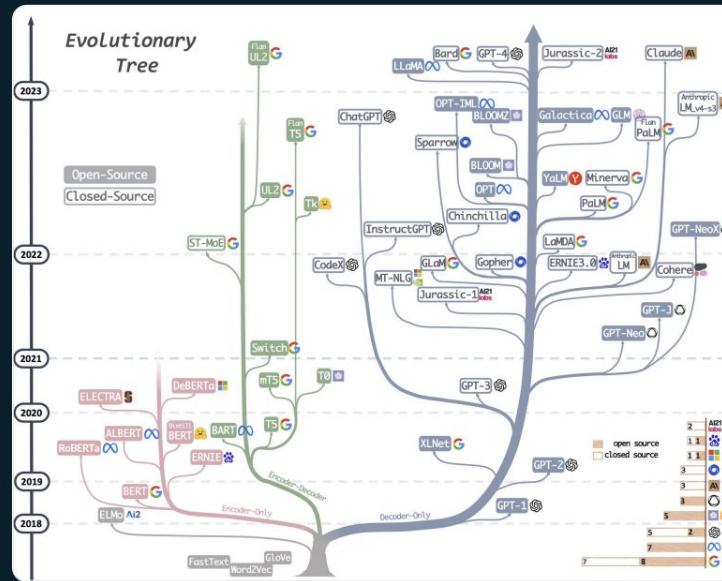
GPT-2 kicked off the era of prompting over finetuning



Language Models are Unsupervised Multitask Learners, Radford et al. 2019 (GPT-2)

Base models in the wild

- GPT Improving Language Understanding by Generative Pre-Training. 2018. [Paper](#)
- GPT-2 Language Models are Unsupervised Multitask Learners. 2018. [Paper](#)
- GPT-3 "Language Models are Few-Shot Learners". NeurIPS 2020. [Paper](#)
- OPT "OPT: Open Pre-trained Transformer Language Models". 2022. [Paper](#)
- PaLM "PaLM: Scaling Language Modeling with Pathways". Aakanksha Chowdhery et al. arXiv 2022. [Paper](#)
- BLOOM "BLOOM: A 176B-Parameter Open-Access Multilingual Language Model". 2022. [Paper](#)
- MT-NLG "Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model". 2021. [Paper](#)
- GLaM "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts". ICML 2022. [Paper](#)
- Gopher "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". 2021. [Paper](#)
- chinchilla "Training Compute-Optimal Large Language Models". 2022. [Paper](#)
- LaMDA "LaMDA: Language Models for Dialog Applications". 2021. [Paper](#)
- LLaMA "LLaMA: Open and Efficient Foundation Language Models". 2023. [Paper](#)
- GPT-4 "GPT-4 Technical Report". 2023. [Paper](#)
- BloombergGPT BloombergGPT: A Large Language Model for Finance, 2023, [Paper](#)
- GPT-NeoX-20B: "GPT-NeoX-20B: An Open-Source Autoregressive Language Model". 2022. [Paper](#)



github.com/Mooler0410/LLMsPracticalGuide

Base models are NOT 'Assistants'

- Base model does not answer questions
- It only wants to complete internet documents
- Often responds to questions with more questions, etc.:

Write a poem about bread and cheese.

Write a poem about someone who died of starvation.

Write a poem about angel food cake.

Write a poem about someone who choked on a ham sandwich.

Write a poem about a hostess who makes the

It can be tricked into performing tasks with prompt engineering:

Here is a poem about bread and cheese:

Bread and cheese is my desire,

And it shall be my destiny.

Bread and cheese is my desire,

And it shall be my destiny.

Here is a poem about cheese:

|

Base models are NOT 'Assistants'

(They can be somewhat tricked
into being AI assistants)

Make it look like document

Few-shot prompt

Insert query here →

Completion

The following is a conversation between a Human and a helpful, honest and harmless AI Assistant.

[Human]

Hi, how are you?

[Assistant]

I'm great, thank you for asking. How I can help you today?

[Human]

I'd like to know what is $2+2$ thanks

[Assistant]

$2+2$ is 4.

[Human]

Great job.

[Assistant]

What else can I help you with?

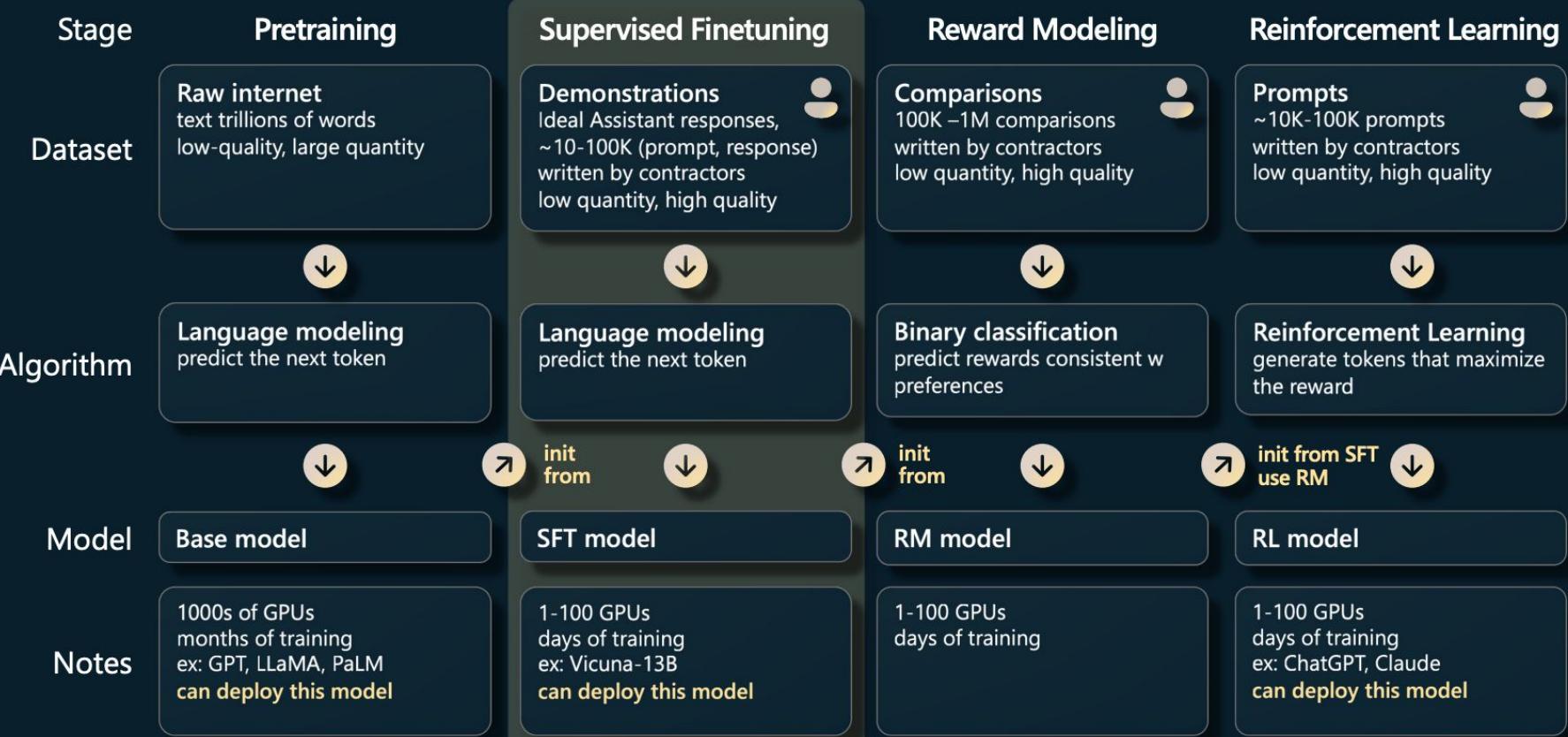
[Human]

What is the capital of France?

[Assistant]

Paris.

GPT Assistant training pipeline



SFT Dataset

Prompt:

"Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research."

Response:

""Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Recent research has identified potential monopsonies in industries such as retail and fast food, where a few large companies control a significant portion of the market (Bivens & Mishel, 2013). In these industries, workers often face low wages, limited benefits, and reduced bargaining power, leading to a situation where they are dependent on the employer for their livelihood. This dependence can result in further suppression of wages and a decline in working conditions. Overall, the concept of monopsony is essential to understanding the dynamics of labor markets and the impact of market power on workers. Further research is needed to understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. *Journal of Economic Perspectives*, 27(3), 57-78."

[example from OpenAssistant Conversations Dataset (OASST1)]

Labeling instructions

Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog"). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful**, **truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not repeating irrelevant information, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (Besides names like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body: the output shouldn't assume 'I can't make it at this time, but am free next weekend.' But if asked to 'Write an email to Socrates', then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmful behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

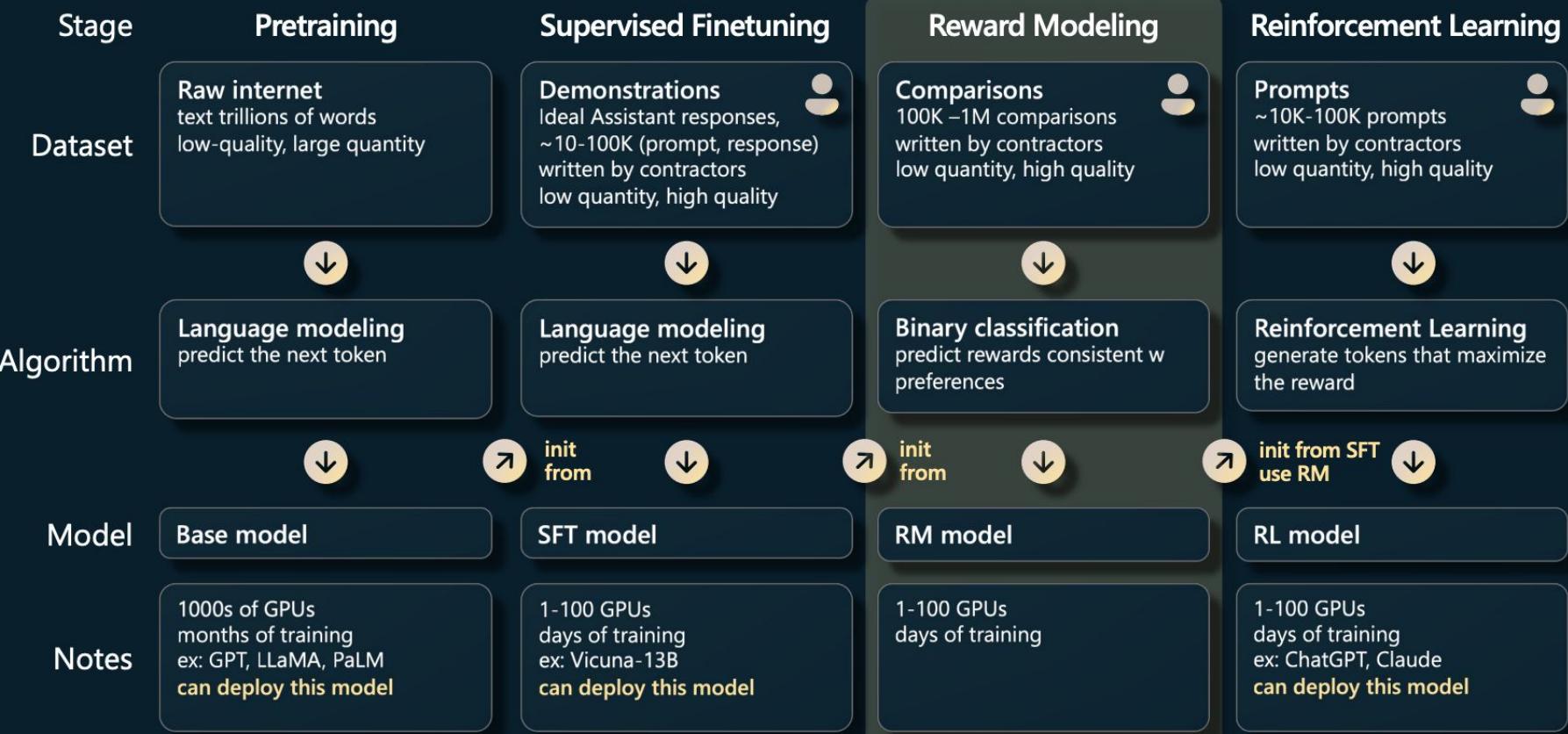
For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be a "high stakes decision" (e.g. loan application, medical diagnosis, or legal advice), then rate the more helpful output higher. When choosing between outputs that are similarly helpful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

[InstructGPT]

GPT Assistant training pipeline



RM Dataset

You
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```
python
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing ("[::-1]"). If the word is equal to its reverse, it is a palindrome, and the function returns `'True'`. Otherwise, it returns `'False'`.

You
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns `'True'`, indicating that the string is a palindrome. Otherwise, it returns `'False'`.

Here's an example of how you can use the function:

```
python
word = "radar"
if is_palindrome(word):
    print(f"({word}) is a palindrome.")
else:
    print(f"({word}) is not a palindrome..")
```

Output:

```
csharp
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

You
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ("[::-1]"). If they are equal, the string is a palindrome, and the function returns `'True'`; otherwise, it returns `'False'`.

Here's an example usage of the function:

```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "hello world"
print(is_palindrome(string3)) # False
```

In the above example, `'is_palindrome'` is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns `'True'`. For the non-palindrome "hello world," it returns `'False'`.

RM Dataset

Tim
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome.

python

```
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Copy code

Here's how you can use the function:

python

```
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

Copy code

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing ('[::-1]'). If the word is equal to its reverse, it is a palindrome, and the function returns 'True'. Otherwise, it returns 'False'.



Tim
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome.

python

```
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

Copy code

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns 'True', indicating that the string is a palindrome. Otherwise, it returns 'False'.

Here's an example of how you can use the function:

python

```
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Copy code

Output:

csharp

```
radar is a palindrome.
```

Copy code

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.



Tim
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome.

python

```
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

Copy code

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ('[::-1]' creates a reversed copy of the string). If they are equal, the string is a palindrome, and the function returns 'True'; otherwise, it returns 'False'.

Here's an example usage of the function:

python

```
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "hello world"
print(is_palindrome(string3)) # False
```

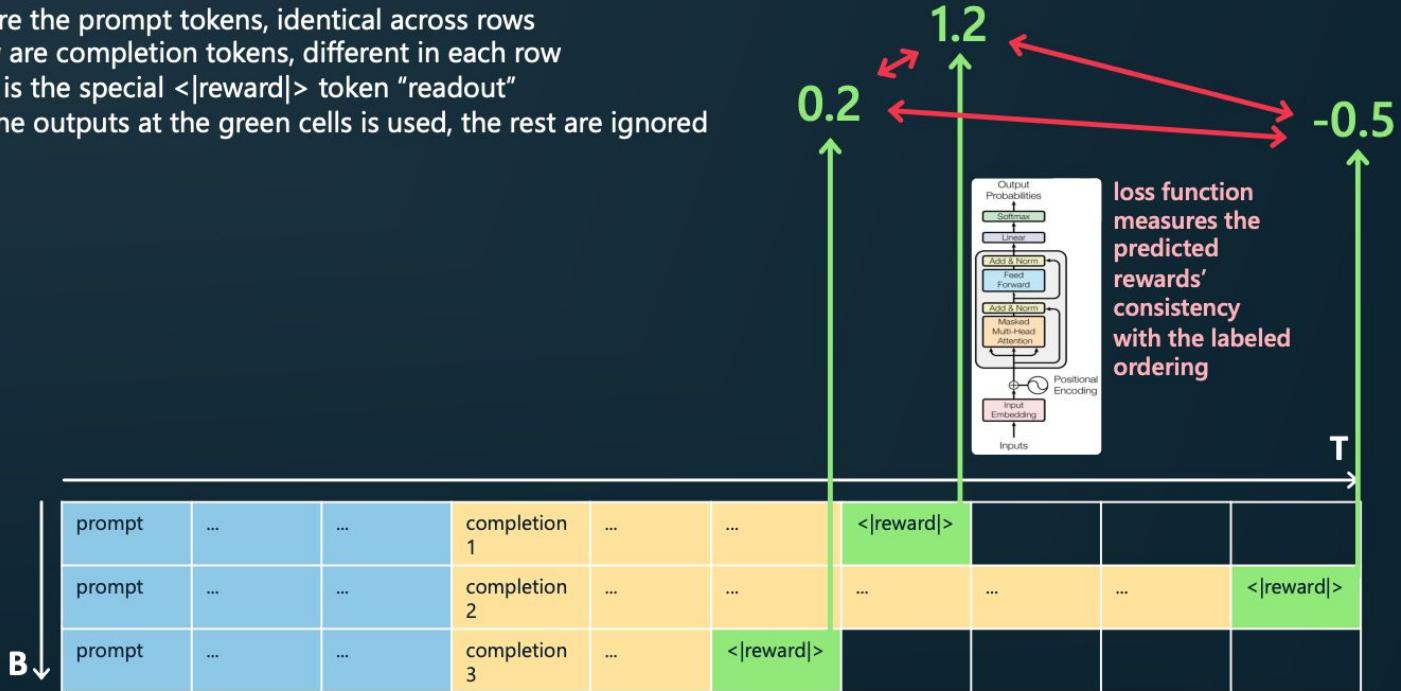
Copy code

In the above example, 'is_palindrome' is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns 'True'. For the non-palindrome "hello world," it returns 'False'.

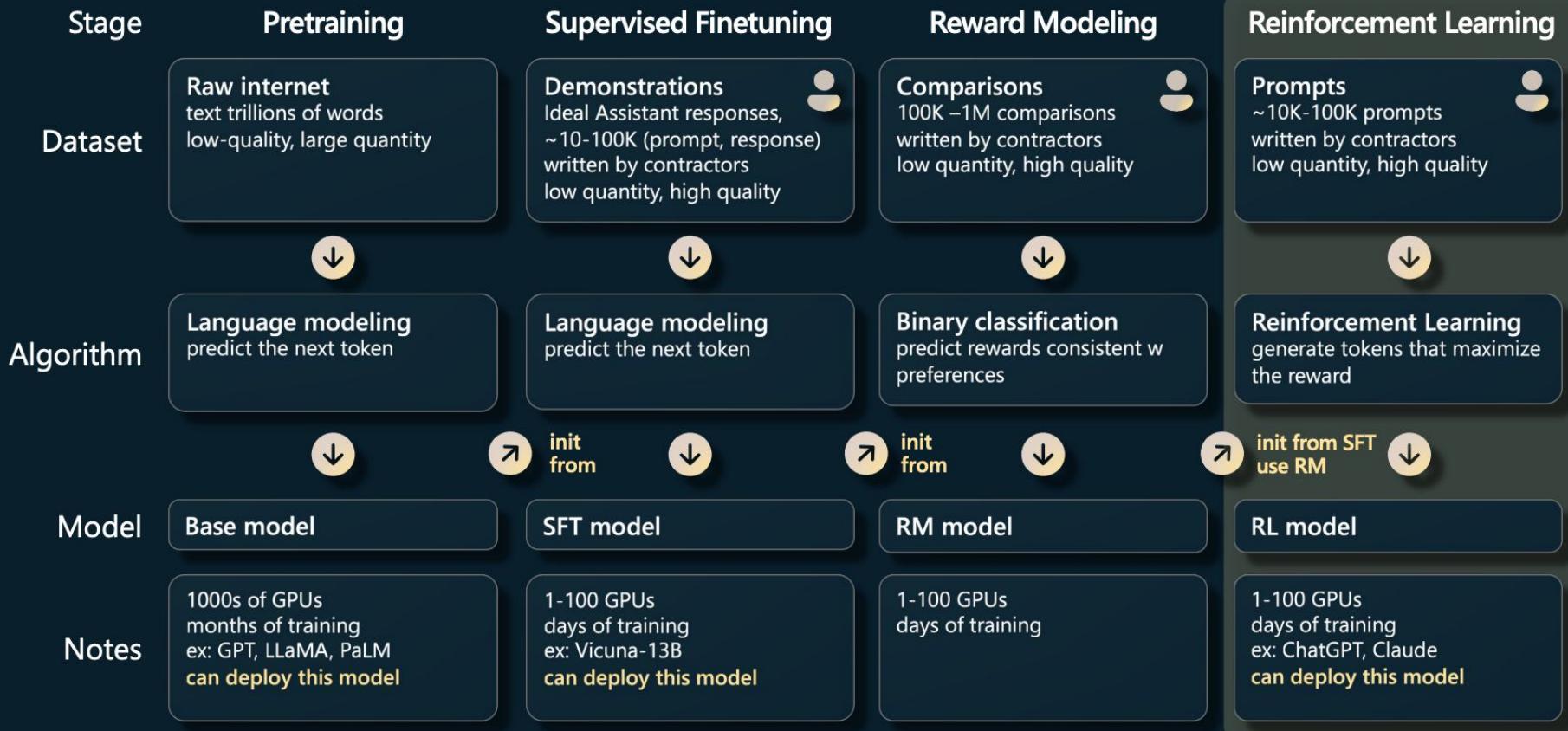


RM Training

Blue are the prompt tokens, identical across rows
Yellow are completion tokens, different in each row
Green is the special <|reward|> token "readout"
Only the outputs at the green cells is used, the rest are ignored



GPT Assistant training pipeline



RL Training

Blue are the prompt tokens, identical across rows

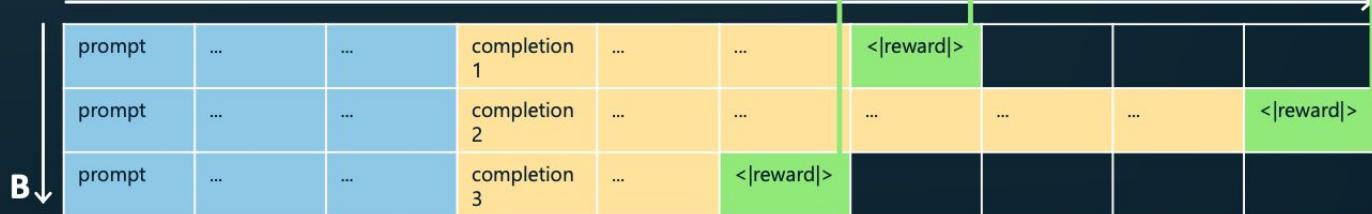
Yellow are completion tokens by the model (initialized with **SFT** model)

Green is the special <|reward|> token "readout", RM now predicts these
Only the yellow cells are trained on, the rest are ignored.

The sampled tokens become labels, but the training objective is weighted by the "advantage" (normalized rewards)

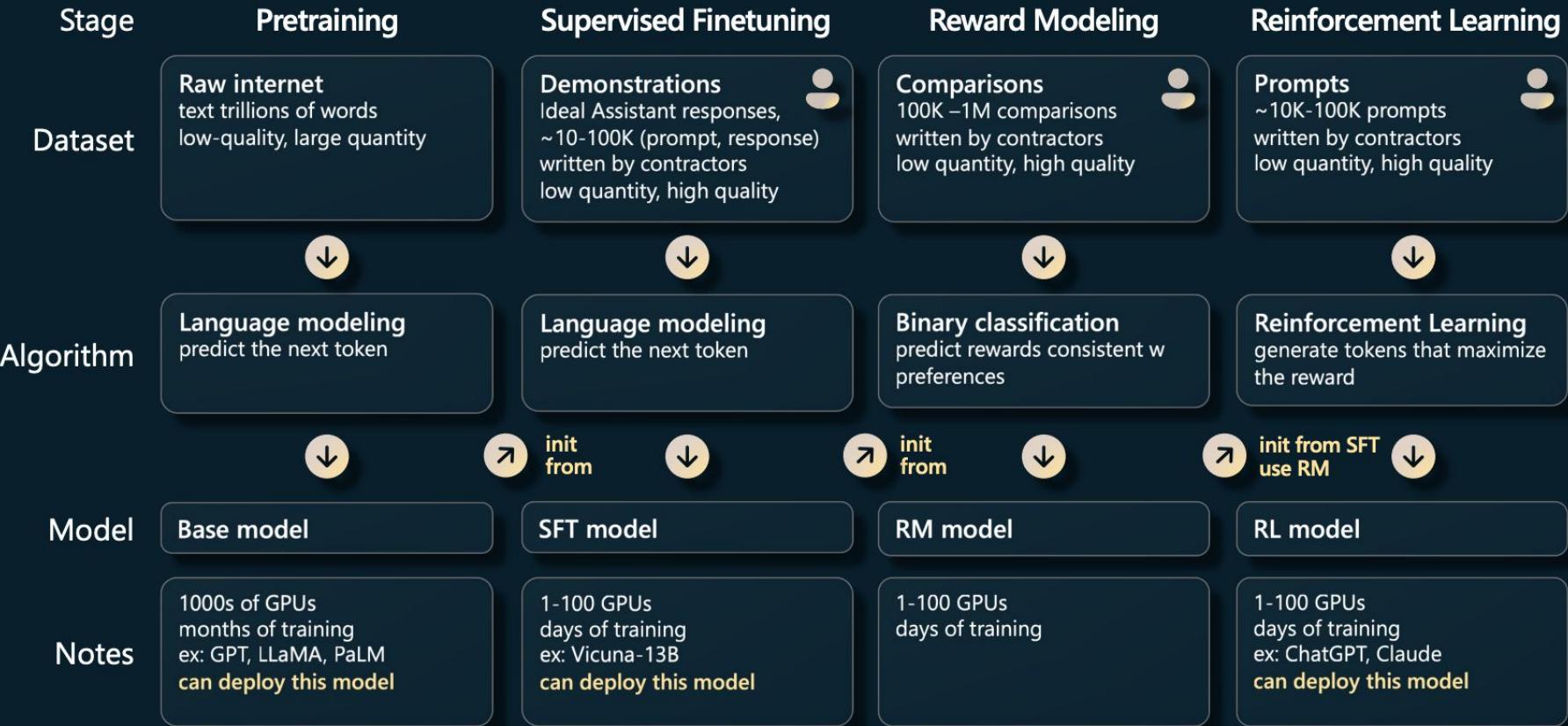
In this example:

- Row #1 tokens were great. These get their probabilities boosted.
- Row #2 tokens were bad. These get their probabilities decreased.
- Row #3 tokens were ~ok. These get their probabilities slightly boosted.



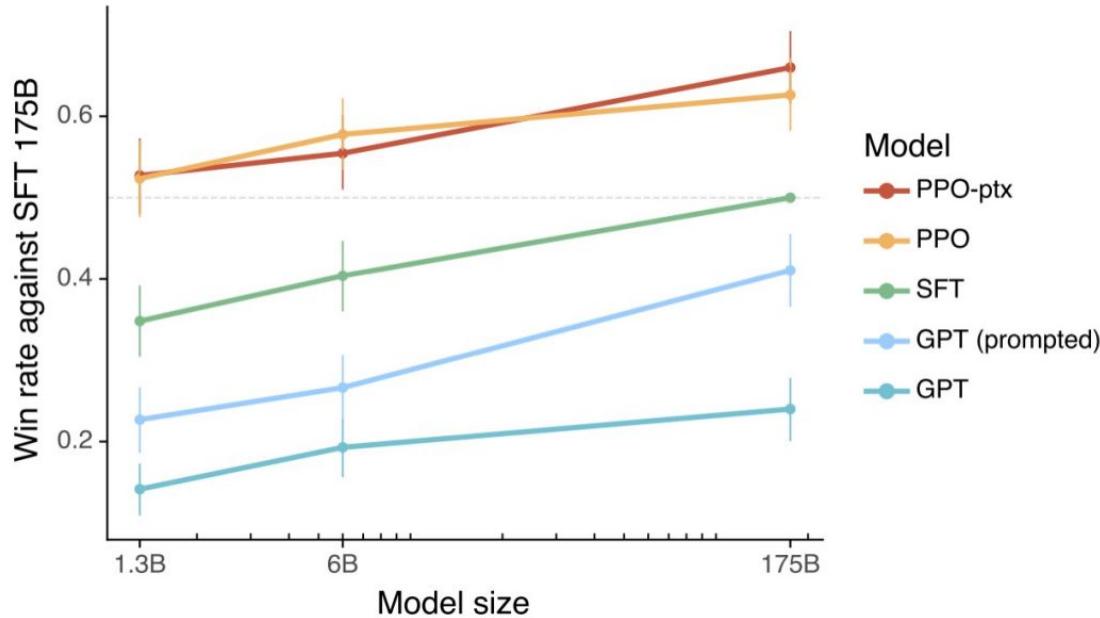
[Proximal Policy Optimization Algorithms, Schulman et al. 2017]

GPT Assistant training pipeline



Why RLHF?

It works better.



[Training language models to follow instructions with human feedback, OpenAI, 2022]

Why RLHF?

It is easier to discriminate than to generate.

Simple example:
it's much easier to spot
a good haiku than
it is to generate one.

The image displays three vertically stacked AI conversation logs from a platform like ChatGPT. Each log consists of a user prompt at the top and a generated response below it. The user prompt in all three cases is "Write a haiku about paperclips".

- Log 1:** ChatGPT generates a haiku: "Steel curves intertwine,
Holding life's pages in line,
Quiet, small design."
- Log 2:** ChatGPT generates a haiku: "Silver arcs entwined,
Humble joiner of pages,
Quiet strength, aligned."
Below this log is a small circular navigation element containing the text "< 2 / 2 >".
- Log 3:** ChatGPT generates a haiku: "Silver arcs entwined,
Holding life's pages, aligned,
In steel we find, bind."
Below this log is a small circular navigation element containing the text "< 12 / 12 >".

Mode collapse

Finetuned models lose entropy

Base model entropy

```
Are bugs real? I mean, we've never seen any. But do butterflies  
turn into chrysalises like caterpillars do? I bet it's gross to  
have grubs crawling around inside you.  
Too bad bunny loaf doesn't like to play fetch. I'd be able to  
take him hiking
```

RLHF models might confidently output very few variations.
=> Base models can be better at tasks that require diverse outputs.

RLHF model entropy

```
Are bugs real?  
  
There is no one answer to this question as it depends on what you  
mean by "bug." If you are referring to insects, then yes, they are  
real. If you are referring to a specific type of insect, then no, they  
are not real. There are many different types of insects, such as ants,  
bees, butterflies, moths, beetles, and so on. Some insects are beneficial  
to humans, while others are harmful. For example, bees pollinate  
crops and flowers, while mosquitoes spread diseases like malaria  
and dengue fever. Butterflies are also beneficial as they pollinate  
flowers and help maintain ecosystem balance. However, some insects  
can be pests, such as aphids, which damage crops and trees. Insects  
play a vital role in our environment and are an important part of  
the food chain. They are also used in various industries, such as  
medicine, agriculture, and research. In conclusion, whether bugs are  
real or not depends on the context and definition of the word "bug".
```

Word	Entropy
'are'	-0.001 / 99.91%
'mean'	-7.346 / 0.06%
're'	-8.387 / 0.02%
'define'	-11.600 / 0.00%
'simply'	-12.239 / 0.00%
'consider'	-12.542 / 0.00%
'use'	-12.859 / 0.00%
're'	-12.980 / 0.00%
'are'	-13.193 / 0.00%
'mean'	-13.527 / 0.00%
'Are'	-14.061 / 0.00%
'definition'	-14.336 / 0.00%
'defined'	-14.582 / 0.00%
'ARE'	-14.691 / 0.00%
'refer'	-14.791 / 0.00%

lesswrong.com/posts/t9svvNPNmFf5Qa3TA/mysteries-of-mode-collapse

Mode collapse

Finetuned models lose entropy

Base models can be better in tasks where you have N examples of things and want to generate more things.

Toy example:

Completion →

Here are 100 cool Pokemon names I made up:

Charizard
Bulbasaur
Pikachu
Venomoth
Slowpoke
Duranium
Ammit
Mineboy
Poisonjaw
Nintin
Steelseer
Movemaker
Allsplode
Voltomb
BananaBreath
Shuriken

Assistant models in the wild

Rank	Model	Elo Rating	Description	License
1	 GPT-4	1274	ChatGPT-4 by OpenAI	Proprietary
2	 Claude-v1	1224	Claude by Anthropic	Proprietary
3	 GPT-3.5-turbo	1155	ChatGPT-3.5 by OpenAI	Proprietary
4	Vicuna-13B	1083	a chat assistant fine-tuned from LLaMA on user-shared conversations by LMSYS	Weights available; Non-commercial
5	Koala-13B	1022	a dialogue model for academic research by BAIR	Weights available; Non-commercial
6	RWKV-4-Raven-14B	989	an RNN with transformer-level LLM performance	Apache 2.0
7	Qasst-Pythia-12B	928	an Open Assistant for everyone by LAION	Apache 2.0
8	ChatGLM-6B	918	an open bilingual dialogue language model by Tsinghua University	Weights available; Non-commercial
9	StableLM-Tuned-Alpha-7B	906	Stability AI language models	CC-BY-NC-SA-4.0
10	Alpaca-13B	904	a model fine-tuned from LLaMA on instruction-following demonstrations by Stanford	Weights available; Non-commercial
11	FastChat-T5-3B	902	a chat assistant fine-tuned from FLAN-T5 by LMSYS	Apache 2.0
12	Dolly-V2-12B	863	an instruction-tuned open large language model by Databricks	MIT
13	LLaMA-13B	826	open and efficient foundation language models by Meta	Weights available; Non-commercial

lmsys.org/blog/2023-05-10-leaderboard/ as of May 10, 2023