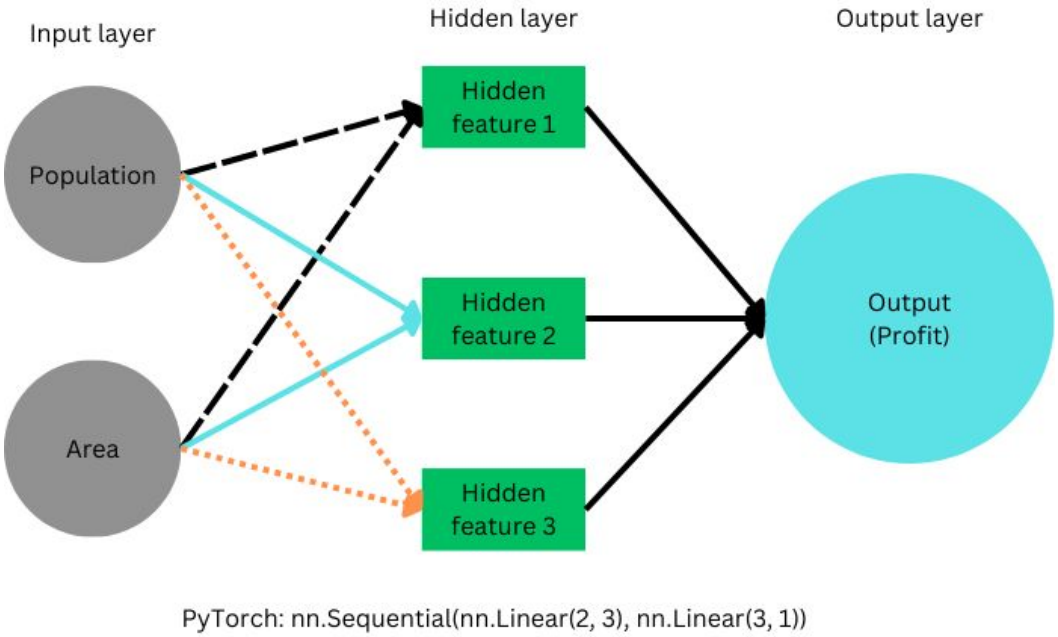
Recurrent Neural Networks

Recap - Linear regression

- Good for data without much structure



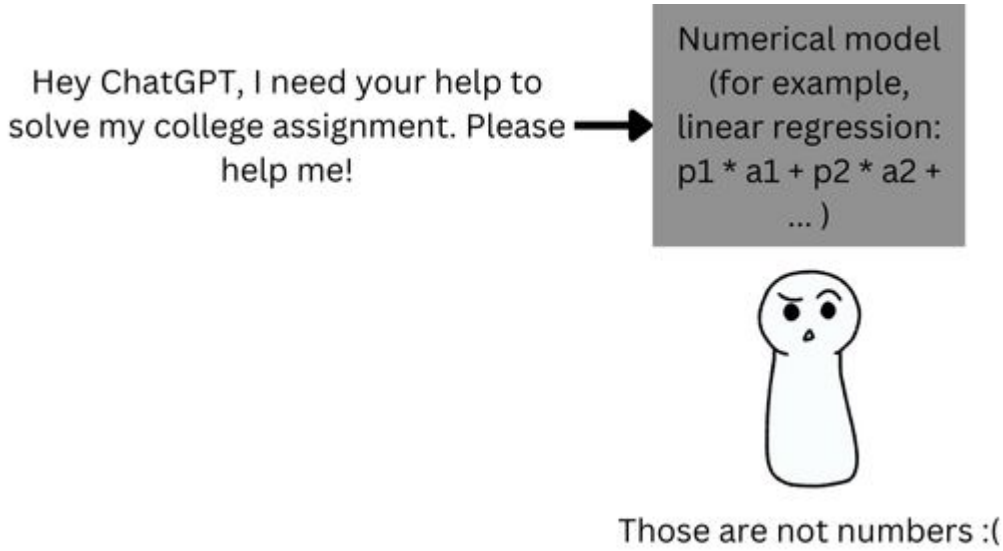
PyTorch: nn.Sequential(nn.Linear(2, 3), nn.Linear(3, 1))

Recap - Convolutions

- Good for images, which have a 2D structure

What about text?

- Text is inherently not numerical



Turning text into numbers

“Hey	ChatGPT,	help	me	solve	this	exercise.	Help	me	or	I	will	fail	this	exam.”
0	1	2	3	4	5	6	2	3	7	8	9	10	5	11

- Replace each word (or letter, or part of word, or whatever we decide: these are called tokens) with a number
- ChatGPT: less common words like masterful are split into master and ful, so the word gets replaced by 2 numbers / tokens
- Whenever a word appear multiple times, it is assigned the same index

Turning text into numbers

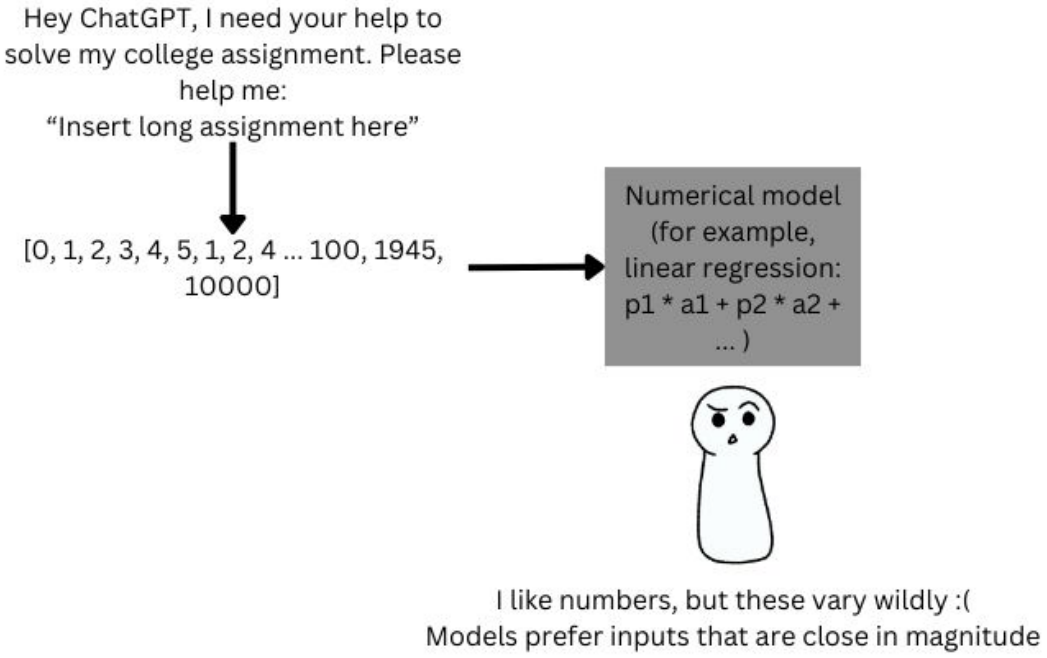
What about the text **I passed this exam** ?:

- a) 0, 1, 2, 3
- b) 12, 13, 14, 15
- **c) 8, 12, 5, 11**
- d) 8, 10, 4, 12

“Hey	ChatGPT,	help	me	solve	this	exercise.	Help	me	or	I	will	fail	this	exam.”
0	1	2	3	4	5	6	2	3	7	8	9	10	5	11

Giving the numbers to a model

- Over 100.000 words in english language
- Model could receive as inputs numbers 1 and 99.999
- Models prefer inputs of **similar scale**



One-hot Text Embeddings

- Replace each number with a vector
- If we have the sentence “**I like to ride my bike**”:
  - I => 0 => [1, 0, 0, 0, 0, 0]
  - like => 1 => [0, 1, 0, 0, 0, 0]
  - to => 2 => [0, 0, 1, 0, 0, 0]
  - ride => 3 => [0, 0, 0, 1, 0, 0]
  - my => 4 => [0, 0, 0, 0, 1, 0]
  - bike => 5 => [0, 0, 0, 0, 0, 1]

One-Hot embeddings

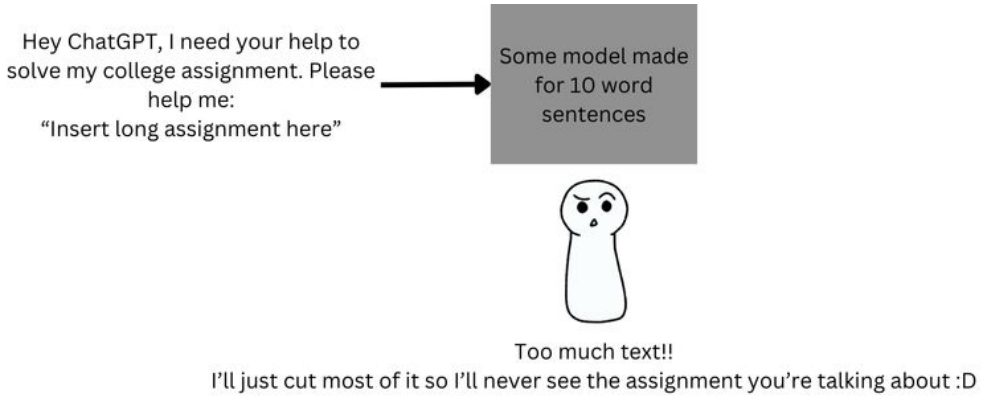
- No matter how many different words our model supports, each of them will be represented as a vector with a lot of zeros and just one **1**
- **Intuition**: would be nice if the embeddings of **cat** and **pet** would be close, in some sense
- One-hot embeddings are: **simple** and **fast**
- One-hot embeddings are not: **informative**, they tell us **nothing** about the words they represent

Embeddings question

What is the **one-hot embedding** of the number 5?

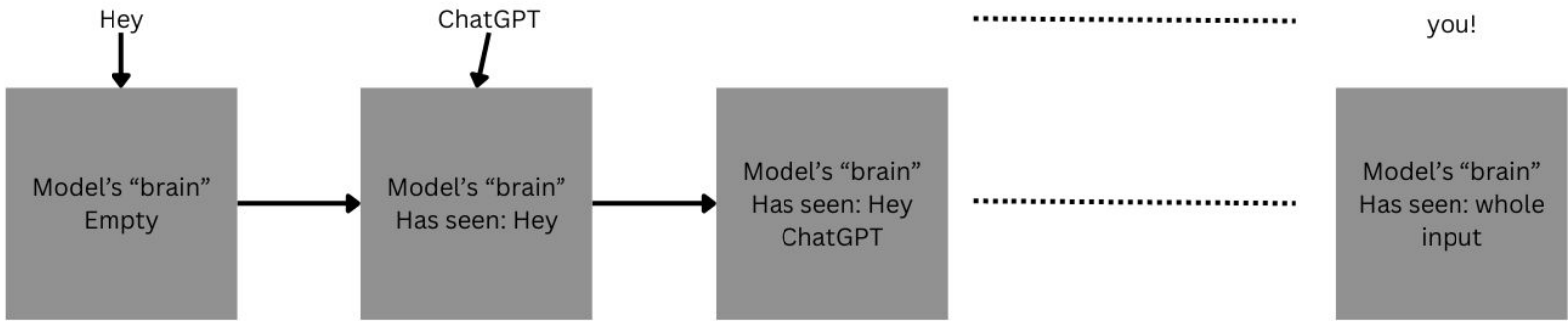
Text models and sentence length

- Basic linear regression models: fixed number on inputs and outputs (population and area as input, profit as output)
- Convolutions: work on many image resolutions, but usually followed by linear regression so they have somewhat fixed input
- Text models: must support inputs and outputs of **any sizes**



Solution: hidden state models

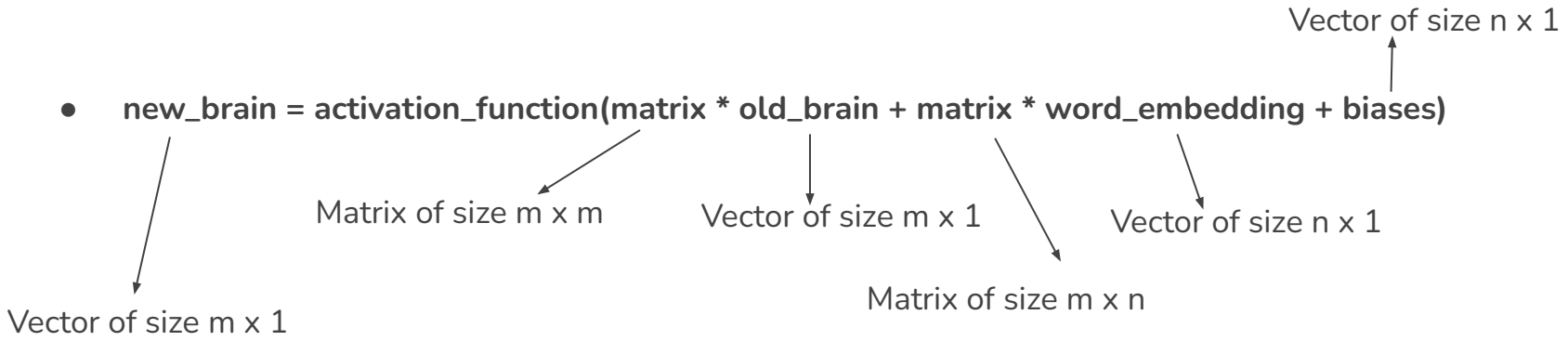
Hey ChatGPT, I need your help to solve my college assignment. Please help me:
“Insert long assignment here”
Thank you!



Modelling the model’s brain

- model’s brain is a **vector of m elements**, for some **m** we decide
- Big **m**: big brain, but slow model
- Small **m**: small brain, fast but dumb

Updating the brain with each word



- General remark: above, the sizes of the matrices are chosen specifically so that the output of the multiplication has the same size as the state. Often in Machine Learning we will be making choices such that sizes match up

Hidden state model / Brain - updating model in PyTorch

- `model = nn.RNN(10, 15)`

  Size of model's brain / memory / hidden state

  Size of input embeddings

- `model(torch.rand(4, 10))` => tensor of shape 4 x 15 and tensor of shape 1 x 15

  4 words represented as vectors of 10 elements

  the 4 hidden states of the model

  the final hidden state of the model

What do we want to do with the model?

- input: [“A”, “cat”, “is”, “an”]
- predict the next word in the sentence
- wanted output: [“cat”, “is”, “an”, **“animal”**]

next word in the sentence

How do we use the values of the hidden state?

- first hidden state obtained from “A”
- Use first hidden state to predict “cat”
- Second hidden state obtained from “A cat”
- Use second hidden state to predict “is”
- Third hidden state obtained from “A cat is”
- Use third hidden state to predict “an”
- Fourth / final hidden state obtained from “A cat is an”
- **Use fourth state to predict “animal”**

How do we predict the next word from the hidden state?

- Hidden state is a vector of shape 1 x n, for some n we decide
- Our model knows a total of m words
- **Multiply** the hidden state by a matrix of shape **n x m**
- Output is a vector of shape **1 x m** => contains **m** values
- [0.01, -0.2, 0.9, 1, 1.3, -1.5, ...]

Largest value, means model predicts word number **4**

Probability next word in the sentence is word number **1**

Probability next word in the sentence is word number **0**

RNN + prediction in PyTorch

```
class RNNAndLinear(nn.Module):
    def __init__(self, input_size, hidden_state_size):
        super().__init__()
        self.rnn = nn.RNN(input_size, hidden_state_size)
        self.linear = nn.Linear(hidden_state_size, input_size)

    def forward(self, sentence, initial_state = None):
        if initial_state is None:
            hidden_states, final_hidden_state = self.rnn(sentence)
        else:
            hidden_states, final_hidden_state = self.rnn(sentence, initial_state)
        output = self.linear(hidden_states)
        return output
```

RNN (with its hidden state)

From hidden state to probabilities of each possible word

More difficulties: incomplete sentences

- Dataset: “RNN is a model with updatable state”
- Input: “RNN is a”
- Expected output: “is a **model**”
- **Model** is a reasonable prediction

- Input: “model with updatable”
- Expected output: “with updatable state”
- **State** is a much less reasonable prediction: our model has only seen “model with updatable” and has **no idea about “RNN is a”**: ***lost context***
- *Parameters* would be a much more reasonable completion for the model

Solution for “losing context”

- Train with “RNN is a”
- Save the final state
- Use final state as initial state when training with “model with updatable”
- `model = nn.RNN(128, 256)`
- `model(torch.rand(32, 10, 128), torch.rand(1, 10, 256))`

Batch of 10 “sentences”, where each sentence contains 32 words, each word being represented as an embedding vector of 128 elements

10 initial states, of 256 elements each
We have 10 sentences, and process each word by word, hence we need 10 initial states

Loss function - CrossEntropyLoss

- Applied for each predicted word
- Penalizes model by `-ln(p)`, where p is the probability of the correct class (in our case, the correct word)
- **Example**: pass the model 5 words. We have 100 unique words. We expect as output the last 4 words that we gave as input + the word that completes (extends) the sentence.
- Assume the indices of the output words are 1, 5, 89, 40 and 36
- Model outputs 5 vectors of 100 elements each (because we have 100 unique words)
- We only care about (and hence penalize the model based on) the first, 5th, 89th, 40th and 36th value in those vectors

Using the model

- **Prompt**: “What is a mountain?”
- Update state with the prompt
- Use state to generate word “Elevated”
- Use “Elevated” to update the state
- Continue however much you like :)
- **Extra question**: how do we know when to stop?

Model state	Model output
Initial zero state	Useless
Has seen “What”	Ignored
Has seen “What is”	Ignored
Has seen “What is a”	Ignored
Has seen “What is a mountain?”	Predicts next word: “Elevated”
Has seen “What is a mountain? Elevated”	Predicts next word: “portion”
Has seen “What is a mountain? Elevated portion”	Predicts next word: “of”
Has seen “What is a mountain? Elevated portion of”	Predicts next word: “earth”