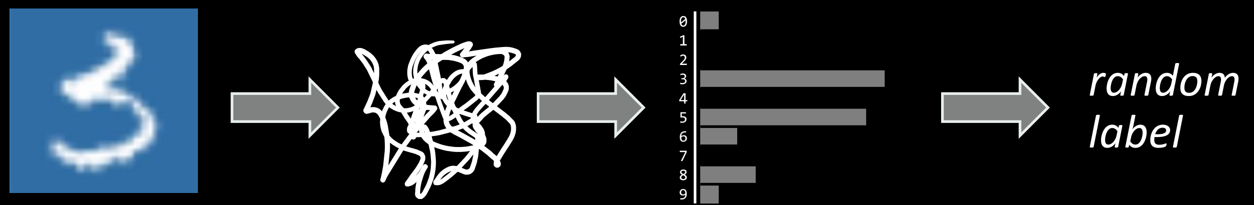
Supervised learning

- Labelled data
- Learn to predict the label

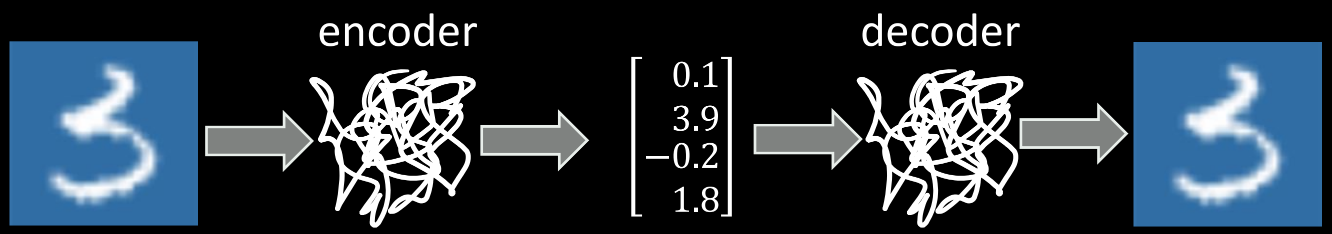


Generative modelling

- Unlabelled data
- Learn to generate new values, similar to those in the dataset

Autoencoder

- Unlabelled data
- Learn to reconstruct values, with a "low-dimensional bottleneck"

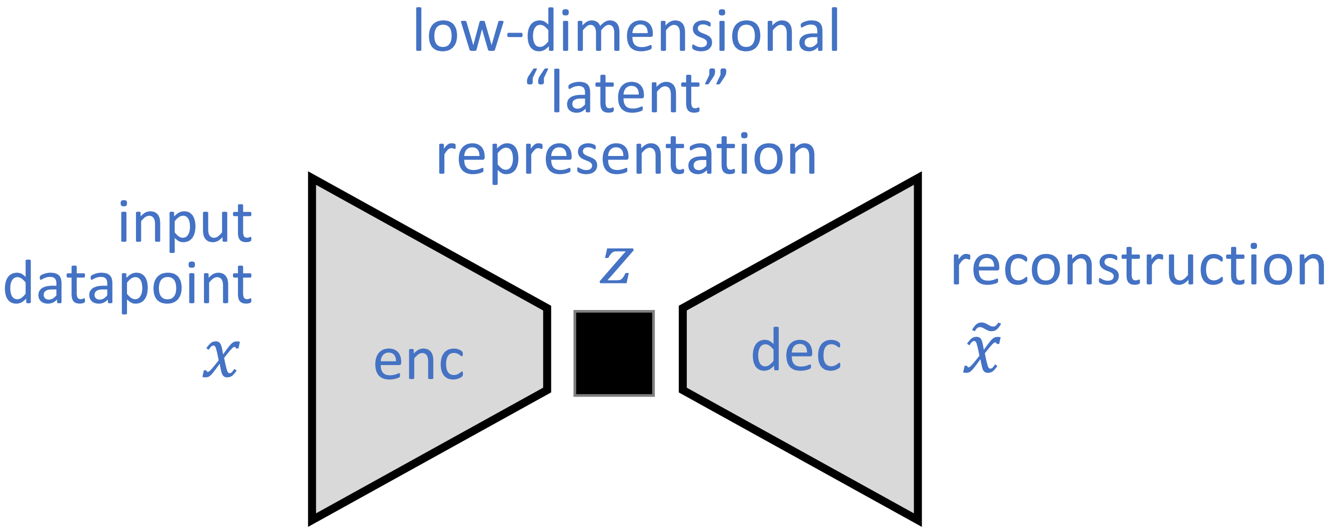


What on earth is the point of training a neural network to simply reproduce its input? Isn't that a simple task?

It's not a simple task, *if* we force it to go via a low-dimensional bottleneck.

This low-dimensional variable will have to contain all the information that's needed to reconstruct the input. Therefore, surely, it will have to capture *only the essential features* of the input.

We call it a "latent representation" of the input. The word "latent" means "hidden". It's hidden from us, and we have to learn what it should be.

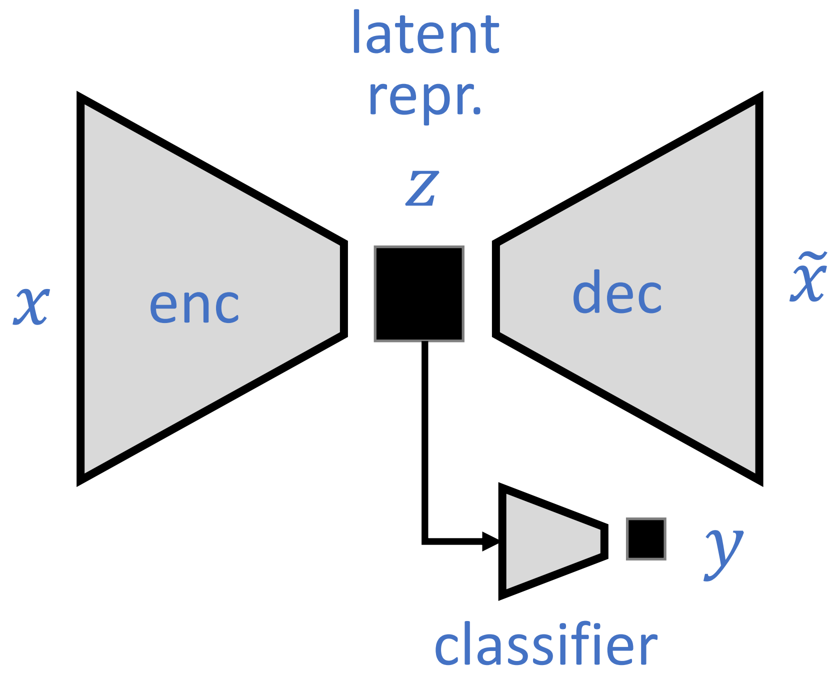


What can we do with a low-dimensional representation?

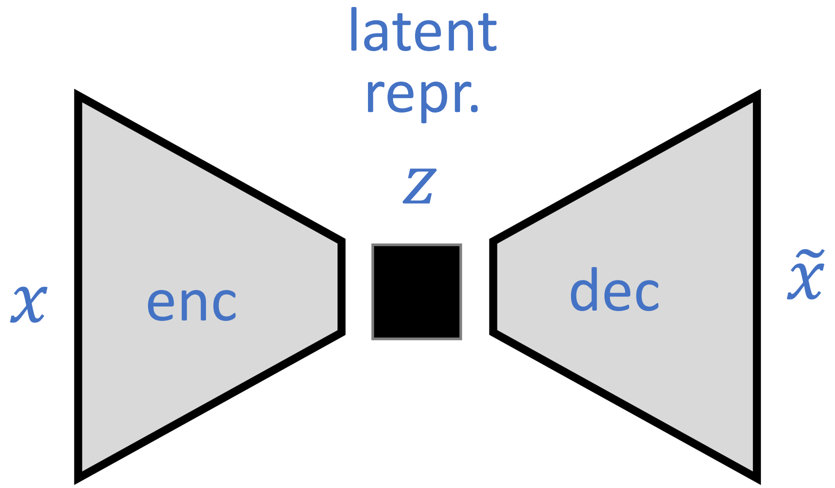
Use case 1: it can make it easier to train a classifier.

- Suppose we have lots of unlabelled data, and only a little bit of labelled data, and we want to train a classifier.
- We can train an autoencoder on unlabelled data. We have lots of data, so this should be easy. We'll learn to encode each datapoint x_i into a low-dimensional representation z_i.
- Now, train a classifier to predict the label y_i from z_i. This should be easier than training a full classifier from scratch, since z_i has already been condensed into only the essential features. Thus, we shouldn't need very much labelled data to train the classifier.

- This method is also useful for fully labelled data, if the labels have only a little bit of information, e.g. sentiment classification of text. If we tried to train using only the labels, it might take a gigantic amount of data for the network to learn what features are useful.

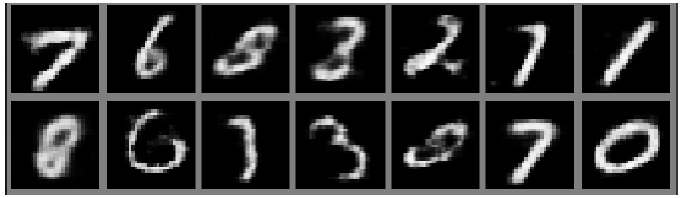


What can we do with a low-dimensional representation?



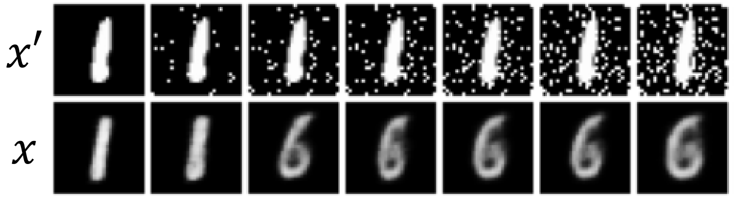
Use case 2: it's a good way to build a generator.

- Ignore the encoder, and simply generate novel outputs by creating random Z and feeding it into the decoder. If Z really is a good low-dimensional representation, then *every* Z that we might create should be decodable into a decent output.



Use case 3: denoising the input.

- Take a corrupted source image x', encode it to get $z = \mathrm{enc}(x')$, then decode to get $x = \mathrm{dec}(z)$. This should clean up the image, assuming that the encoder has learnt to keep only the important details.



What do we hope the latent representation will contain?

We hope that the low-dimensional latent representation will contain meaningful dimensions, and that we can set each dimension separately and tweak aspects of the datapoint.

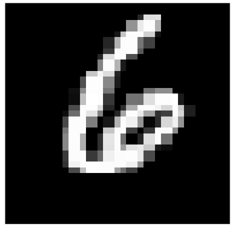
Use case 4: smooth interpolation.

- Take two source images x_1 and x_2, and generate a new image x^* by

$$z_1 = \mathrm{enc}(x_1)$$
$$z_2 = \mathrm{enc}(x_2)$$
$$x^* = \mathrm{dec}(0.5z_1 + 0.5z_2)$$

This should generate a smooth interpolation between the two inputs, where each intermediate looks "nice".

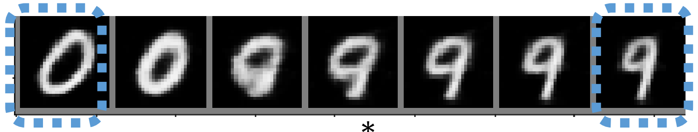
MNIST image



A 4-dimensional representation

```
{'digit': 6,
 'slant': UPRIGHT,
 'weight': MEDIUM,
 'style': LOOSE}
```

x_1



x_2

x^*

- Pick a random z, and
  This should let us synthes
  images.

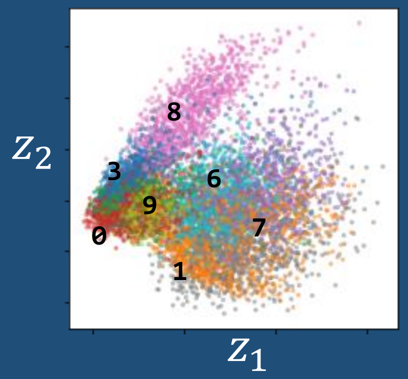
- Take two source ima
  $$z_1 = \mathrm{enc}(x_1)$$
  $$z_2 = \mathrm{enc}(x_2)$$
  $$x^* = \mathrm{dec}(0.5z_1 +$$
  This should generate a s
  interpolation between th
  where each intermediate

- Take a corrupted sou
  encode it to get $z =$
  decode to get $x = \mathrm{dec}(z)$.
  This should clean up the image, assuming z only contains relevant details.

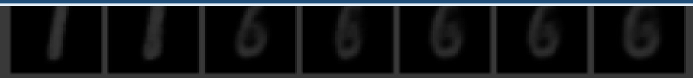
2

The dream of autoencoding:

Neural networks can learn **meaningful representations** of their inputs.

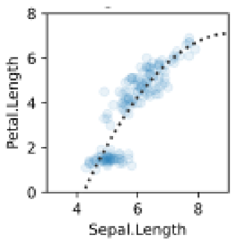


But nothing comes easy ...

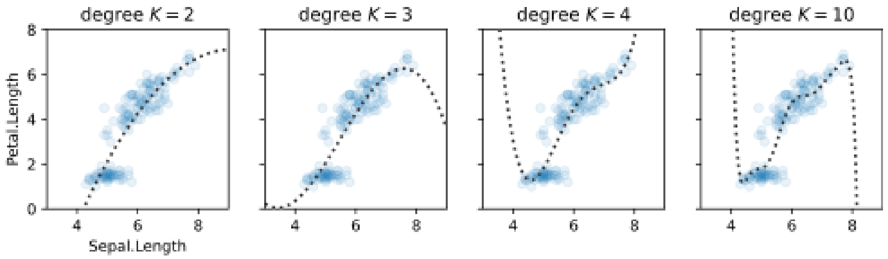
x


SECTION 10.1. THE CURSE OF OVERFITTING for supervised learning

NON-LINEAR RESPONSE



$$\text{Petal.Length} \approx \alpha + \beta\ \text{Sepal.Length} + \gamma(\text{Sepal.Length})^2$$



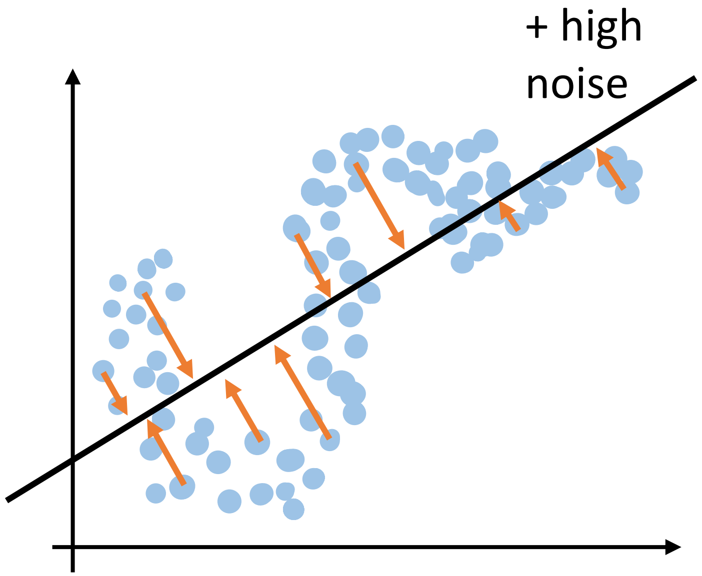
$$\text{Petal.Length} \approx \beta_0 + \sum_{k=1}^{K} \beta_k (\text{Sepal.Length})^k$$

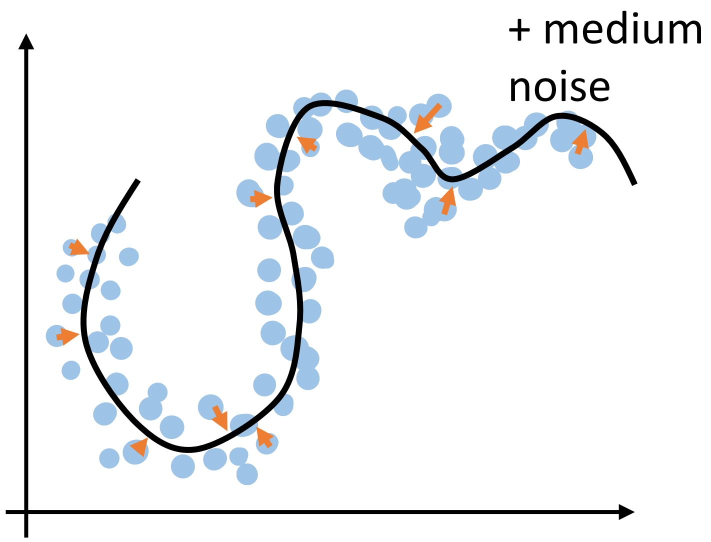
If our model is too rich (too many parameters, too many layers), it will overfit the training data.

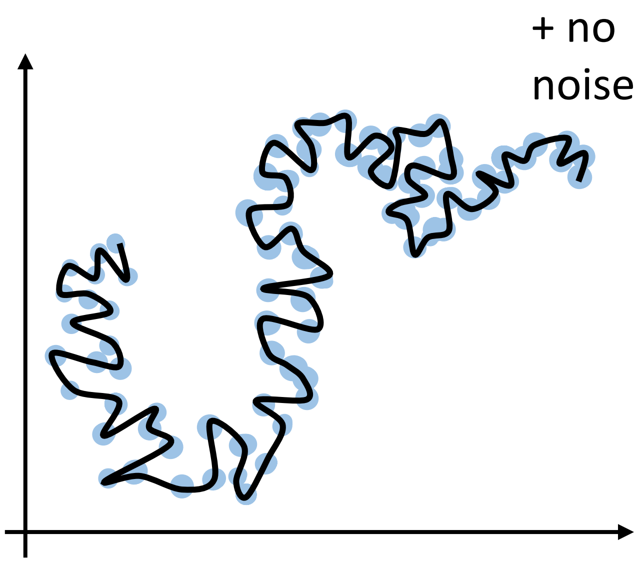
And then it will perform badly on new data.

SECTION 10.1. THE CURSE OF OVERFITTING for generative models

Suppose we have a dataset of points in $\mathbb{R}^2$, and we want to learn a generative model of the form $X = f(Z) + \text{noise}$.

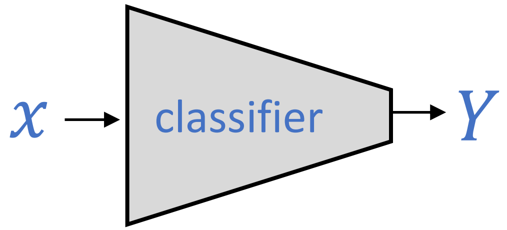






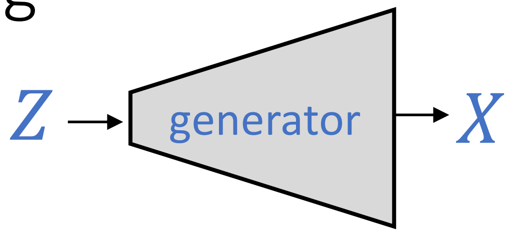
If our model is too rich, it can learn to overfit the training data. It'll probably be an unhelpful model.

Supervised learning



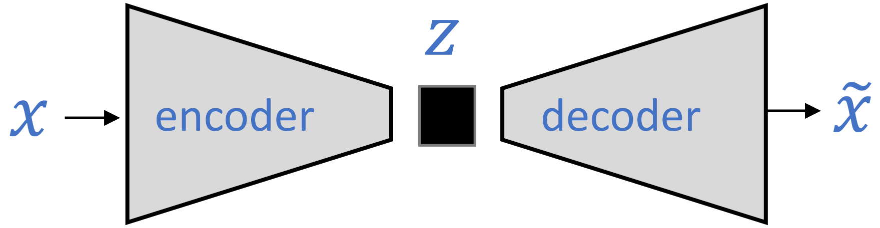
If the classifier neural network is too rich,
then our model will overfit

Generative modelling



If the generator neural network is too rich,
then our model will overfit

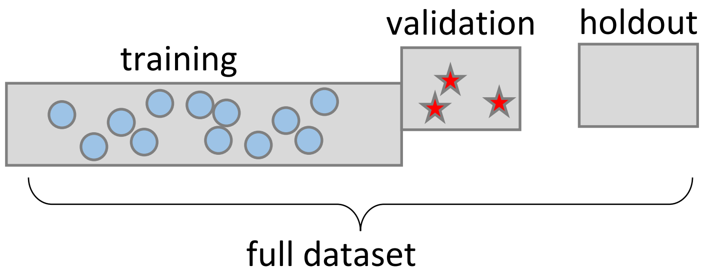
Autoencoder

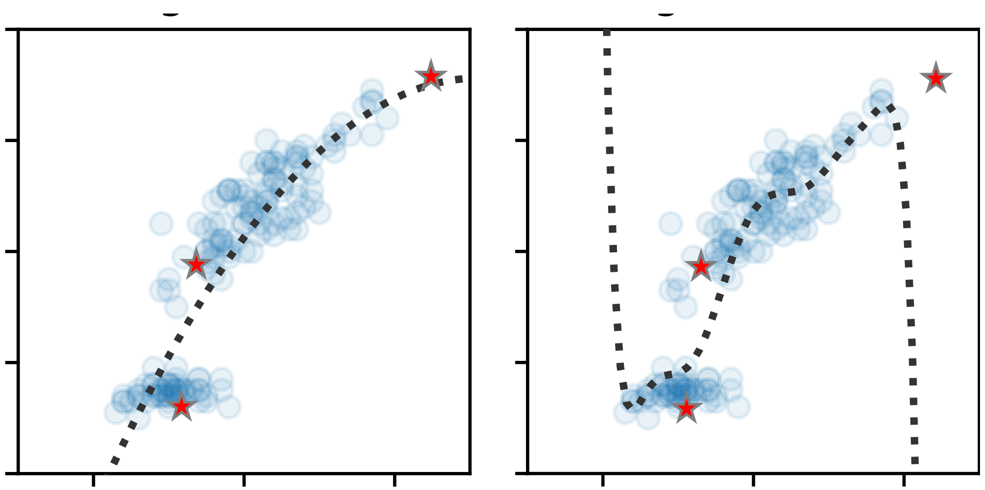


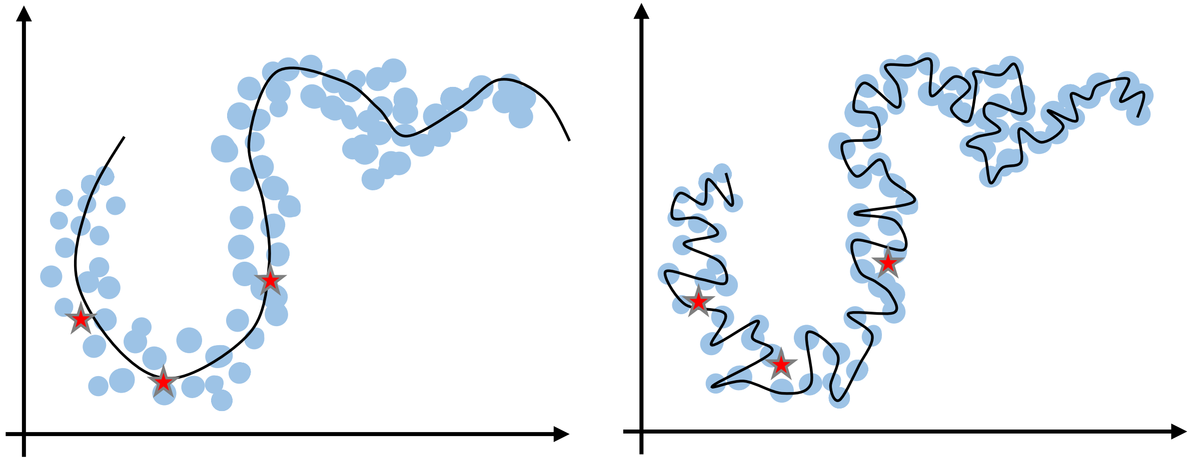
QUESTION. If we trained a very rich encoder and decoder, what would they learn?

We should test our model on a validation set, and tune our model's complexity so that it does well on this set.

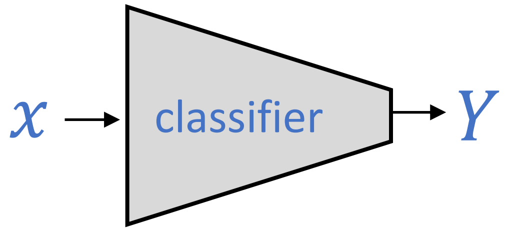
If it does well on validation, it'll likely do well on holdout data.







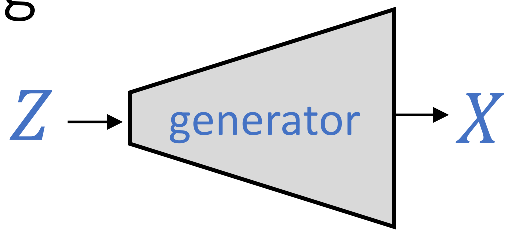
Supervised learning



If the classifier neural network is too rich,
then our model will overfit

and do badly on the validation set, so we can learn to avoid overfitting

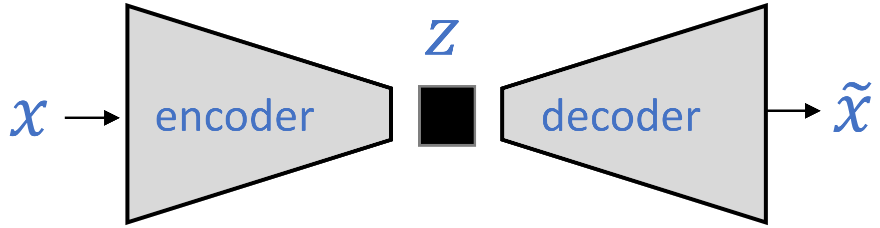
Generative modelling



If the generator neural network is too rich,
then our model will overfit

and do badly on the validation set, so we can learn to avoid overfitting

Autoencoder

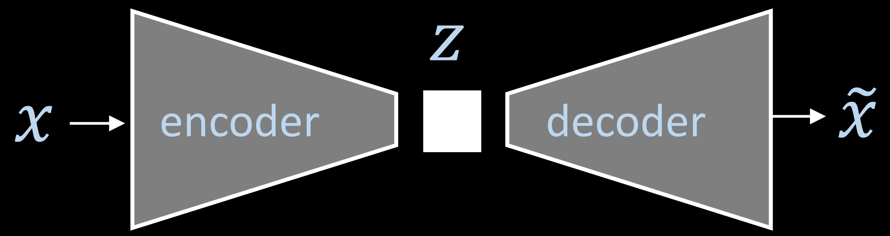


If the neural networks are too rich, then they will learn to encode x perfectly in z, which would be useless

but it'd still score perfectly on validation!

If we simply train an autoencoder to reconstruct its input, it won't learn a useful representation.

What's a better training objective?
What's a better way to think of autoencoders?



SECTION 6.4.

Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

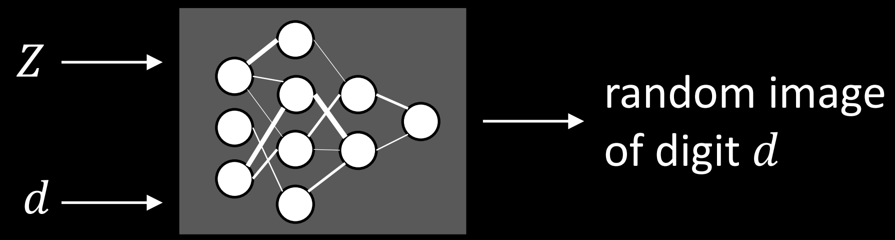
How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

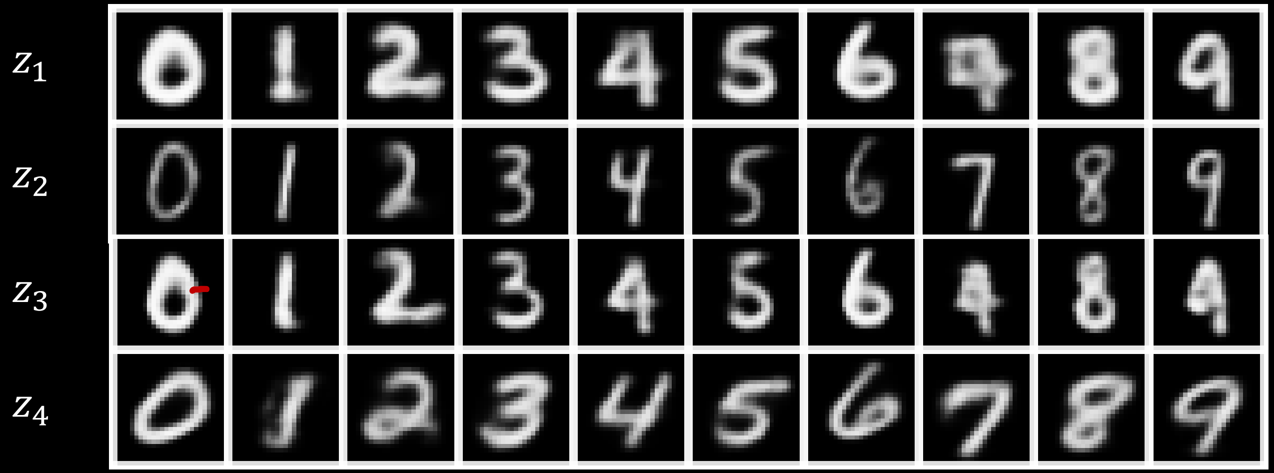
The solution:

Don’t try to build an autoencoder.
Instead, just build a better generator –
and the encoder will come “for free”.

SECTION 6.4. WARNING: MASTERS-LEVEL MATHS

In the Advanced Coursework, you will be asked to build a neural network for generating a *font* of handwritten digits. For this sort of creative extension, we need to understand deeply the maths of the variational autoencoder.





Brain teaser

Let $X \sim \mathrm{Bin}(n=2, p=0.9)$. What is $\Pr_X(X)$?

X is a random variable , $X \sim \mathrm{Bin}(2, 0.9)$

$\mathbb{P}(X=0) = 0.01$

$\mathbb{P}(X=1) = 0.18$

$\mathbb{P}(X=2) = 0.81$

$$X = \begin{cases} 0 & \text{with prob. } 0.01 \\ 1 & \text{with prob. } 0.18 \\ 2 & \text{with prob. } 0.81 \end{cases}$$

$\Pr_X(0) = 0.01$

$\Pr_X(1) = 0.18$

$\Pr_X(2) = 0.81$

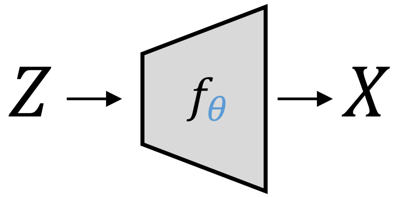
$$\underbrace{\Pr_X}_{f}(X) = f(X) = \begin{cases} 0.01 & \text{with prob } 0.01 \\ 0.18 & \text{with prob } 0.18 \\ 0.81 & \text{with prob. } 0.81 \end{cases}$$

Recall: latent-variable generative modelling (SECTION 3.4)

I have a collection of datapoints in $\mathbb{R}^d$, $x_1, \dots, x_n$.

Q. How might I model this dataset?

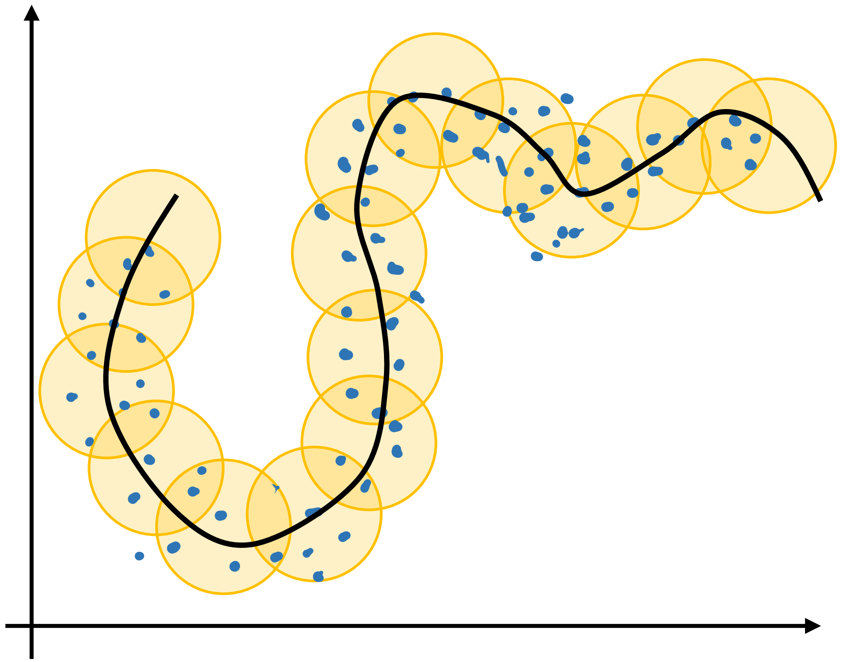
A. Model the datapoints as samples from
$X \sim N(f_\theta(Z), \sigma^2)$ where $Z \sim N(0,1)$



Z measures distance along the line

$f_\theta(Z)$ specifies the shape of the line

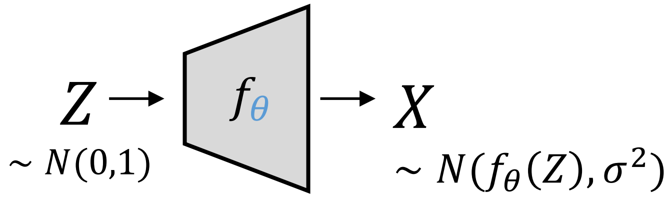
σ is noise around the line



Q. How should I learn the parameters θ and σ?

A. Fit the model, i.e. choose θ and σ to maximize the log likelihood of the dataset

$$\log \operatorname{lik}(\text{data}; \theta, \sigma) = \frac{1}{n} \sum_{i=1}^{n} \log \Pr_X(x_i; \theta, \sigma)$$



$$\log \text{lik}\,(\text{data}) = \sum_{i=1}^{n} \log \Pr_X(x_i)$$

$$= \sum_{i=1}^{n} \log \int_z \overbrace{\Pr_X(x_i|Z=z)}^{h(z)} \Pr_Z(z)dz$$

Law of Total Probability

$$= \sum_{i=1}^{n} \log[\mathbb{E}_{z\sim Z} \overbrace{\Pr_X(x_i|Z=z)}^{h(z)}]$$

rewrite integral as expectation

$$\approx \sum_{i=1}^{n} \log \left[\frac{1}{m} \sum_{j=1}^{m} \Pr_X(x_i|Z=z_j) \right]$$

Monte Carlo approximation, where z_j are sampled from Z

$$\approx \sum_{i=1}^{n} \log \left[\frac{1}{m} \sum_{j=1}^{m} \left\{ \overbrace{\Pr_X(x_i|Z=z_j)}^{h(z_j)} \frac{\Pr_Z(z_j)}{\Pr_{\tilde{Z}}(z_j)} \right\} \right]$$

Importance Sampling approximation, where z_j are sampled from $\tilde{Z}$

$$\approx \sum_{i=1}^{n} \log \left\{ \Pr_X(x_i|Z=z) \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \right\}$$

If $\tilde{Z}$ is well chosen, we can get away with just using a single sample from $\tilde{Z}$

$$= \sum_{i=1}^{n} \left\{ \log \Pr_X(x_i|Z=z) + \log \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \right\}$$

X is a random variable. So it has a likelihood function, $\Pr_X$.

$$\mathbb{E}\, h(Z) \approx \frac{1}{m} \sum_{j=1}^{m} h(z_j)$$

where z_j sampled from Z

Recall: importance sampling (SECTION 6.3)

Let Z be a random variable, let h be a real-valued function, and let $\tilde{Z}$ be any distribution. Then, if we sample $z_1, \ldots, z_m$ from $\tilde{Z}$,

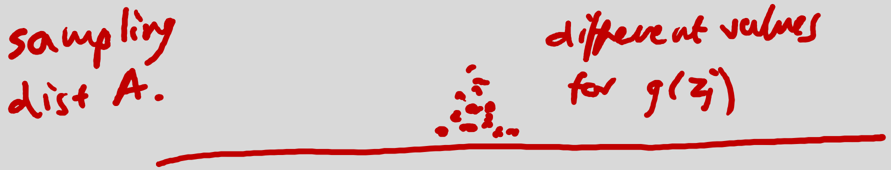
$$\mathbb{E}h(Z) \approx \frac{1}{m}\sum_{j=1}^{m} \underbrace{h(z_j)\frac{\Pr_Z(z_j)}{\Pr_{\tilde{Z}}(z_j)}}_{g(z_j)}$$

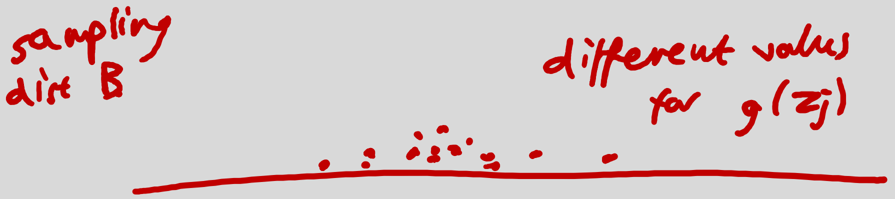
This works for *any* sampling distribution $\tilde{Z}$.

But it will only be useful if we choose a sensible sampling distribution!

The more samples we take, the better the approximation should be.

QUESTION. How could we choose the sampling distribution $\tilde{Z}$ so that we don't need very many samples?





we want $g(z) \approx$ const.

$$\Rightarrow \frac{h(z)\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \approx \text{const}$$

$$\Rightarrow \Pr_{\tilde{Z}}(z) \approx \text{const} \times h(z) \times \Pr_Z(z)$$

Recall: importance sampling (SECTION 6.3)

Let Z be a random variable, let h be a real-valued function, and let $\tilde{Z}$ be any distribution. Then, if we sample $z_1, \dots, z_m$ from $\tilde{Z}$,

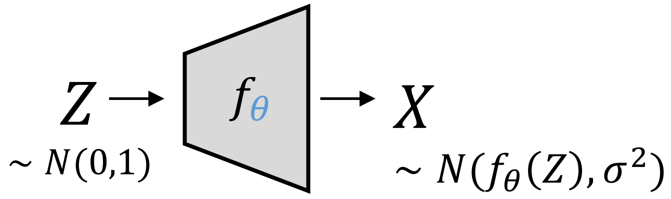
$$\mathbb{E}h(Z) \approx \frac{1}{m}\sum_{j=1}^{m} h(z_j)\frac{\Pr_Z(z_j)}{\Pr_{\tilde{Z}}(z_j)}$$

This works for *any* sampling distribution $\tilde{Z}$.

But it will only be useful if we choose a sensible sampling distribution!

If we choose $\tilde{Z}$ so that $h(z)\frac{\Pr_{Z(z)}}{\Pr_{\tilde{Z}(z)}}$ is roughly constant, then we can get away with just a few samples.

$$h(z)\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \approx \text{const} \quad \Longrightarrow \quad \Pr_{\tilde{Z}}(z) \approx \text{const} \times h(z)\Pr_Z(z)$$



$$\log \text{lik}\,(\text{data}) = \sum_{i=1}^{n} \log \Pr_X(x_i)$$

$$= \sum_{i=1}^{n} \log \int_z \Pr_X(x_i|Z=z)\Pr_Z(z)dz$$

Law of Total Probability

$$= \sum_{i=1}^{n} \log[\mathbb{E}_{z\sim Z} \Pr_X(x_i|Z=z)]$$

rewrite integral as expectation

$$\cancel{\approx \sum_{i=1}^{n} \log\left[\frac{1}{m}\sum_{j=1}^{m} \Pr_X\left(x_i|Z=z_j\right)\right]}$$

Monte Carlo approximation, where z_j are sampled from Z

$$\approx \sum_{i=1}^{n} \log\left[\frac{1}{m}\sum_{j=1}^{m} \left\{\Pr_X\left(x_i|Z=z_j\right)\frac{\Pr_Z(z_j)}{\Pr_{\tilde{Z}}(z_j)}\right\}\right]$$

Importance Sampling approximation, where z_j are sampled from $\tilde{Z}$

$$\approx \sum_{i=1}^{n} \log\left\{\Pr_X(x_i|Z=z)\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)}\right\}$$

If $\tilde{Z}$ is well chosen, we can get away with just using a single sample z from $\tilde{Z}$

$$= \sum_{i=1}^{n} \left\{\log \Pr_X(x_i|Z=z) + \log\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)}\right\}$$

$$\approx \sum_{i=1}^{n} \log\left\{\Pr_X(x_i|Z=z)\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)}\right\}$$

If $\tilde{Z}$ is well chosen, we can get away with just using a single sample z from $\tilde{Z}$

$$= \sum_{i=1}^{n} \left\{\log \Pr_X(x_i|Z=z) + \log\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)}\right\}$$

QUESTION. How should we choose $\tilde{Z}$?

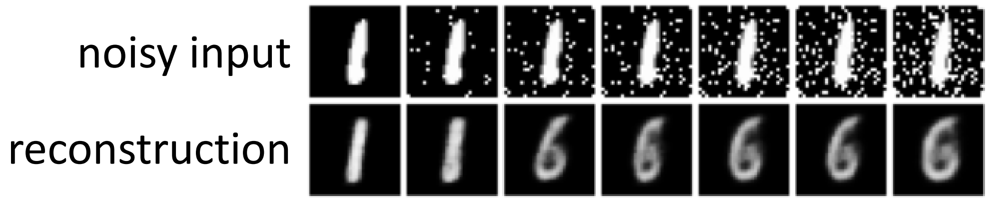
$$\approx \sum_{i=1}^{n} \log\left\{ \Pr_X(x_i|Z=z)\, \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \right\}$$

If $\tilde{Z}$ is well chosen, we can get away with just using a single sample z from $\tilde{Z}$

$$= \sum_{i=1}^{n} \left\{ \log \Pr_X(x_i|Z=z) + \log \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \right\}$$

$$= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\Big(x_i - \mathrm{dec}\big(\overbrace{\mathrm{enc}(x_i) + \mathrm{noise}}^{z}\big)\Big)^2$$

This term measures how well our networks can reconstruct a noisy input.

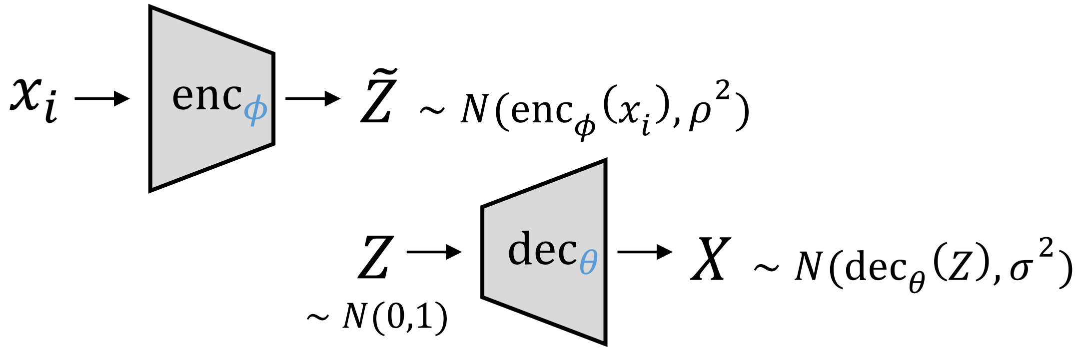


We should ideally choose $\tilde{Z}$ so that $\boxed{\Pr_{\tilde{Z}}(z) \approx \text{const} \times \Pr_X(x_i|Z=z)\, \Pr_Z(z).}$

But it's really tricky to find const and if we could it's still tricky to sample from this ideal distribution…

So let's use a neural network to choose its own sampling distribution!

(We just need to make sure that the network is given x_i as an input.)



```python
class BernoulliImageGenerator(nn.Module):
    def __init__(self, d=4):
        self.p = nn.Sequential( … )

    def loglik(self, x, z):
        xr = self.p(z)
        return (x*torch.log(xr) + (1-x)*torch.log(1-xr)).sum((1,2,3))

class GaussianEncoder(nn.Module):
    def __init__(self, decoder):
        self.dec = decoder
        self.enc = nn.Sequential( … )

    def forward(self, x):
        μτ = self.enc(x)
        μ,τ = μτ[:,:self.decoder.d], μτ[:,self.decoder.d:]
        return μ, torch.exp(τ/2)

    def loglik_lb(self, x):
        μ,ρ = self(x)
        kl = 0.5 * (μ**2 + ρ**2 - torch.log(ρ**2) - 1).sum(1)
        ε = torch.randn_like(ρ)
        ll = self.decode.loglik(x, z=μ+ρ*ε)
        return ll – kl
```

$$\underset{\mathbb{R}^d}{z} \to \boxed{\text{dec}} \to \underset{\mathbb{R}^{28\times 28}}{p} \longrightarrow \underset{\mathbb{R}^{28\times 28}}{X \sim \text{Bin}(1,p)}$$

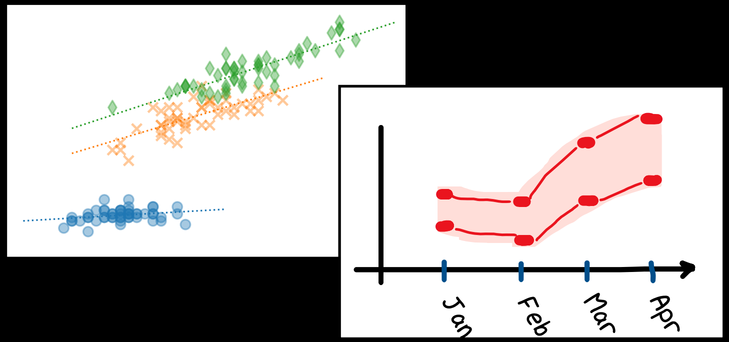
$$\underset{\mathbb{R}^{28\times 28}}{x} \to \boxed{\text{enc}} \to \underset{\mathbb{R}^{4}}{\mu,\rho} \longrightarrow \underset{\mathbb{R}^{4}}{\tilde{Z} \sim N(\mu,\rho^2)}$$

$$\log \Pr_X(x)$$
$$\geq \log \Pr_X(x|Z=z) + \mathbb{E}_{Z\sim\tilde{Z}} \log \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)}$$

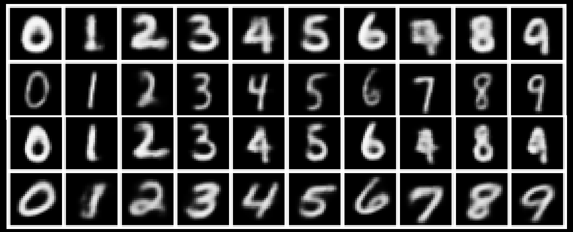
See sections 6.4 and 6.5 of lecture notes for more details.

```python
dataset = ...
model = GaussianEncoder(BernoulliImageGenerator(d=4))
optimizer = optim.Adam(model.parameters())

for epoch in range(10):
    for imgs in batched(dataset):
        optimizer.zero_grad()
        loglik_lb = torch.mean(model.loglik_lb(imgs))
        (-loglik_lb).backward()
        optimizer.step()
```



- Exam on 10 August, open book
  - — linear models [lecture 2]
  - — confidence ribbon [lecture 3]
  - — fitting a sequence model [lecture 5]
  - — hypothesis testing [lecture 5]
- Presentation on 16 August, group work
  - — inventing models [lecture 2]
  - — Markov chain calculations [lecture 4]
- Advanced coursework, not assessed
  - — variational autoencoder [lecture 4]



随机 suíjī

隨機 (traditional)

隨
comply,
vary according to

遀
follow

辶
move

機
machine

幾
attend to
subtle things

𢆶
little things

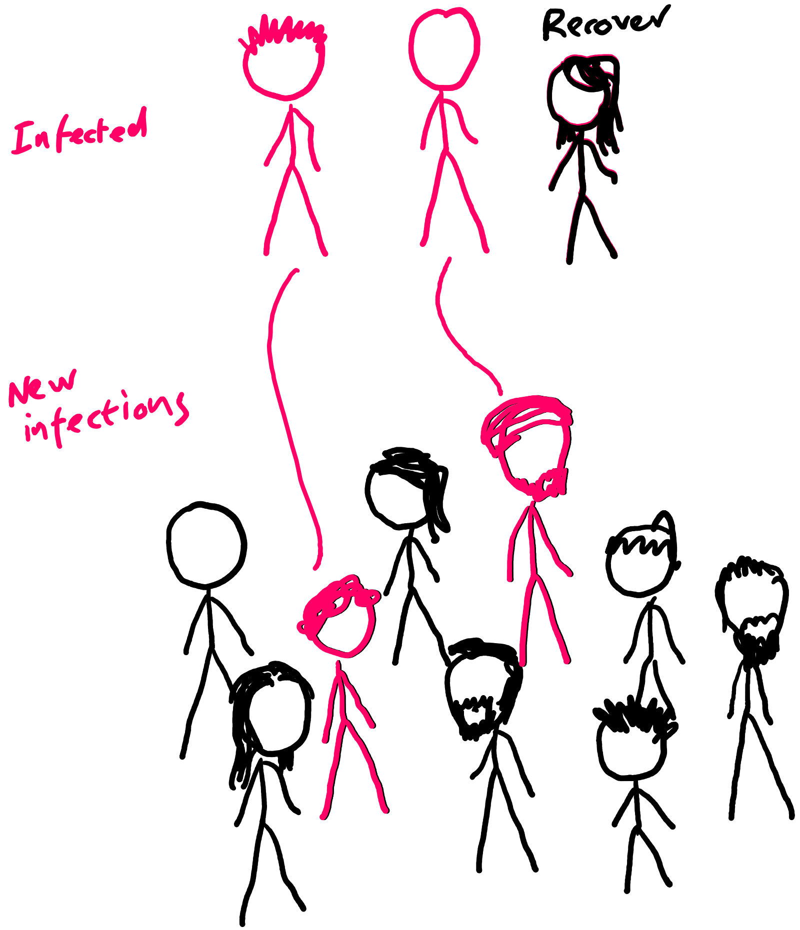
戍
guarding

Example 12.1.2: epidemic model

Let $X_n \in \mathbb{N}$ be the number of infected people on day n, and let it evolve according to

$$X_{n+1} = X_n - \text{Recoveries}_n + \text{Infections}_n$$

Day 2: #infected = 3 + 2 - 1 = 4

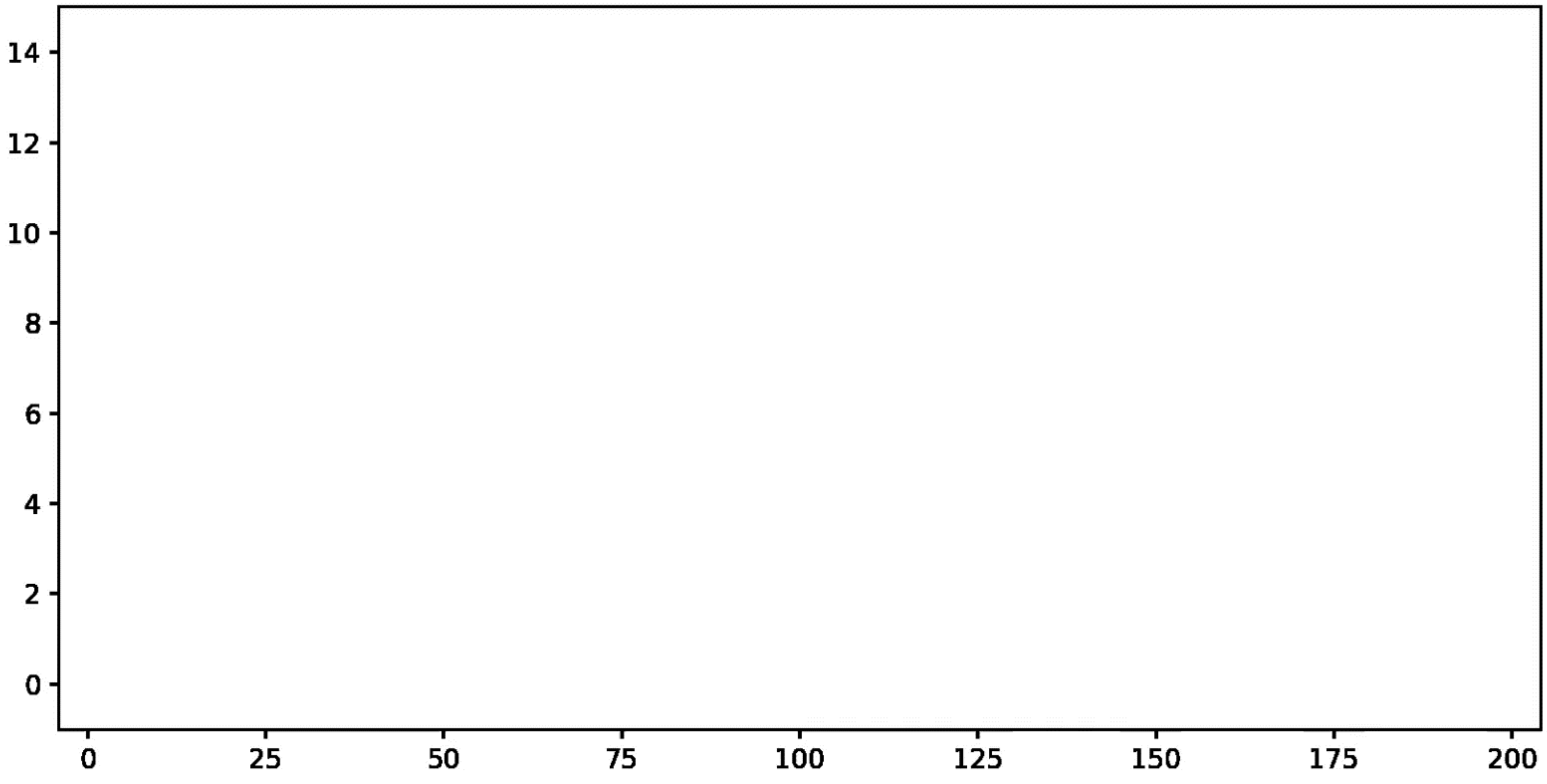


Example 12.1.2: epidemic model

Let $X_n \in \mathbb{N}$ be the number of infected people on day n, and let it evolve according to

$$X_{n+1} = X_n - \text{Recoveries}_n + \text{Infections}_n$$

num. infected X_n
(5 simulation runs)

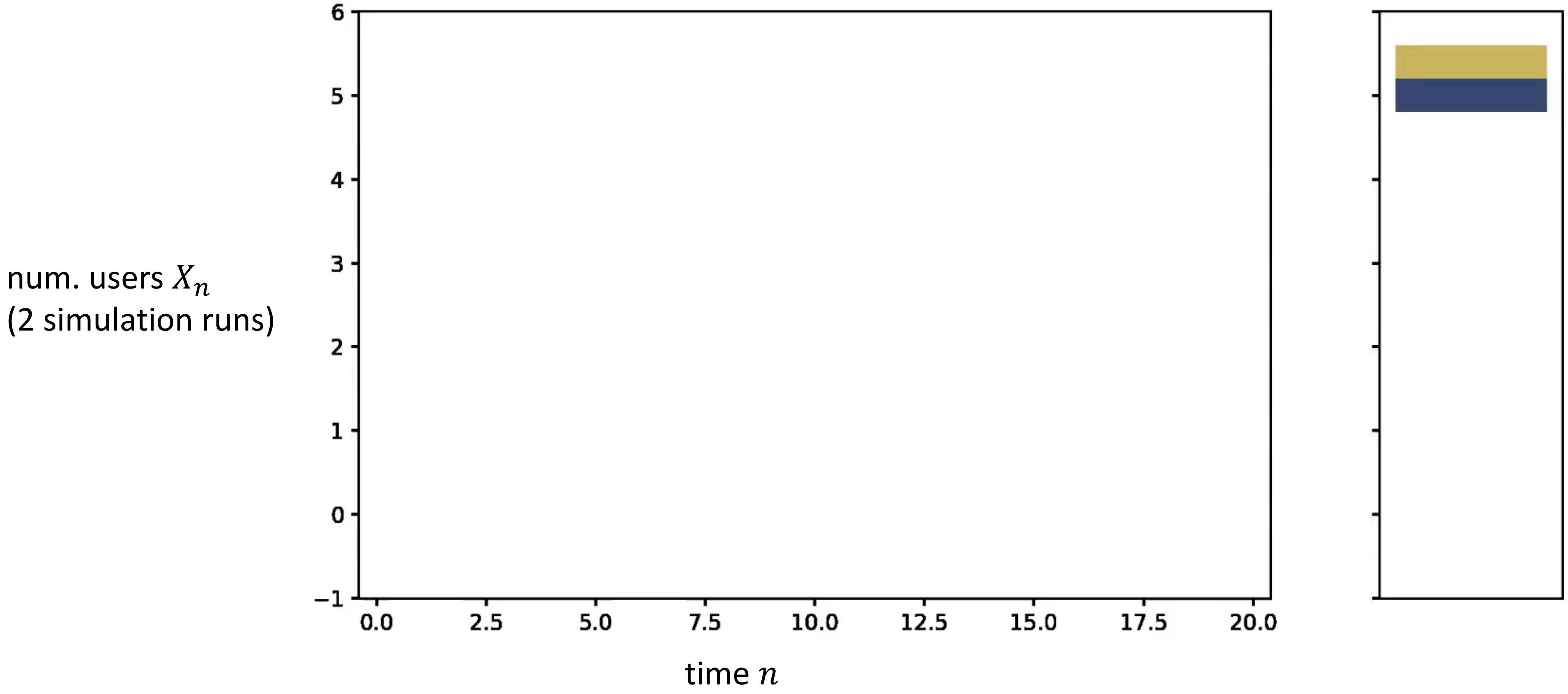


time n

Example 12.1.3 (active users)

Let $X_n \in \mathbb{N}$ be the number of users currently using an online platform at timestep n, and let it evolve according to

$$X_{n+1} = X_n + \text{Newusers}_n - \text{Departures}_n$$



SECTION 12

Random process:
any system whose state changes over time, with probabilistic dynamics.

$X_0, X_1, X_2, \dots$

Famous applications:
ChatGPT, CoPilot

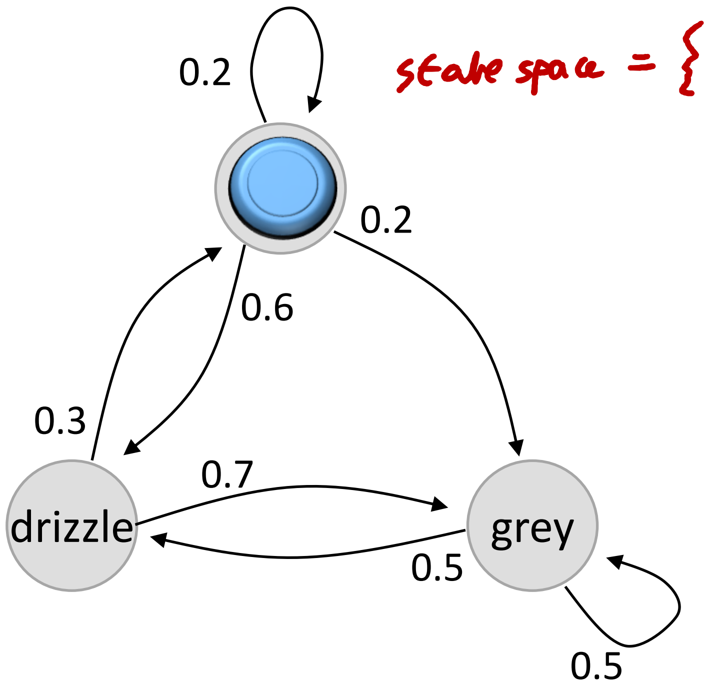
Markov chain:
a random process in which each X_i is generated based **only** on the preceding state X_{i-1}.

$X_0 \to X_1 \to X_2 \to \cdots$

Famous applications:
epidemic modelling, queueing theory,
stock market, Google PageRank, Manhattan Project

SECTION 12.1. Three ways to specify a Markov chain model

STATE SPACE DIAGRAM



state space = { rain, drizzle, grey }

TRANSITION MATRIX

$$P = \begin{array}{c} \\ \text{rain} \\ \text{drizzle} \\ \text{grey} \end{array} \begin{array}{c} \begin{array}{ccc} \text{rain} & \text{drizzle} & \text{grey} \end{array} \\ \begin{bmatrix} .2 & .6 & .2 \\ .3 & 0 & .7 \\ 0 & .5 & .5 \end{bmatrix} \end{array}$$

$$P_{ij} = \mathbb{P}\left(\begin{matrix} \text{next state} \\ \text{is } j \end{matrix} \middle| \begin{matrix} \text{in state} \\ i \end{matrix} \right)$$

CAUSAL DIAGRAM

Each X_i is generated based only on the preceding state X_{i-1}:

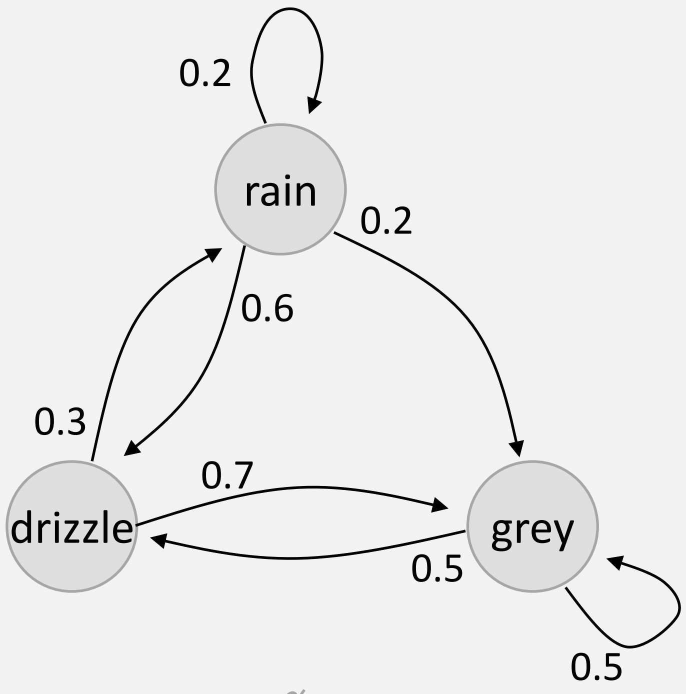
$$X_1 \to X_2 \to X_3 \to \cdots$$

Example 12.2.1
(Multi-step transition probabilities)

If it's grey today, what's the chance of rain two days from now?

MEMORYLESSNESS

$$X_1 \to X_2 \to X_3 \to \cdots$$



$$P = \begin{array}{r} \\ \text{rain} \\ \text{drizzle} \\ \text{grey} \end{array} \begin{array}{c} \begin{array}{ccc} \text{rain} & \text{drizzle} & \text{grey} \end{array} \\ \begin{bmatrix} .2 & .6 & .2 \\ .3 & 0 & .7 \\ 0 & .5 & .5 \end{bmatrix} \end{array}$$

r = rain
g = grey
d = drizzle

$$\mathbb{P}(X_2 = r \mid X_0 = g)$$

$$= \sum_x \mathbb{P}(X_2 = r \mid X_1 = x, X_0 = g)\ \mathbb{P}(X_1 = x \mid X_0 = g)$$

by Law of Total Prob. with baggage

baggage is $X_0 = g$

$$= \sum_x \mathbb{P}(X_2 = r \mid X_1 = x)\ \mathbb{P}(X_1 = x \mid X_0 = g)$$

since X_2 is generated based only on X_1, so the state at time 0 is irrelevant (once we know the state at time 1).

$$= \sum_x P_{xr} P_{gx} = \sum_x P_{gx} P_{xr} = [P \times P]_{gr}$$

Law of Total Probability

$$\mathbb{P}(A = a) = \sum_{b} \mathbb{P}(A = a \mid B = b)\, \mathbb{P}(B = b)$$

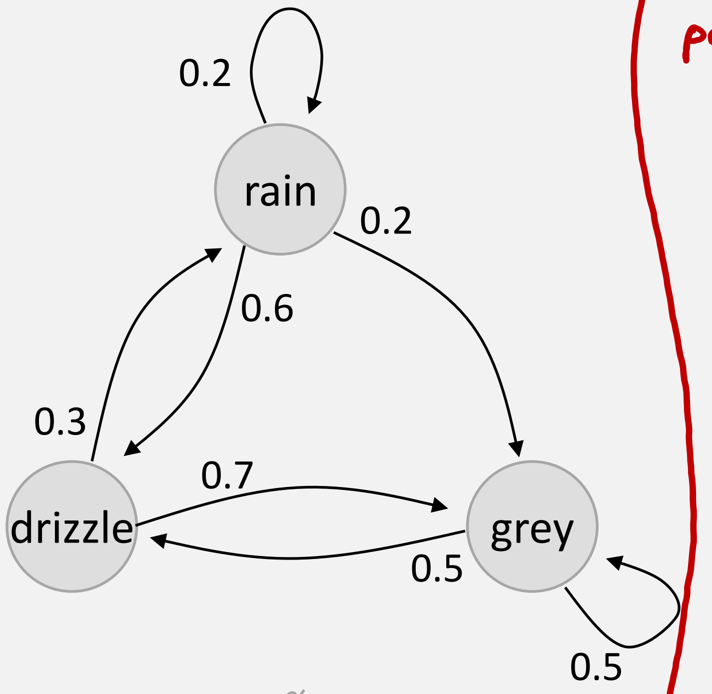
Law of Total Probability with baggage $\{C = c\}$

$$\mathbb{P}(A = a \mid C = c) = \sum_{b} \mathbb{P}(A = a \mid B = b, C = c)\, \mathbb{P}(B = b \mid C = c)$$

Exercise

Given that yesterday was rain, and tomorrow is rain, what's the chance that today is drizzle?

$$X_1 \to X_2 \to X_3 \to \cdots$$



$$P = \begin{array}{c} \\ \text{rain} \\ \text{drizzle} \\ \text{grey} \end{array} \begin{array}{c} \begin{array}{ccc} \text{rain} & \text{drizzle} & \text{grey} \end{array} \\ \begin{bmatrix} .2 & .6 & .2 \\ .3 & 0 & .7 \\ 0 & .5 & .5 \end{bmatrix} \end{array}$$

$X_0 \to ? \to X_2$

$$\mathbb{P}(X_1 = x_1 \mid X_0 = x_0,\ X_2 = x_2)$$

$$= \frac{\mathbb{P}(X_1 = x_1,\ X_0 = x_0,\ X_2 = x_2)}{\mathbb{P}(X_0 = x_0,\ X_2 = x_2)}$$

by definition of conditional probability.

top part:

$$\mathbb{P}(X_2 = x_2,\ X_1 = x_1,\ X_0 = x_0)$$

$$= \mathbb{P}(X_2 = x_2 \mid X_1 = x_1,\ X_0 = x_0)\ \mathbb{P}(X_1 = x_1,\ X_0 = x_0)$$

$$= \mathbb{P}(X_2 = x_2 \mid X_1 = x_1,\ \cancel{X_0 = x_0})\ \mathbb{P}(X_1 = x_1 \mid X_0 = x_0)\ \mathbb{P}(X_0 = x_0)$$

$$= \mathbb{P}(X_0 = x_0)\ P_{x_0 x_1}\ P_{x_1 x_2}$$

bottom part:

$$= \sum_y \mathbb{P}(X_0 = x_0)\ P_{x_0 y}\ P_{y x_2}$$

$$= \frac{P_{x_0 x_1}\ P_{x_1 x_2}}{\sum_y P_{x_0 y}\ P_{y x_2}}$$

Helpful rules for working with Markov chains

Law of Total Probability

$$\mathbb{P}(A = a) = \sum_b \mathbb{P}(A = a \mid B = b)\, \mathbb{P}(B = b)$$

Law of Total Probability with baggage $\{C = c\}$

$$\mathbb{P}(A = a \mid C = c) = \sum_b \mathbb{P}(A = a \mid B = b, C = c)\, \mathbb{P}(B = b \mid C = c)$$

Bayes's rule

$$\mathbb{P}(A = a \mid B = b) = \frac{\mathbb{P}(A = a)\, \mathbb{P}(B = b | A = a)}{\mathbb{P}(B = b)}$$

Bayes's rule with baggage $\{C = c\}$

$$\mathbb{P}(A = a \mid B = b, C = c) = \frac{\mathbb{P}(A = a | C = c)\, \mathbb{P}(B = b | A = a, C = c)}{\mathbb{P}(B = b | C = c)}$$

Definition of independence

If A and B are independent then

$$\mathbb{P}(A = a \mid B = b) = \mathbb{P}(A = a)$$

Definition of conditional independence

If A and B are conditionally independent given $\{C = c\}$ then

$$\mathbb{P}(A = a \mid B = b, C = c) = \mathbb{P}(A = a \mid C = c)$$

Calculating with Markov Chains

The chain is memoryless

$$X_0 \to X_1 \to \cdots$$

i.e. each item is generated based only on the previous item

The most important thing about Markov chains is **memorylessness**.

Whenever we're doing calculations with Markov chains, we have to wrangle our expression into a form where we can use memorylessness (plus the transition probability matrix).

Remember memorylessness as "conditional on the present, the future is independent of the past".

$$\mathbb{P}(\overset{\text{future}}{X_3 = x_3} \mid \overset{\text{present}}{X_2 = x_2}, \overset{\text{past}}{\cancel{X_1 = x_1, X_0 = x_0}}) = \mathbb{P}(X_3 = x_3 \mid X_2 = x_2)$$

$$\mathbb{P}(\overset{\text{future}}{X_3 = x_3} \mid \overset{\text{present}}{X_1 = x_1}, \overset{\text{past}}{X_0 = x_0}) = \mathbb{P}(X_3 = x_3 \mid X_1 = x_1)$$

$$\mathbb{P}(\overset{\text{future}}{X_3 = x_3} \mid \overset{\text{present}}{X_2 = x_2}, \overset{\text{past}}{X_0 = x_0}) = \mathbb{P}(X_3 = x_3 \mid X_2 = x_2)$$

The full sequence $X = [X_0 X_1 X_2 \cdots]$ is a random variable, with a likelihood function,

$$\Pr_X(x_0 x_1 x_2 \cdots x_n) = \mathbb{P}(X_0 = x_0, X_1 = x_1, \cdots, X_n = x_n)$$

We'll need the likelihood for machine learning

Random process:
any system whose state changes over time, with probabilistic dynamics.

$X_0, X_1, X_2, \ldots$

$$\Pr(x_0 x_1 x_2 x_3 \cdots x_n)$$
$$= \mathbb{P}(x_0) \times \mathbb{P}(x_1|x_0) \times \mathbb{P}(x_2|x_1, x_0) \times \mathbb{P}(x_3|x_2, x_1, x_0) \times \cdots$$

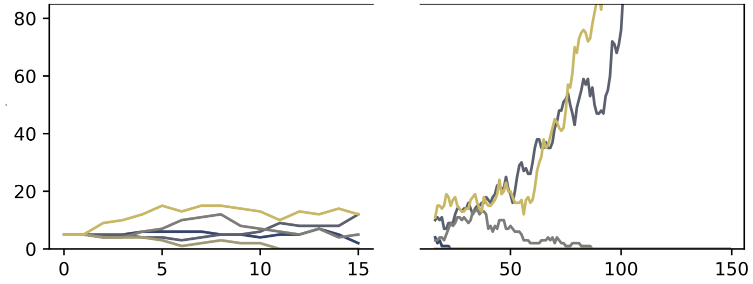
Markov chain:
a random process in which each X_i is generated based **only** on the preceding state X_{i-1}.

$X_0 \to X_1 \to X_2 \to \cdots$

$$\Pr(x_0 x_1 x_2 x_3 \cdots x_n)$$
$$= \mathbb{P}(x_0) \times \mathbb{P}(x_1|x_0) \times \mathbb{P}(x_2|x_1) \times \mathbb{P}(x_3|x_2) \times \cdots$$

SECTION 12.4–12.6. Analysis of Markov chains

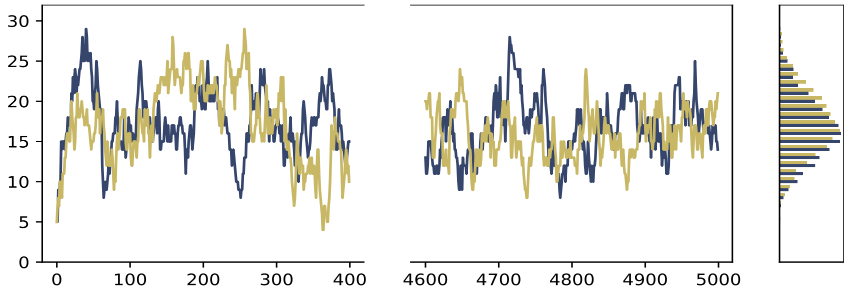
EPIDEMIC MODEL



- How likely is it that the epidemic dies out?
- If it doesn't die out, how does it progress?

How can we learn the growth rate?

ACTIVE USERS MODEL



- What's the average number of active users?

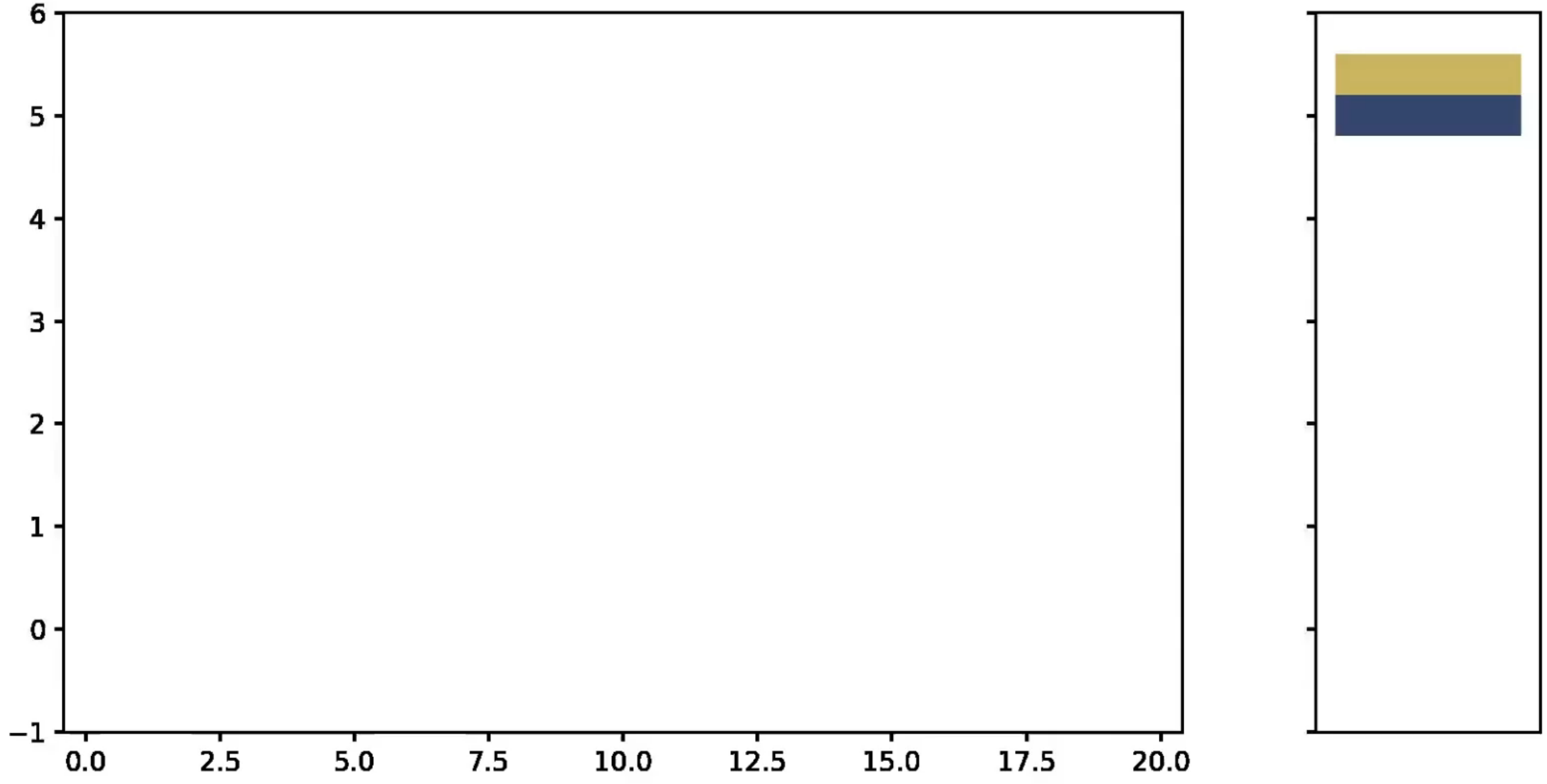
How can we learn this distribution?

Example 12.1.3 (active users)

Let $X_n \in \mathbb{N}$ be the number of users currently using an online platform at timestep n, and let it evolve according to

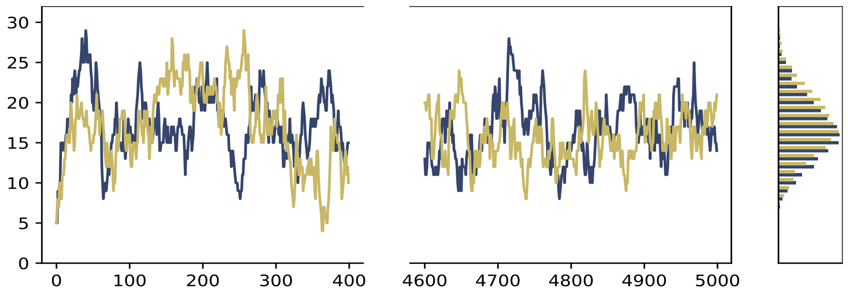
$$X_{n+1} = X_n + \mathrm{Newusers}_n - \mathrm{Departures}_n$$

num. users X_n
(2 simulation runs)



time n

ACTIVE USERS MODEL



It looks like this distribution is **stable** i.e. unchanging over time

Can we find a probability distribution π such that, if $X_0 \sim \pi$, then $X_1 \sim \pi$?
(and so $X_2 \sim \pi$, and $X_3 \sim \pi$, and …)

$X_i \sim \pi$ means:
$\mathbb{P}(X_i = x) = \pi_x$ for all x in the state space

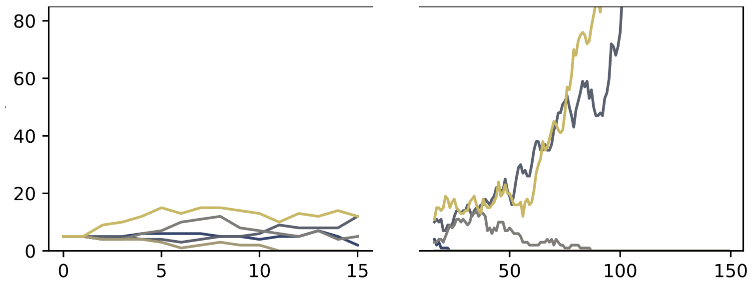
Let's assume $X_0 \sim \pi$, and calculate $\mathbb{P}(X_1 = x)$:

$$\mathbb{P}(X_1 = x) = \sum_{x_0} \mathbb{P}(X_1 = x \mid X_0 = x_0)\, \mathbb{P}(X_0 = x_0)$$
$$= \sum_{x_0} P_{x_0 x}\, \pi_{x_0} = \sum_{x_0} \pi_{x_0}\, P_{x_0 x} = [\pi P]_x$$

Now, if π is a stable distribution, then
$$\mathbb{P}(X_1 = x) = \pi_x \quad \text{for all } x$$
thus

$$\pi = \pi P$$

This is called the *stationarity equation*. It's a simple matrix equation; we can solve it to find the stable distribution.



Drift analysis

… is a nice simple back-of-the-envelope way to get a rough idea of how a Markov chain X_n is likely to behave.

Drift formula: $\delta(x) = \mathbb{E}(X_{n+1} - X_n \mid X_n = x)$

Drift model: solution to $x_{n+1} = x_n + \delta(x_n)$

Simple epidemic (example 12.1.2)

Let $X_n \in \mathbb{N}$ be the number of infected people on day n, and let it evolve according to

$$X_{n+1} = X_n + \mathrm{Poisson}(rX_n/d) - \mathrm{Bin}(X_n, 1/d)$$

Suppose the R-number (r) is > 1. What's the growth rate of the epidemic?

Why these particular distributions? Explained in notes, example 12.1.2.

Drift formula: $\delta(x) = \mathbb{E}(X_{n+1} - X_n \mid X_n = x)$

$$= \mathbb{E}\left[\mathrm{Poisson}\left(\frac{rx}{d}\right) - \mathrm{Bin}\left(x, \frac{1}{d}\right)\right]$$

$$= \frac{rx}{d} - \frac{x}{d}$$

$$= x\left(\frac{r-1}{d}\right)$$

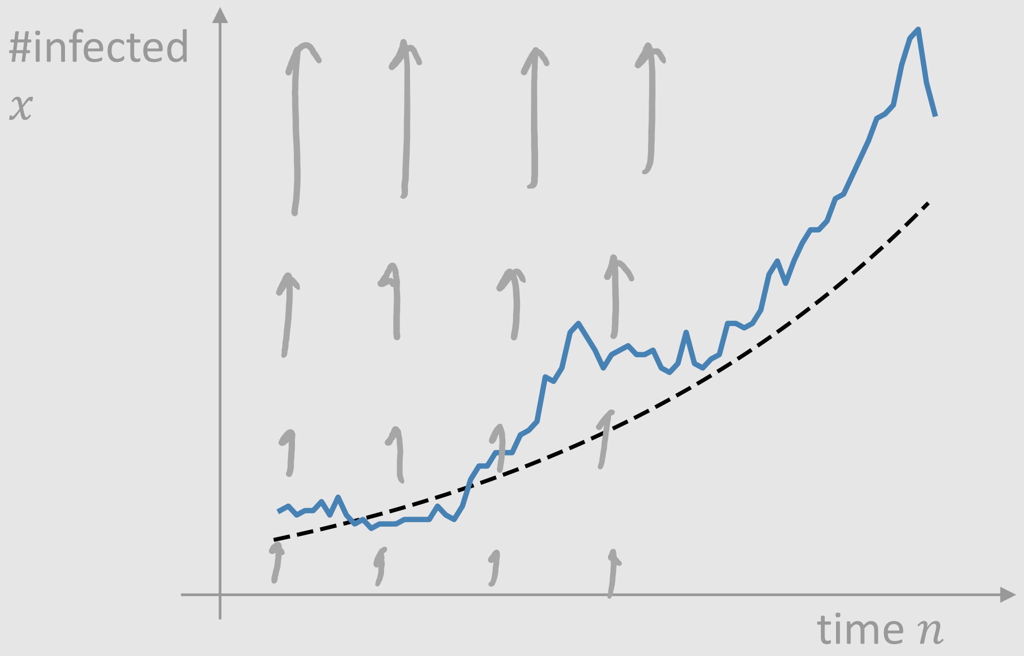
Drift model: solution to $x_{n+1} = x_n + \delta(x_n)$

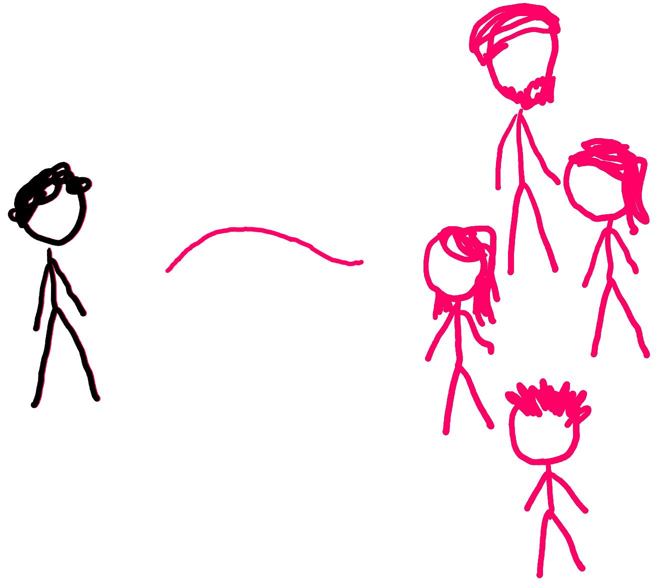
$$x_1 = x_0\left(1 + \frac{r-1}{d}\right)$$

$$x_2 = x_1\left(1 + \frac{r-1}{d}\right) = x_0\left(1 + \frac{r-1}{d}\right)^2$$

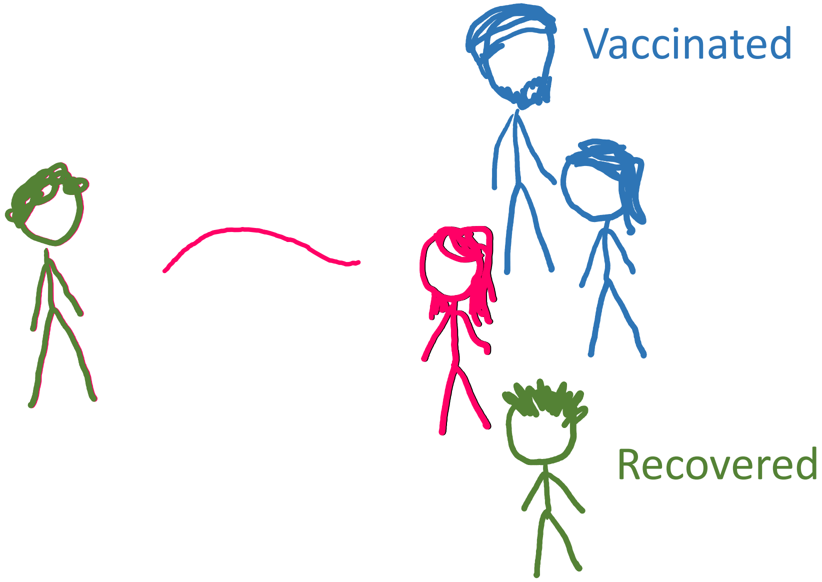
$$\cdots$$

$$x_n = x_0\left(1 + \frac{r-1}{d}\right)^n$$





R=4
each infected person infects
4 others on average

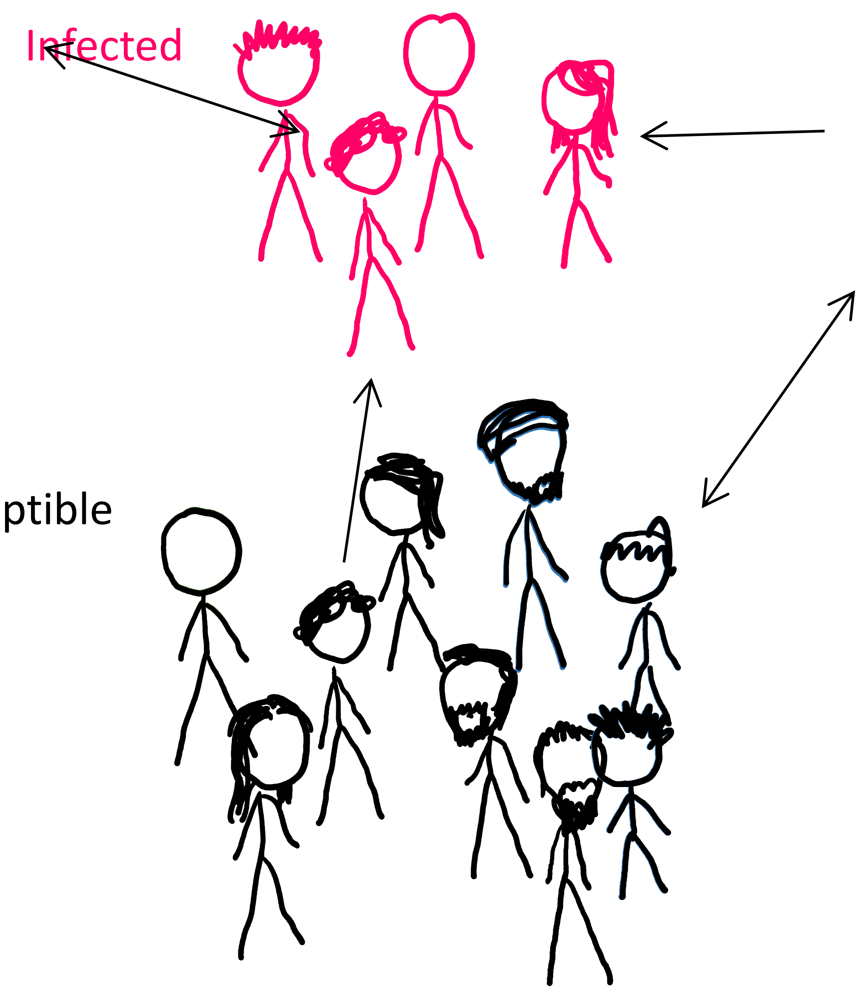


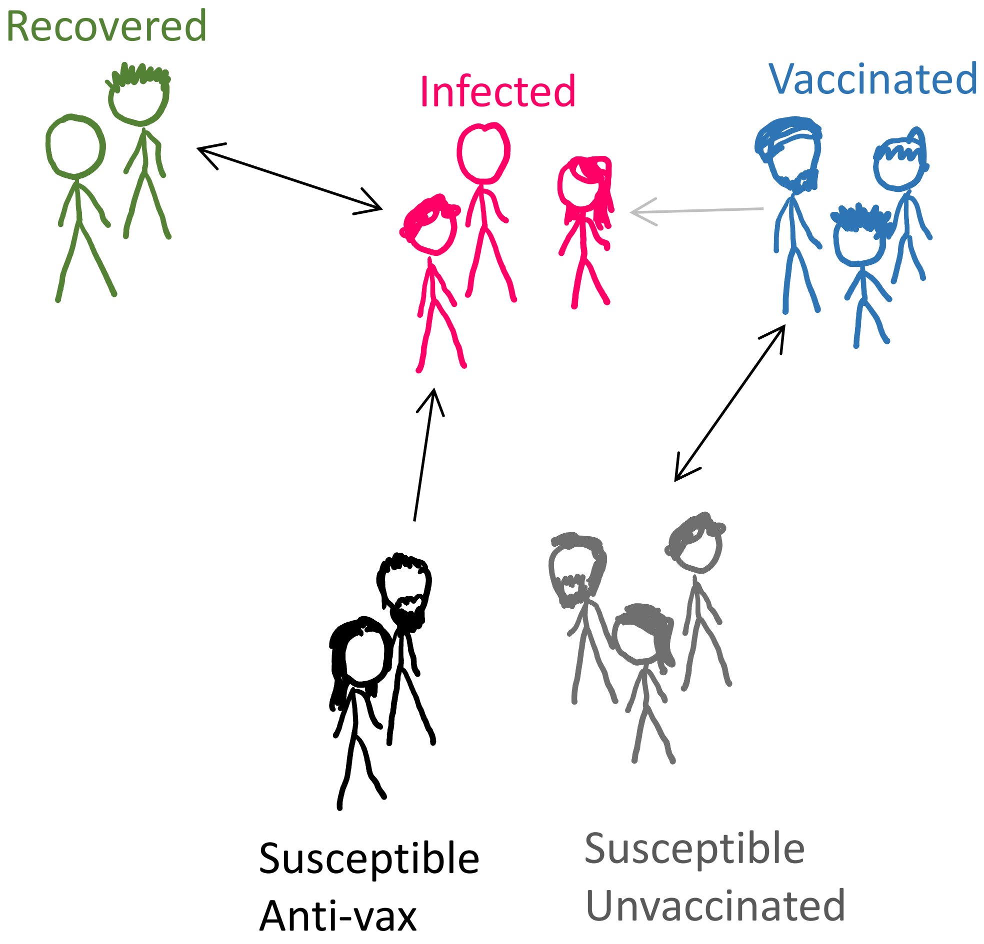
R_0=4, q=75%

each infected person infects

4 others on average

$R_0(1-q)$ others on average

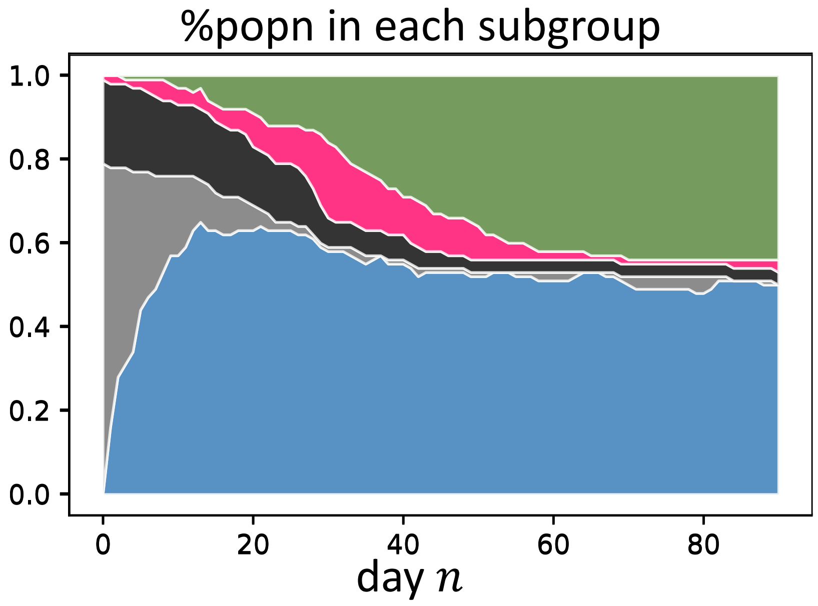

Recovered
Vaccinated
Infected
Infected
Susceptible
Susceptible
Anti-vax
Susceptible
Unvaccinated



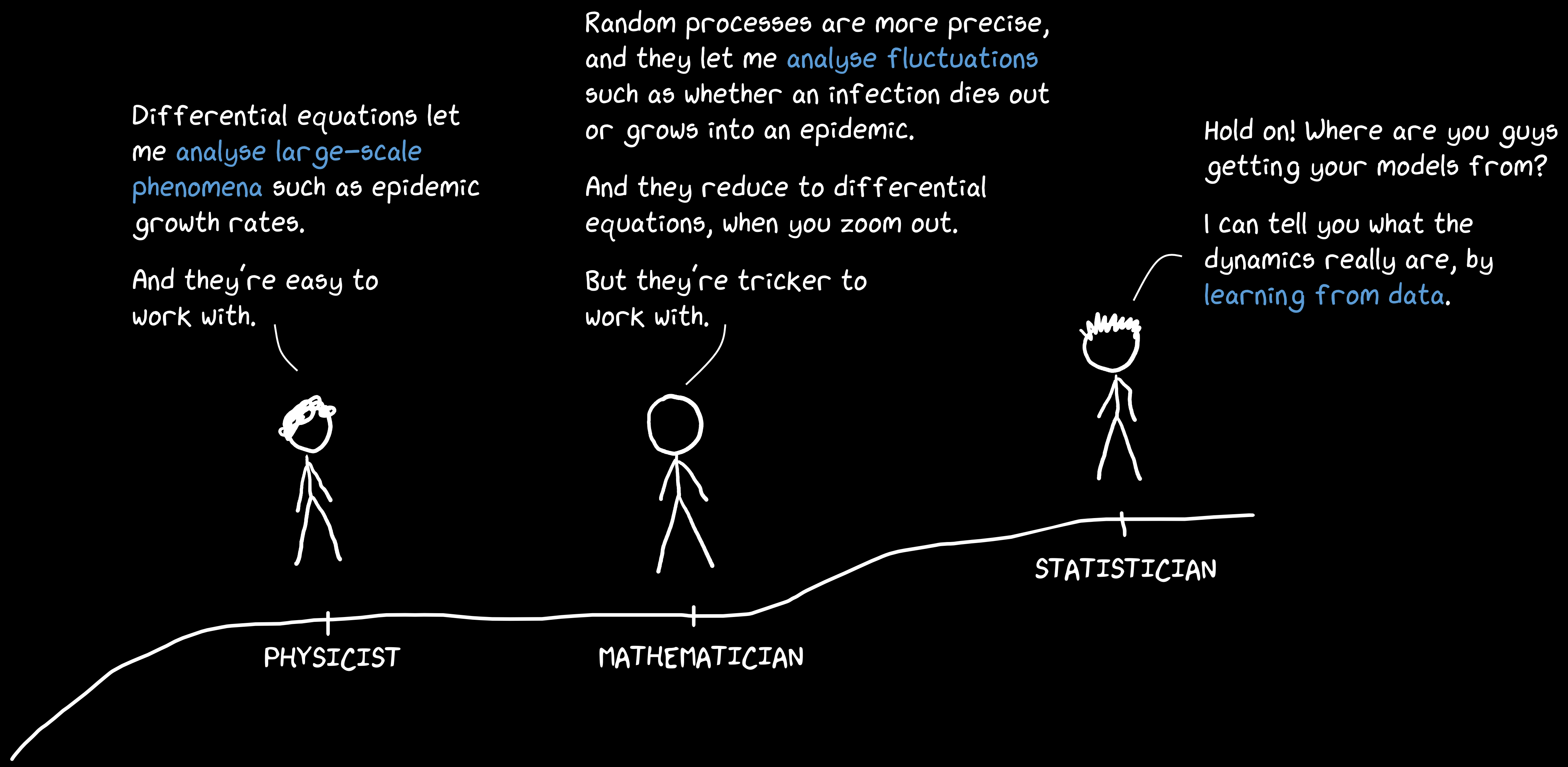
Let $X_n = (A_n, U_n, V_n, R_n, I_n)$, and let total population be N.

Model the epidemic as follows: the update each timestep is

- Infections in subgroup A: $\text{Poisson}(rI_nA_n/Nd)$
- Infections in subgroup U: $\text{Poisson}(rI_nU_n/Nd)$
- Infections in subgroup V: $\text{Poisson}(rI_n\,(1-p_v)V_n/Nd)$
- Infections in subgroup R: $\text{Poisson}(rI_n\,(1-p_r)R_n/Nd)$
- Recoveries: $\text{Bin}(I_n, 1/d)$
- Vaccination elapses: $\text{Bin}(V_n, \lambda_e)$
- Jabs: $\text{Bin}(U, \lambda_v)$



population 10,000


Differential equations let me analyse large-scale phenomena such as epidemic growth rates.
And they're easy to work with.
Random processes are more precise, and they let me analyse fluctuations such as whether an infection dies out or grows into an epidemic.
And they reduce to differential equations, when you zoom out.
But they're tricker to work with.
Hold on! Where are you guys getting your models from?
I can tell you what the dynamics really are, by learning from data.
PHYSICIST
MATHEMATICIAN
STATISTICIAN