Mass Spectrum Protein Matching with Neural Networks

Group 7

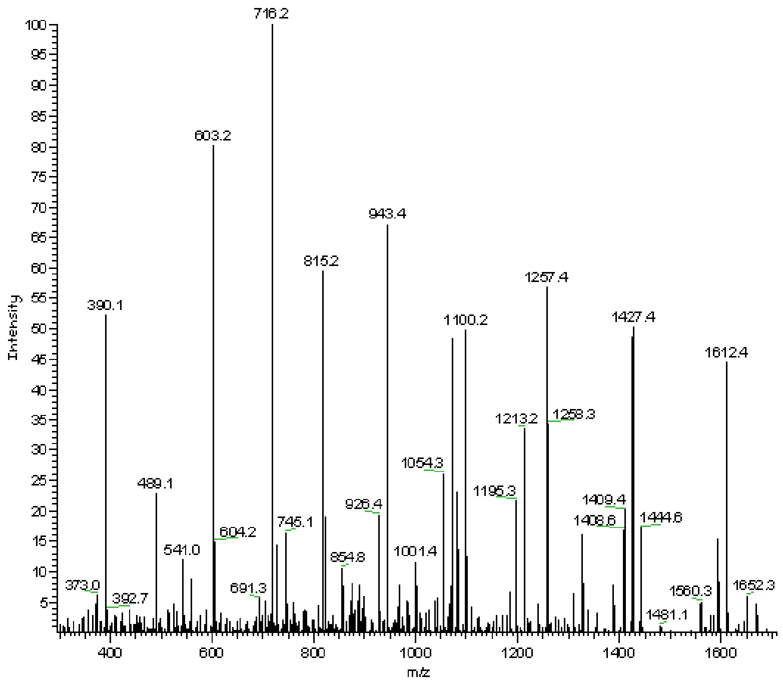
Jonah Kubath
Matt Peter

Abstract

Increasing the rate at which we can identify proteins via their peptide sequences is one of the many computational problems present in the world today. Over the past few decades, protein mass spectrometry has become the primary way of identifying proteins. However, due to the complexity and quantity of proteins, as well as the large amount of “noise” that comes with gathering data, identifying one with an extremely high degree of certainty quickly becomes a daunting task. To solve this problem, large amounts of research have been put into finding ways to decrease the amount of time it takes to identify proteins, while still maintaining a decent level of accuracy. For our project, we decided to specifically focus on using mass-to-charge ratios to assist a neural network in labeling mass spectrometry data with the given protein. In the end, with the limited time and resources we had, we achieved promising results that showed that neural networks would be something to perform further research in when it comes to protein identification.

1. Protein Background

In the field of proteomics, “the study of all proteins in a biological system”, mass spectrometry is a technique that is used “to detect, identify and quantitate molecules based on their mass-to-charge (m/z) ratio” [1]. In order to do this, a protein must first be broken down into a set of peptides using enzymes. After this is done, the first stage of mass spectrometry (MS1) takes place, where peptides are further broken down into ions using an ion source and are separated by m/z ratio. In the second stage of mass spectrometry (MS2), ions of a particular m/z ratio are selected and fragmented, creating fragment ions which are then separated and detected [2]. This collection of fragment ions can then be quantified in MS2 data as “spectra”, a collection of “peaks” that shows the ions’ corresponding intensities at various m/z ratios.

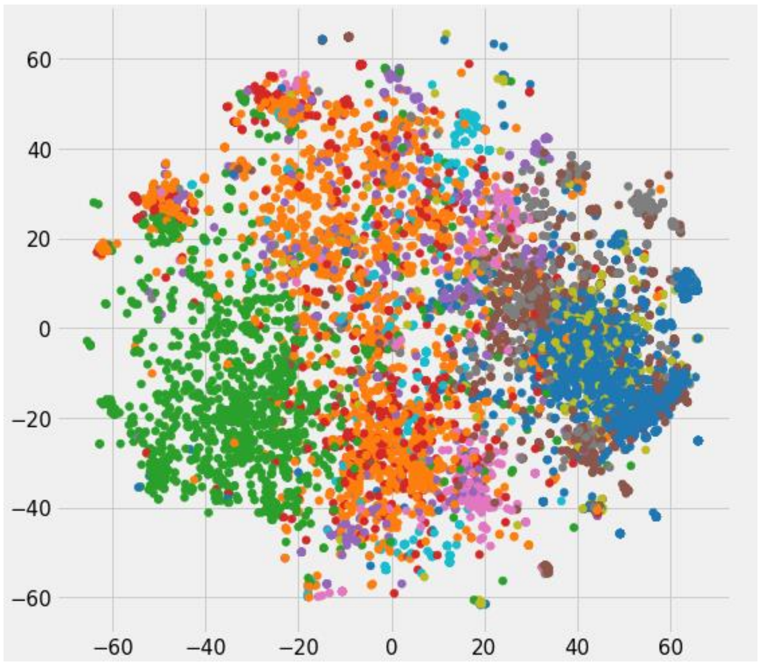


MS2 Data

This MS2 data is then analyzed in order to determine what the original peptides and proteins were. Unfortunately, only a small percentage of the peaks in a spectrum are useful in determining its peptide, as lot of the peaks are either noise or simply not helpful for the given spectrum [3]. This means that a large portion of the time spent analyzing the spectra does not actually improve the end result. In an effort to reduce the amount of time needed to accurately identify a peptide, we've implemented an embedding layer in an attempt to group similar peptides based on their MS2 data.

2. Neural Network Embedding Layer

The main layer of the neural network implemented in this project is called the embedding layer. The embedding layer works to represent the input "words" in a vector space. As more examples are learned, this layer will then move the words around in the vector space so that similar words are closer together. This technique of word embeddings has been used in other areas of research such as document classification or finding similar words. As an example, when embeddings layers are used in regular text documents, the words "king" and "queen" should be close to each other in the vector space as they are commonly used in similar sentences/situations.



2D Embedding Layer Visual Example

The goal of using an embedding layer in this project is to train it to the point where it can take a peptide sequence and identify a set of other peptide sequences that it relates to. In order to implement an embedding layer in code, we use the Keras API which is built on top of TensorFlow and offers an embedding layer function. With the Keras Tokenize class, the “words” in the sentences are tokenized, converted from string to integer values, padded to the length of the longest m/z ratio string, and then fed into the embedding layer.

3. Data Augmentation

When it came to deciding what MS2 data to use for training our neural network, we selected the yeast protein database mostly due to it being widely used in all areas of proteomics as a testing standard. In order to generate MS2 data for the proteins, we first had to generate the peptides, which are subsequences of the protein. To do this, we simply took each protein and recorded all possible subsequences ranging from length 6 to 50 and labeled them with their original protein identification. These peptides were then fed through a mass simulator that would create the theoretical MS2 data for the given peptide. The MS2 data, which comes in the form of a set of m/z ratios and their intensities, is cleaned by taking only the m/z ratios and binning them to the nearest integer value:

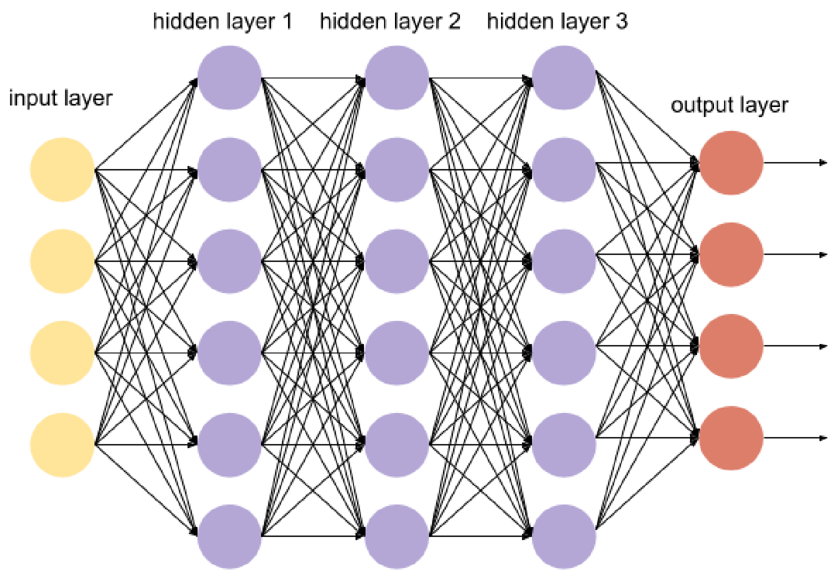
Example MS2 m/z data:	10.005, 13.02, 20.53, 47.92, …
Cleaned MS2 m/z data:	10, 13, 21, 48, …

The binned input data is then read in and concatenated to form the sentences for a given protein sequence. A given sentence would be similar to the cleaned MS2 m/z data listed above (Ex. “10, 13, 21, 48”).

4. Neural Network Design

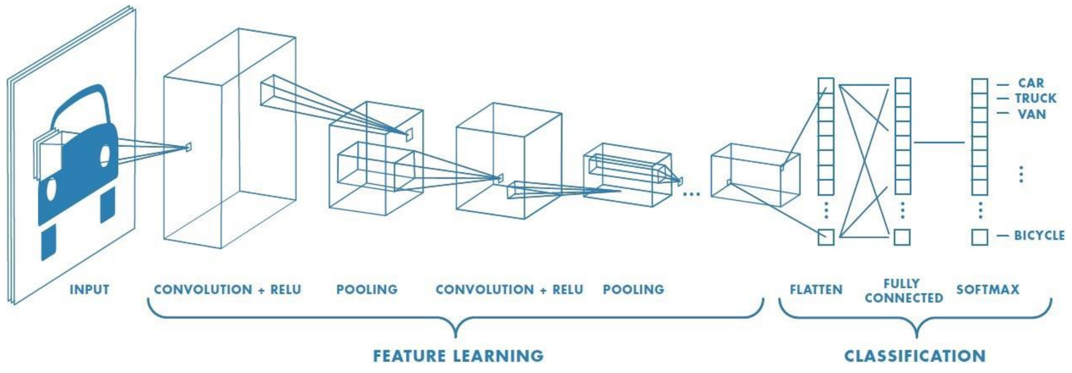
For the overall design of the neural network, we start by first passing the cleaned MS2 m/z data to the embedding layer, where the similarity of the vectors is assessed and they are grouped based on their relation to each other. The weights that are used to group the vectors then become the input for the next layer of the network. The main factor that needs to be considered when it comes to optimizing the embedding layer is how many dimensions the input data should be reduced to. After doing literature search of others' implementations of the embedding layer, dimensions of 50, 100, 200, and 1000 were chosen. After implementing and testing each dimension size, 100 ended up doing slightly better than the lowest value of 50, but the two highest dimensions of 200 and 1000 did not improve the accuracy. This resulted in 100 dimensions being used as the number of dimensions in the embedding layer.

For the remaining portion of the network, two main methods were tested. The first was densely connected layers, where each layer creates *n* nodes and each node is connected to every node in the previous layer. In our testing, the number of nodes in a dense layer varied from 50 to 5000. The layer depth was also changed based on the number of nodes in each layer due to computation time. When each layer had 1000 to 5000 nodes, the number of layers was kept low, ranging from 2 to 4, whereas when layers had 50 to 1000 nodes, the number of layers ranged from 2 to 12. To put it into perspective, the number of parameters in the computational tests ranged from roughly 20 million parameters in most of the smaller tests to over 200 million parameters in some of the largest tests. After testing the network with the varying numbers of layers and nodes, a tradeoff was noticed between accuracy and computation time. When testing with layers that had 50 to a few hundred nodes, the overall computation time was small due to the reduced number of connections and trainable parameters, but overall accuracy was very low, presumably due to their limited training and learning capacity. On the other hand, when it came to testing with layers that had 5000 nodes, the improved accuracy did not suffice for the immensely higher training and computation time that it took. In the end, the best choice for the number of nodes per densely connected layer ended up being around 500 nodes due to its balance of accuracy and computation time. However, choosing 500 nodes also meant that the neural network couldn't be very deep due to memory constraints. A final decision of one or two hidden, densely connected layers feeding to the final densely connected output layer was chosen. This allowed for a train time of roughly 300 seconds per epoch using around 300,000 theoretical peptide MS2 sequences as training data.



Densely Connected Neural Network

The second method that was used to build upon the embedding layer was convolutional 1D layers. Convolutional neural networks are commonly used in other areas where small sections of the data can be analyzed. As Nils Ackermann says in *Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences*, “This applies well to the analysis of time sequences of sensor data (such as gyroscope or accelerometer data). It also applies to the analysis of any kind of signal data over a fixed-length period (such as audio signals). Another application is NLP (although here LSTM networks are more promising since the proximity of words might not always be a good indicator for a trainable pattern)” [4]. Convolutional layers, along with pooling layers, were used to drastically decrease the computation time as the number of parameters is reduced. Unlike densely connected layers, convolutional layers do not have access to all the nodes from the previous layer, removing a portion of the computations. When it comes to customizing a convolutional layer, the parameters are the output dimensions and the kernel size. For output dimensions, since the embedding layer already adds a dimension reduction to the theoretical peptide sequences, smaller values were not used in order to avoid bottlenecking. Instead, values ranging from 256 to 1000 were tested for the number of output dimensions, with 256 to 512 performing the best in terms of accuracy and computation time. From there, different numbers of convolutional layers with different pooling layer frequencies were tested. The number of convolutional layers ranged from 1 to 4 and the number of pooling layers ranged from a single pooling layer to a pooling layer after every convolutional layer. Along with the number of pooling layers, each layer also has a choice of pooling size, but increasing this led to a loss of dimensions and decreased accuracy. As a result, this value was left at 2 throughout the course of the training.



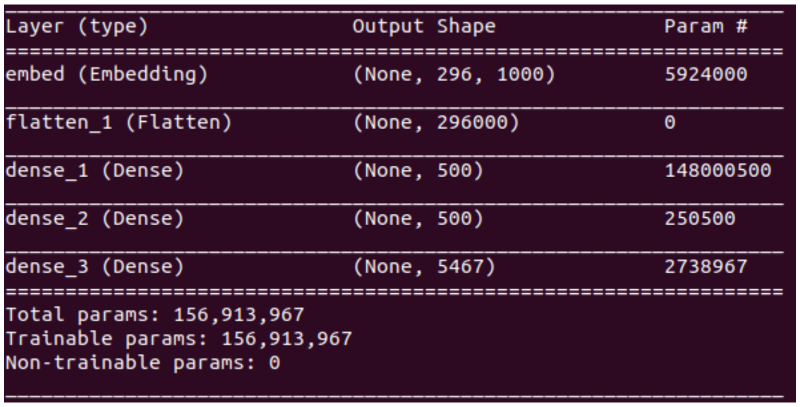
Convolutional Neural Network

5. Results

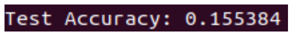
When doing our testing, the best results ended up coming from the following network configuration:

1. Embedding layer: 1000-dimension output
2. 2 convolutional layers
    a. Filter = 256
    b. Kernel size = 8
3. Pooling layer – pool size of 2
4. Flatten layer to reduce dimension to one
5. Densely connected output layer
    a. Output nodes = number of protein labels

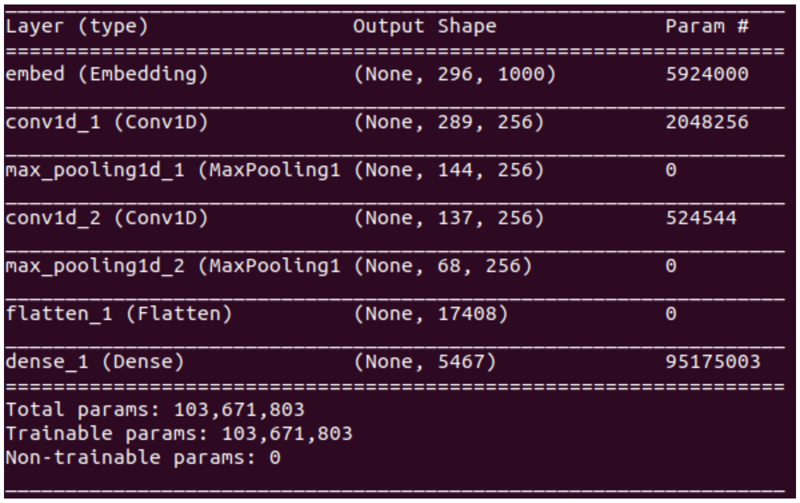
Using this model, roughly 300,000 different peptide strings along with their protein labels were used as training data. When tested against never seen theoretical MS2 for other peptides, the model was able to predict the protein label with 26% accuracy. This is of great improvement over our tests with a smaller data set of only 10,000 peptide strings and protein labels, which had a resulting accuracy of less than 0.01%.



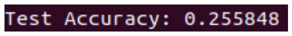
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embed (Embedding)            (None, 296, 1000)         5924000
_________________________________________________________________
flatten_1 (Flatten)          (None, 296000)            0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               148000500
_________________________________________________________________
dense_2 (Dense)              (None, 500)               250500
_________________________________________________________________
dense_3 (Dense)              (None, 5467)              2738967
=================================================================
Total params: 156,913,967
Trainable params: 156,913,967
Non-trainable params: 0
_________________________________________________________________
```



```
Test Accuracy: 0.155384
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embed (Embedding)            (None, 296, 1000)         5924000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 289, 256)          2048256
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 144, 256)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 137, 256)          524544
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 68, 256)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 17408)             0
_________________________________________________________________
dense_1 (Dense)              (None, 5467)              95175003
=================================================================
Total params: 103,671,803
Trainable params: 103,671,803
Non-trainable params: 0
_________________________________________________________________
```



```
Test Accuracy: 0.255848
```

6. Further Research

The next steps that can be taken in this area of research is expanding the training data. We have generated data sets for the entire yeast database of proteins, but this results in a training data set with over 2.7 million peptide sequences, far more than what our current devices our capable of handling. Having more resources would not only allow us to handle larger training sets, but also much larger neural networks. Increasing the size of convolutional and pooling layers along with adding additional dense layers before the output layer could greatly increase the network's learning capacity. These options were attempted, but physical memory limitations prevented any results. In the end though, the small tests that we were able to perform with the resources we had showed signs of being able to learn patterns in the protein data. With continued research, these models could drastically reduce the time and computation that the current standard of database searching requires.

References

[1] “Overview of Mass Spectrometry | Thermo Fisher Scientific - US.” *Overview of Mass Spectrometry | Thermo Fisher Scientific - US*, www.thermofisher.com/us/en/home/life-science/protein-biology/protein-biology-learning-center/protein-biology-resource-library/pierce-protein-methods/overview-mass-spectrometry.html.

[2] “Tandem Mass Spectrometry.” *Wikipedia*, Wikimedia Foundation, 18 Mar. 2019, en.wikipedia.org/wiki/Tandem_mass_spectrometry.

[3] Gul Awan, Muaaz, and Fahad Saeed. “MS-REDUCE: an Ultrafast Technique for Reduction of Big Mass Spectrometry Data for High-Throughput Processing.” *OUP Academic*, Oxford University Press, 21 Jan. 2016, academic.oup.com/bioinformatics/article/32/10/1518/1743195.

[4] Ackermann, Nils. “Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences.” *Good Audience*, Good Audience, 4 Sept. 2018, blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.