**What does the embedding layer do?**

[M](https://www.coursera.org/learn/neural-networks/profiles/72252698753cf95d937e2fef376f9b9b)

Jonathan MerrimanMentor · [9 months ago](https://www.coursera.org/learn/neural-networks/discussions/weeks/5/threads/WRem1e_aEeatJhLWwJKM8g/replies/Cl06UvCEEeaGLRKpFVYwTg)

The embedding layer actually reduces dimensionality from the original bag-of-words or one-hot encoding representations. In practice, by tens of thousands or more into hundreds. It creates a bottleneck that forces words to share dimensions to encode syntactic and semantic information, which puts similar words nearby each other (in terms of cosine distance).

The embedded representations are learned by setting up a prediction task; the prediction task influences what information the embedding representations would need to encode. Predicting words given a context or vice versa does not require hand-labeling of data and yet give generally useful embedding representations. You can set up more specialized tasks and reuse pre-trained vectors as well.

**6 Upvotes**Hide 1 Reply

[DC](https://www.coursera.org/learn/neural-networks/profiles/22b262ed05c4be3c9983e1801de354d0)

Dan Chernoff · [9 months ago](https://www.coursera.org/learn/neural-networks/discussions/weeks/5/threads/WRem1e_aEeatJhLWwJKM8g/replies/Cl06UvCEEeaGLRKpFVYwTg/comments/YFgfYPCVEea15xLglo07cA)

OK, that's very helpful and obviously my intuition about the embedding layer was wrong. If I understand your second paragraph correctly, you do need a prediction task to train the embedding layer; the end-to-end task of word prediction is a form of supervised learning but the data does not need to be hand-labeled; and it is perfectly possible to reuse the weights in the embedding later for other tasks, but whether that's a good idea depends on whether the new task is related to the same language corpus as the original.

**1 Upvote**

**How does the embedding layer work?**

Let us assume that an embedding layer has a dimension of 250 x 50. Here, every word is encoded in a 50-dimensional vector. In other words, every row represents an encoding of 50 values that collectively represent a word.

**Epoch:** One pass of the network through the entire training set

**Gradients**: column-wise sum of incoming unit’s derivates

**1st backpropagation derivative**

**2nd backpropagation derivative** = weights b/w hid. & emb. layer 1st backpropagation derivative

## Assignment 2

**Reference code**: fprop.m

**Code snippet**: embedding\_layer\_state = reshape( word\_embedding\_weights ( reshape ( input\_batch, 1, []), : )', numhid1 \* numwords, [] );

Let us start constructing the embedding layer. We have an input dimension of (250x50). This is the word\_embedding\_weights matrix. We know that we have 250 words in the vocab. In this matrix, every word is encoded in a 50-dimensional vector of weights. In other words, every row represents an encoding of 50 values that collectively represents a word. This is our vocab lookup table.

Now, one batch of train\_input (3x100) has 100 samples where every sample contains 3 words (their indices) such that row 1 is word 1, row 2 is word 2 etc. and every column is a different combination of this 3-gram. We can flatten this matrix into a (1x300) vector of words. Note that this vector will contain many duplicates because the same word may appear multiple times in different 3-grams (training samples). We have rearranged our training input here.

Next, we perform the word vs weight lookup. To do this, we simply fetch the weights of each word from our training sample by matching its indices with the row numbers of the lookup table above (e.g. row 10 stores the weights of word 10 in the vocab). We now have a (50x300) matrix that contains 50 weights for each word in our training batch. In short, we have an encoded representation of our training input.

Finally, we reconstruct (reshape) this (50x300) matrix into a (150x100) matrix and feed this as input to the hidden layer above. If I understand correctly, this is a more efficient representation of the embedding layer that also conforms with the structure of the hidden layer.

Jean Tsao · [6 months ago](https://www.coursera.org/learn/neural-networks/discussions/weeks/5/threads/ZM-fWBOhEeeskRI8P5CzrA/replies/htnBii_FEeeqRw77WotmkA)

Tomer,

Here are some thoughts:

yiis also a function of all of the other zk’s from all other branches, not just one branch where yiis located. Cross entropy is across the entire set of output classes (e.g. vocabsize in Programming Assignment #2)

(1) C = - ∑j=1:n tj \* log(yj)

(2) ∂C/∂zk = - ∑j=1:n tj \* ∂log(yj)/∂zk

(3) We need to find expression for ∂log(yj)/∂zk

(4) Starting with yj = zj/∑I exp(zi)

(5) Take log of yj and we obtain log yj =zj –log(∑I exp(zi)

(6) Take partial derivative of (5)

∂log(yi)/∂zk= δjk- 1 /∑I exp(zi) \* ∂(∑I exp(zi))/ ∂zk

where δjk is kronecker\_delta\_function that equal 1 when j =k and 0 otherwise.

(7) The last partial derivative of (6) is ∂(∑I exp(zi))/ ∂zk= ∂(∑I exp(zi)\*δjk= exp(zk)

(8) So (6) now becomes ∂log(yi)/∂zk= δjk- (1 /∑I exp(zi) )\* exp(zk)

(9) But (1 /∑I exp(zi) )\* exp(zk) in (8) is just expression for yk

(10) So ∂log(yi)/∂zk= δjk - yk

(11) Plug expression (9) for ∂log(yi)/∂zjinto ∂C/∂zjin (2) we obtain

(12) ∂C/∂zk = ∑j=1:n tj \* (yk– δjk)

= (yk\*∑j=1:n tj)-(∑j=1:n tj\* δjk)

(13) ∑j=1:n tj in the first term yk\*∑j=1:n tj is 1. So the first term becomes yk

(14) Because δjk in the second term is 1 and not zero only when j = k, the second term becomes tk

(15) Plug (12) and (13) back into (11), we obtain ∂C/∂zk = yk- tk

What do you think?

2 UpvotesHide 1 Reply

[VS](https://www.coursera.org/learn/neural-networks/profiles/7f303dc3d0ce9930e725101fe66f3b80)

Vinay Kumar Sisodia · [5 months ago](https://www.coursera.org/learn/neural-networks/discussions/weeks/5/threads/ZM-fWBOhEeeskRI8P5CzrA/replies/htnBii_FEeeqRw77WotmkA/comments/KHWHZ0NdEee1HRJ1vKtk8A)

Hi Jean, I guess in step 4, the correct expression should be:

yj = exp(zj)/∑I exp(zi)

and not yj = zj/∑I exp(zi)

Thanks for the explanation by the way!