Zaid Al-Shayeb

Professor Shanker

Advanced Machine Learning

5 December 2022

Word Embedding

Summary

We can use a neural network on a supervised task to learn word embeddings. The embeddings are weights that are tuned to minimize the loss on the task. For example, given 50K words from a collection of movie reviews, we might obtain a 100-dimensional embedding to predict sentiment. Words signifying positive sentiment will be closer in the vector space. Since embeddings are tuned for a task, selecting the right task is important.

Word embeddings can be learnt from a standalone model and then applied to different tasks. Or it could be learnt jointly with a task-specific model. For good embeddings, we would need to train on millions or even billions of words. An easier approach is to use pretrained word embeddings (word2vec or GloVe). They can be used "as is" if they suit the task at hand. If not, they can be updated while training your own model.

Predictive neural network models (word2vec) and count-based distributional semantic models (GloVe) are different means to achieve the same goal. There's no qualitative difference. Word2vec has proven to be robust across a range of semantic tasks.

For syntactic tasks such as named entity recognition or POS tagging, a small dimensionality is adequate. For semantic tasks, higher dimensionality may prove more effective. It's also been noted that pretrained embeddings give better results. It's been commented that 8 dimensions might suffice for small datasets and as many as 1024 dimensions for large datasets.

More generally, it's been said the deep learning isn't sample efficient. Perhaps we need something better than deep learning to tackle language with compositional properties.

Problems

Current word embedding algorithms tend to identify synonyms efficiently. As a result, the vectors for the words house and home share a cosine similarity of 0.63, what means that they are alike to some extent. Thus, the vectors for like and love are expected to be similar too. Nevertheless, they show a cosine similarity of just 0.41, what is surprisingly low. That is because the word like is not only a verb but also, a preposition, an adverb, and even a noun. In other words, all these terms are homographs: different words sharing the same spelling. Since there is no way to distinguish between these identical words, the vector used for the word like must include all the contexts where the word appears resulting, then, in an average of all vectors. That is why the vector for like is not as close to love as expected. When put into practice, this reality can significantly impact on the performance of ML systems posing a potential problem for conversational agents and text classifiers.

Another problem challenging standard word embeddings are word inflections (alterations of a word to express different grammatical categories). When looking, for instance, at the verbs find and locate they present a similarity of 0.68, almost as close as expected. However, if the inflected forms (past tense or participle, for example) of those verbs are compared, an unusual similarity of 0.42 between found and located comes up. That's because some word inflections appear less frequently than others in certain contexts. As a result, there are fewer examples of those 'less common' words in context for the algorithm to learn from them resulting, therefore, in 'less similar' vectors. For all that, a far bigger issue emerges when using languages with a greater level of inflection. While English verbs may have a maximum of 5 different forms (e.g., go, goes, went, gone, going), Spanish verbs present over 50 inflections and Finnish over 500. No matter how large these amounts of training data are, there will not be enough examples of the 'less common' forms to help the algorithm generate useful vectors.

Techniques

Term frequency-inverse document frequency is the machine learning algorithm that is used for word embedding for text. It comprises two metrics, namely term frequency (TF) and inverse document frequency (IDF). This algorithm works on a statistical measure of finding word relevance in the text that can be in the form of a single document or various documents that are referred to as corpus. The term frequency (TF) score measures the frequency of words in a particular document. In simple words, it means that the occurrence of words is counted in the documents. The inverse document frequency or the IDF score measures the rarity of the words in the text. It is given more importance over the term frequency score because even though the TF score gives more weightage to frequently occurring words, the IDF score focuses

on rarely used words in the corpus that may hold significant information. TF-IDF algorithm finds application in solving simpler natural language processing and machine learning problems for tasks like information retrieval, stop words removal, keyword extraction, and basic text analysis. However, it does not capture the semantic meaning of words efficiently in a sequence.

A bag of words is one of the popular word embedding techniques of text where each value in the vector would represent the count of words in a document/sentence. In other words, it extracts features from the text. We also refer to it as vectorization.

Word2Vec method was developed by Google in 2013. Presently, we use this technique for all advanced natural language processing (NLP) problems. It was invented for training word embeddings and is based on a distributional hypothesis. In this hypothesis, it uses skip-grams or a continuous bag of words (CBOW). These are basically shallow neural networks that have an input layer, an output layer, and a projection layer. It reconstructs the linguistic context of words by considering both the order of words in history as well as the future. The method involves iteration over a corpus of text to learn the association between the words. It relies on a hypothesis that the neighboring words in a text have semantic similarities with each other. It assists in mapping semantically similar words to geometrically close embedding vectors. It uses the cosine similarity metric to measure semantic similarity. Cosine similarity is equal to Cos(angle) where the angle is measured between the vector representation of two words/documents.

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. The advantage of GloVe is unlike Word2Vec that relies on local context information of words, but also incorporates global statistics (word co-occurence, count-based models) to obatin word vectors.

Conclusions

We explored the evolution of neural-based modelling for NLP or machine translation. We covered word embedding, a method to find semantic, syntactic and linear relationship in the vocabulary. Although some of the methods mentioned earlier are no longer used, they lay the basic foundation of the subject, and make further learning easier.

In contrast, some experiments were done using different combinations of these vectors. As a result, it was found that the sum between these vectors can introduce additional knowledge, but doesn’t always provide a better result. Also, from geometrical interpretation, we know that the summer and averaged vectors will behave almost the same.

Furthermore, there's no evidence for using concatenation operations between vectors. Finally, we recommend using only word embeddings, without context vectors, since this is a well-known practice and context vectors won't give significantly better results. However, for research purposes, the sum or average might be worth exploring.

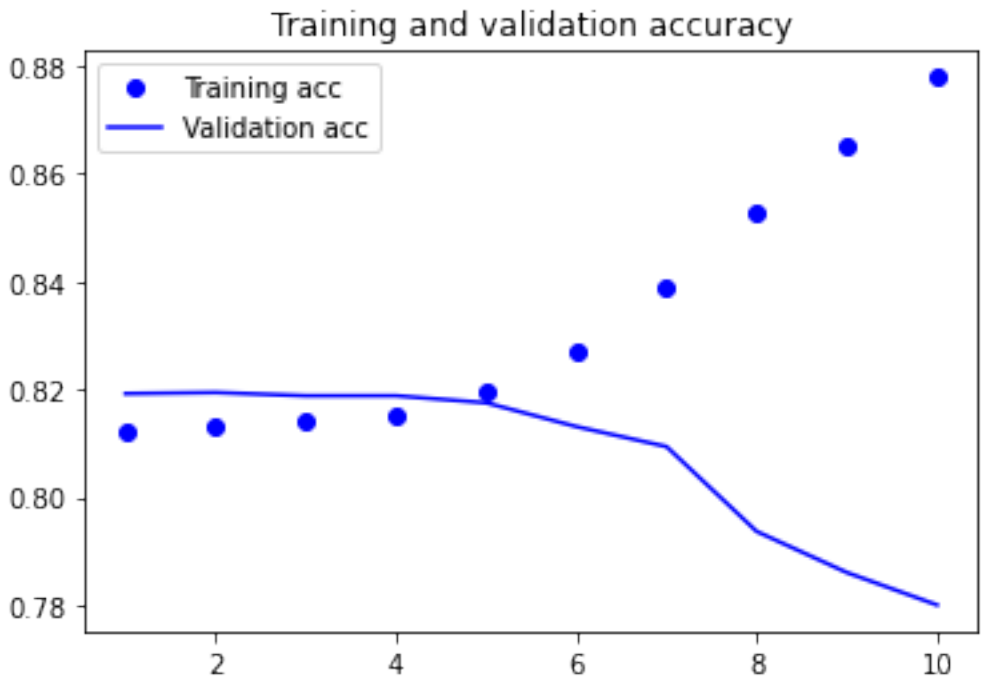
Word Embeddings are an important part of text interpretation. A robust Word Embedding model is essential to be able to understand the dialogues in a contact center and to improve the agent and customer experience. In Agent Assist, Word Embeddings help us understand the meaning of each word, which can be used to recommend articles, suggest automations, and enable more features based on the dialogue meaning.

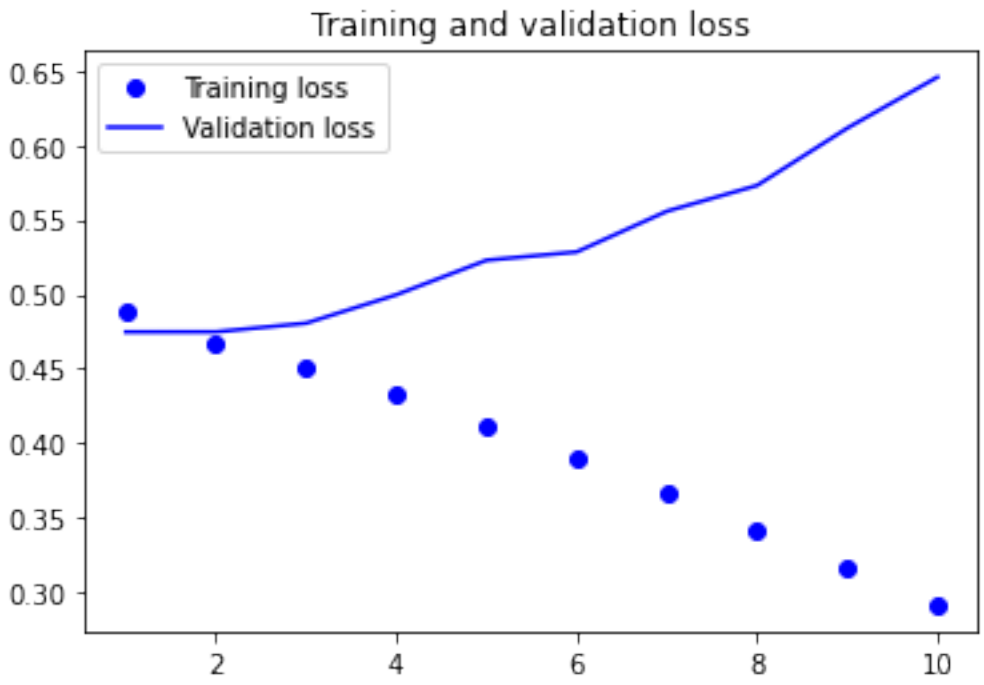
Contributions

All modern NLP techniques use neural networks as a statistical architecture. Word embeddings are mathematical representations of words, sentences and (sometimes) whole documents. Embeddings allow us to compare text, visualize it, and improve the accuracy of newer models. Word Vectors are often used as a fundamental component for downstream NLP tasks, e.g. question answering, text generation, translation, etc., so it is important to build some intuitions as to their strengths and weaknesses. Here, you will explore two types of word vectors: those derived from co-occurrence matrices, and those derived via word2vec. Word2vec is a framework for learning word vectors. It's a very simple and scalable way of learning vector representations of words. Ultimately, we want a model that gives a reasonably high probability estimate to all words that occur in the context (fairly often). The terms "word vectors" and "word embeddings" are often used interchangeably. The term "embedding" refers to the fact that we are encoding aspects of a word's meaning in a lower dimensional space.

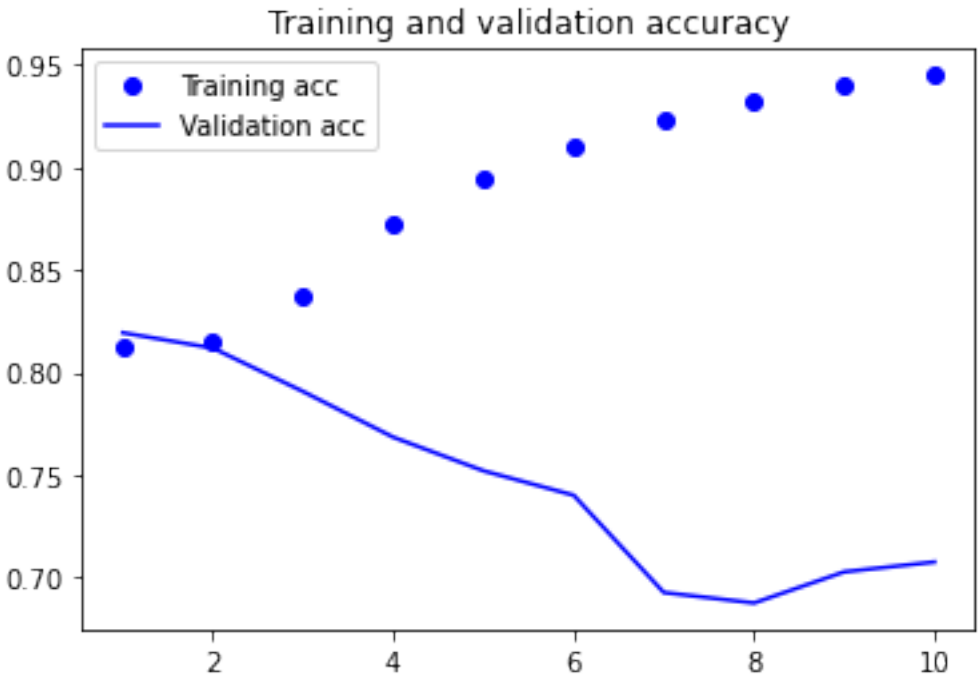
Code

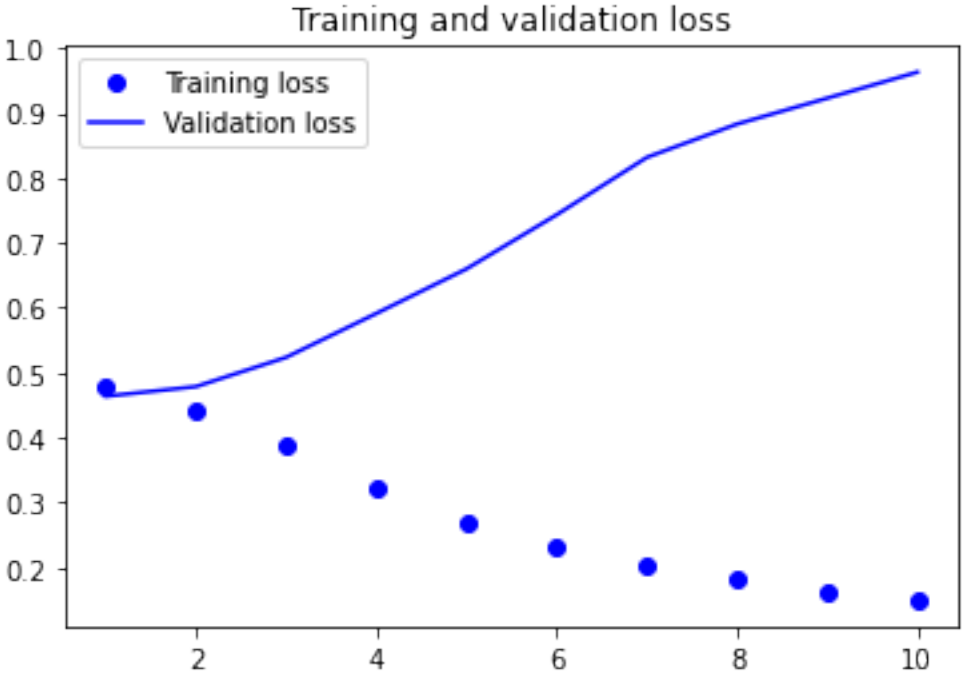
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



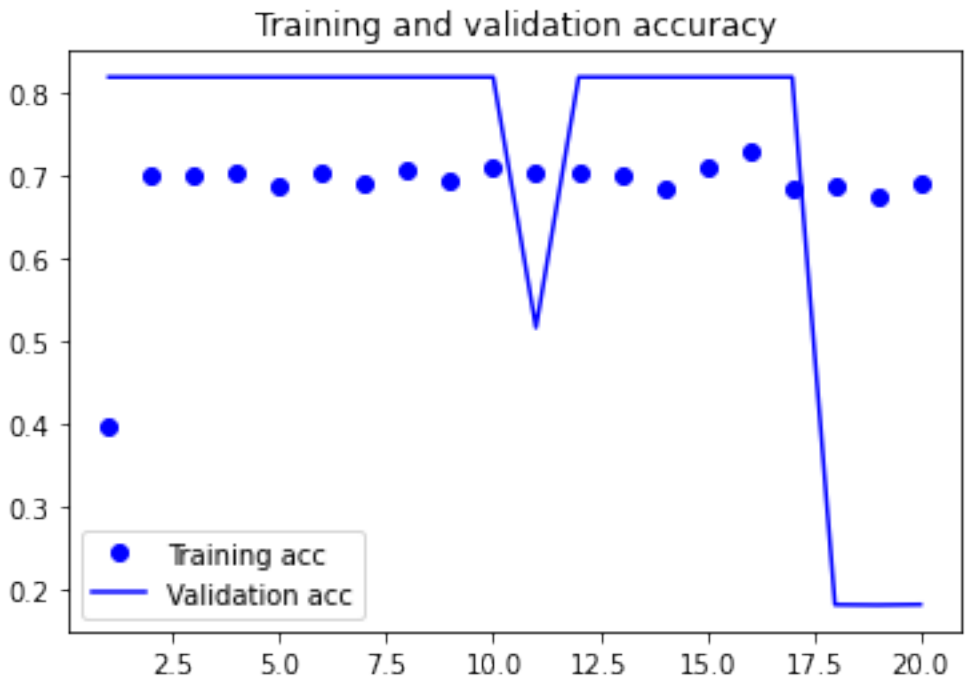


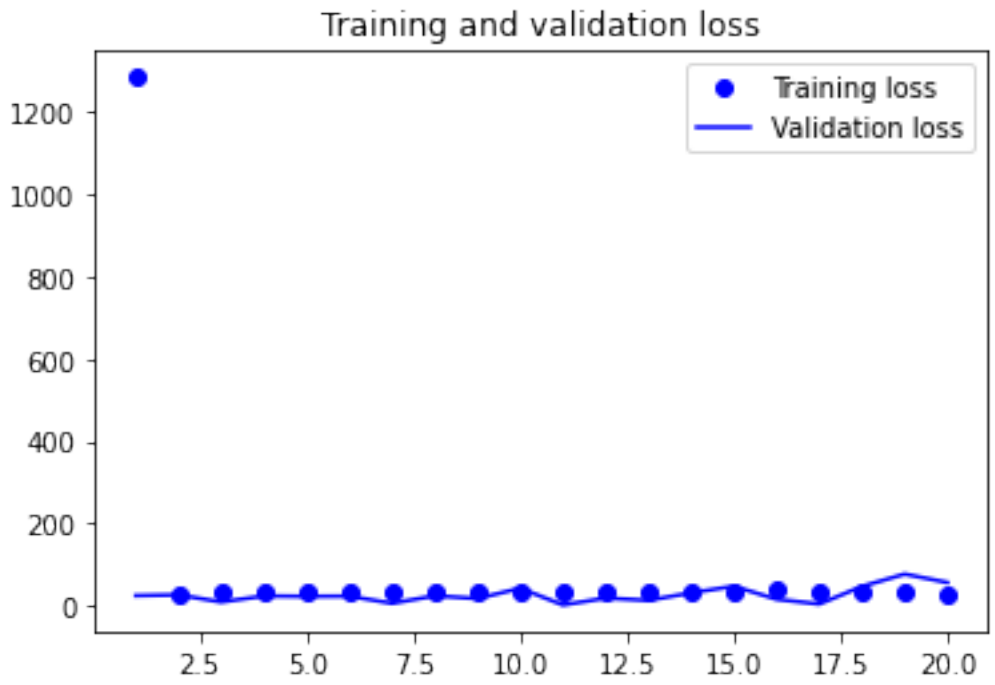

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.95
0.90
0.85
0.80
0.75
0.70
2
4
6
8
10

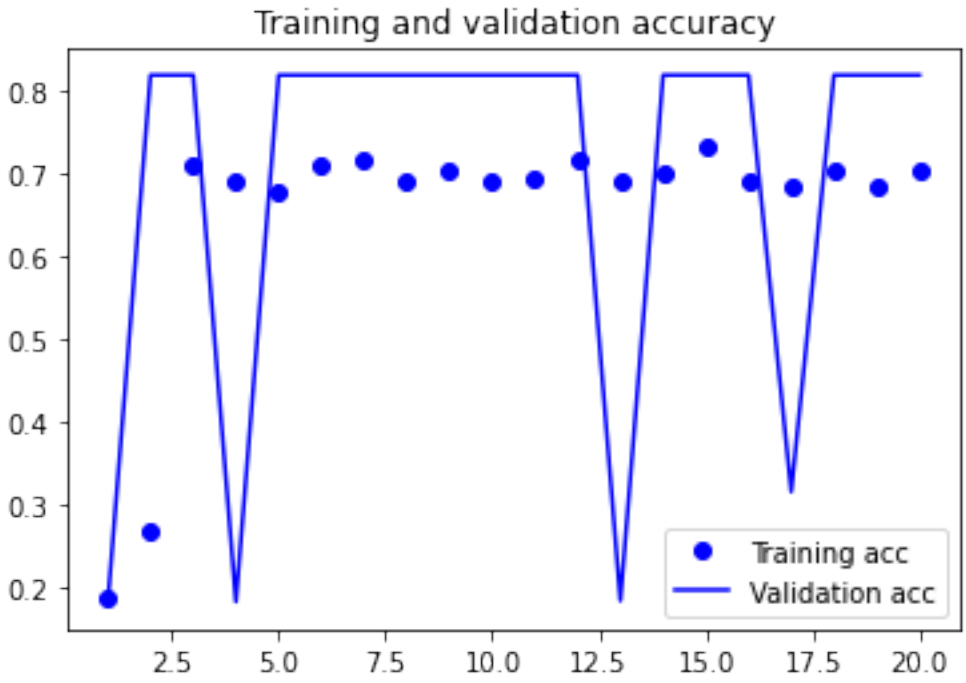

Training and validation loss
Training loss
Validation loss
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10

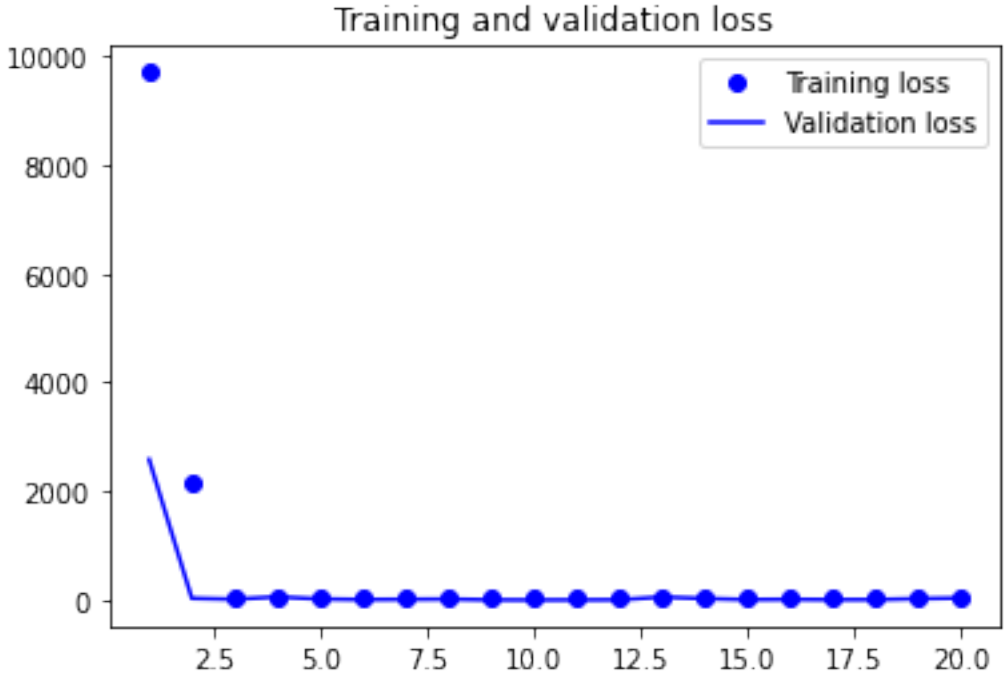
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



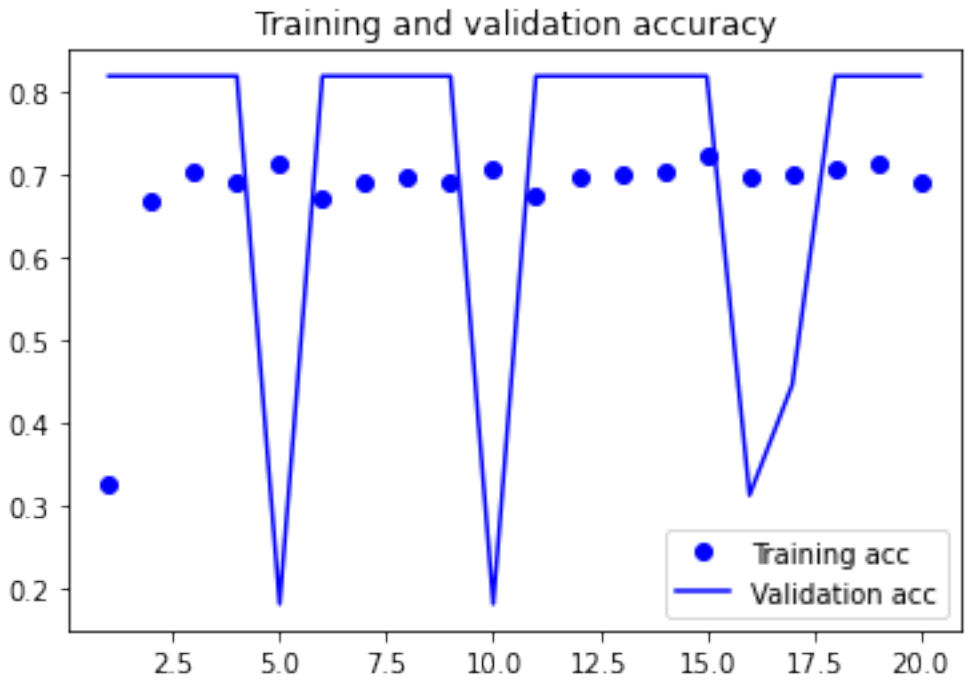


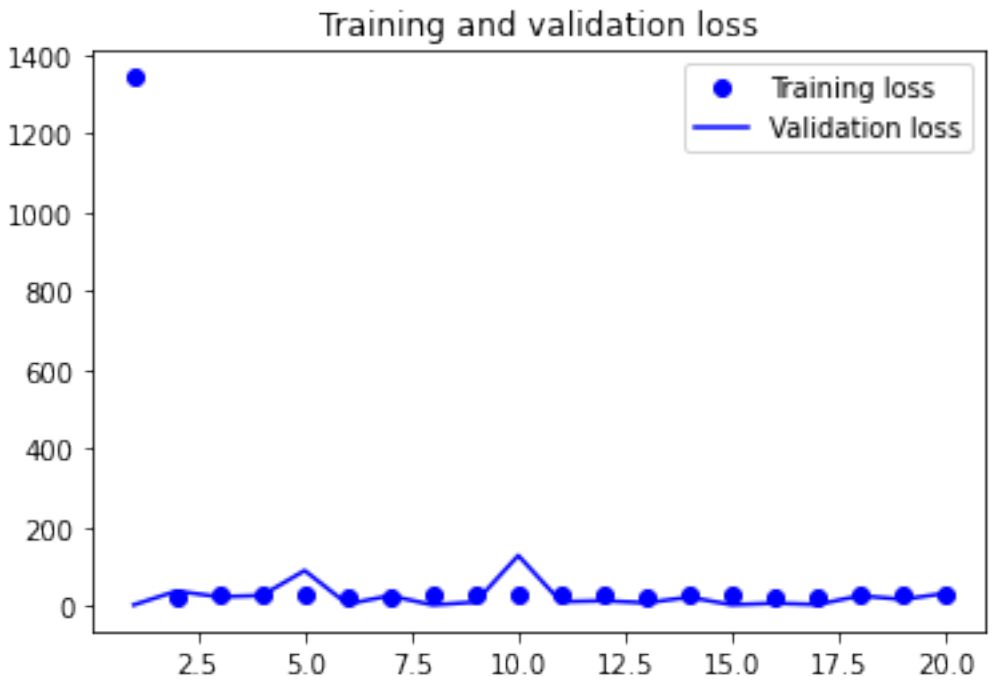

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training acc
Validation acc

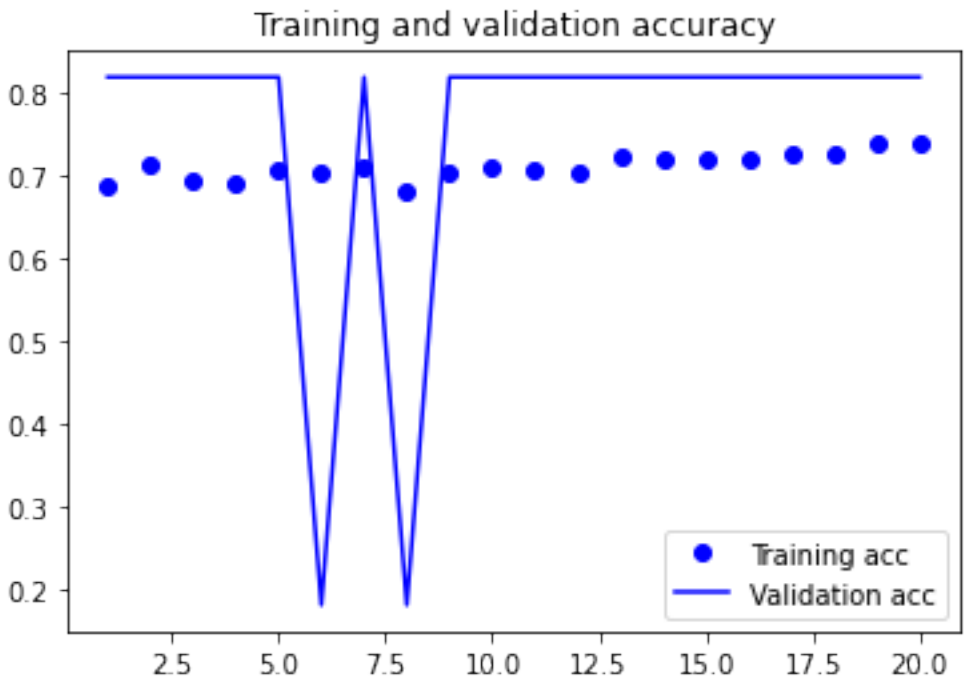

Training and validation loss
10000
8000
6000
4000
2000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training loss
Validation loss

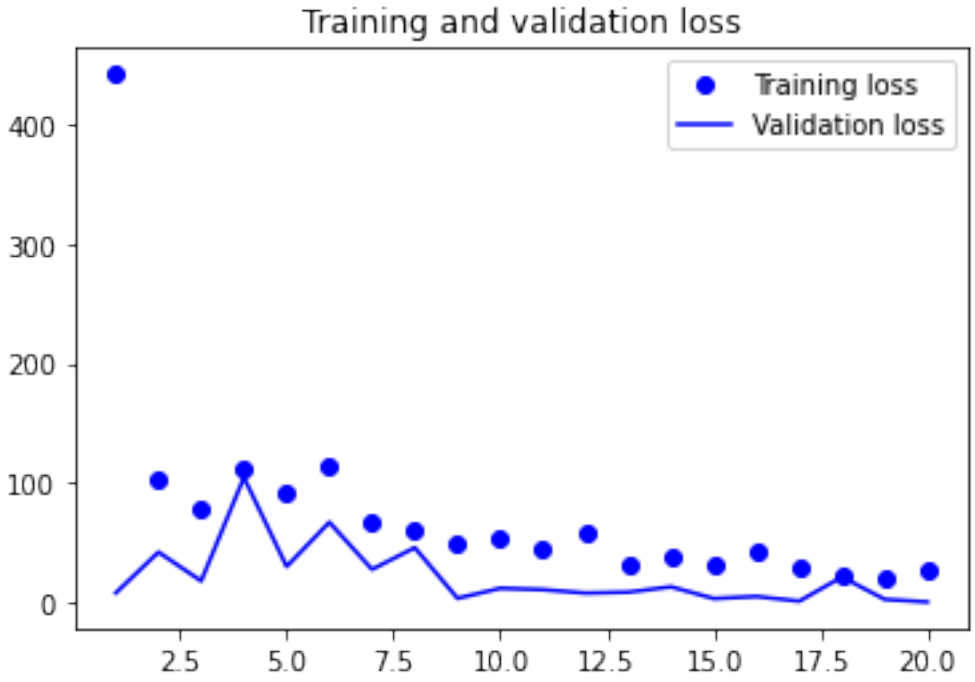
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```






```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training acc
Validation acc


Training and validation loss
400
300
200
100
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training loss
Validation loss