



Help



Personal



Playground

Your presets

Save

View code

Share



The following is a classic Chinese poem from the Tang dynasty, translated into English.



The dawn light strikes the head of my bed
I see leaves of green in the morning
The singing birds soar through the sky
Where is my home, beyond the horizon?

Submit



495

GPT is a model for sequences.

- ❖ It sees text as a sequence of tokens $\underline{x} = x_0 x_1 x_2 \cdots x_N$
- ❖ Its training dataset is a collection of sequences $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(n)}\}$

The following is a classic Chinese poem from the Tang dynasty, translated into English.

The dawn light strikes the head of my bed
I see leaves

TEXT TOKEN IDS

[464, 1708, 318, 257, 6833, 3999, 21247, 422, 262, 18816, 30968, 11, 14251, 656, 3594, 13, 198, 198, 464, 17577, 1657, 8956, 262, 1182, 286, 616, 3996, 198, 40, 766, 5667, 220]

TEXT TOKEN IDS

GPT is a probability model for sequences of tokens

- ❖ Let $\underline{X} = X_0X_1X_2 \cdots X_N$ be a random sequence of tokens, of random length N
- ❖ GPT has been trained to fit a probability model for \underline{X} to its training dataset $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(n)}\}$
- ❖ Once we have a trained probability model, we can use it for completion. We give it an input prompt $\underline{x} = x_0x_1 \cdots x_m$ and it generates a sample of

$$(\underline{X} \mid x_0x_1 \cdots x_m)$$

SECTION 13

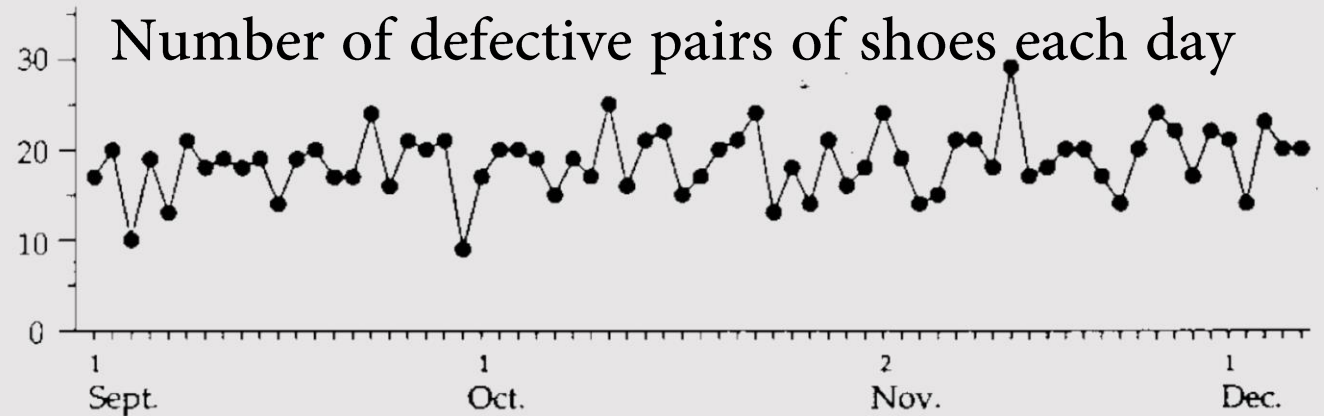
How can we learn a
probability model
for sequences?

Example 13.1.1: fitting a Markov model

Let (x_0, x_1, \dots, x_n) be a time series which we believe is generated by

$$X_{i+1} = a + b X_i + N(0, \sigma^2).$$

Estimate a , b , and σ using maximum likelihood estimation.



What's the likelihood of a sequence $\underline{x} = x_0 x_1 x_2 \dots x_n$?

$$\Pr(x_0 x_1 x_2 \dots x_n) = \Pr(x_0) \Pr(x_1 | x_0) \Pr(x_2 | x_1) \times \dots \times \Pr(x_n | x_{n-1})$$

$$= \Pr(x_0) \prod_{i=1}^n \Pr(x_i | x_{i-1})$$

The question doesn't tell us anything about the model for X_0 . So all we can do is leave this term un-fitted.

$$= \dots \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-[x_i - (a + bx_{i-1})]^2 / 2\sigma^2}$$

The question tells us

$$X_{i+1} \sim a + bX_i + N(0, \sigma^2)$$

therefore

$$X_i \sim N(a + bX_{i-1}, \sigma^2)$$

and so the likelihood is

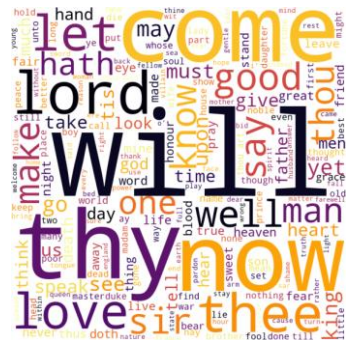
$$\Pr_{X_i}(x_i | X_{i-1} = x_{i-1}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-[x_i - (a + bx_{i-1})]^2 / 2\sigma^2}$$

To fit this model (i.e. to learn a, b, σ) we simply maximize this likelihood, as usual.

We've seen this sort of learning before! It's just supervised learning, and in particular it's a simple linear model $x_{i+1} \approx a + bx_i$. We can fit it with sklearn, using the feature vectors $[1, 1, \dots, 1]$ and $[x_0, x_1, \dots, x_{n-1}]$ and response vector $[x_1, x_2, \dots, x_n]$.

(This is called an *autoregressive* model, because it's a regression (i.e. supervised learning with numerical response), and it's 'auto' because it's a regression of x using x itself as a predictor.)

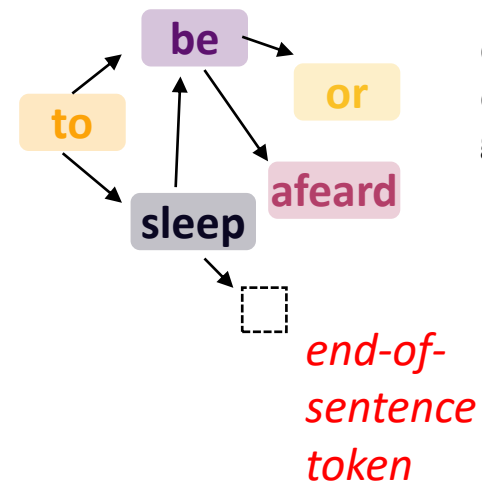
SECTION 13.2. Models for text sequences



Bag-of-words text generation

Generate a sentence (X_1, \dots, X_N) by choosing words at random from the entire corpus

```
"us the incite o'er a land-damn are peace  
incardinate take him worthy quick generals □"
```



Markov model

Generate the next word based on the preceding word. In other words, let (X_1, \dots, X_N) be a random walk on the graph of words, with weighted edges for word pairs.

"to foreign princes lie in your blessing god who shall have the prince of rome □"

Markov chain on state space $\mathbb{V} = \{w_1, w_2, \dots, w_W, \square\}$, where W is the vocabulary size. Generate \underline{X} by starting at \square and jumping from word to word until we hit \square again.

$$\square \rightarrow X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_N \rightarrow \square$$

$$\Pr_{\underline{X}}(x_1 x_2 \dots x_n) = p(x_1|\square) \times p(x_2|x_1) \times \dots \times p(x_n|x_{n-1}) \times p(\square|x_n)$$

It's easy to estimate p , the word-to-word transition probabilities, by simple counting. (Formally, this is an autoregressive model, and fitting it with MLE gives us the obvious p estimates.)



Andrei Markov (1856–1922)

be contented **to be** what they
 who is **to be** executed this
 in him **to be** truly touched
 took occasion **to be** quickly woo'd

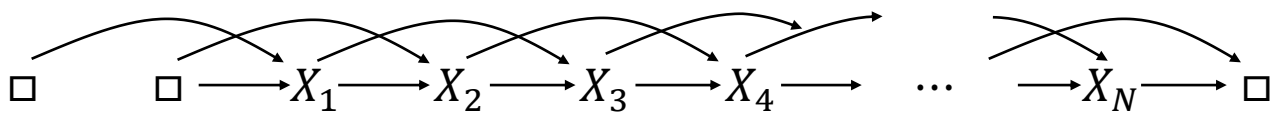
Markov's trigram model

Generate a sequence (X_1, \dots, X_N) by looking at the last *two* words, seeing where they appear in the corpus and which word comes next, and generating the next word at random from these options.

"to be wind-shaken we will be glad to receive at once for the example of thousands □"

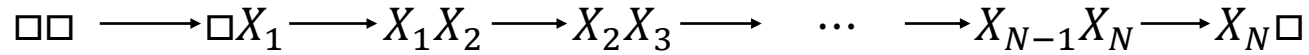
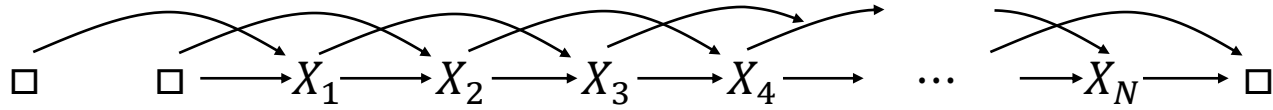
Generate \underline{X} by starting with □□ and repeatedly generating the next word based on the preceding **two**, until we produce □.

$$\Pr_{\underline{X}}(x_1 x_2 \dots x_n) = p(x_1 | \square \square) p(x_2 | \square x_1) p(x_3 | x_1 x_2) \times \dots \times p_{\theta}(x_n | x_{n-2} x_{n-1}) p_{\theta}(\square | x_{n-1} x_n)$$

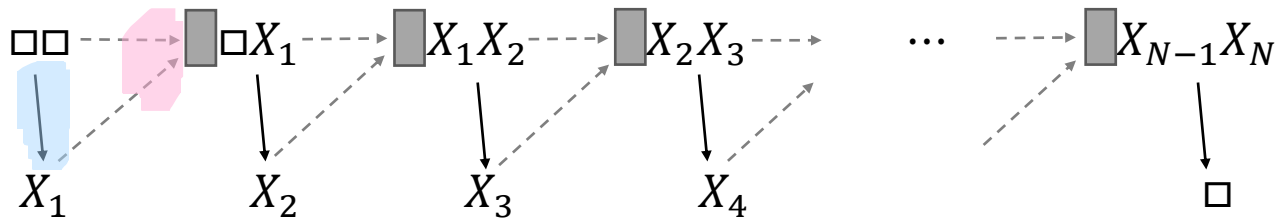


It's easy to estimate p , the (word,word)-to-word transition probabilities, by simple counting. (Before counting, preprocess the dataset by putting □□ at the start and □ at the end of every sentence.)

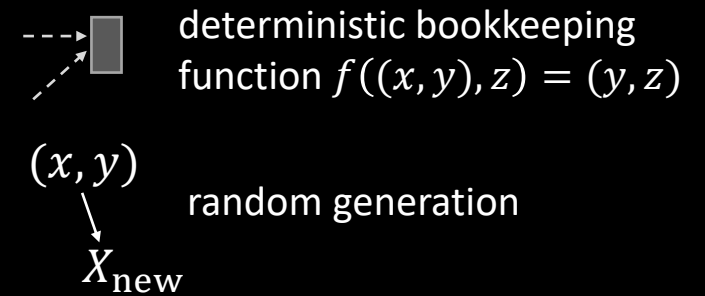
Different ways to write the trigram model:



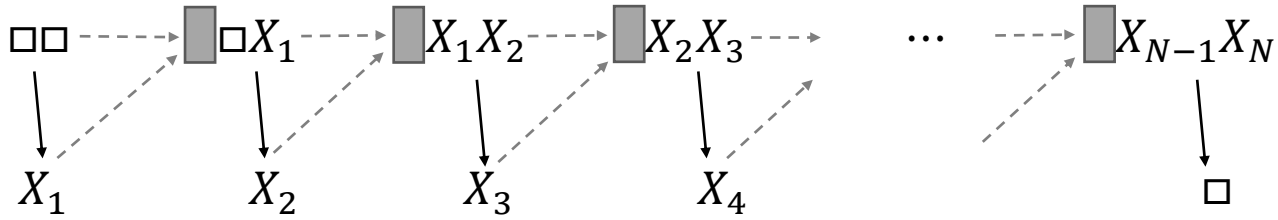
random generation
deterministic book-keeping



A Markov chain on state space \mathbb{V}^2 , where all transitions $(a, b) \rightarrow (c, d)$ with $b \neq c$ have probability 0

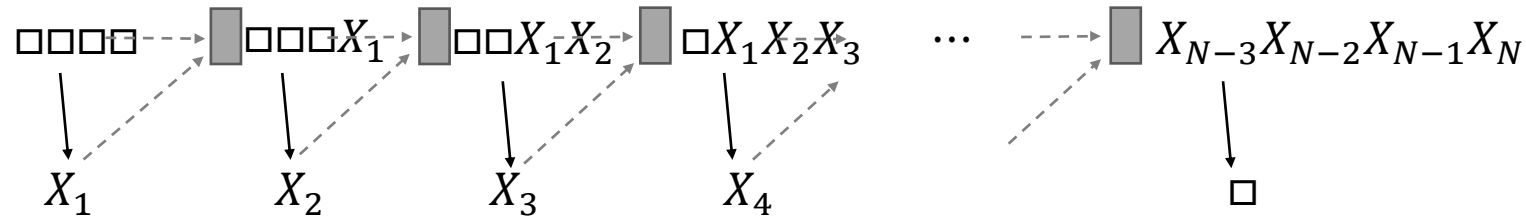


Can we get a better model by using more history?



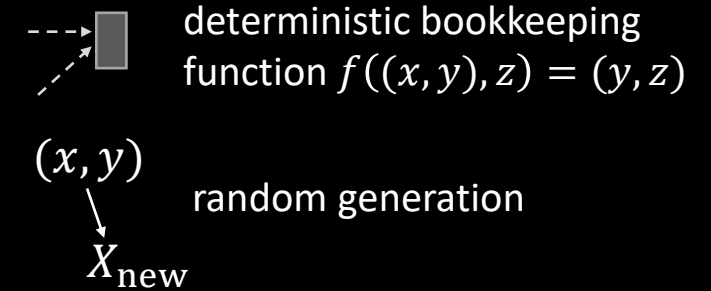
Trigram character-by-character model trained on Shakespeare:

"on youghtlee for vingiond do my not whow'd no crehout withal
deeper forand a but thave a doses?"



5-gram character-by-character model trained on Shakespeare:

"once is pleasurely. though the the with them with
comes in hand. good. give and she story tongue."

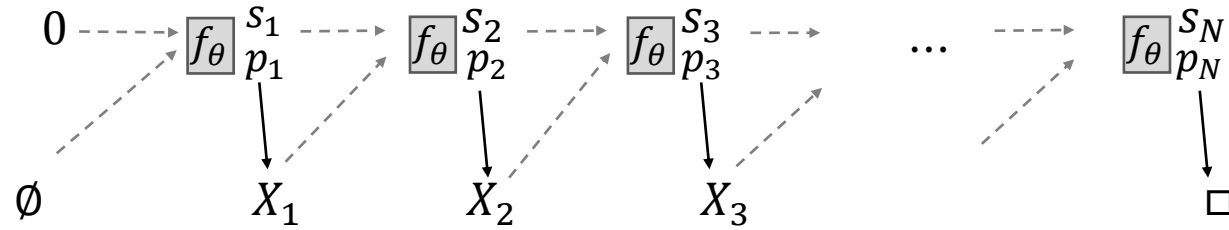


QUESTION. What are the advantages and disadvantages of a long history window?

QUESTION. Can we do better than using a fixed history window?

Recurrent Neural Network (RNN)

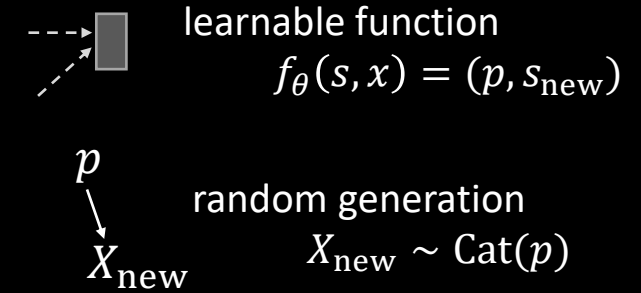
Let's use a neural network to learn an appropriate history digest. This is more flexible than choosing a fixed history window.



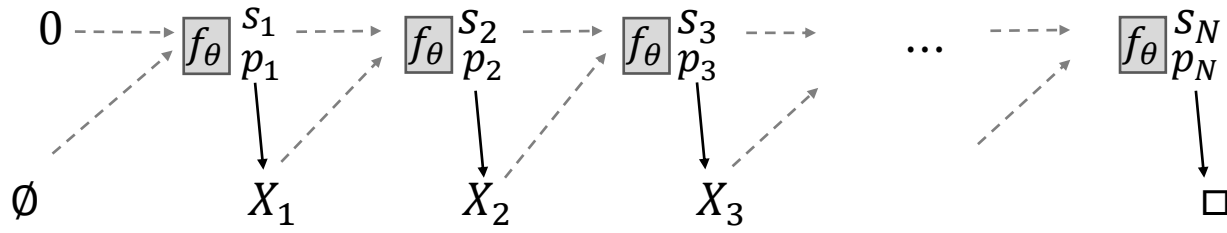
RNN character-by-character model trained on Shakespeare
[due to Andrej Karpathy]:

"PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep."



A Recurrent Neural Network (RNN) is a probability model for generating a random sequence \underline{X} .



$$X_i \sim \text{Cat}(p_i)$$

$$(s_{i+1}, p_{i+1}) = f_\theta(s_i, X_i)$$

We can train it in the usual way, by maximizing the log likelihood of our dataset. This is easy, because there's a simple explicit formula for the likelihood of a datapoint:

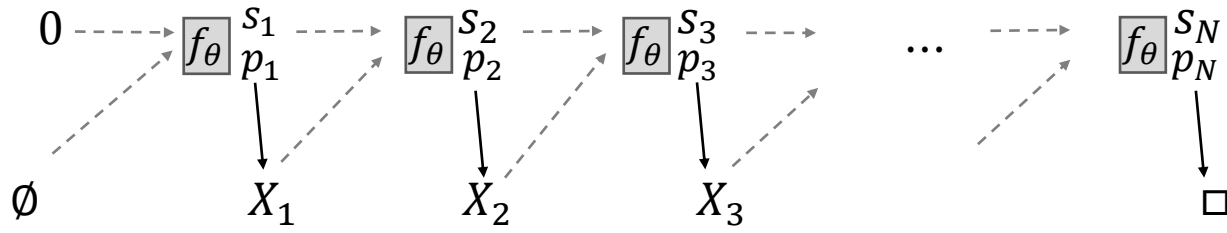
$$\Pr_{\underline{x}}(x_1 \dots x_n) = \Pr_{x_1}(x_1) \Pr_{x_2}(x_2 | x_1) \Pr_{x_3}(x_3 | x_1, x_2) \times \dots \times \Pr_{x_n}(x_n | x_1 \dots x_{n-1}) \\ \times \Pr_{x_{n+1}}(\square | x_1 \dots x_n)$$

$$= [p_1]_{x_1} [p_2]_{x_2} \times \dots \times [p_n]_{x_n} [p_{n+1}]_{\square}$$

where each p_i is a function of $x_1 \dots x_{i-1}$

```
def loglik(xstr):
    res = 0
    s, x = 0, \square
    for x_next in xstr + "\square":
        s, p = f_\theta(s, x)
        res += log(p[x_next])
        x = x_next
    return res
```

A Recurrent Neural Network (RNN) is a probability model for generating a random sequence \underline{X} .



$$X_i \sim \text{Cat}(p_i)$$

$$(s_{i+1}, p_{i+1}) = f_\theta(s_i, X_i)$$

We can train it in the usual way, by maximizing the log likelihood of our dataset. This is easy, because there's a simple explicit formula for the likelihood of a datapoint.

It's also easy to generate new strings (or to complete prompts).

```
def generate():
    xstr = ""
    x, v =  $\square$ ,  $\emptyset$ 
    while true:
        s, p =  $f_\theta(s, x)$ 
        x = np.random.choice(VOCABULARY+ $\square$ , p)
        if x ==  $\square$ : break
        xstr += x
    return xstr
```

Exercise

Given a dataset of strings, how can we generate new strings of the same general type?

abbas	abigail	andrew
abbott	abraham	anne
abby	adlai	ahab
abel	adria	...

See the notebook `nn.ipynb` for code.

The history of random sequence models

Markov
chains

RNN

LSTM

Transformers

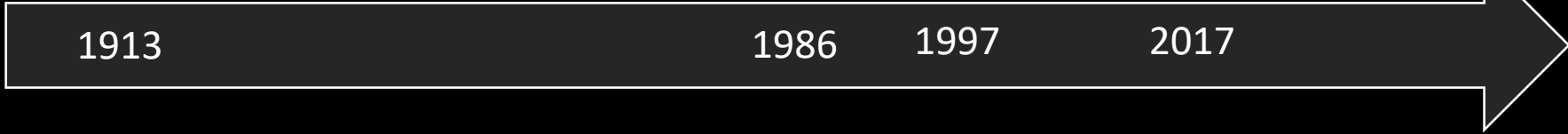
all trained by
maximum likelihood

1913

1986

1997

2017

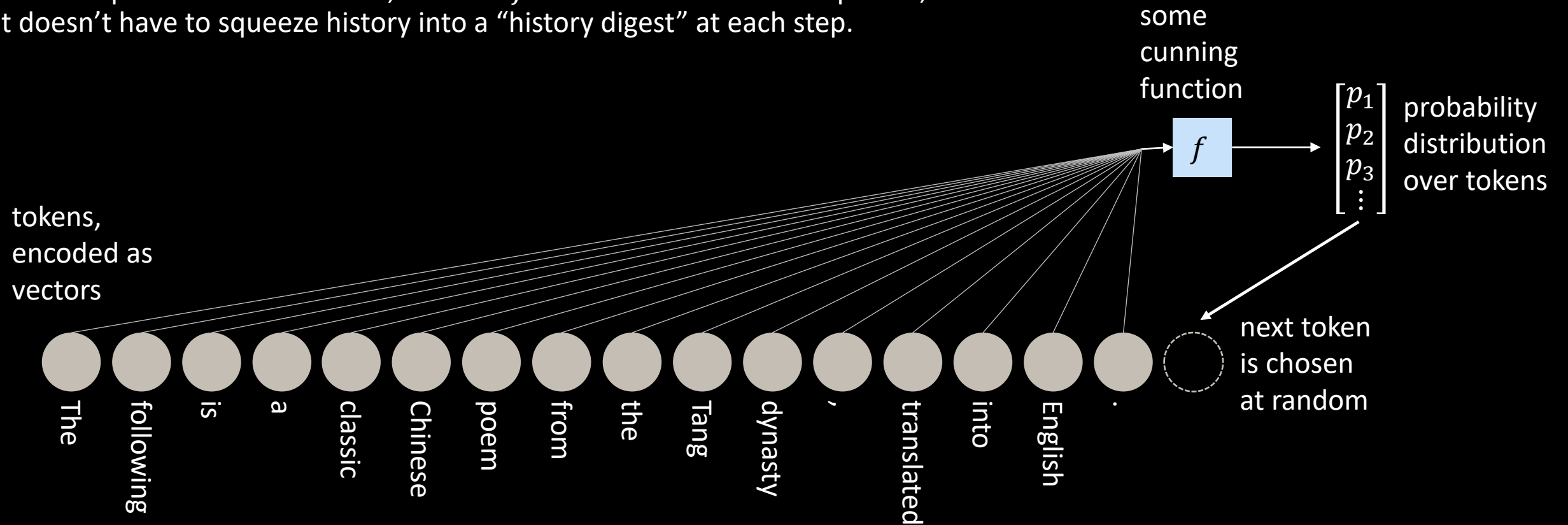


Transformer architecture

This is a probability model for a random sequence \underline{X} .

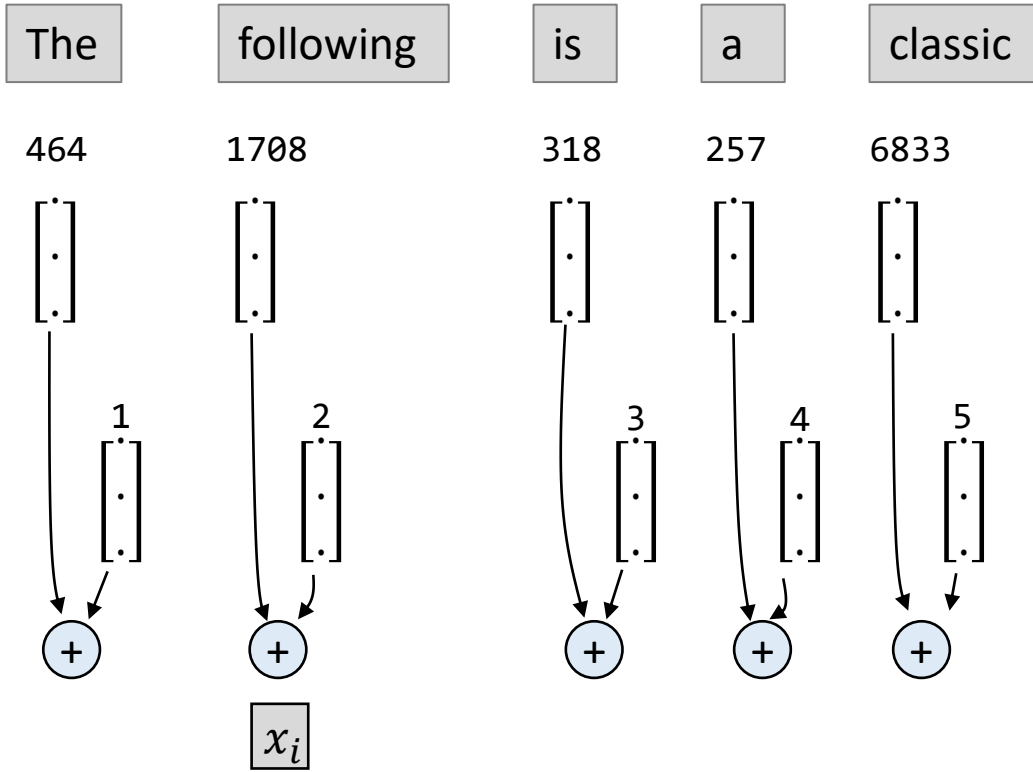
Like the RNN, there's a simple explicit formula for the log likelihood $\Pr_{\underline{X}}(\underline{x})$, so it's easy to train.

It's more powerful than an RNN, because f has access to the full sequence; it doesn't have to squeeze history into a "history digest" at each step.



The following is a classic Chinese poem from the Tang dynasty, translated into English.

What does f look like? How is it built out of differentiable functions?



$W = \text{vocabulary size}$

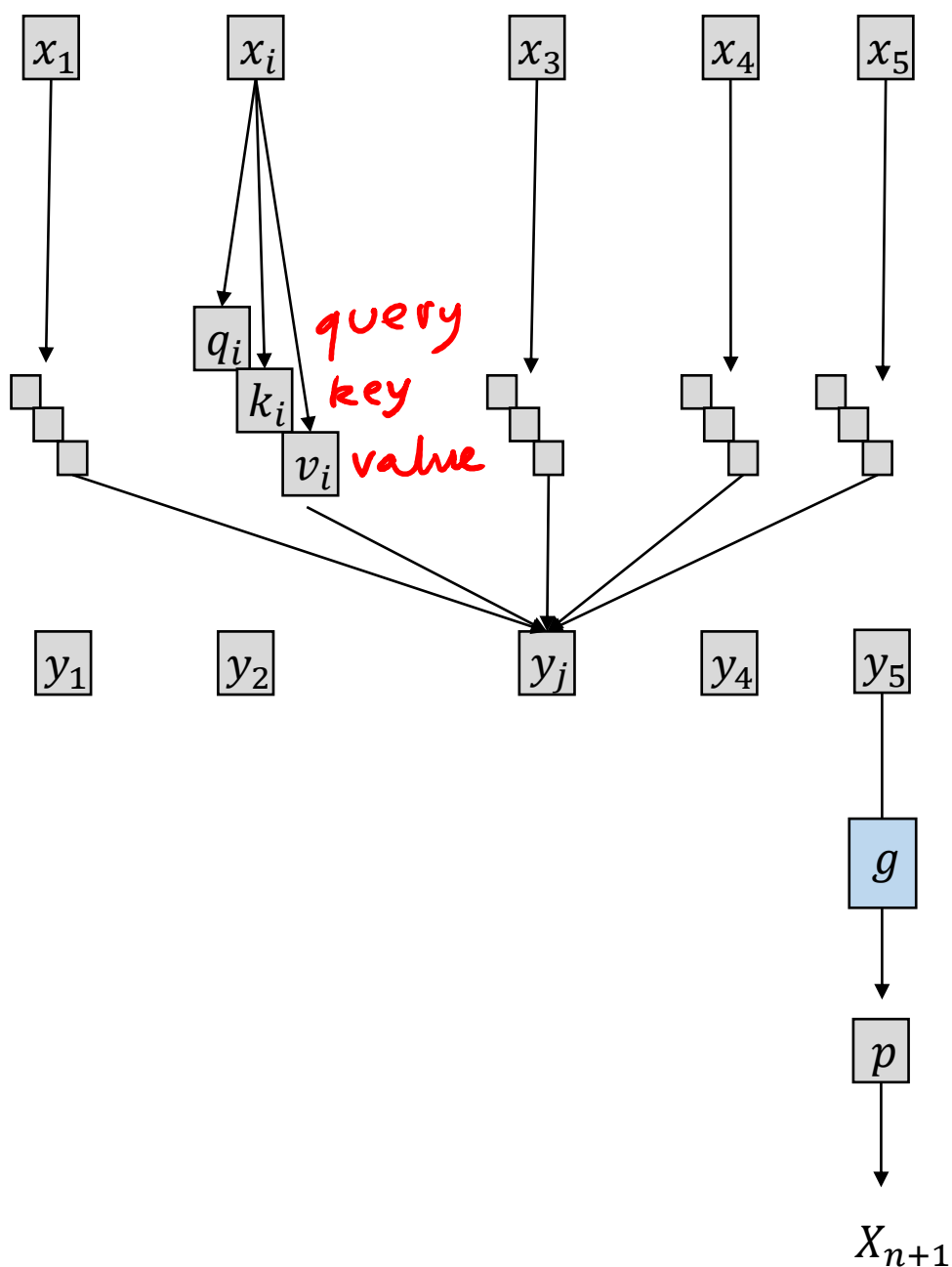
Split the text into tokens $t_i \in \{1, \dots, W\}$
Turn each token into a vector $e_i \in \mathbb{R}^d$
by looking up an embedding matrix $E \in \mathbb{R}^{W \times d}$

E is a matrix to be learnt

For each position $i \in \{1, \dots, n\}$
create a position-embedding vector $t_i \in \mathbb{R}^d$

$$\begin{bmatrix} \sin(i) \\ \cos(i) \\ \sin(i/2) \\ \cos(i/2) \\ \vdots \end{bmatrix}$$

Let $x_i = e_i + t_i \in \mathbb{R}^d$



For each position $i \in \{1, \dots, n\}$,
 let $q_i = Qx_i \in \mathbb{R}^e$, let $k_i = Kx_i \in \mathbb{R}^e$, let $v_i = Vx_i \in \mathbb{R}^d$

Q, K, V are matrices to be learnt

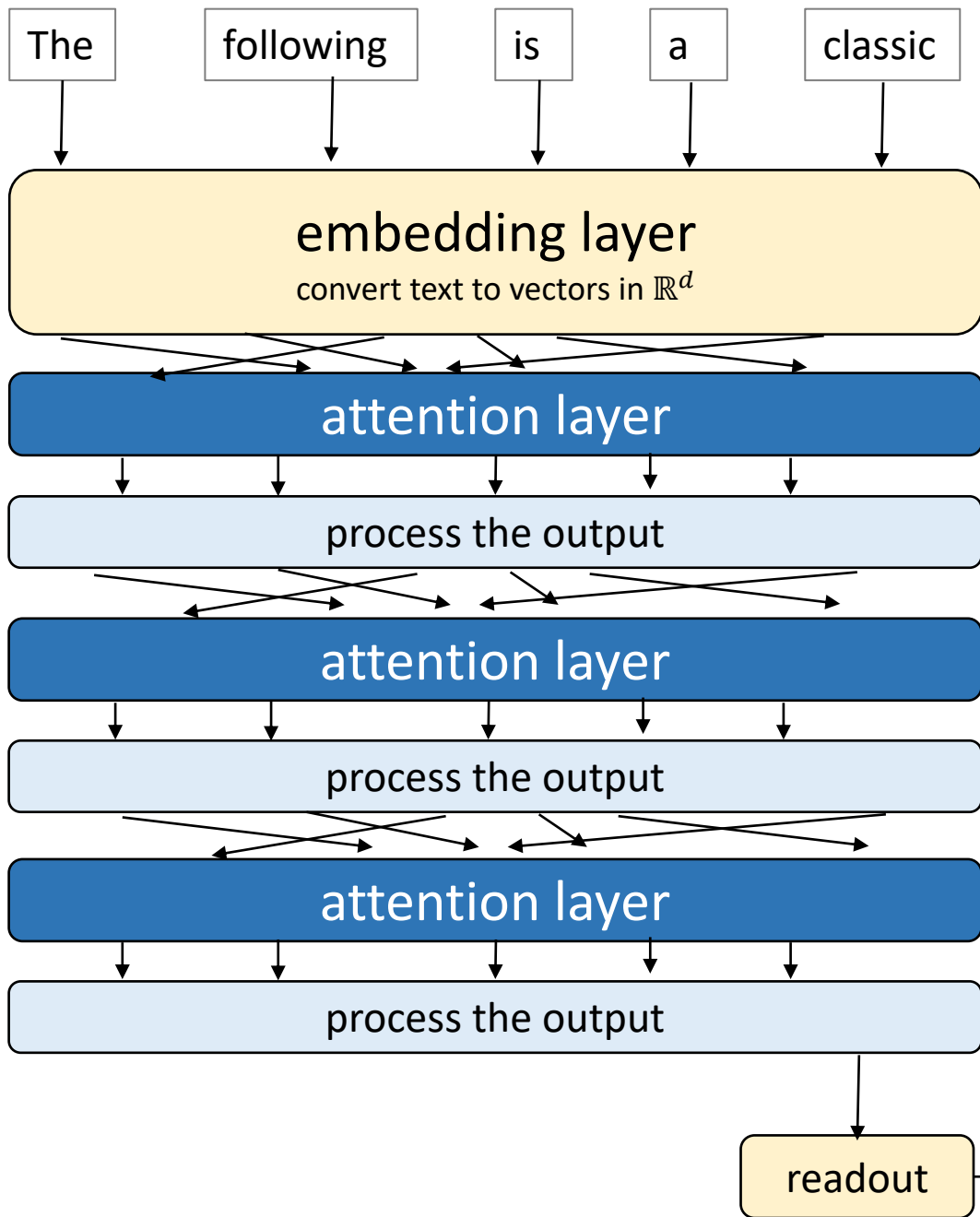
For each position $j \in \{1, \dots, n\}$ we'll produce
 an output vector $y_j \in \mathbb{R}^d$, as follows:

1. let $s_{ji} = q_j \cdot k_i$ and $a_{j*} = \text{softmax}(s_{j*}/\sqrt{e})$
2. let $y_j = \sum_i a_{ji} v_i$

a_{ji} is the attention
 that we should give
 to input x_i when
 computing output y_j

From the final value y_n , compute $p = g(y_n) \in \mathbb{R}^W$
 where g is some straightforward neural network

Generate the next token by $X_{n+1} \sim \text{Cat}(p)$



In practice, it's useful to use several passes of the attention mechanism.

The history of random sequence models

Markov
chains

RNN

LSTM

Transformers

1913

1986

1997

2017

linguistic
theories

non-
probabilistic
metrics

larger
scale

prompt
engineering

Better models of the data
All trained by maximizing the
log likelihood of the data

The history of random sequence models

Markov
chains

RNN

LSTM

Transformers

1913

1966

1986

1997

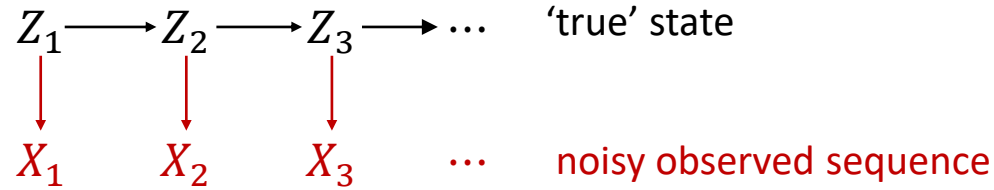
2017

Hidden
Markov
models

Better models of the data
All trained by maximizing the
log likelihood of the data



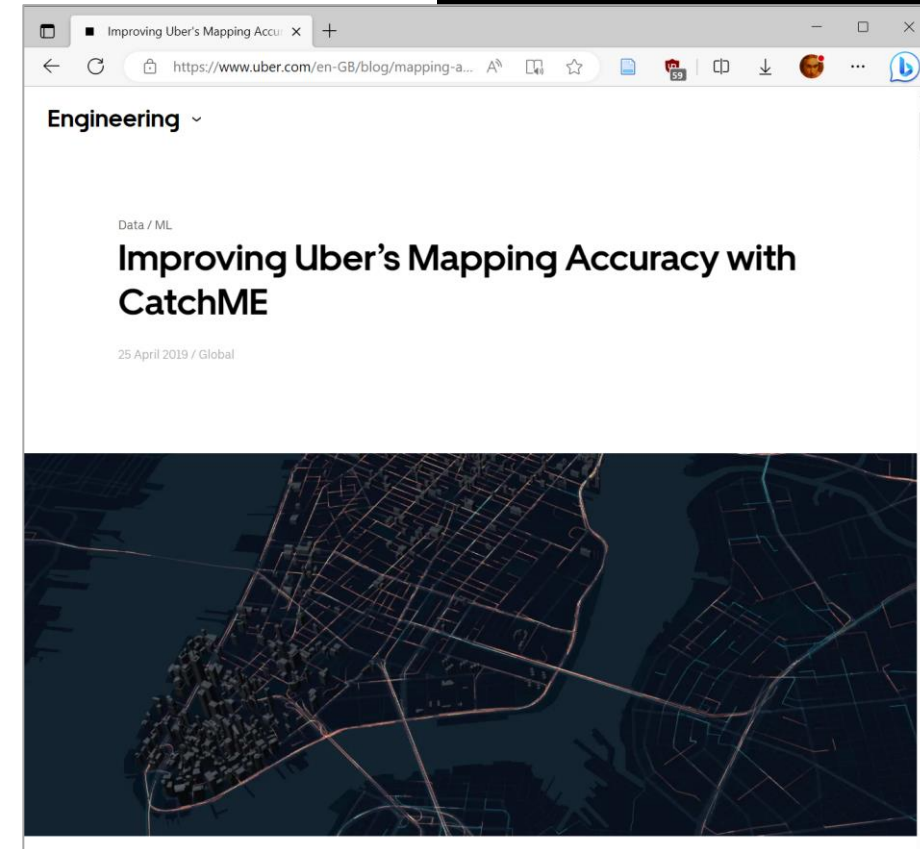
Hidden Markov models



For a hidden Markov model, the likelihood function $\Pr_{\underline{x}}(\underline{x})$ is nasty, and it's pretty much impossible to learn the model from \underline{x} data.

So why are they useful?

- Uber collects precise logs (both \underline{z} and \underline{x}) from a few drivers, so it can learn the full probability model for how \underline{Z} and \underline{X} are generated using straightforward supervised learning
- Then, for regular trips (only \underline{x} data available), it can infer \underline{Z} using Bayes's rule



一只可爱的白鼬



Group project

Our friend Data Stoaat has gone missing!

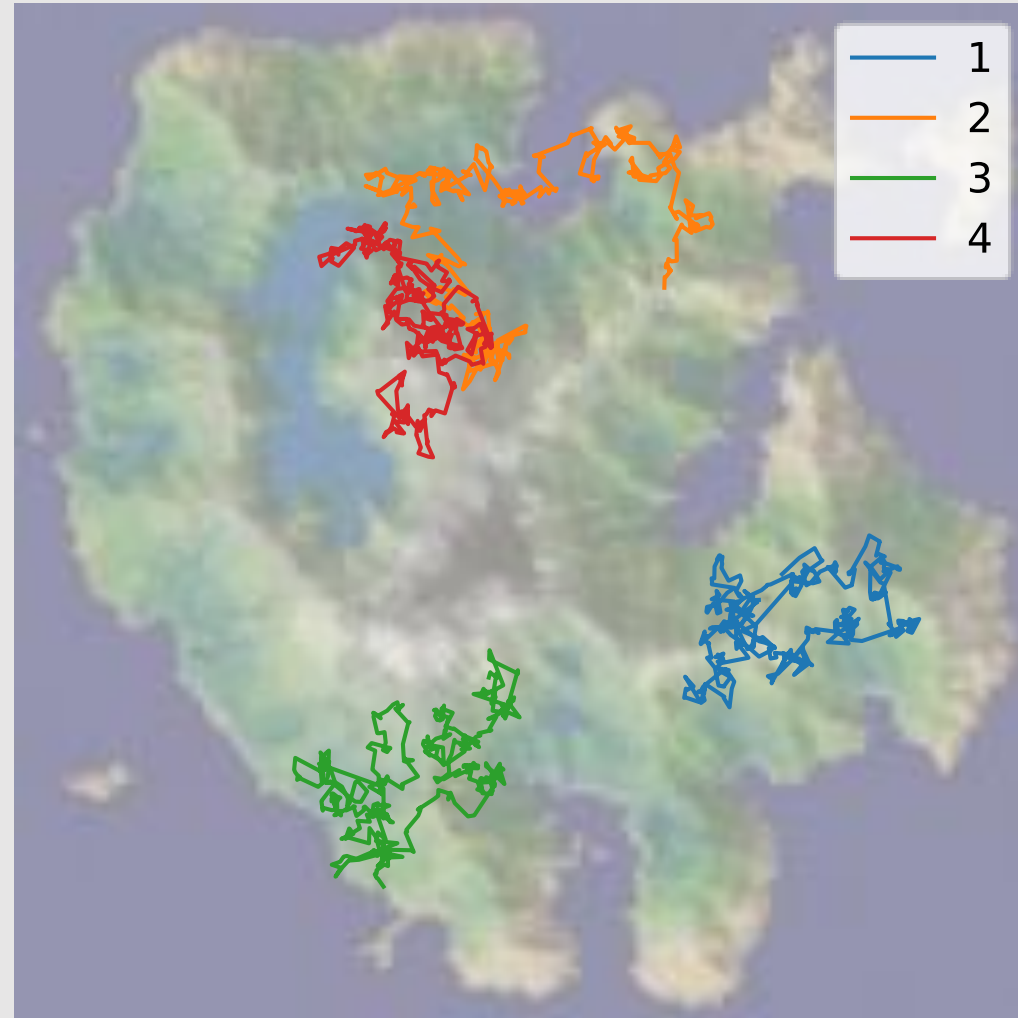
The GPS sensor that he normally carries has stopped working. But he still has a low-res camera with mobile uplink, so we know what sort of scenery they're in.

Can you help find Data Stoaat?

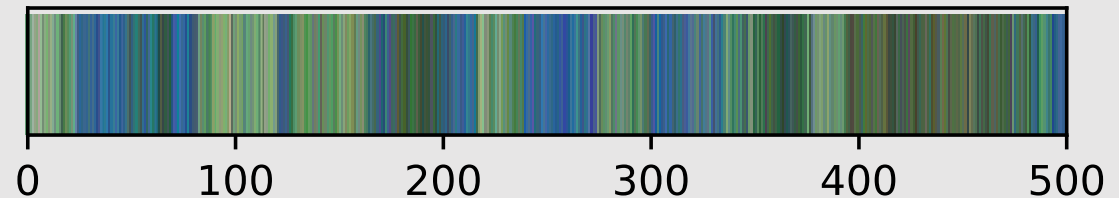


Your task: (1) use data from animals 1–4 (for which we know both \underline{z} and \underline{x}) to learn the probability model (2) use computational Bayes to find the distribution of \underline{Z} given $\underline{X} = \underline{x}$.

Animals 1--4, GPS tracks



Animal id=0, camera only



Exploring
and comparing
models

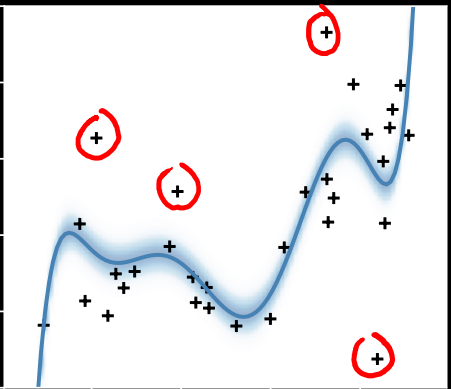
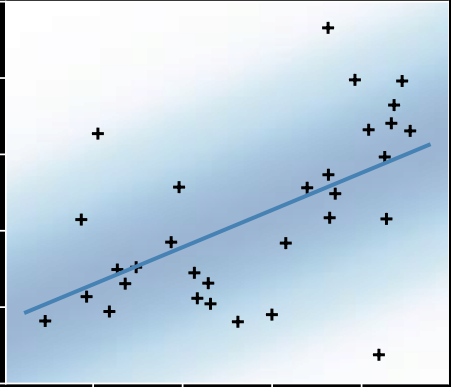
BIG IDEA 1

Log likelihood measures

how well a model fits

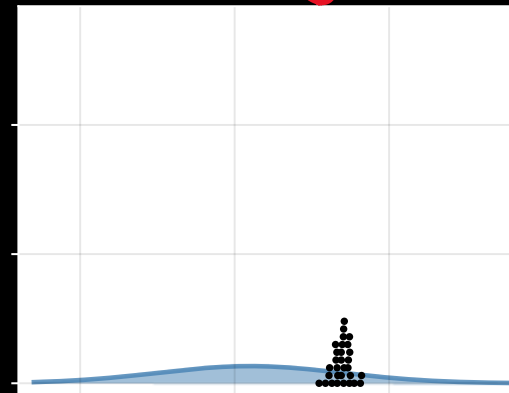
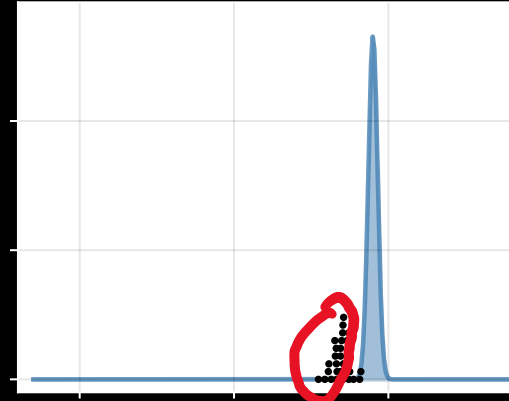
your data

SUPERVISED LEARNING



This model has a low log likelihood score, because it's a bad fit for the outlier datapoints.

GENERATIVE MODELLING



This model has a low log likelihood score, because it's a bad fit for most of the datapoints.

BIG IDEA 1

Log likelihood measures
how well a model fits
your data

→ **Model diagnostics**

Look at datapoints with low likelihood.
They'll suggest what you need to fix.

→ **Model comparison**

Pick the model that
has the higher log
likelihood

BIG IDEA 2

Evaluate your model on a
holdout dataset (if you can)



“Every genuine scientific theory must be falsifiable.

“It is easy to obtain evidence in support of virtually any theory; the evidence only counts if it is the positive result of a genuinely risky prediction.”

which is what holdout sets are there to test

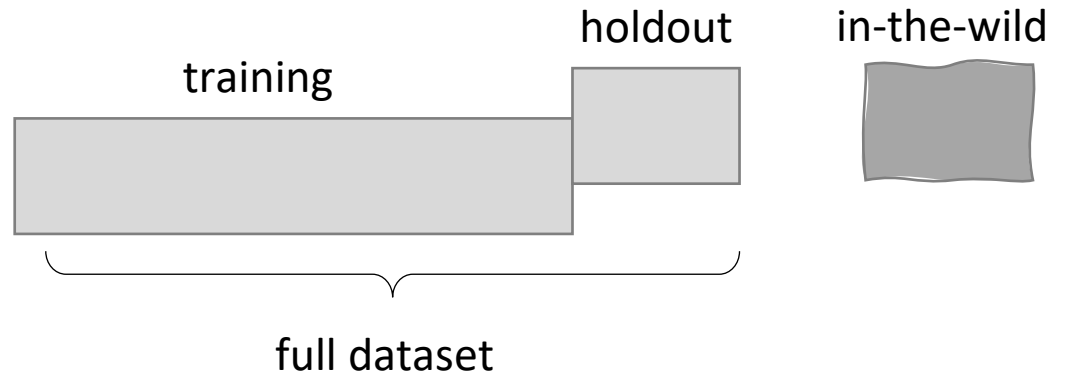
Karl Popper (1902–1994)

What’s the alternative to Popper’s philosophy? Another view is that in science we gather evidence that supports our theories. But consider the theory “all swans are white”, which is logically equivalent to “all non-white things are not swans”. Thus, a black pen is evidence in support of the theory. This is absurd! It’s why Popper doesn’t like “supporting evidence” and prefers “prediction”.



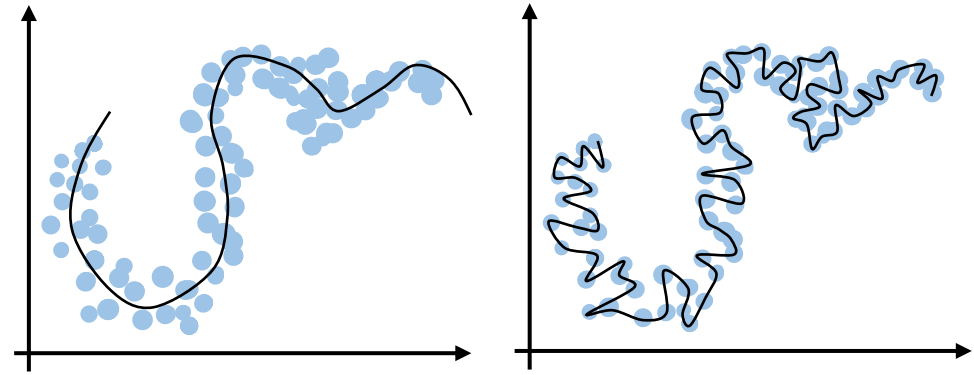
❖ What we care about is how well our model will work in the future, on *in-the-wild* data that it hasn't seen before

❖ We use holdout data as a proxy for in-the-wild data
(and so we MUST NOT PEEK at holdout data during training)



- ❖ A model is said to be *overfitted* if it's a great fit for the training data but a bad fit for holdout data

Likelihood maximization means “seek the model with the best fit”, so it *wants* to overfit



- ❖ To avoid overfitting, we need to take a step back from pure likelihood maximization

- only use low-complexity models?
- add a penalty term to the training objective?
- make life hard for gradient descent, by adding jitter (such as dropout)?

It's silly to limit ourselves unnecessarily!

There's an interesting link with Bayesianism.

Suppose we're Bayesianists, and we've proposed a model with unknown parameters θ , and we've found the posterior distribution

$$\Pr(\theta|\text{data}) = \text{const} \times \Pr(\theta) \Pr(\text{data}|\theta)$$

A simple way to summarize this posterior distribution is by reporting the MAP (Maximum A Posteriori) estimate, i.e. the value of θ that maximizes the posterior distribution. In other words, we pick θ to maximize

$$\log \Pr(\text{data}|\theta) + \log \Pr(\theta)$$

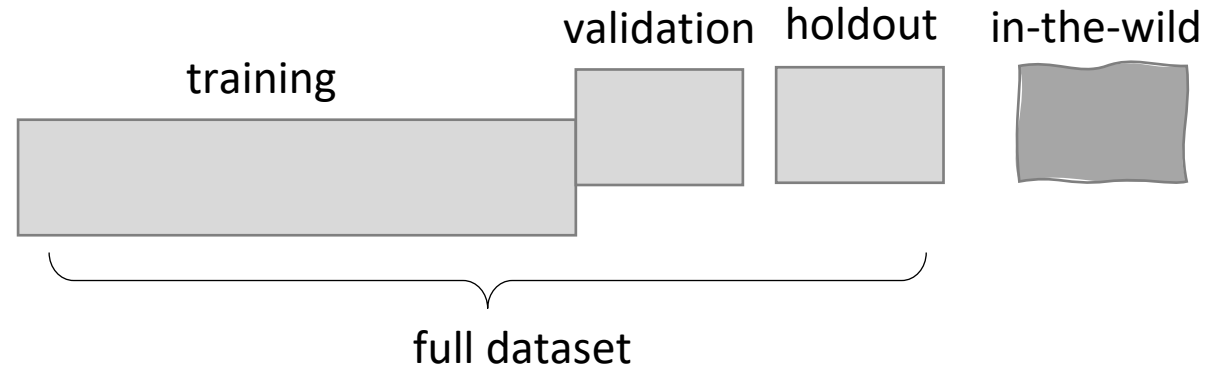
This is similar to likelihood maximization, but we've added a regularizer term $\log \Pr(\theta)$ to the objective function. In other words, our Bayesian prior belief about θ acts as a regularizer.

These are all called “regularization methods”

❖ How much regularization should we add?

Work it out by experiment!

Create a validation set (not used for fitting),
and choose the regularizer that gives best
performance on this validation set.

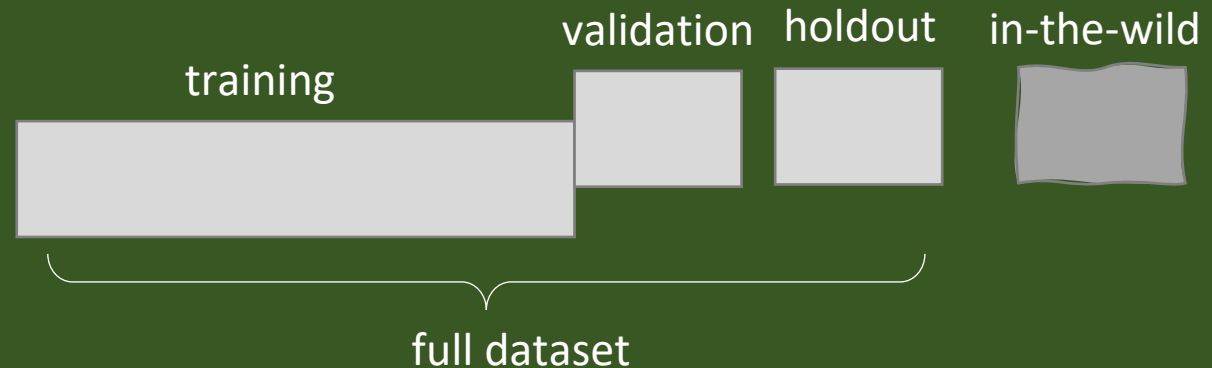


BIG IDEA 2

Evaluate your model on a holdout dataset (if you can)

→ **Cross-validation**

Regularize your training. To choose how much regularization, choose whatever works best on a validation set.



Let's poke holes
in these two big
ideas.

BIG IDEA 1

Log likelihood measures
how well a model fits
your data

→ **Model diagnostics**

Look at datapoints with low likelihood.
They'll suggest what you need to fix.

→ **Model comparison**

Pick the model that
has the higher log
likelihood

Thought experiment 1

I have a dataset of binary sequences

$\underline{x}^{(1)} = 11111111111111110111$

$\underline{x}^{(2)} = 00101010011000110101$

...

and I propose a simple model: each sequence is made of independent $\text{Bin}(1, \frac{1}{2})$ random variable.

QUESTION. Which sequence has higher likelihood, $\underline{x}^{(1)}$ or $\underline{x}^{(2)}$?

Thought experiment 2

I have a dataset of binary values

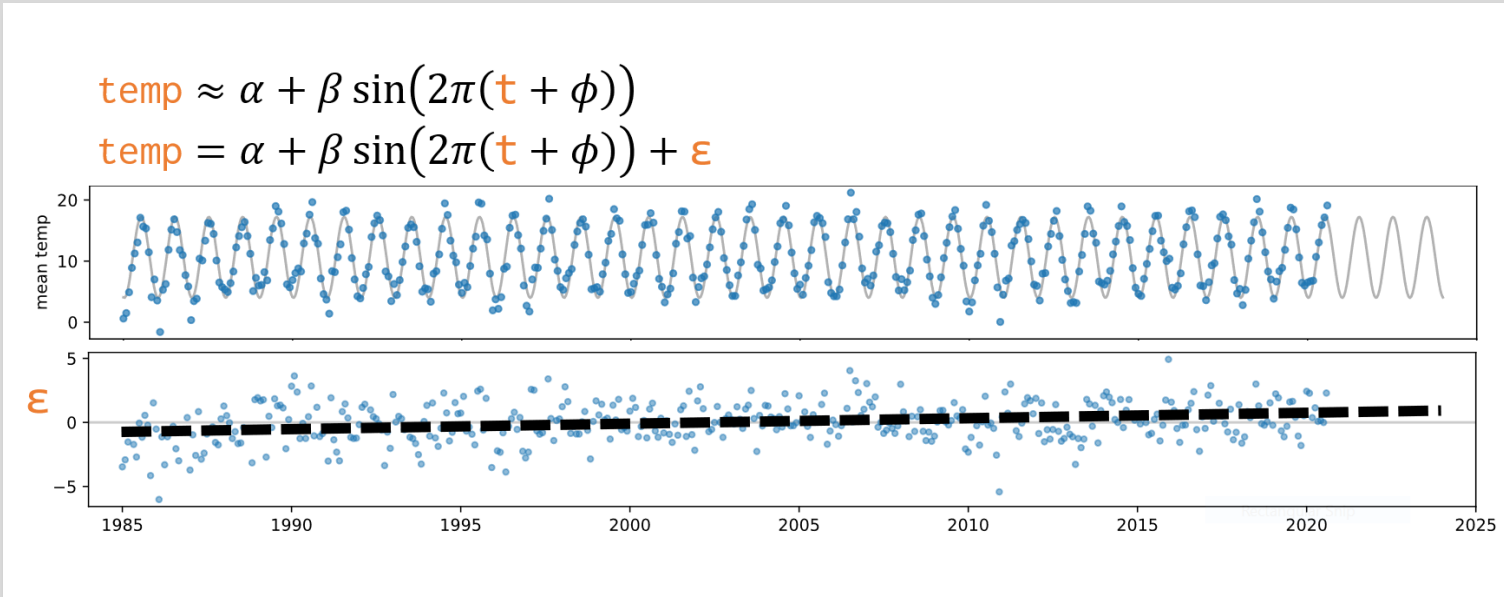
data = 11111111111111110111

and I propose a simple model: each item is an independent $\text{Bin}(1, \frac{1}{2})$ random variable.

QUESTION. What is the likelihood of each of the points in my dataset?

Is my model a good model?

In Lecture 2, to diagnose what was wrong with a model, we plotted prediction errors:



A single large prediction error, or a single low-likelihood datapoint, isn't a worry.

What's worrying is a lot of errors that all point in the same direction.

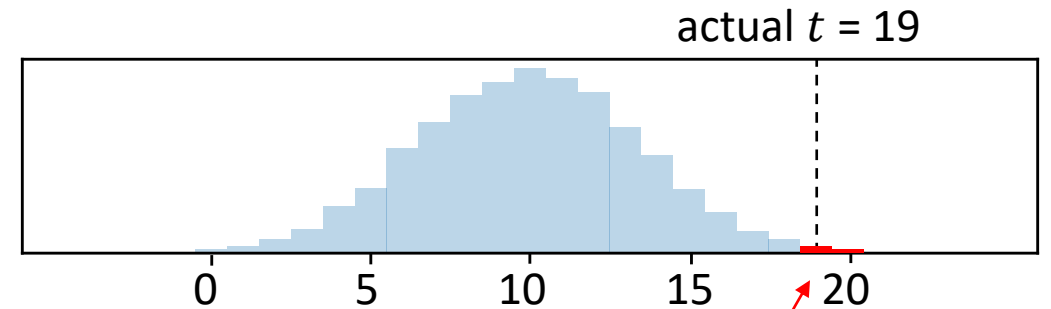
The Hypothesis Testing approach

This is a tool for looking for systematic errors in a model. It's not limited to prediction problems.

1. Propose a test statistic t . This can be any function at all that maps your dataset to a real number.
2. Using your fitted model, generate lots of synthetic datasets, and evaluate t on each of them. Plot a histogram.
3. Mark on the t of the actual dataset, and count what fraction of your simulated t are as extreme or more so than the actual t . This is the p -value.

t = number of 1s in the sequence

Generate lots of sequences of length 20, each made up of $\text{Bin}(1, \frac{1}{2})$ values.

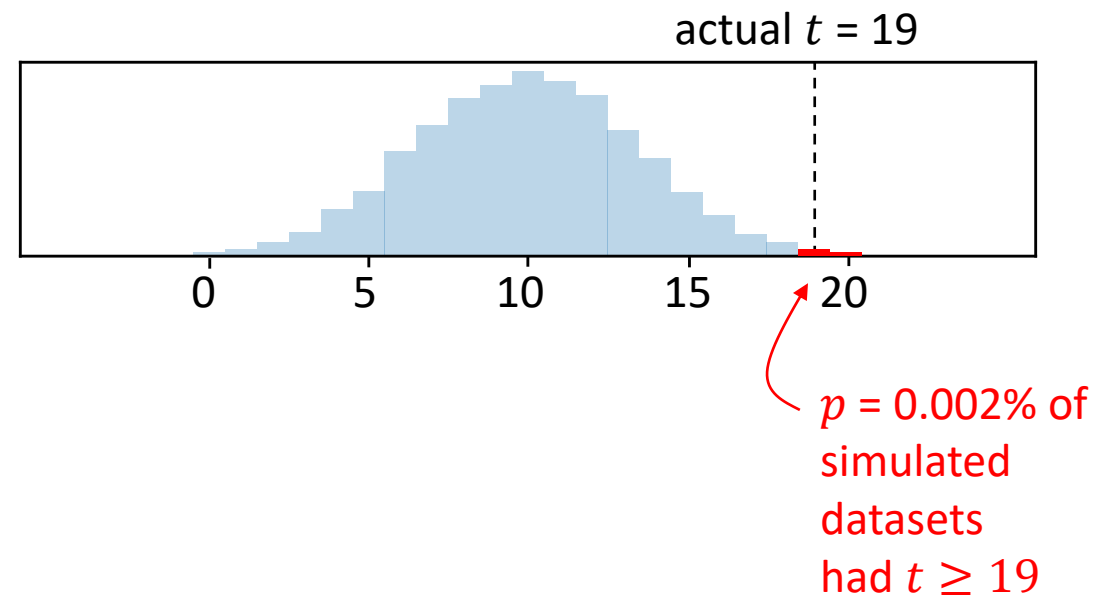


$p = 0.002\%$ of simulated datasets had $t \geq 19$

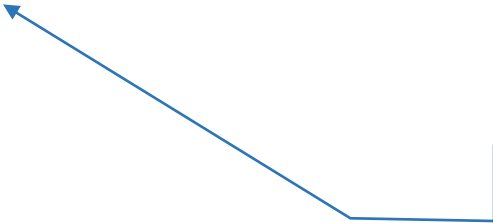
The Hypothesis Testing approach

The p -value measures “What is the chance of seeing something as extreme as my dataset, assuming my model is true?”

If the p -value is very small (e.g. <5%), your model is probably wrong.



1. Propose a test statistic t . This can be any function at all that maps your dataset to a real number.



It's up to you to choose whatever test statistic you think will be useful.

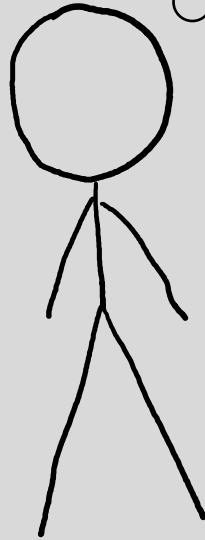
Use hypothesis testing when you have spotted a possible problem with your model, and you want to know if it's worth inventing a new model

Thought experiment 2

I have a dataset of binary values

```
data = 11111111111111110111
```

and I propose a simple model: each item is an independent $\text{Bin}(1, \frac{1}{2})$ random variable.



Hold on! If this model were true, I wouldn't expect to see so many 1s.

The p -value is 0.002%, for the test statistic "number of 1s".

So I should invent a better model!

If we want to **decide between two models**, we can use log likelihood.

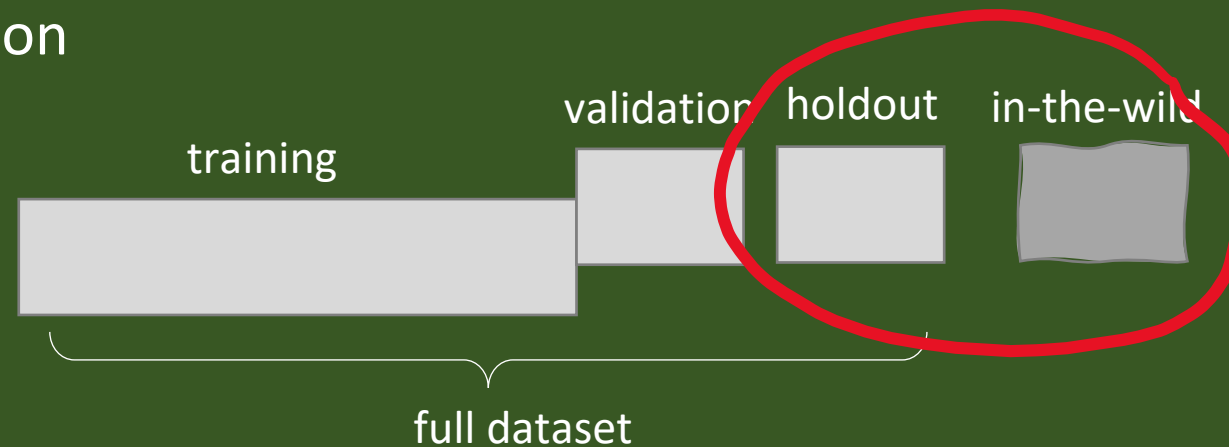
If we want to **test the fit of a single model**, we can use hypothesis testing. We don't need to propose an alternative model.

BIG IDEA 2

Evaluate your model on a holdout dataset (if you can)

→ Cross-validation

Regularize your training. To choose how much regularization, choose whatever works best on a validation set.



“All science is either physics or stamp
collecting.”

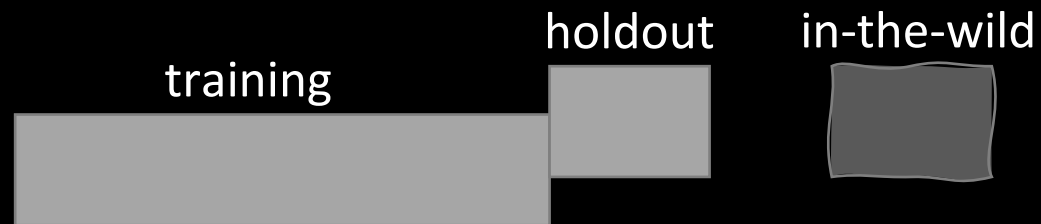
Ernest Rutherford

Table 2: Results on HotpotQA distractor (dev). (+hyperlink) means usage of extra hyperlink data in Wikipedia. Models beginning with “–” are ablation studies without the corresponding design.

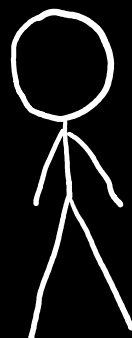
Model	Ans EM	Ans F_1	Sup EM	Sup F_1	Joint EM	Joint F_1
Baseline [53]	45.60	59.02	20.32	64.49	10.83	40.16
DecompRC [29]	55.20	69.63	N/A	N/A	N/A	N/A
QFE [30]	53.86	68.06	57.75	84.49	34.63	59.61
DFGN [36]	56.31	69.69	51.50	81.62	33.62	59.82
SAE [45]	60.36	73.58	56.93	84.63	38.81	64.96
SAE-large	66.92	79.62	61.53	86.86	45.36	71.45
HGN [14] (+hyperlink)	66.07	79.36	60.33	87.33	43.57	71.03
HGN-large (+hyperlink)	69.22	82.19	62.76	88.47	47.11	74.21
<i>BERT (sliding window) variants</i>						
BERT Plus	55.84	69.76	42.88	80.74	27.13	58.23
LQR-net + BERT	57.20	70.66	50.20	82.42	31.18	59.99
GRN + BERT	55.12	68.98	52.55	84.06	32.88	60.31
EPS + BERT	60.13	73.31	52.55	83.20	35.40	63.41
LQR-net 2 + BERT	60.20	73.78	56.21	84.09	36.56	63.68
P-BERT	61.18	74.16	51.38	82.76	35.42	63.79
EPS + BERT(large)	63.29	76.36	58.25	85.60	41.39	67.92
CogLTX	65.09	78.72	56.15	85.78	39.12	69.21
– multi-step reasoning	62.00	75.39	51.74	83.10	35.85	65.35
– rehearsal & decay	61.44	74.99	7.74	47.37	5.36	37.74
– train-test matching	63.20	77.21	52.57	84.21	36.11	66.90

stamp
collecting
(as Rutherford
would say)

Results. Table 2 shows that CogLTX outperforms most of previous methods and all 7 BERT variants solutions on the leaderboard.⁴ These solutions basically follow the framework of aggregating the results from sliding windows by extra neural networks, leading to bounded performances attributed to insufficient interaction across paragraphs.



full dataset



“My classification algorithm achieves 93.7% accuracy on the holdout set.”

What your readers think:

For a new in-the-wild datapoint x ,
 $\mathbb{P}(\text{classify } x \text{ correctly}) = 93.7\%$

What you actually meant:

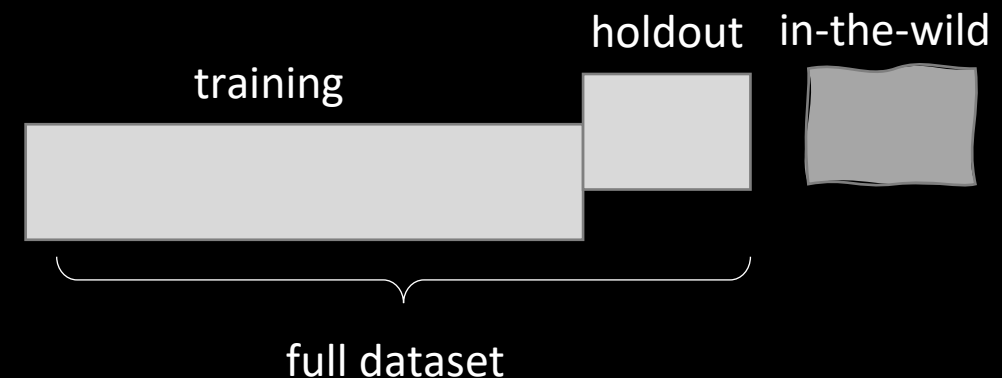
When I take a bunch of new in-the-wild datapoints (**matching the composition of my holdout set**) then, averaged across this bunch,

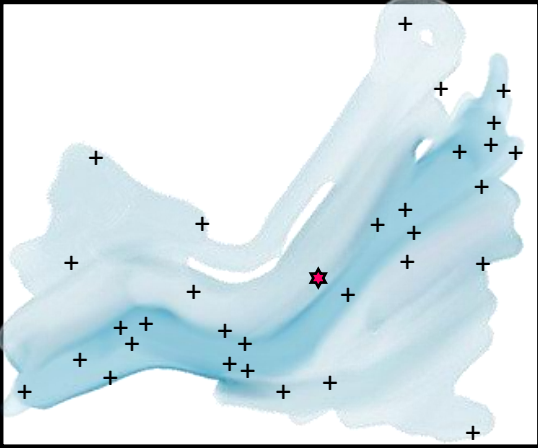
fraction classified correctly = 93.7%

“Every genuine scientific theory must be falsifiable.

“It is easy to obtain evidence in support of virtually any theory; the evidence only counts if it is the positive result of a genuinely **risky prediction.**”

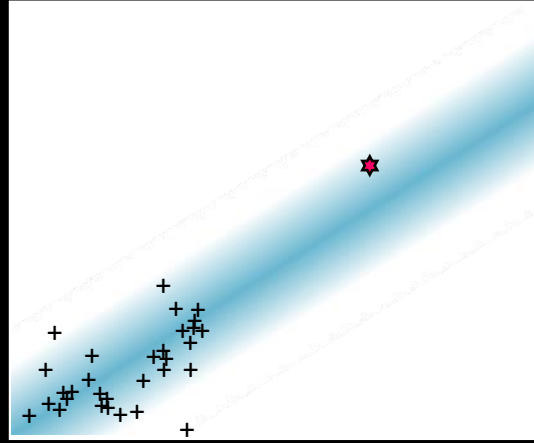
Karl Popper (1902–1994)





Machine learning approach

“The job of a model is to **generalize to new data**. I’ll split my data into training + holdout, and measure how accurate it is on the holdout set.”



Scientist's approach

“The job of a model is to **generalize to novel situations**. Any model that’s not based on well-grounded scientific concepts will probably make bad predictions.”

Everything should be made
as simple as possible,
but not simpler.

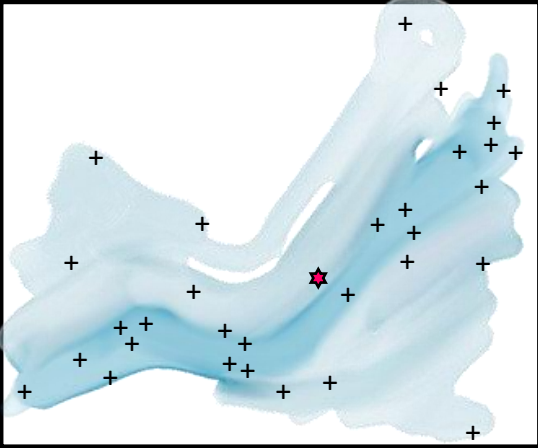
Albert Einstein

“It can scarcely be denied that the supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience.”

*the Herbert Spencer Lecture, Oxford, 10 June
1933*

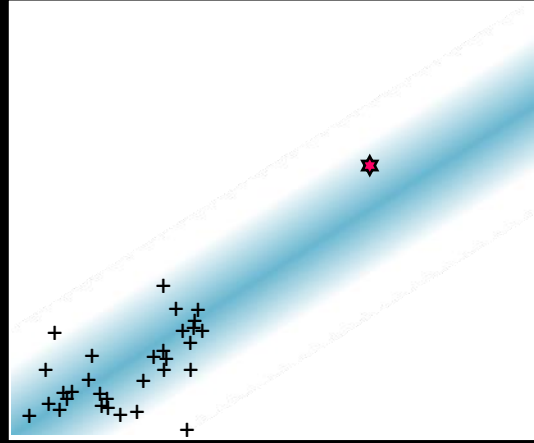
“It is more important to have
beauty in one’s equations than
to have them fit experiment.”

Paul Dirac



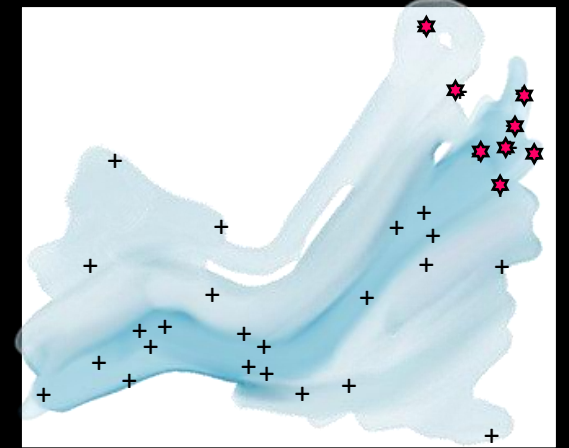
Machine learning approach

“The job of a model is to **generalize to new data**. I’ll split my data into training + holdout, and measure how accurate it is on the holdout set.”



Scientist's approach

“The job of a model is to **generalize to novel situations**. Any model that's not based on well-grounded scientific concepts will probably make bad predictions.”



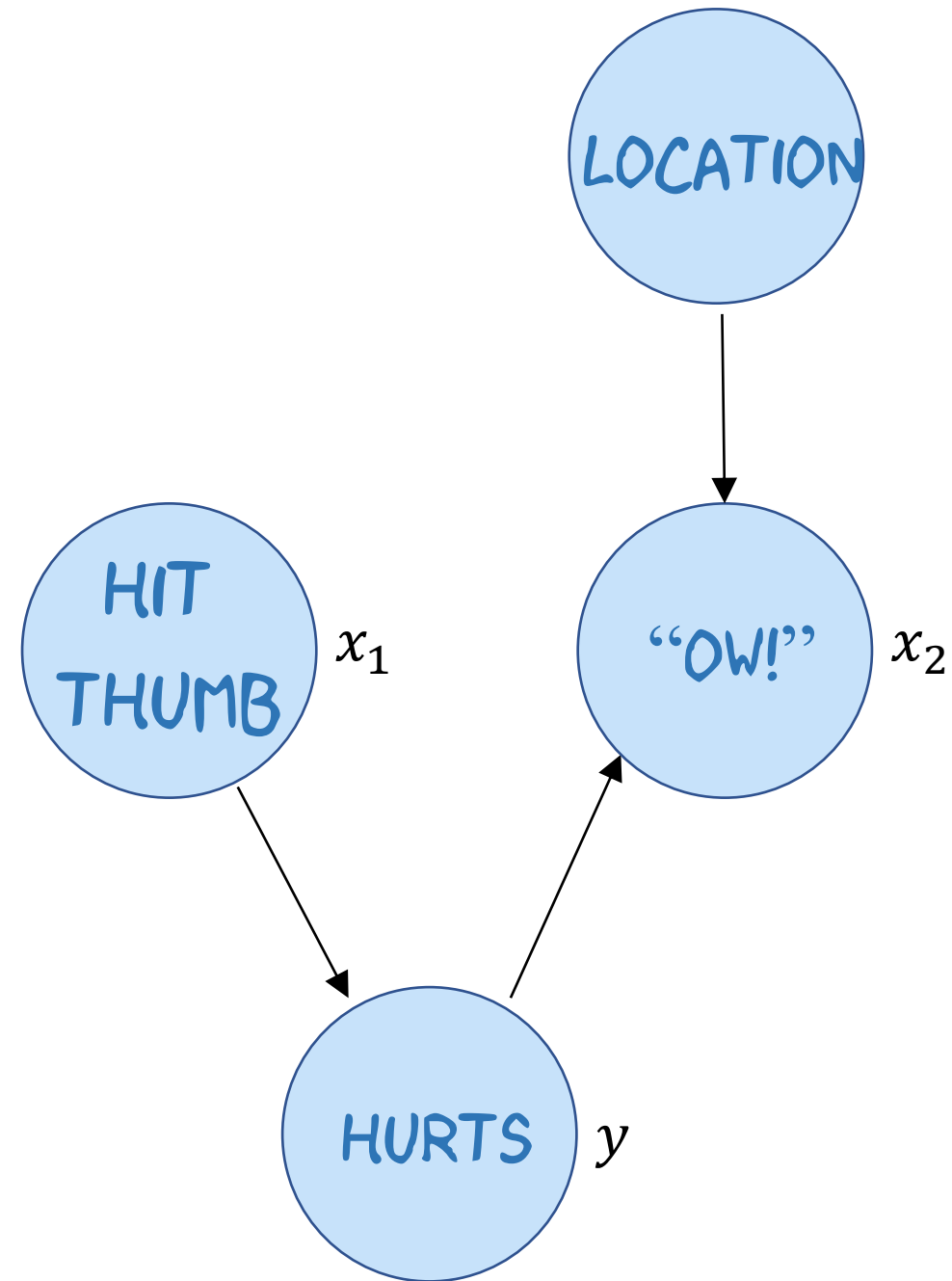
Simple tip

Don't choose your holdout set by random shuffling. Perhaps choose it to be the most extreme 10%?

Throughout this course, I've tried to persuade you that machine learning *is* probability modelling.

But how can probability modelling possibly be enough to address the big questions?

- ❖ Safety
- ❖ Alignment, fairness
- ❖ Explainability, latent knowledge
- ❖ Domain shift, meta learning
- ❖ Adaptive learning



Suppose we're given a dataset of (x_1, x_2, y) and we're asked to predict y .

We go ahead and train a model, and find that shouting "Ow!" predicts hurt (99.5% accuracy).

We deploy our model – in a library. Now it makes rubbish predictions.

This is called *domain shift*.

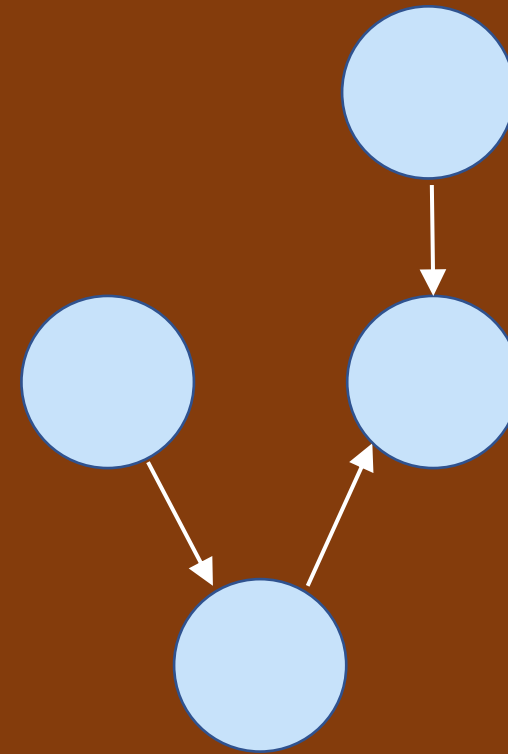
We retrain our model on library data. Now it makes bad predictions on our original dataset.

This is called *catastrophic forgetting*.

Throughout this course, I've tried to persuade you that machine learning *is* probability modelling.

The next big thing in machine learning is building systems that can learn *causal models* from data.

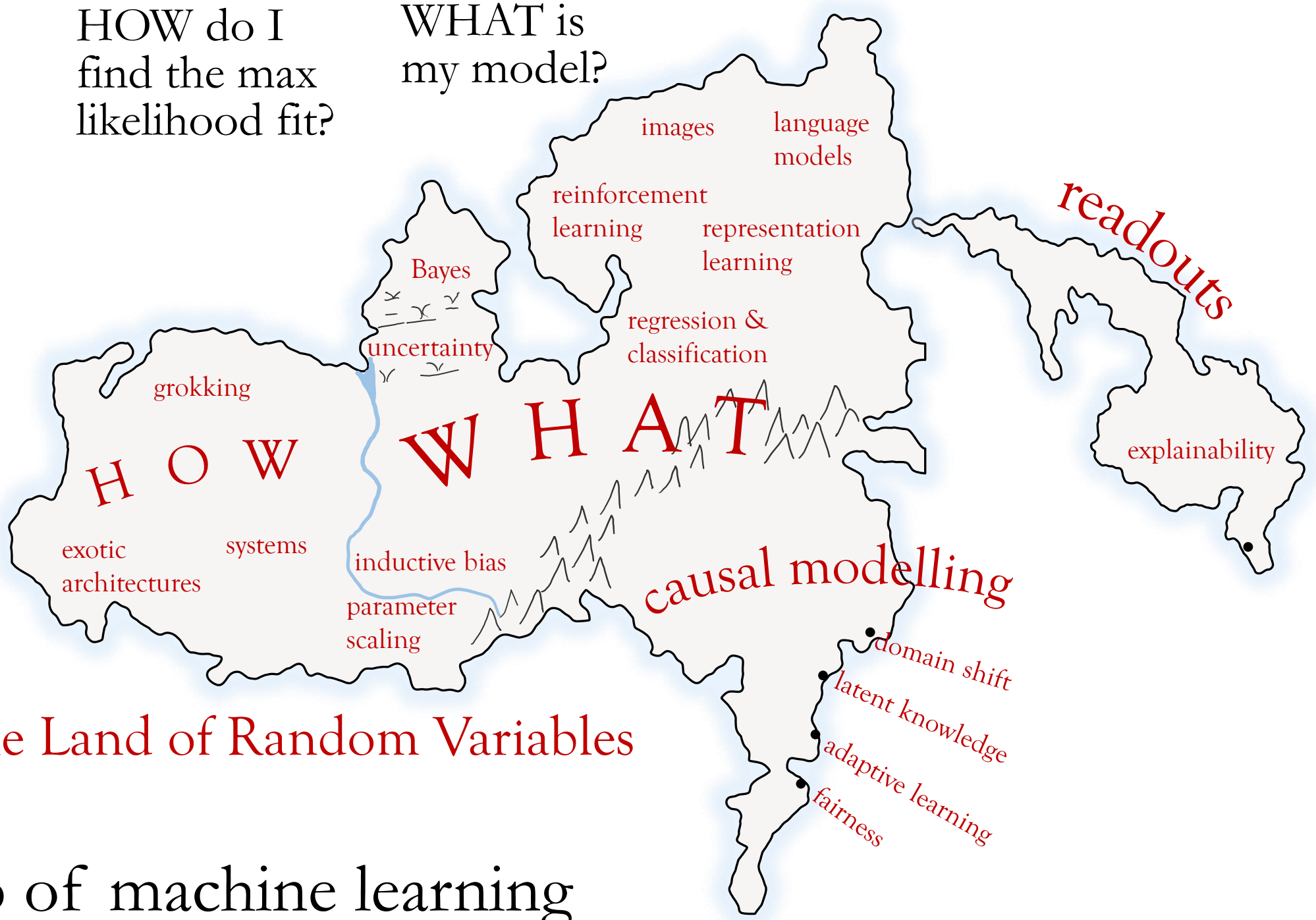
(They'll still have to be probabilistic models, of course. That's the only sort of model that works robustly.)



non-probabilistic models

HOW do I find the max likelihood fit?

WHAT is my model?



The Land of Random Variables

A map of machine learning