Topic 4: Hidden representations

The hidden layers of a neural network are typically very difficult to interpret and for this reason we say that neural networks are black box models, that is, you can't inspect what it is that they're doing internally to come up with their output. However, even if we cannot do a perfect job of inspecting the inner representations, we can still manipulate them to our advantage.

1. Word embedding

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow/04_-_Hidden_representations/01_-_Word_embedding.py

How can you feed a word or several words to a neural network as an input (say, to predict the next word in a sentence)? Neural networks can only work with numbers. Will you express the individual letters as a vector of Unicode numbers? That would imply that Unicode 97 ('a') is equally similar to Unicode 98 ('b') and 96 ('`').

The most common way to encode words (or letters) is to use an embedding matrix with a row vector for every word (or letter). Each word is given an index number which would be the row number in the embedding matrix. In order to keep the number of words finite, we choose a fixed vocabulary made from frequent words found in the training set. We typically exclude rare words as it's difficult to infer the meaning of a rare word.

Any word that is not part of the vocabulary is replaced with a pseudo-word called an unknown token. This allows us to use words that were not in the vocabulary when using the neural network after training.

The embedding matrix is trained just like any other weight matrix in the neural network and an interesting thing about it is that, after training, the similarity between two word vectors tends to give hints on the similarity between the two words. In fact, neural networks are constructed and trained just to be able to extract this embedding matrix as it is very useful information about the words. An example of this is Word2Vec, which is a neural network that is trained to predict the middle word from its neighbouring words in a sequence of words taken from a corpus, or predict the surrounding word from the middle word. See this paper for more information on Word2Vec.

2. Transfer learning

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow/04_-_Hidden_representations/02_-_Transfer_learning.py

Of course, we don't just collect word vectors for linguistic reasons, we also do it to pass on the embedding matrix into other neural networks. If you have to train a neural network to perform a task such as part-of-speech tagging (classify if a word is a noun, a verb, etc.) then you are likely to have a relatively small dataset of tagged words. This makes it difficult for the neural network to infer the meaning of the words. On the other hand, it is very easy to find lots and lots of unlabelled text to train a neural network to predict the middle word from a sequence of words. The embedding matrix learned by the second neural network is likely to be very high quality and it would be nice to copy over the knowledge encoded in it to other neural networks.

To do this we can initialise the embedding matrix of the part-of-speech tagger (called the target model) with the embedding matrix extracted from the pre-trained word predictor (called the source model). The embedding matrix can then either be kept as a constant (freezing the parameters) in order to prevent it from changing during training of the target model or can be kept as a variable (fine-tuning the parameters) in order to allow it to be optimised during training. Freezing is useful to avoid overfitting whilst fine-tuning is useful to make the representation specialise to the new task

This process of transferring parameters between tasks or data domains is called transfer learning and is one of the most important fields in deep learning as it allows you to reuse pre-trained neural networks to train new ones. It also makes sense to train a model starting from existing knowledge rather than from randomly set parameters as that is how we usually learn (you wouldn't start from zero every time you wanted to learn something new).

3. Multi-task learning

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow/04_-_Hidden_representations/03_-_Multi-task_learning.py

Rather than extracting the parameters from a pre-trained neural network, we can instead train one neural network on two datasets. Of course, the neural network would need separate inputs and outputs for each task, but the hidden layers can be shared, that is, both tasks make use of the exact same parameters internally (although not all parameters need to be used for both tasks). This means that a particular hidden layer would be trained to create useful features for two tasks at once rather than for a single task, which would make it harder for the model to overfit since it will need to overfit on two different tasks.

This is called multi-task learning. One very interesting use of multi-task learning was in a single neural network that performs image object recognition, image description generation, text translation, part-of-speech tagging, and speech recognition. Another example is a single neural network that can translate between many different languages at once.