1 Word Embeddings

We have an upgraded word_embeddings.ipynb notebook from last week's recitation.

This version has two $vocab\_size \times 2$ matrices of parameters, U and V, used to encode and decode tokens, respectively. But rather than trying to decode the same word as the input like an autoencoder, we are trying to predict the next word.

Another way to think about these two parameter matrices is U maps *prev* token indices into a 2-dimensional space and V maps *next* token indices into the same 2-dimensional space. In that 2-dimensional space, we can match previous tokens to their corresponding next tokens by using the cosine similarity metric, $\mathbf{u}^\top \mathbf{v}$.

This notebook loads the lines in Green Eggs and Ham and trains our word embeddings model for the corpus from that story. Our goal is to use this as a language model to generate the likely next token, given a specific previous token.

1. Run the notebook and observe the embedded vectors change during the training loop for 100 epochs.

   The notebook then loads the pre-trained word embeddings after 27,000 epochs. Compare the visualization of the purple V next word embeddings after 100 epochs to after 27,000 epochs. What organization do you observe in the trained embedded space?

   Use this simple Word2vec example as a language model to generate new text!

   (a) Current context, e.g., "i will not"
   (b) Previous word: "not" (last word of current context)
   (c) Previous word index from `token_to_vocab_index`
   (d) Use our model to get $\hat{y}$, i.e., the next-word probabilities for each possible vocab word
   (e) Select the most probable next token (`np.argmax`)

   Give an example of some of the text that you generate:

   As a bonus, sample the next token using the `yhat` probabilities (`np.choice`, using the optional argument `p=yhat`). Give an example of some of the text that you generate:

2 MinGPT

MinGPT is a simple implementation of the GPT architecture. In following notebooks, pico version and femto version, we can explore how this model works. For some context, GPT utilizes transformers. A transformer is a type of neural network architecture that focuses on processing sequential data by emphasizing relationships between different parts of the sequence. It achieves this by employing a mechanism called "self-attention," which, in the context of large language models, allows the model to weigh the importance of each word in the prompt (context) when computing representations.

1. Look at the model architect of either pico or femto version of minGPT. Do you see anything familiar?

   Yes. Notice how there are word embedding layers at the beginning of the model. The model also features various linear layers.

2. Compare minGPT pico and minGPT femto architectures. How many attention heads are their in each model? What are the dimension of the word embeddings?

   Note: Having multiple attention heads occurs when the attention mechanism is replicated several times in parallel, each time with different learned parameters. Think of it like having multiple channels in convolutions.

   Femto is the simpler model. It uses 1 attention head. It has 1 layer and an embedding size of 2.

   On the other hand, Pico has 3 attention heads. It has 3 layers, and an embedding size of 6.

3. Now it's time to play around with the model. First, run the untrained pico model on the given context, "I do not like." Generate a sample from the model. What does the model predict?

   The output reads, "I do not like fox say with say Let dark , eggs fox do Not , fox . mouse eggs fox Sam say Let anywhere Eat fox Here fox fox Sam do with with with with with with with with with with with with with with with with with with with with with with".

   The model prediction is pretty bad, as expected from an untrained model!

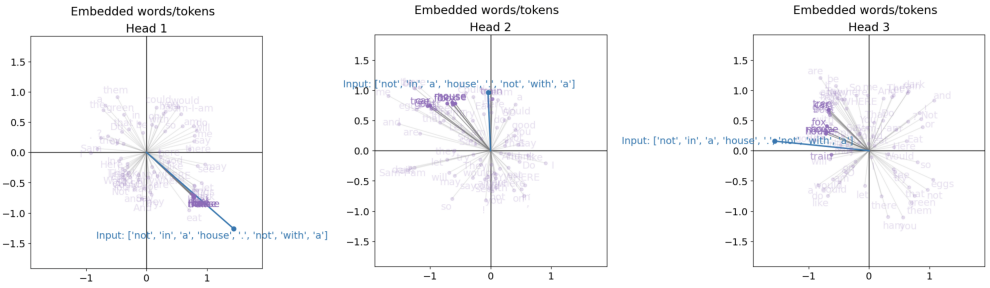
4. Now, train the model for 100 iterations. Do the word embeddings look the same for the vocab? Does the word embedding look the same for the context?

   After training for 100 iterations, the embeddings look very different than the ones with zero training. Both the vocab embeddings and the context embedding have shifted significantly.

5. Now, let's use the pretrained model on 10,000 iterations and run it with context = "not with in a house , not with a." Sample from this model. What does the model predict? What do the word embeddings for each attention head look like?

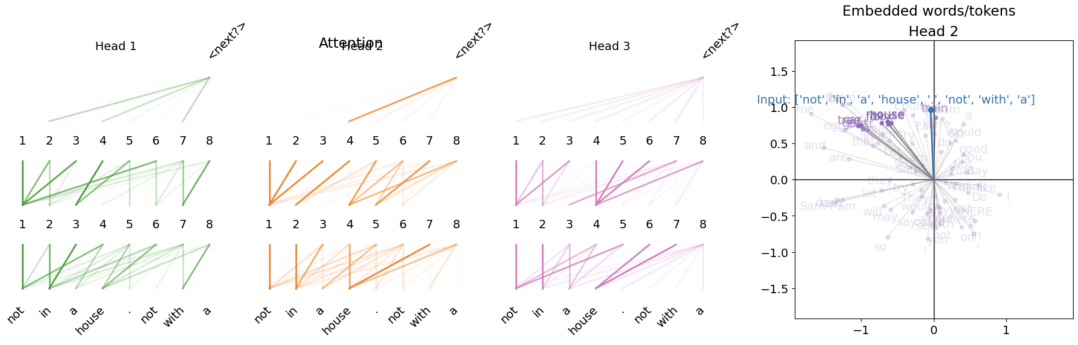
The model outputs, "not with in a house , not with a house . I will not eat them here or there . I will not eat them anywhere . I will not eat them anywhere . I do not like them anywhere . I do not like them , Sam-I-am . Would you , Sam-I-am . Would you , Sam-I-am ."

The word embeddings look pretty good. All three attention heads seem to have found words that would likely come next given the context.



6. For the pretrained model run from the previous part, which attention head appears to be picking up on rhyming?

Head 2



7. Now, compare the results from minGPT pico and minGPT femto. Which model is better?

Pico performs much better. Femto is a very small model and does not appear to be learning very successfully.

Finally, in the notebooks, explore using either model how it generates new text!

3 Naive Bayes

Naive Bayes is a probabilistic algorithm that uses Bayes' theorem to classify data. Reminder that Bayes Theorem states

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}$$

In addition to Bayes Theorem, Naive Bayes makes use of something called the Naive Bayes assumption, which states that features of a data point are conditionally independent given the class label. That is, all X_i are conditionally independent given Y or

$$P(X_1, X_2, \ldots, X_i \mid Y) = \prod_{i=1}^{I} P(X_i \mid Y)$$

The **Naive Bayes Algorithm** works as follows:

1. Calculate the prior probability of each class label, $P(Y = y)$.
2. For each feature X_i of each class label, calculate the conditional probability $P(X_i \mid Y = y)$.
3. Given a new data point X, calculate the posterior probability of each class label using Bayes' theorem and the conditional probabilities from step (b). The class label with the highest posterior probability is the predicted class label.

   In other words, assuming there are M features, the classifier can be written as

$$\underset{y}{\operatorname{argmax}}\, P(Y = y) \prod_{i=1}^{M} P(X_i \mid Y = y)$$

Now, let's walk through an example where our goal is to classify whether an email is SPAM or not. Consider the following training samples:

SPAM	Email Body
1	Money is free now
0	Pat teach 315
0	Pat free to teach
1	Sir money to teach
1	Pat free money now
0	Teach 315 now
0	Pat to teach 315

The vocabulary consists of the following words: {315, free, is, money, now, Pat, Sir, teach, to, tomorrow}. The words in the vocabulary are the features. Compute the following probabilities:

1. Fill in the tables below:

$P(Y = 1)$	$P(Y = 0)$

$P(Y = 1) = 3/7$

$P(Y = 0) = 4/7$

j	$P(X_j = 1 \mid Y = 1)$	$P(X_j = 1 \mid Y = 0)$
315	0/3	3/4
free	2/3	1/4
is	1/3	0/4
money	3/3	0/4
now	2/3	1/4
Pat	1/3	3/4
Sir	1/3	0/4
teach	1/3	4/4
to	1/3	2/4
tomorrow	0/3	0/4

2. Consider the following email body: X = Pat teach now. Fill in the table below:

j	x	$P(X_j = x \mid Y = 1)$	$P(X_j = x \mid Y = 0)$
315	0	3/3	1/4
free	0	1/3	3/4
is	0	2/3	4/4
money	0	0/3	4/4
now	1	2/3	1/4
Pat	1	1/3	3/4
Sir	0	2/3	4/4
teach	1	1/3	4/4
to	0	2/3	2/4
tomorrow	0	3/3	4/4

3. Reminder that with naive Bayes, $P(Y, X_1, \ldots, X_M) = \prod P(X \mid Y)P(Y)$

$P(Y = 1, X_1, ..., X_M)$	$P(Y = 0, X_1, ..., X_M)$

$P(Y = 1, X_1, ..., X_M) = \prod P(X \mid Y = 1)P(Y = 1)$
3/3 * 1/3 * 2/3 * 0/3 * 2/3 * 1/3 * 2/3 * 1/3 * 2/3 * 3/3 * 3/7
=0

$P(Y = 0, X_1, ..., X_M)$
1/4 * 3/4 * 4/4 * 4/4 * 1/4 * 3/4 * 4/4 * 4/4 * 2/4 * 4/4 * 4/7
=0.01004

4.

$P(Y = 1 \mid X_1, ..., X_M)$	$P(Y = 0 \mid X_1, ..., X_M)$
0 / (0.01004 + 0) = 0	0.01004 / (0.01004 + 0) = 1

5. Is this email classified as SPAM or not?

No