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?
GPT is a model for sequences.

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

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

<table>
<thead>
<tr>
<th>TEXT</th>
<th>TOKEN IDS</th>
</tr>
</thead>
</table>
GPT is a probability model for sequences of tokens

- Let $X = X_0X_1X_2 \cdots X_N$ be a random sequence of tokens, of random length $N$
- GPT has been trained to fit a probability model for $X$ to its training dataset $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$
- 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 $\left( X \mid x_0x_1 \cdots x_m \right)$

GPT playground: https://platform.openai.com/playground?mode=complete
SECTION 13

How can we learn a probability model for sequences?
Example 13.1.1: fitting a Markov model

Let \((x_0, x_1, \ldots, x_n)\) 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 \(\sigma\) using maximum likelihood estimation.

The question tells us \(X_{i+1} \sim a + b X_i + N(0, \sigma^2)\)

therefore \(X_i \sim N(a + b X_{i-1}, \sigma^2)\)

and so the likelihood is

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

To fit this model (i.e. to learn \(a\), \(b\), \(\sigma\)) 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 \(x_{i+1} \approx a + b x_i\). We can fit it with sklearn, using the feature vectors \([1,1,\ldots,1]\) and \([x_0, x_1, \ldots, x_{n-1}]\) and response vector \([x_1, x_2, \ldots, x_n]\).

(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.)
Bag-of-words text generation

Generate a sentence \((X_1, \ldots, X_N)\) 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 \((X_1, \ldots, X_N)\) 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 \(\mathcal{V} = \{w_1, w_2, \ldots, w_W, \square\}\), where \(W\) is the vocabulary size. Generate \(X\) by starting at \(\square\) and jumping from word to word until we hit \(\square\) again.

\[\square \to X_1 \to X_2 \to \cdots \to X_N \to \square\]

\[
\Pr_X(x_1x_2\cdots x_n) = p(x_1|\square) \times p(x_2|x_1) \times \cdots \times p(x_n|x_{n-1}) \times p(\square|x_n)
\]

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)
Markov’s trigram model
Generate a sequence \((X_1, \ldots, X_N)\) 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 \(X\) by starting with □□ and repeatedly generating the next word based on the preceding two, until we produce □.

\[
Pr_X(x_1x_2 \cdots x_n) = p(x_1|\square\square) p(x_2|\square x_1) p(x_3|x_1x_2) \times \cdots \times p_\theta(x_n|x_{n-2}x_{n-1}) p_\theta(\square|x_{n-1}x_n)
\]

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:

A Markov chain on state space $\mathbb{V}^2$, where all transitions $(a, b) \to (c, d)$ with $b \neq c$ have probability 0.
Can we get a better model by using more history?

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.

\[
\begin{align*}
0 & \rightarrow f_\theta s_1 \rightarrow f_\theta s_2 \rightarrow f_\theta s_3 \rightarrow \ldots \rightarrow f_\theta s_N \\
\emptyset & \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow \square
\end{align*}
\]

- learnable function \( f_\theta (s, x) = (p, s_{\text{new}}) \)
- random generation \( X_{\text{new}} \sim \text{Cat}(p) \)

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 srain would be attain’d into being never fed,
And who is but a chain and subjects of his death,
I should not sleep."

\( p \)
\( X_{\text{new}} \)
A Recurrent Neural Network (RNN) is a probability model for generating a random sequence $X$.

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

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

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:

$$P_{\theta}(x_1, \ldots, x_n) = P_{\theta_{x_1}}(x_1) P_{\theta_{x_2}}(x_2 | x_1) P_{\theta_{x_3}}(x_3 | x_2) \cdots P_{\theta_{x_n}}(x_n | x_{i-1}, \ldots, x_1)$$

$$\times P_{\theta_{x_{n+1}}}(\emptyset | x_1, \ldots, x_n)$$

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

where each $p_i$ is a function of $x_1, \ldots, x_{i-1}$.
A Recurrent Neural Network (RNN) is a probability model for generating a random sequence $X$.

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

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

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).

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

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

<table>
<thead>
<tr>
<th>abbas</th>
<th>abigail</th>
<th>andrew</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbott</td>
<td>abraham</td>
<td>anne</td>
</tr>
<tr>
<td>abby</td>
<td>adlai</td>
<td>ahab</td>
</tr>
<tr>
<td>abel</td>
<td>adria</td>
<td>...</td>
</tr>
</tbody>
</table>

See the notebook nn.ipynb for code.
The history of random sequence models

- Markov chains
  - 1913
- RNN
  - 1986
- LSTM
  - 1997
- Transformers
  - 2017
- All trained by maximum likelihood
Transformer architecture

This is a probability model for a random sequence $X$.

Like the RNN, there's a simple explicit formula for the log likelihood $\Pr_X(x)$, 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.

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?

Split the text into tokens $t_i \in \{1, ..., W\}$

Turn each token into a vector $e_i \in \mathbb{R}^d$ by looking up an embedding matrix $E \in \mathbb{R}^{W \times d}$

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

Let $x_i = e_i + t_i \in \mathbb{R}^d$
For each position $i \in \{1, \ldots, n\}$, let $q_i = Qx_i$, let $k_i = Kx_i$, let $v_i = Vx_i \in \mathbb{R}^e$. 

$Q, K, V$ are matrices to be learnt.

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

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

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** (1913)
- **RNN** (1986)
- **LSTM** (1997)
- **Transformers** (2017)

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

- Linguistic theories
- Non-probabilistic metrics
- Larger scale
- Prompt engineering
The history of random sequence models

- Markov chains: 1913
- RNN: 1966
- LSTM: 1986
- Transformers: 1997
- Better models of the data: 2017

All trained by maximizing the log likelihood of the data.
Hidden Markov models

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

So why are they useful?

- Uber collects precise logs (both $Z$ and $X$) from a few drivers, so it can learn the full probability model for how $Z$ and $X$ are generated using straightforward supervised learning.
- Then, for regular trips (only $X$ data available), it can infer $Z$ using Bayes’s rule.
一只可爱的白鼬
Group project
Our friend Data Stoat 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 Stoat?

Your task: (1) use data from animals 1–4 (for which we know both $z$ and $x$) to learn the probability model (2) use computational Bayes to find the distribution of $Z$ given $X = x$. 
Exploring and comparing models
BIG IDEA 1
Log likelihood measures how well a model fits your data
This model has a low log likelihood score, because it's a bad fit for most of the datapoints.

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

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”.

which is what holdout sets are there to test
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 PEEEK 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)?

These are all called “regularization methods”.

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 $\theta$, 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 $\theta$ that maximizes the posterior distribution. In other words, we pick $\theta$ 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 $\theta$ acts as a regularizer.
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 2
I have a dataset of binary values
\[ \text{data} = 1111111111111110111 \]
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?

Thought experiment 1
I have a dataset of binary sequences
\[ x^{(1)} = 1111111111111110111 \]
\[ x^{(2)} = 00101010011000110101 \]
...
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, \( x^{(1)} \) or \( x^{(2)} \)?
In Lecture 2, to diagnose what was wrong with a model, we plotted prediction errors:

\[ \text{temp} \approx \alpha + \beta \sin(2\pi(t + \phi)) \]
\[ \text{temp} = \alpha + \beta \sin(2\pi(t + \phi)) + \epsilon \]

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 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.

The actual $t = 19$.
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

\[ \text{data} = 111111111111110111 \]

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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Ans EM</th>
<th>Ans $F_1$</th>
<th>Sup EM</th>
<th>Sup $F_1$</th>
<th>Joint EM</th>
<th>Joint $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline [53]</td>
<td>45.60</td>
<td>59.02</td>
<td>20.32</td>
<td>64.49</td>
<td>10.83</td>
<td>40.16</td>
</tr>
<tr>
<td>DecompRC [29]</td>
<td>55.20</td>
<td>69.63</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>QFE [30]</td>
<td>53.86</td>
<td>68.06</td>
<td>57.75</td>
<td>84.49</td>
<td>34.63</td>
<td>59.61</td>
</tr>
<tr>
<td>DFGN [36]</td>
<td>56.31</td>
<td>69.69</td>
<td>51.50</td>
<td>81.62</td>
<td>33.62</td>
<td>59.82</td>
</tr>
<tr>
<td>SAE [45]</td>
<td>60.36</td>
<td>73.58</td>
<td>56.93</td>
<td>84.63</td>
<td>38.81</td>
<td>64.96</td>
</tr>
<tr>
<td>SAE-large</td>
<td>66.92</td>
<td>79.62</td>
<td>61.53</td>
<td>86.86</td>
<td>45.36</td>
<td>71.45</td>
</tr>
<tr>
<td>HGN [14] (+hyperlink)</td>
<td>66.07</td>
<td>79.36</td>
<td>60.33</td>
<td>87.33</td>
<td>43.57</td>
<td>71.03</td>
</tr>
<tr>
<td>HGN-large (+hyperlink)</td>
<td>69.22</td>
<td>82.19</td>
<td>62.76</td>
<td>88.47</td>
<td>47.11</td>
<td>74.21</td>
</tr>
</tbody>
</table>

**BERT (sliding window) variants**

<table>
<thead>
<tr>
<th>Model</th>
<th>Ans EM</th>
<th>Ans $F_1$</th>
<th>Sup EM</th>
<th>Sup $F_1$</th>
<th>Joint EM</th>
<th>Joint $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT Plus</td>
<td>55.84</td>
<td>69.76</td>
<td>42.88</td>
<td>80.74</td>
<td>27.13</td>
<td>58.23</td>
</tr>
<tr>
<td>LQR-net + BERT</td>
<td>57.20</td>
<td>70.66</td>
<td>50.20</td>
<td>82.42</td>
<td>31.18</td>
<td>59.99</td>
</tr>
<tr>
<td>GRN + BERT</td>
<td>55.12</td>
<td>68.98</td>
<td>52.55</td>
<td>84.06</td>
<td>32.88</td>
<td>60.31</td>
</tr>
<tr>
<td>EPS + BERT</td>
<td>60.13</td>
<td>73.31</td>
<td>52.55</td>
<td>83.20</td>
<td>35.40</td>
<td>63.41</td>
</tr>
<tr>
<td>LQR-net 2 + BERT</td>
<td>60.20</td>
<td>73.78</td>
<td>56.21</td>
<td>84.09</td>
<td>36.56</td>
<td>63.68</td>
</tr>
<tr>
<td>P-BERT</td>
<td>61.18</td>
<td>74.16</td>
<td>51.38</td>
<td>82.76</td>
<td>35.42</td>
<td>63.79</td>
</tr>
<tr>
<td>EPS + BERT (large)</td>
<td>63.29</td>
<td>76.36</td>
<td>58.25</td>
<td>85.60</td>
<td>41.39</td>
<td>67.92</td>
</tr>
<tr>
<td>CogLTX</td>
<td>65.09</td>
<td>78.72</td>
<td>56.15</td>
<td>85.78</td>
<td>39.12</td>
<td>69.21</td>
</tr>
<tr>
<td>– multi-step reasoning</td>
<td>62.00</td>
<td>75.39</td>
<td>51.74</td>
<td>83.10</td>
<td>35.85</td>
<td>65.35</td>
</tr>
<tr>
<td>– rehearsal &amp; decay</td>
<td>61.44</td>
<td>74.99</td>
<td>7.74</td>
<td>47.37</td>
<td>5.36</td>
<td>37.74</td>
</tr>
<tr>
<td>– train-test matching</td>
<td>63.20</td>
<td>77.21</td>
<td>52.57</td>
<td>84.21</td>
<td>36.11</td>
<td>66.90</td>
</tr>
</tbody>
</table>

**Results.** Table 2 shows that CogLTX outperforms most of previous methods and all 7 BERT variants solutions on the leaderboard. 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.
"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,
\[
\text{fraction classified correctly} = 93.7\%
\]
“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 is more important to have beauty in one’s equations than to have them fit experiment.”

*Paul Dirac*

“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*
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.)
The Land of Random Variables

WHAT is my model?
HOW do I find the max likelihood fit?

A map of machine learning