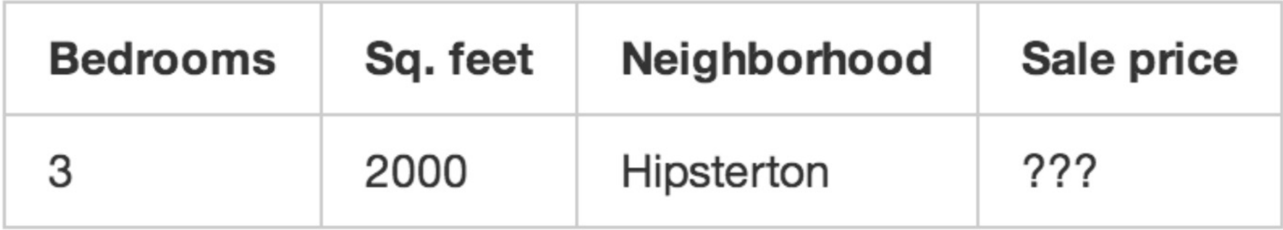
Neural Networks

Previously…

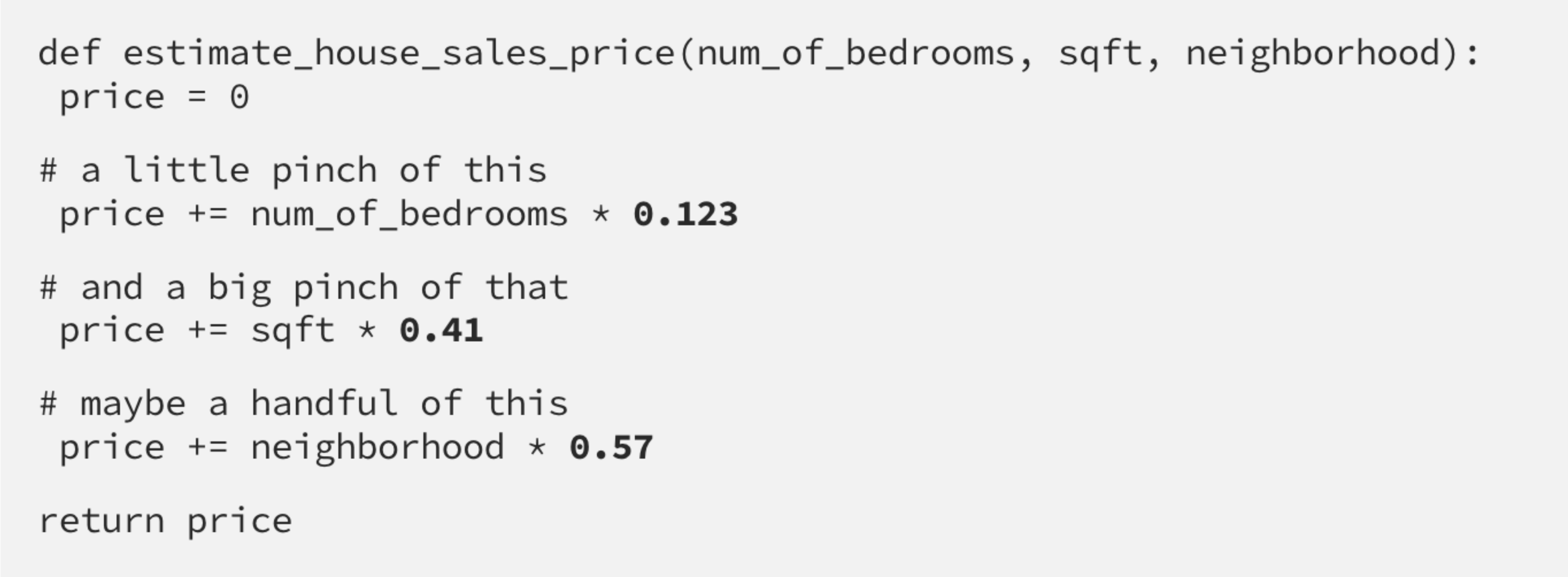
- In the last workshop, we created a simple algorithm that estimated the value of a house based on its attributes.
- Given data about a house like this:



Bedrooms	Sq. feet	Neighborhood	Sale price
3	2000	Hipsterton	???

Previously…

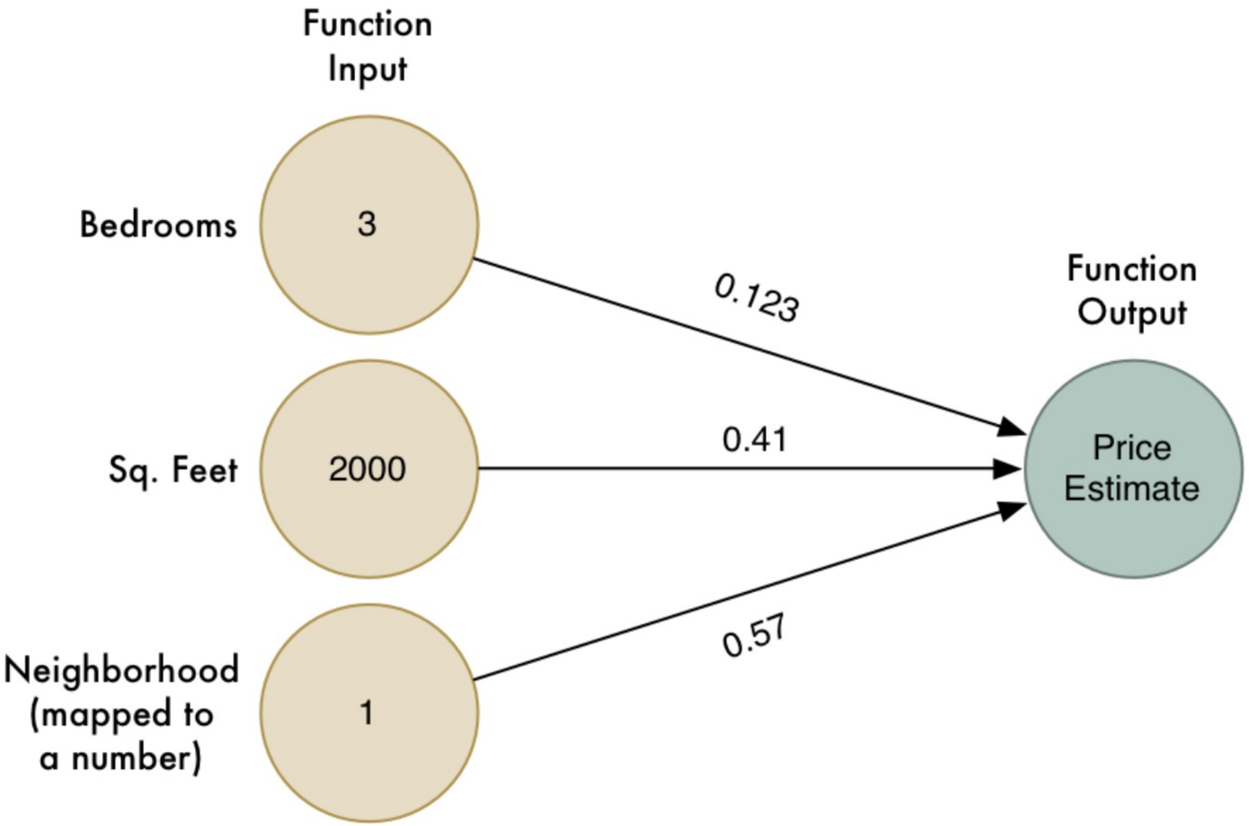
- We ended up with this simple estimation function



```
def estimate_house_sales_price(num_of_bedrooms, sqft, neighborhood):
 price = 0

# a little pinch of this
 price += num_of_bedrooms * 0.123

# and a big pinch of that
 price += sqft * 0.41

# maybe a handful of this
 price += neighborhood * 0.57

return price
```

- In other words, we estimated the value of the house by multiplying each of its attributes by a weight. Then we just added those numbers up to get the house’s value.
- Instead of using code, let’s represent that same function as a simple diagram

Function as a diagram

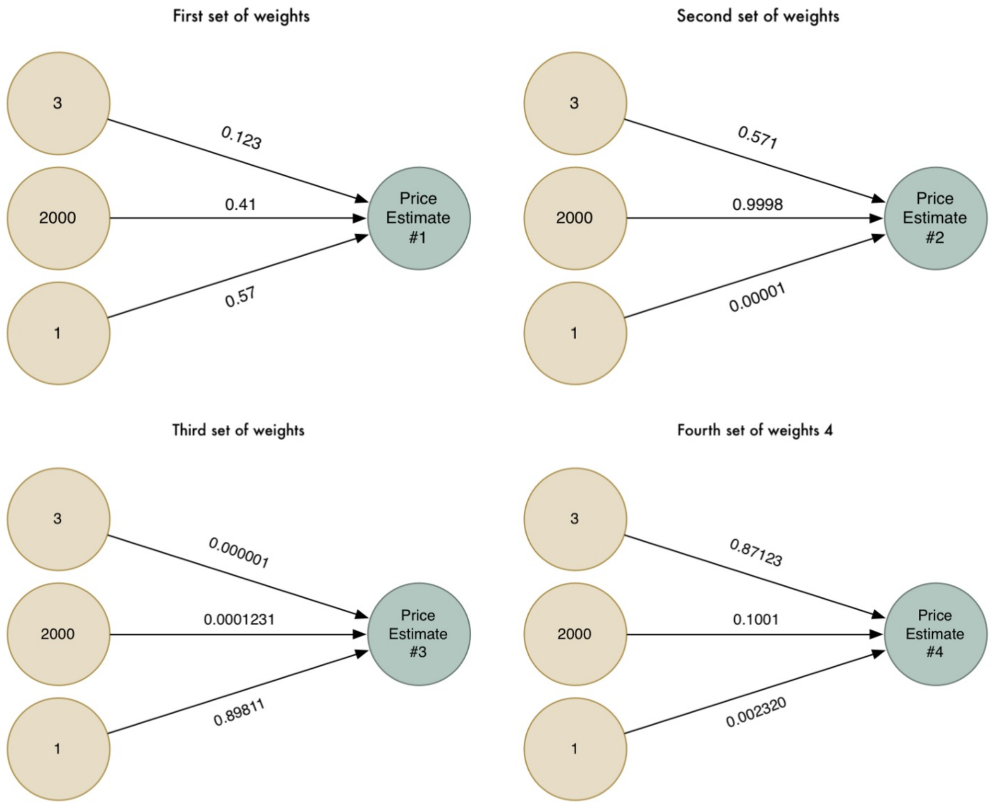


The arrows represent the weights in our function.

Function as a diagram

- However, this algorithm only works for simple problems where the result has a linear relationship with the input.
- What if the truth behind house prices isn’t so simple?
- For example, maybe the neighborhood matters a lot for big houses and small houses but doesn’t matter at all for medium-sized houses.
- How could we capture that kind of complicated detail in our model?

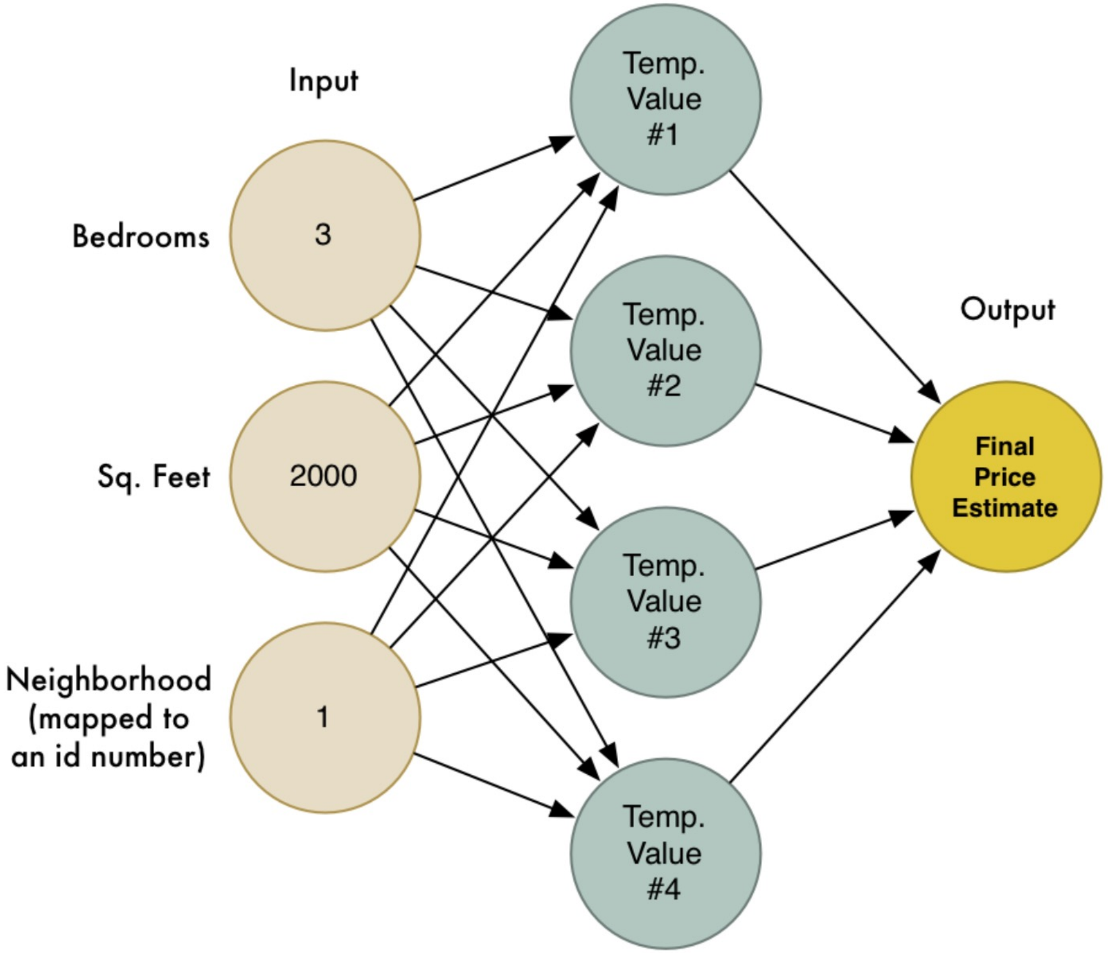
We could run this algorithm multiple times with different of weights that each capture different edge cases

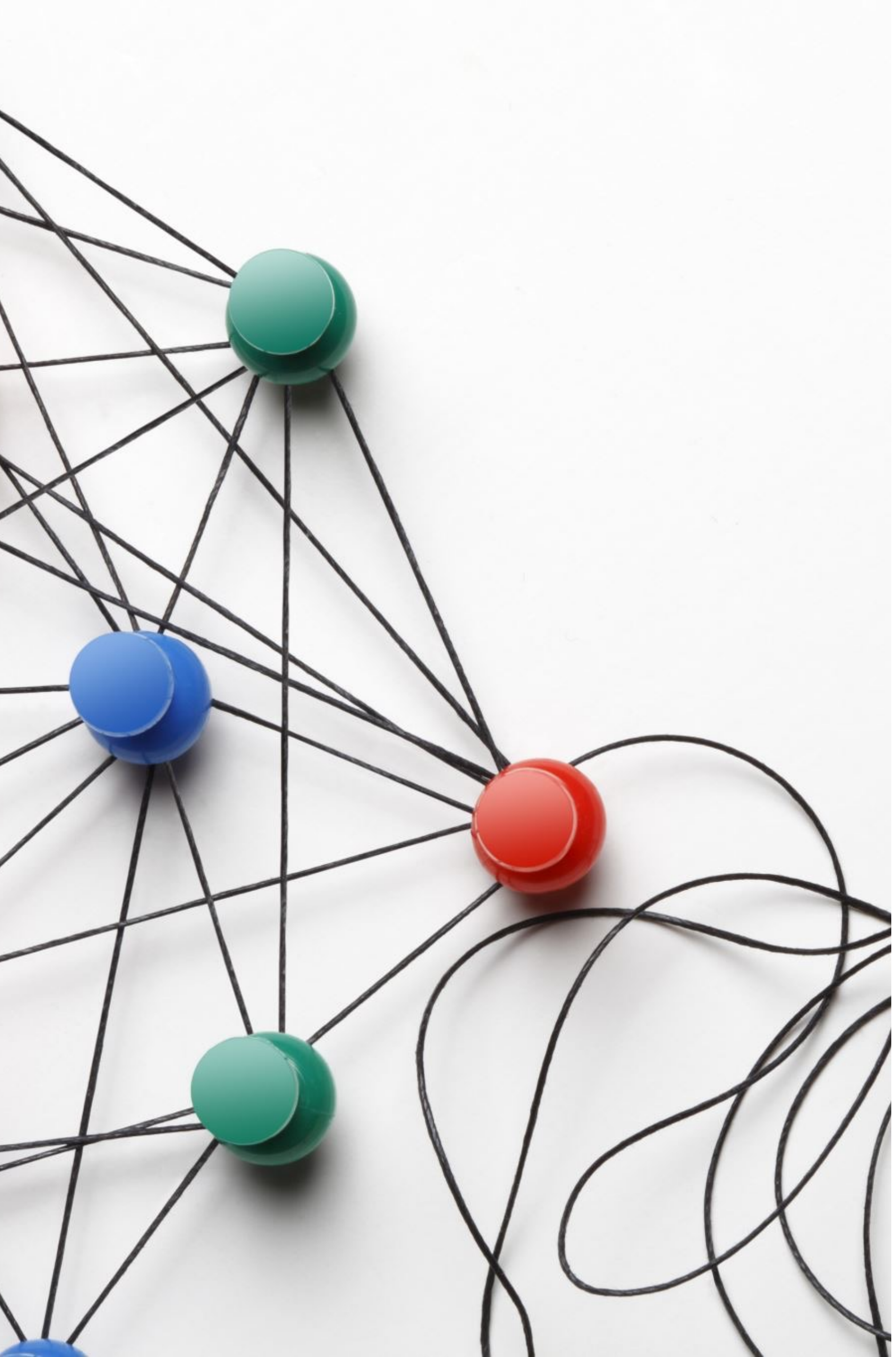


Let's try solving the problem four different ways

- Now we have four different price estimates. Let’s combine those four price estimates into one final estimate. We’ll run them through the same algorithm again (but using another set of weights)!
- Our new Super Answer combines the estimates from our four different attempts to solve the problem. Because of this, it can model more cases than we could capture in one simple model.

Let’s combine our four attempts to guess into one big diagram





Neural Networks

- This is a neural network!
- Each node knows how to take in a set of inputs, apply weights to them, and calculate an output value.
- By chaining together lots of these nodes, we can model complex functions.

Basics revisited

- We made a simple estimation function that takes in a set of inputs and multiplies them by weights to get an output. Call this simple function a **neuron**.
- By chaining lots of simple **neurons** together, we can model functions that are too complicated to be modeled by one single neuron.

The power of neurons

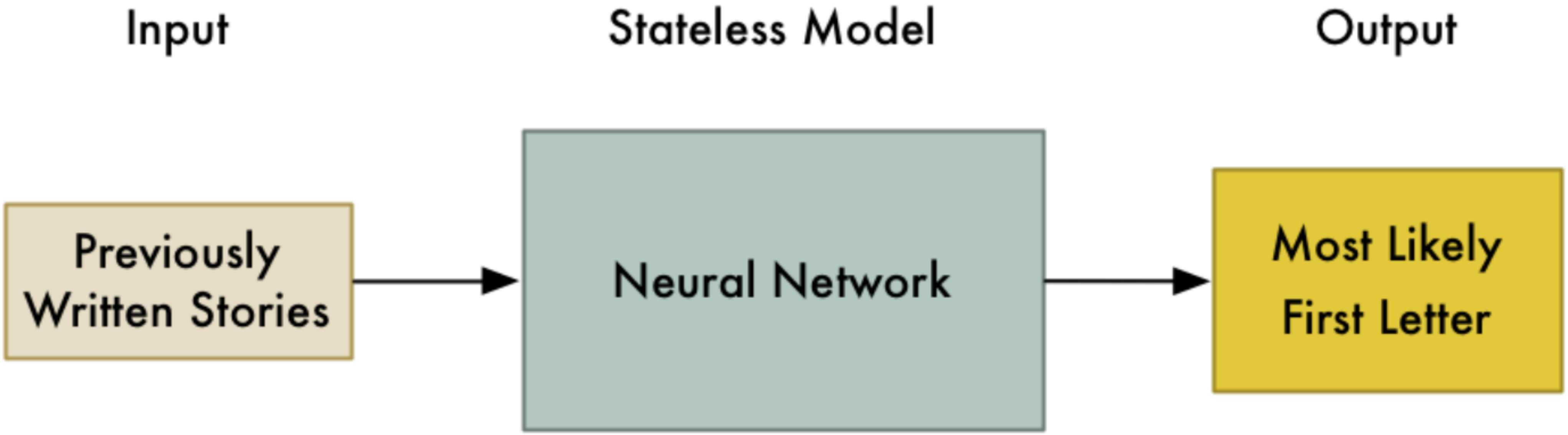
- It's just like LEGO! We can't model much with one single LEGO block, but we can model anything if we have enough basic LEGO blocks to stick together

Giving our Neural Network a Memory

- The neural network we've seen always returns the same answer when you give it the same inputs.
- It has no memory. In programming terms, it's a stateless algorithm.
- In many cases (like estimating the price of house), that's exactly what you want. But the one thing this kind of model can't do is respond to patterns in data over time.

Giving our Neural Network a Memory

- Imagine I handed you a keyboard and asked you to write a story. But before you start, my job is to guess the very first letter that you will type. What letter should I guess?
- If I looked at stories you wrote in the past, I could narrow it down based on the words you usually use at the beginning of your stories.
- Once I had all that data, I could use it to build a neural network to model how likely it is that you would start with any given letter.



Input

Stateless Model

Output

Previously Written Stories

Neural Network

Most Likely First Letter

Making the problem more interesting…

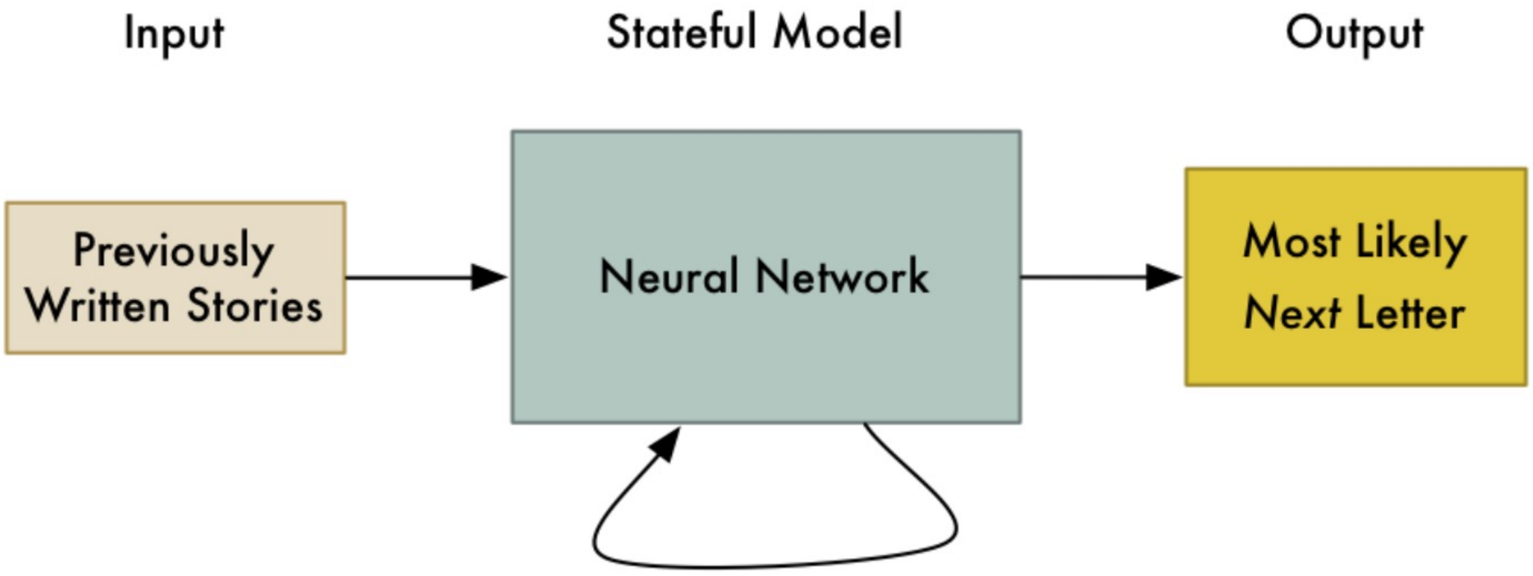
- Let's say I need to guess the next letter you are going to type at any point in your story.
- Let's use the first few words of Ernest Hemingway's ***The Sun Also Rises*** as an example:

*Robert Cohn was once middleweight boxi_*

Giving our Neural Network a Memory

- It's easy to guess the next letter if we take into account the sequence of letters that came right before it and combine that with our knowledge of the rules of English.
- To solve this problem with a neural network, we need to add *state* to our model. Each time we ask our neural network for an answer, we also save a set of our intermediate calculations and re-use them the next time as part of our input.
- That way, our model will adjust its predictions based on the input that it has seen recently.

Giving our Neural Network a Memory



Save the model's current state
and use that as part
of our next calcuation.

Recurrent Neural Networks

- This is the basic idea of a Recurrent Neural Network. We are updating the network each time we use it.
- This allows it to update its predictions based on what it saw most recently.
- It can even model patterns over time as long as we give it enough of a memory.

Pushing our algorithm…

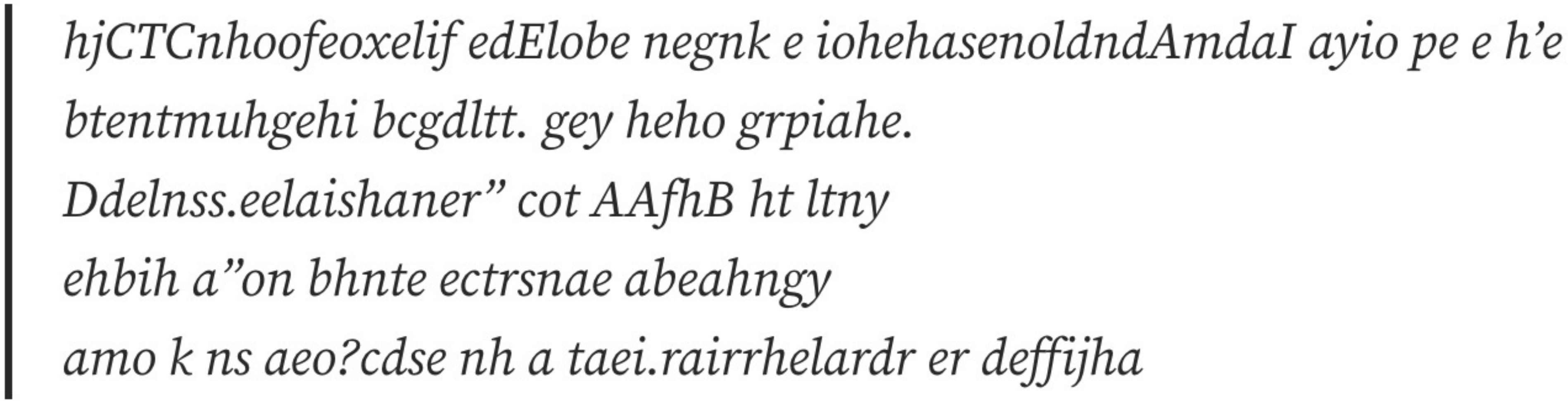
- What if we took this idea to the extreme? What if we asked the model to predict the next most likely character over and over – forever?
- We’d be asking it to write a complete story for us!

Generating a story

- Let's try generating a whole story in the style of Hemingway.
- We are going to use the Recurrent Neural Network implementation that Andrej Karpathy wrote.
- We'll create our model from the complete text of **The Sun Also Rises** – 362,239 characters using 84 unique letters (including punctuation, uppercase/lowercase, etc).
- This data set is actually really small compared to typical real-world applications

Generating a story

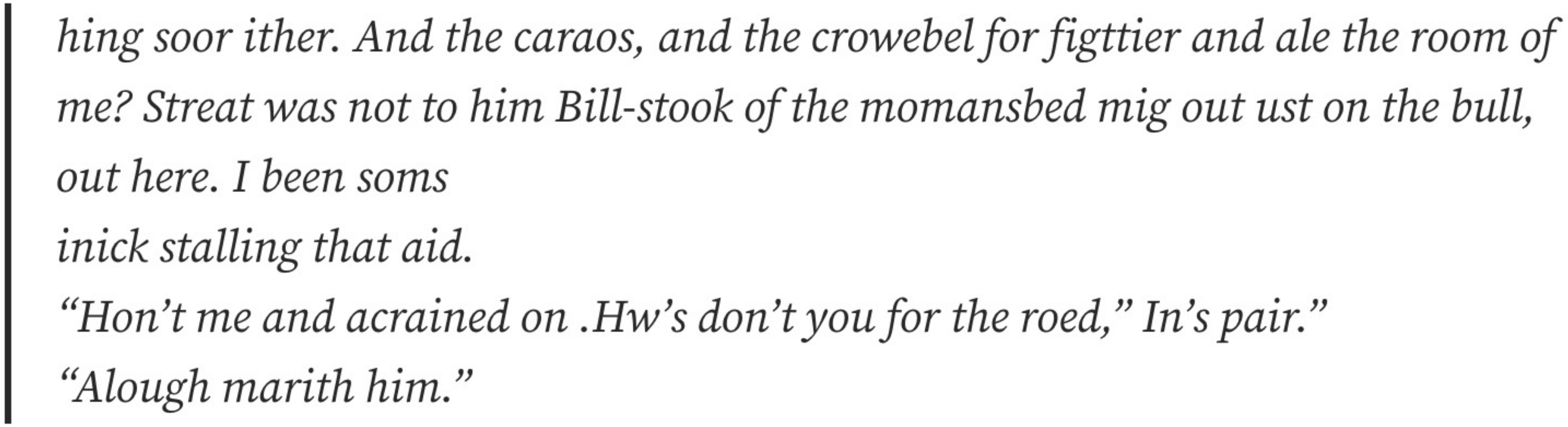
- As we just start to train the RNN, it’s not very good at predicting letters. Here’s what it generates after a 100 loops of training.



```
hjCTCnhoofeoxelif edElobe negnk e iohehasenoldndAmdaI ayio pe e h’e
btentmuhgehi bcgdltt. gey heho grpiahe.
Ddelnss.eelaishaner” cot AAfhB ht ltny
ehbih a”on bhnte ectrsnae abeahngy
amo k ns aeo?cdse nh a taei.rairrhelardr er deffijha
```

Generating a story

- After about 1000 iterations, things are looking more promising:

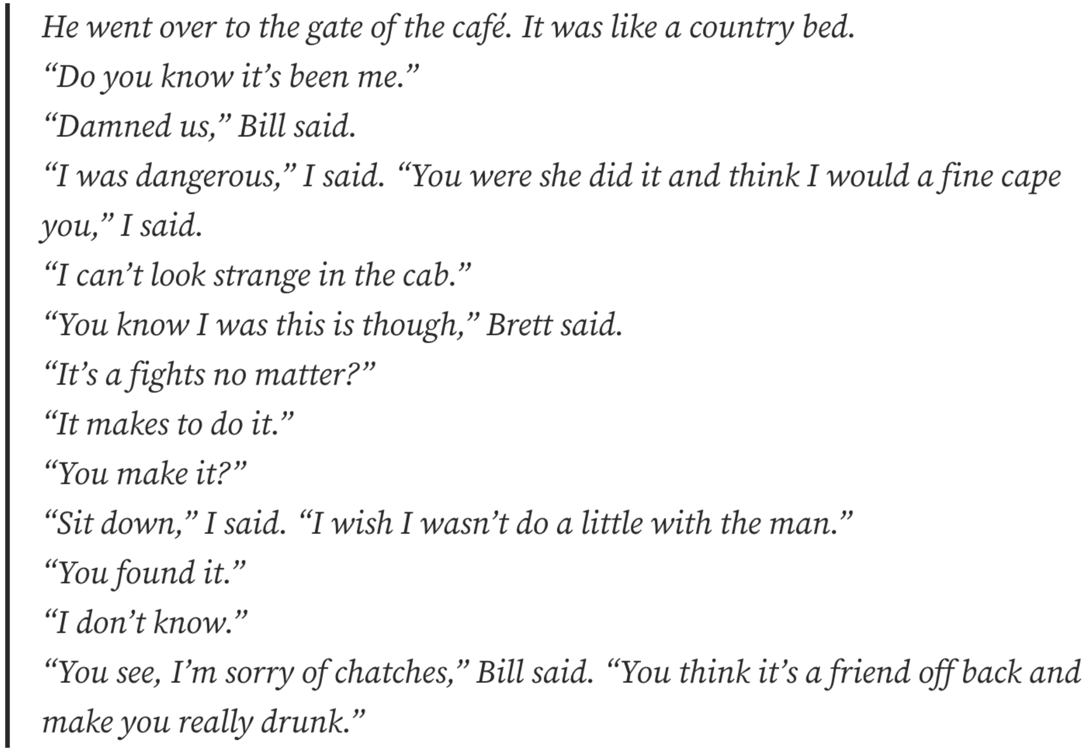


> *hing soor ither. And the caraos, and the crowebel for figttier and ale the room of me? Streat was not to him Bill-stook of the momansbed mig out ust on the bull, out here. I been soms*
>
> *inick stalling that aid.*
>
> *"Hon't me and acrained on .Hw's don't you for the roed," In's pair."*
>
> *"Alough marith him."*

- The model has started to identify the patterns in basic sentence structure. It's adding periods at the ends of sentences and even quoting dialog. A few words are recognizable, but there's also still a lot of nonsense.

Generating a story

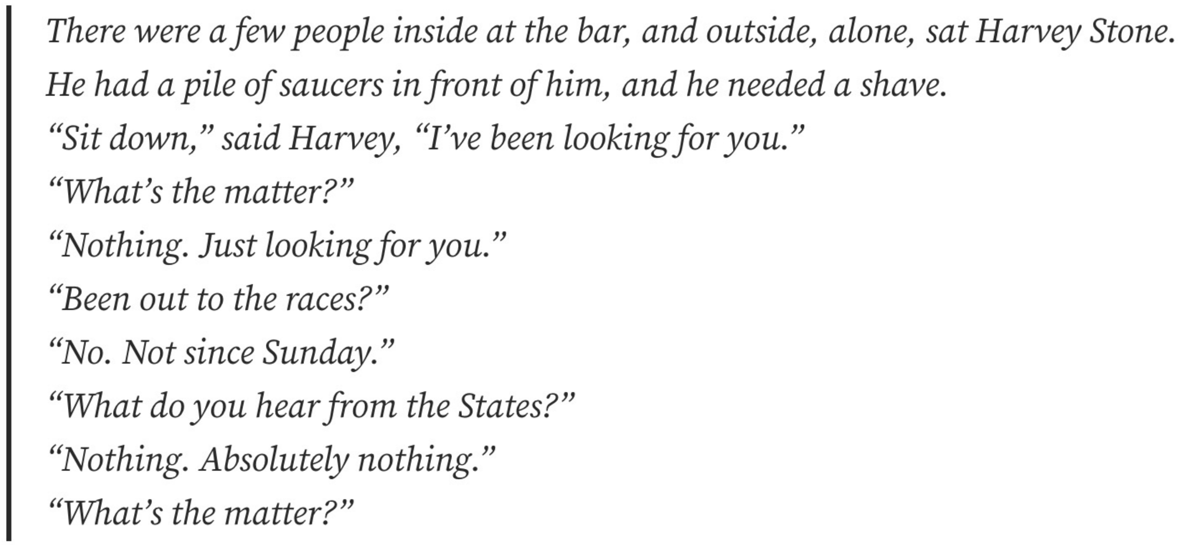
- After several thousand more training iterations, it looks pretty good:



> *He went over to the gate of the café. It was like a country bed.*
> *“Do you know it’s been me.”*
> *“Damned us,” Bill said.*
> *“I was dangerous,” I said. “You were she did it and think I would a fine cape you,” I said.*
> *“I can’t look strange in the cab.”*
> *“You know I was this is though,” Brett said.*
> *“It’s a fights no matter?”*
> *“It makes to do it.”*
> *“You make it?”*
> *“Sit down,” I said. “I wish I wasn’t do a little with the man.”*
> *“You found it.”*
> *“I don’t know.”*
> *“You see, I’m sorry of chatches,” Bill said. “You think it’s a friend off back and make you really drunk.”*

Generating a story

- At this point, the algorithm has captured the basic pattern of Hemingway's short, direct dialog. A few sentences even sort of make sense. Compare that with some real text from the book:



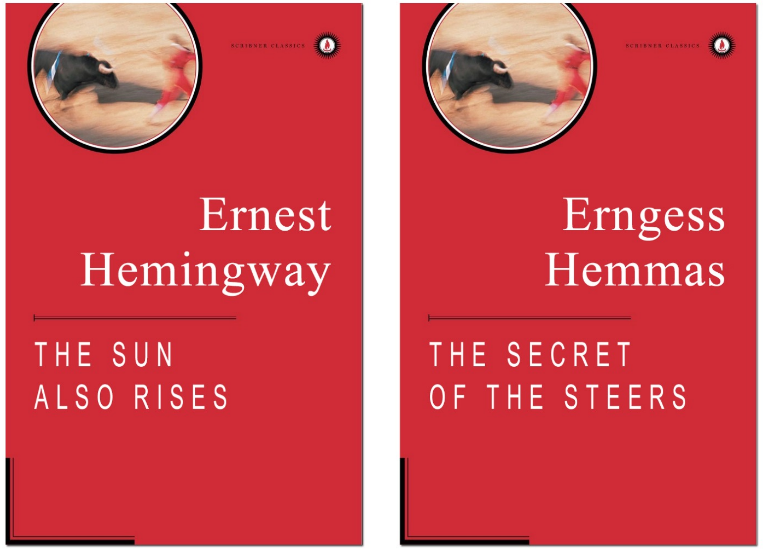
> *There were a few people inside at the bar, and outside, alone, sat Harvey Stone.*
> *He had a pile of saucers in front of him, and he needed a shave.*
> *"Sit down," said Harvey, "I've been looking for you."*
> *"What's the matter?"*
> *"Nothing. Just looking for you."*
> *"Been out to the races?"*
> *"No. Not since Sunday."*
> *"What do you hear from the States?"*
> *"Nothing. Absolutely nothing."*
> *"What's the matter?"*

Generating a story

- Even by only looking for patterns one character at a time, our algorithm has reproduced plausible-looking prose with proper formatting. That is kind of amazing!
- We don’t have to generate text completely from scratch, either. We can seed the algorithm by supplying the first few letters and just let it find the next few letters.

Generating a story

- For fun, let’s make a fake book cover for our imaginary book by generating a new author name and a new title using the seed text of “Er”, “He”, and “The S”:



The real book is on the left and our silly computer-generated nonsense book is on the right.

Recurrent Neural Networks

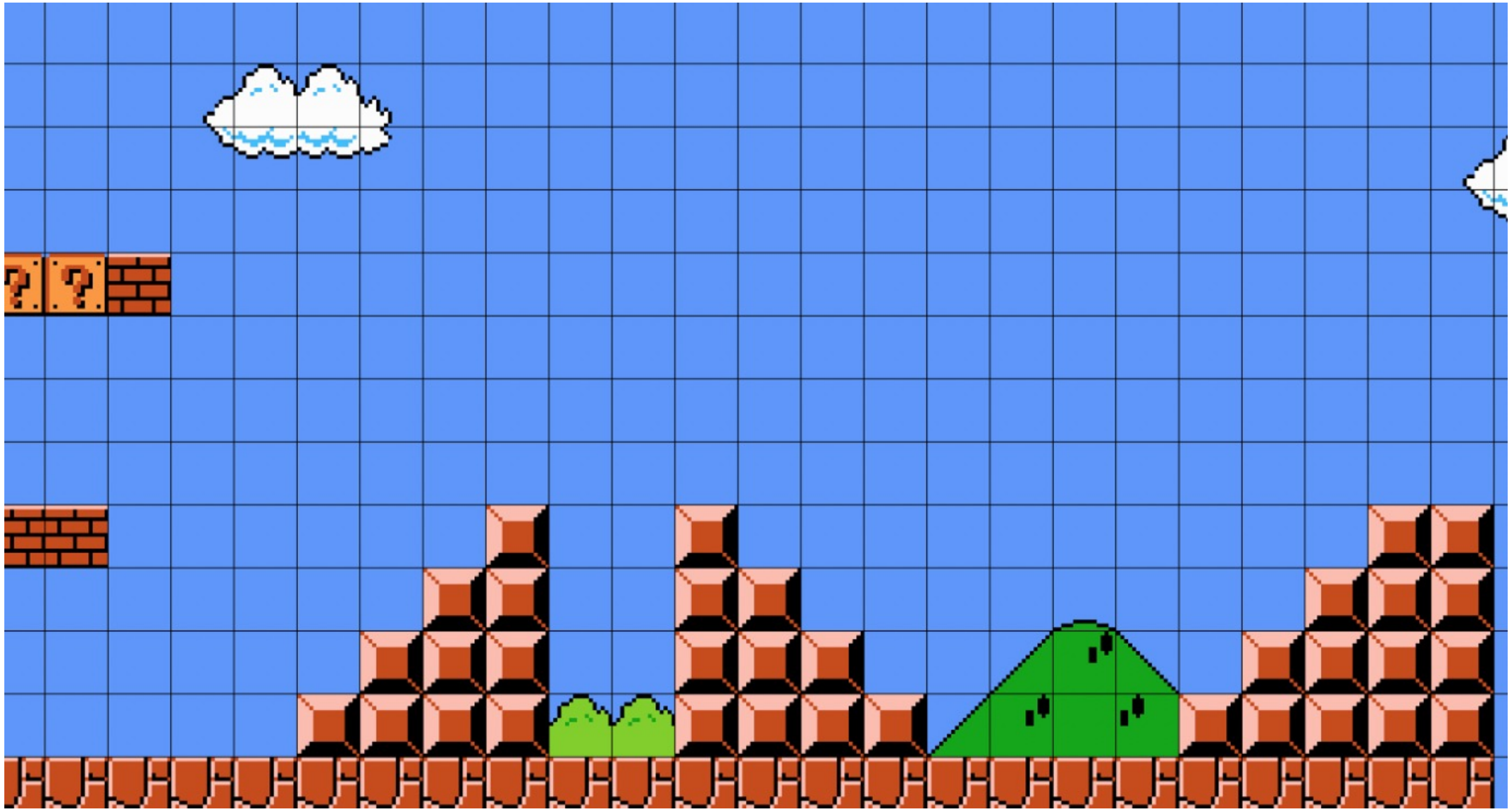
- This algorithm can figure out patterns in any sequence of data!
- It can easily generate real-looking recipes or fake Obama speeches.
- But why limit ourselves human language? We can apply this same idea to any kind of sequential data that has a pattern.

Making Mario without actually Making Mario

- Can we use the same model that generated fake Hemingway text to generate fake Super Mario Brothers levels?
- First, we need a data set for training our model. Let’s take all the outdoor levels from the original Super Mario Brothers game released in 1985

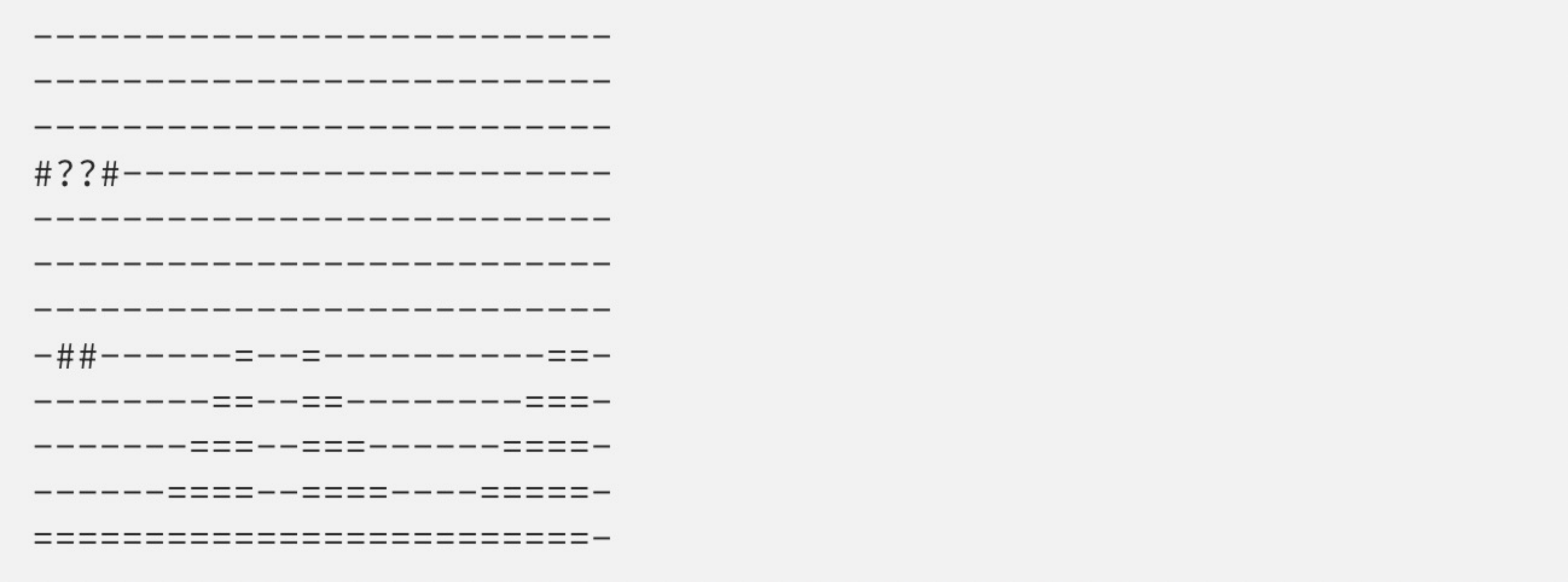
A blast from the past

If we look closely, we can see the level is made of a simple grid of objects



Making Mario without actually Making Mario

- We could just as easily represent this grid as a sequence of characters with one character representing each object

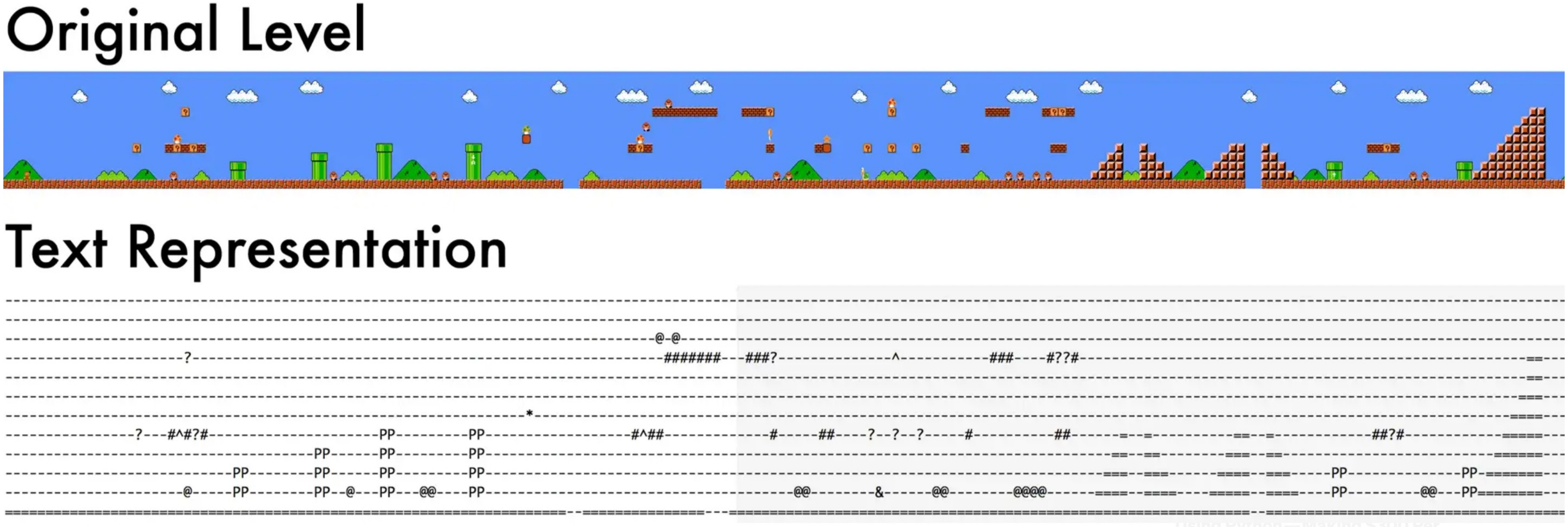


```
--------------------------
--------------------------
--------------------------
#??#----------------------
--------------------------
--------------------------
--------------------------
-##------=--=----------==-
--------==--==--------===-
-------===--===------====-
------====--====----=====-
=========================-
```

Making Mario without actually Making Mario

- We’ve replaced each object in the level with a letter:
  - ‘-’ is a blank space
  - ‘=’ is a solid block
  - ‘#’ is a breakable brick
  - ‘?’ is a coin block
- …and so on, using a different letter for each different kind of object in the level.

Final text files



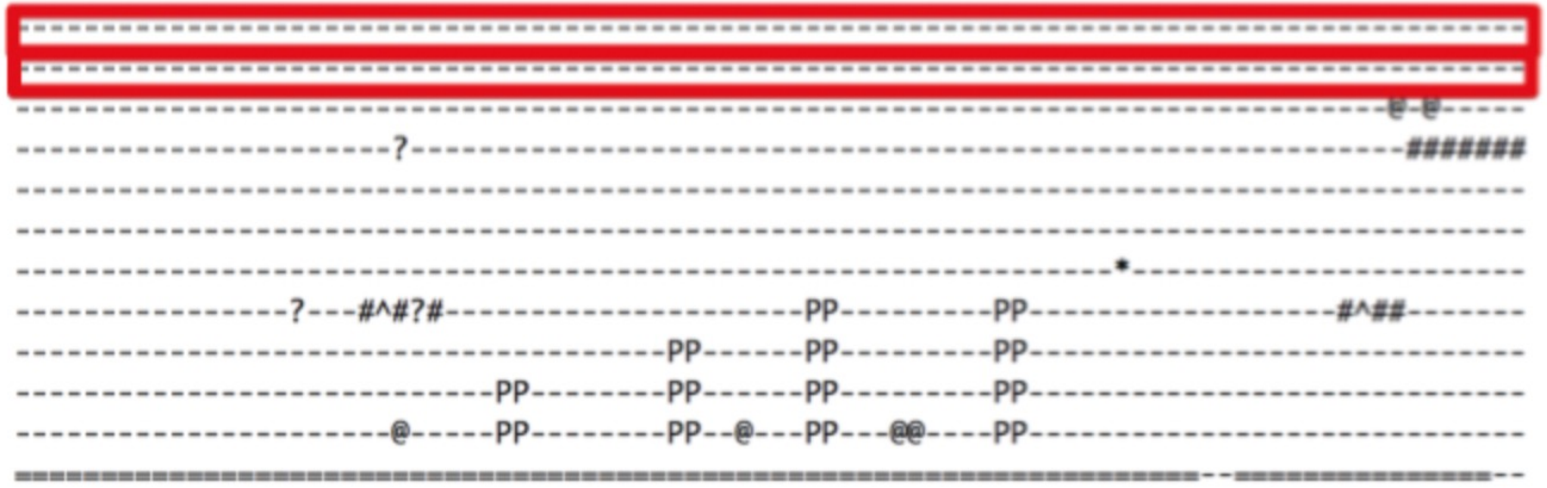
Original Level

Text Representation

```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------@-@-------------------------------------------------------------------------------------------------------------
----------------------?----------------------------------------------------------#######---###?--------------^-----------###----#??#-------------------------------------------------------==---
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------==---
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------===---
----------------------------------------------------------------*------------------------------------------------------------------------------------------------------------------------====---
----------------?---#^#?#---------------------PP---------PP------------------#^##-------------#-----##----?--?--?-----#----------##------=--=----------==--=------------##?#------------=====---
--------------------------------------PP------PP---------PP-----------------------------------------------------------------------------==--==--------===--==--------------------------======---
----------------------------PP--------PP------PP---------PP----------------------------------------------------------------------------===--===------====--===-----PP--------------PP-=======---
----------------------@-----PP--------PP--@---PP---@@----PP--------------------------------------@@--------&------@@--------@@@@------====--====----=====--====----PP---------@@---PP========---
=====================================================================--===============---================================================================--=====================================
```

Making Mario without actually Making Mario

- Looking at the text file, you can see that Mario levels don’t really have much of a pattern if you read them line-by-line:

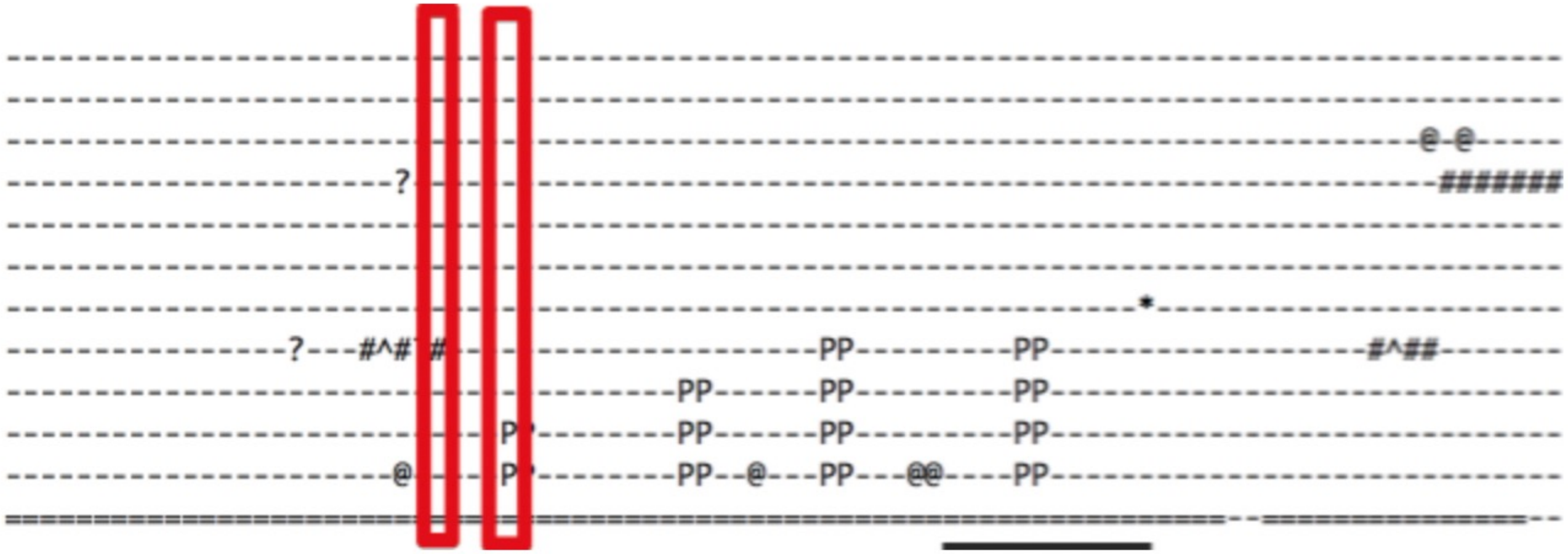


```
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
--------------------------------------------------------------------------------@-@-----
----------------------?----------------------------------------------------------#######
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------*-----------------------
----------------?---#^#?#---------------------PP---------PP------------------#^##-------
--------------------------------------PP------PP---------PP-----------------------------
----------------------------PP--------PP------PP---------PP-----------------------------
----------------------@-----PP--------PP--@---PP---@@----PP-----------------------------
=====================================================================--===============--
```

Reading line-by-line, there’s not really a pattern to capture. Lots of lines are completely blank.

Making Mario without actually Making Mario

- The patterns in a level really emerge when you think of the level as a series of columns



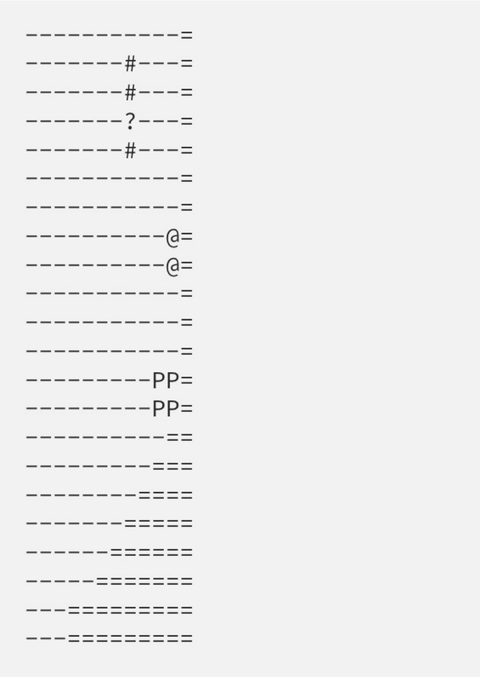
Looking column-by-column, there's a real pattern. Each column ends in a '=' for example.

Feature selection and representation

- In order for the algorithm to find the patterns in our data, we need to feed the data in column-by-column.
- Figuring out the most effective representation of your input data (called feature selection) is one of the keys of using machine learning algorithms well.

Making Mario without actually Making Mario

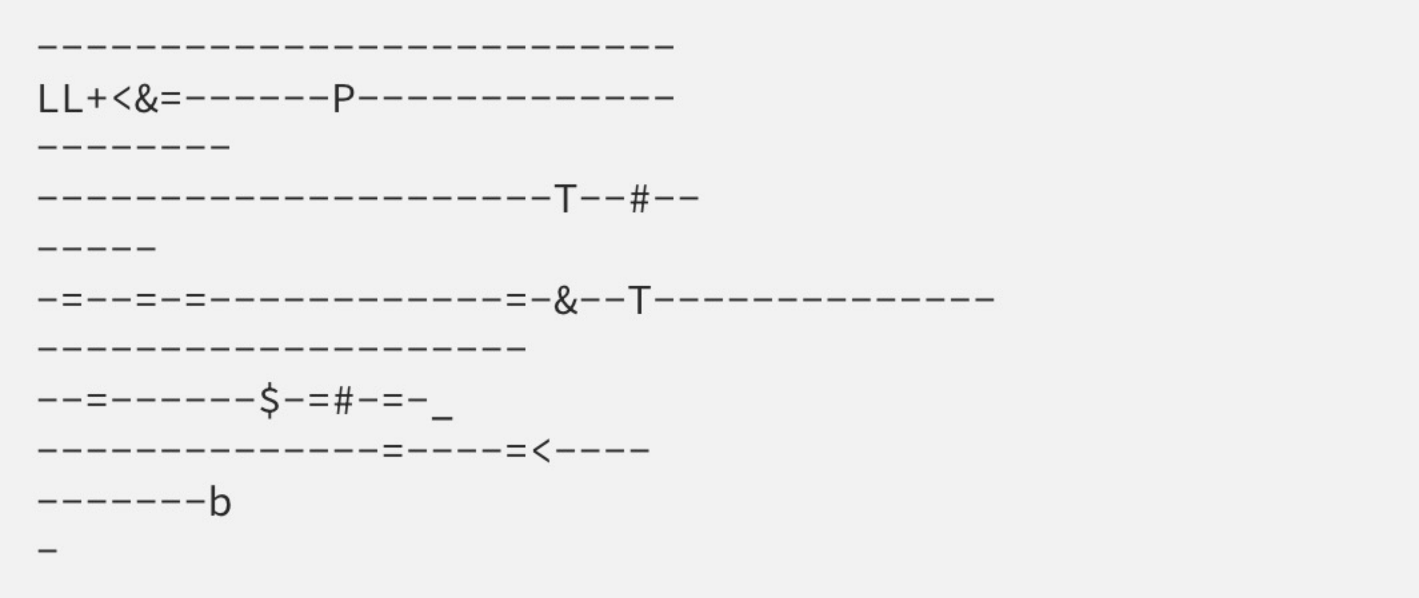
- To train the model, I needed to rotate my text files by 90 degrees. This made sure the characters were fed into the model in an order where a pattern would more easily show up



```
-----------=
-------#---=
-------#---=
-------?---=
-------#---=
-----------=
-----------=
----------@=
----------@=
-----------=
-----------=
-----------=
---------PP=
---------PP=
----------==
---------===
--------====
-------=====
------======
-----=======
---=========
---=========
```

Training Our Model

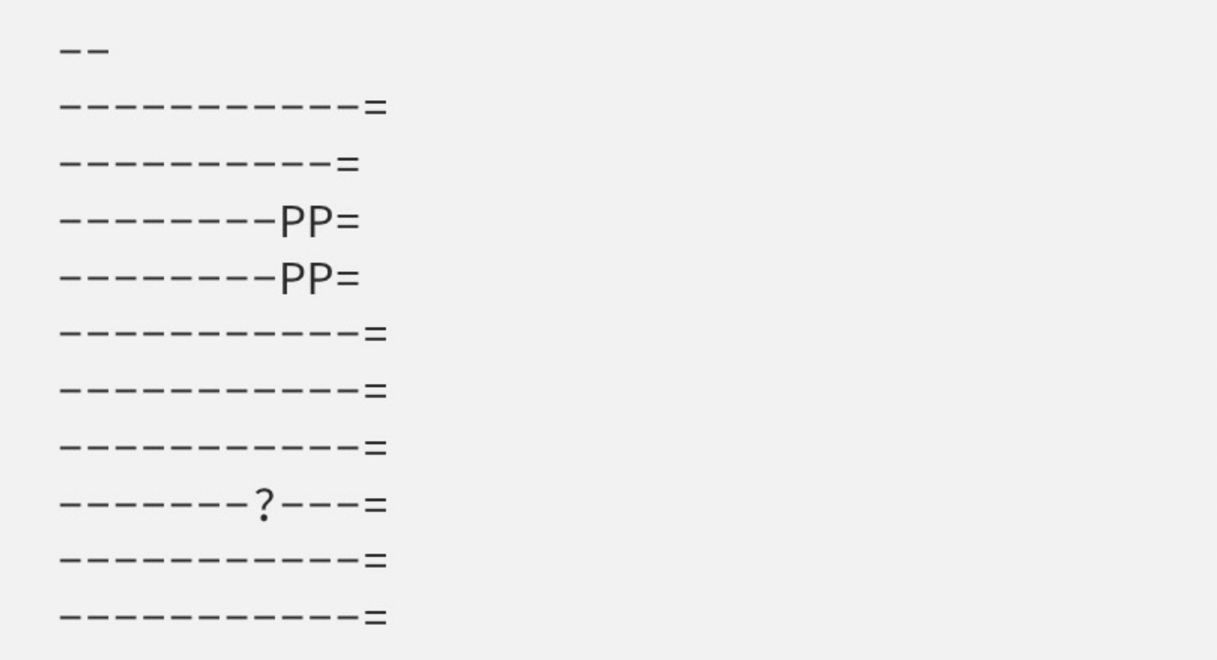
- Just like we saw when creating the model of Hemingway’s prose, a model improves as we train it. After a little training, our model is generating junk.



```
--------------------------
LL+<&=------P-------------
--------
---------------------T--#--
-----
-=--=-=------------=-&--T--------------
--------------------
--=------$-=#-=-_
--------------=----=<----
-------b
-
```

Training Our Model

- After several thousand iterations, it’s starting to look like something:



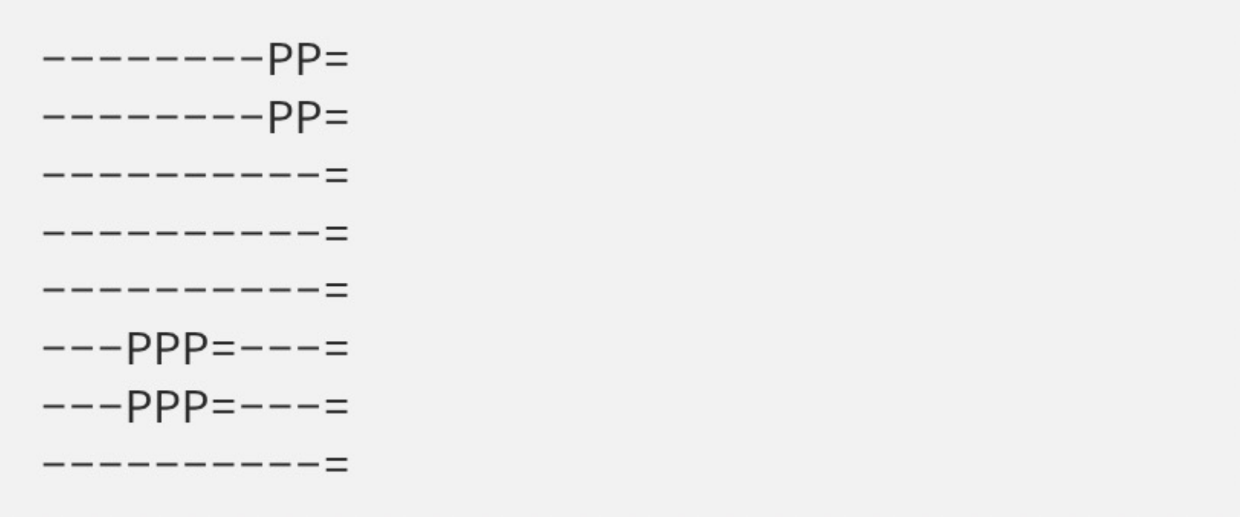
```
--
------------=
-----------=
---------PP=
---------PP=
------------=
------------=
------------=
--------?---=
------------=
------------=
```

Training Our Model: Patterns detected

- The model has almost figured out that each line should be the same length.
- It has even started to figure out some of the logic of Mario: The pipes in mario are always two blocks wide and at least two blocks high, so the “P”s in the data should appear in 2x2 clusters.

Still training…

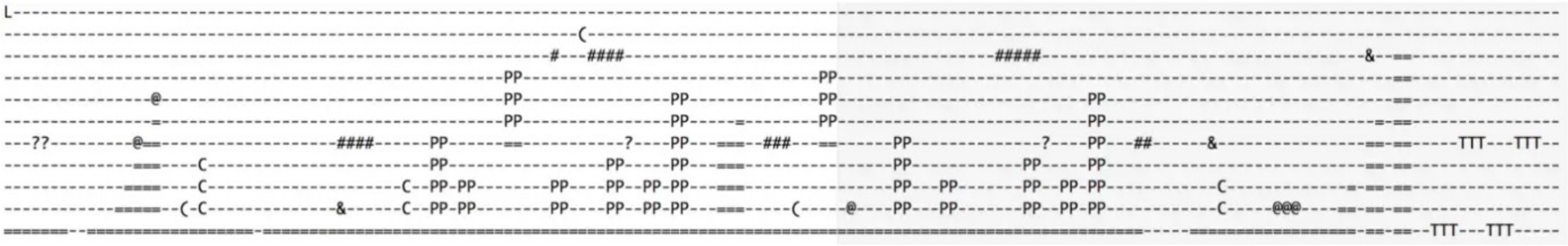
- With a lot more training, the model gets to the point where it generates perfectly valid data



```
--------PP=
--------PP=
----------=
----------=
----------=
---PPP=---=
---PPP=---=
----------=
```

Generating a new level

- Let's sample an entire level's worth of data from our model and rotate it back horizontal



```
L-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------(---------------------------------------------------------------------------------------------------------
-----------------------------------------------------------#---####----------------------------------------#####-----------------------------------&--==----------------
------------------------------------------------------PP--------------------------------PP------------------------------------------------------------==----------------
----------------@-------------------------------------PP----------------PP--------------PP---------------------------PP-------------------------------==----------------
----------------=-------------------------------------PP----------------PP-----=--------PP---------------------------PP-----------------------------=-==----------------
---??---------@==-------------------####------PP------==-----------?----PP---===--###---==------PP--------------?----PP---##------&----------------==-==-----TTT---TTT--
--------------===----C------------------------PP-----------------PP-----PP---===----------------PP------------PP-----PP----------------------------==-==----------------
-------------====----C---------------------C--PP-PP--------PP----PP--PP-PP---===----------------PP---PP-------PP--PP-PP------------C-------------=-==-==----------------
------------=====--(-C--------------&------C--PP-PP--------PP----PP--PP-PP---===-----(-----@----PP---PP-------PP--PP-PP------------C-----@@@----==-==-==----------------
=======--==================-===============================================================================================----===================-==-==--TTT---TTT-----
```

A whole level, generated from our model!

Observing closely…

- This data looks great! There are several awesome things to notice:
- It put a Lakitu (the monster that floats on a cloud) up in the sky at the beginning of the level – just like in a real Mario level.
- It knows that pipes floating in the air should be resting on top of solid blocks and not just hanging in the air.
- It places enemies in logical places.
- It doesn’t create anything that would block a player from moving forward.
- It feels like a real level from Super Mario Bros. 1 because it’s based off the style of the original levels that existed in that game.

Final result

- Let us see how the final result looks like:

RNN generated Mario level

Toys vs. Real World Applications

- The recurrent neural network algorithm we used to train our model is the same kind of algorithm used by real-world companies to solve hard problems like speech detection and language translation.
- What makes our model a ‘toy’ instead of cutting-edge is that our model is generated from very little data.
- There just aren’t enough levels in the original Super Mario Brothers game to provide enough data for a really good model.

Data is precious

- As machine learning becomes more important in more industries, the difference between a good program and a bad program will be how much data you have to train your models. That’s why companies like Google and Facebook need your data so badly!

Conclusion

- Outline of a neural network
- Recurrent Neural Networks
- RNNs for story generation
- RNNs for Mario Level Generation