Supporting Information

Tran et al. 10.1073/pnas.1705691114

SI Text

DeepNovo Model.

**Input processing.** As shown in Fig. 1A, a tandem mass spectrum is often presented as a histogram plot of intensity vs. mass (more precisely, m/z). The underlying raw format (e.g., mgf) is simply a list of pairs of mass and intensity. In DeepNovo, we discretize a spectrum into a vector, called an intensity vector, in which masses correspond to indices and intensities are values. This representation assumes a maximum mass and also depends on a mass resolution parameter. For instance, if the maximum mass is 5,000 Da, and the resolution is 0.1 Da, then the vector size is 50,000, and every 1 Da mass is represented by 10 bins in the vector. In these implementation, we consider two types of data: low resolution (0.1 Da) and high resolution (0.01 Da). High-resolution data often allow de novo peptide sequencing tools to achieve better accuracy.

**Ion-CNN model.** The ion-CNN model is designed to learn features of fragment ions in a spectrum. The input is a prefix (i.e., a sequence including the “start” symbol and the amino acids that have been predicted up to the current iteration). The output is a probability distribution over 20 amino acid residues, their modifications, and three special symbols: start, “end,” and “padding.” In this paper, we consider three variable modifications oxidation and deamidations, hence, a total of 26 symbols for prediction. For example, in Fig. 1B and C, the prefix consists of four symbols: start, “P,” “E,” and P. Symbol “T” is predicted as the next amino acid by sampling or ReLU of the first convolutional layer is +10. Deep and is called the × of the first convolutional layer is +10, 10, 1

Given the input prefix, DeepNovo first computes the prefix mass (i.e., the sum of masses of the N terminus and amino acids in the prefix) (Fig. 1C). Next, DeepNovo tries to add each of 26 symbols to the current prefix and updates its mass accordingly. For each candidate, the corresponding masses of b and y ions are calculated. In this implementation, we use eight ion types: b, y, b(2+), y(2+), b-H₂O, y-H₂O, b-NH₃, and y-NH₃ (24). Given an ion mass, DeepNovo identifies its location on the intensity vector using the mass resolution. For example, the prefix of four symbols start, P, E, and P with the next candidate T will have a b ion of mass 424.2 Da, which corresponds to index 4,240 on the intensity vector of resolution 0.1 Da. DeepNovo then extracts an intensity window of size 10 around the ion location. Thus, for each input prefix, DeepNovo computes a 3D array of shape 26×8×10. Deep learning libraries often process data in batches to take advantage of parallel computing. Here, we use a batch size of 128 (i.e., we process 128 prefixes at the same time, and their arrays are packed into a 4D array of shape 128×8×10×26). We further transpose the shape into 128×8×10×26 (reason will be explained later). This final array, denoted by X¹28x8x10x26, is similar to the common data setting of image processing, where the first dimension is the number of images, the second is the height, the third is the width, and the fourth is the number of channels (e.g., three for red-green—blue or one for black-white).

The ion-CNN model is a CNN with two convolutional layers and two fully connected layers (Fig. 1C) (27, 61). The first convolutional layer uses a 4D kernel H¹16x8x16x32 and a bias term B¹32 to transform the input array X¹28x8x10x26 into a new array Y¹28x8x10x32. This convolution operator slides 26×32=832 receptive fields (filters) of size 1×3 of the kernel W over the input array X and performs a series of dot products and additions as follows:

$$Y_{i,j,k,l} = \sum_{m=1}^{26} \sum_{n=1}^{3} W_{l,n,m,p} X_{i,j,k,n-1,m} + B_{l,k}.$$  \[S1\]

where 1 ≤ i ≤ 128, 1 ≤ j ≤ 8, 1 ≤ k ≤ 10, 1 ≤ l ≤ 32, and the third dimension of X is padded with 0 when needed. The purpose of convolution is to learn as many local features as possible through several different filters. Hence, the kernel W is often called the “feature detector,” and the output Y is called the “feature map.” As can be seen from Eq. S1, we perform convolution along the third dimension of X (i.e., the intensity window) to learn the bell-shaped features (i.e., peaks) (Fig. 1C). We also use different sets of filters for different amino acids. This setting is currently the best setting that we found after trying multiple convolution combinations of ions and/or amino acids. However, more investigations with more data are worth trying in future development.

The linear convolution is followed by an activation with rectified linear unit [ReLU; i.e., f(x) = max(0,x)]. Activation functions are often used to add nonlinearity into neural network models, and ReLU is currently the most favorite because of its many advantages (62). Thus, the output Z of the first convolutional layer is obtained by applying the ReLU function on Y elementwise:

$$Z_{i,j,k} = \text{ReLU}(Y_{i,j,k}).$$  \[S2\]

The second convolutional layer is applied on top of the first convolutional layer in a similar way with another kernel Y¹16x8x16x32. Adding more convolutional layers does not show significant improvement of accuracy, probably because the bell-shaped features are not too complicated to learn. We also apply max-pooling, but it seems not to have much impact, because the dimensionality is not large.

The convolutional layers are followed by a fully connected layer or often-called hidden layer of 512 neuron units (Fig. 1C). As the name suggests, each unit is connected to every output of the previous convolutional layer to process all local features together. This connection is done via a linear matrix multiplication and addition as follows:

$$X_{hidden}^{128x512} = \text{ReLU}(X_{hidden}^{128x512} W^{hidden} + B^{512 hiddenn}).$$  \[S3\]

Notice that the output of the previous convolutional layer with shape 128×8×10×32 is first reshaped into X¹28x8x10x26 to be compatible with the matrix multiplication operator. We also apply ReLU elementwise after the linear operations.

The final fully connected layer has 26 neuron units, which correspond to 26 symbols to predict. It is connected to the previous hidden layer in a similar way as Eq. S3, except that there is no ReLU activation.

We also apply dropout, an important technique to prevent neural networks from overfitting (63). We use dropout after the second convolutional layer with probability 0.25 and after the first fully connected layer with probability 0.5. The idea of dropout is that neuron units are randomly activated (or dropped) at every training iteration so that they do not coadapt. At the testing phase, all units are activated, and their effects are averaged by the dropout probability.

**Spectrum-CNN and LSTM model.** The spectrum-CNN coupled with LSTM model is designed to learn sequence patterns of amino acids of the peptide in association with the corresponding spectrum. We adopt this idea from a recently trending topic of “automatically generating a description for an image.” In that research, a CNN is used to encode or “understand” the image,
and an LSTM RNN (35) is used to decode or “describe” the content of the image (36, 37). Here, we consider the spectrum intensity vector as an image (with one dimension and one channel) and the peptide sequence as a caption. We use the spectrum-CNN to encode the intensity vector and the LSTM to decode the amino acids.

**Spectrum-CNN: Simple version.** The input to the spectrum-CNN is an array of shape $128 \times 1 \times 50,000 \times 1$, where 128 is the batch size and 50,000 is the size of intensity vectors given the maximum mass of 5,000 Da and the resolution of 0.1 Da. Because the input size is too large, we first try a simple version of spectrum-CNN that includes two convolutional layers, each with four filters of size $1 \times 4$ and one fully connected layer of 512 neuron units. We also use ReLU activation, max-pooling, and dropout in the same way as for the ion-CNN model described above.

It should be noted that the pattern recognition problem with tandem mass spectra here is quite different from traditional object recognition problems. Usually, an object is recognized by its shape and its features (e.g., face recognition). However, in a tandem mass spectrum, an amino acid is identified by two bell-shaped signals (i.e., peaks) with distance between them that has to precisely match with the amino acid mass. Because distance is involved, our simple spectrum-CNN and other common CNN models may not be good enough.

**Spectrum-CNN: Advanced version.** To take the distance into account, we slice each input intensity vector into pieces based on the amino acid masses. For instance, given that the mass of Alanine or “A” is 71.0 Da and the resolution is 0.1 Da, we slice the intensity vector from index 710 until the end to create a new vector. We pad the new vector by 0 so that it has the same size as the original one and concatenate the two along the second dimension to obtain an array of shape $128 \times 2 \times 50,000 \times 1$. We repeat this procedure for all 26 symbols and construct a new input array of shape $128 \times 2 \times 50,000 \times 26$.

After preprocessing, we apply the first convolutional layer with the kernel of shape $2 \times 10 \times 26 \times 32$. The idea is to capture two bell-shaped signals in the same filter of size $2 \times 10$. This convolutional layer is followed by another one with kernel of shape $1 \times 5 \times 32 \times 64$ and one fully connected layer of 512 neuron units. Again, we also use ReLU activation, max-pooling, and dropout. Note that here we use max-pooling aggressively, because the intensity vectors are very sparse.

It should be noted that the goal of our spectrum-CNN is not to make accurate predictions of the next amino acid, such as the ion-CNN. Instead, the spectrum-CNN only tries to pick up signals of which amino acids are presented in the spectrum and provide that information to the LSTM model to better learn sequence patterns of amino acids. The spectrum-CNN output is a vector of size 512, corresponding to 512 neuron units of its fully connected layer.

**LSTM model.** LSTM networks, a special kind of RNNs, are the most widely used models to handle sequential data in natural language processing and speech recognition (35). RNNs are called “recurrent,” because they repeat the same computations on every element of a sequence, and the next iteration depends on the networks’ “memory” of previous steps. For example, one could predict the next word in a sentence given the previous words. In the problem de novo peptide sequencing, we want to predict the next amino acid, a symbol, given the previous ones (i.e., the prefix) (Fig. 1 B and C). This assumption is reasonable, because amino acids do not just appear in a random order in protein sequences. In other words, protein sequences may speak their own “languages.”

Because of limited space, we do not try to include all details of LSTM and RNNs in this manuscript. We use the standard LSTM model, which can be found in many articles in the literature, such as refs. 35–37, or online resources. Here, we only discuss some important configurations of our LSTM model. First, we use embedding vectors of size 512 to represent each of 26 symbols, similar to the common word2vec (39) approach that uses embedding vectors to represent words in a vocabulary. The embedding vectors form a 2D array Embedding$^{26 \times 512}$. Thus, the input to the LSTM model at each iteration is a vector of size 512. Second, the output of the spectrum-CNN is used to initialize the LSTM model (i.e., being fed as the zero input). Third, the LSTM architecture consists of one layer of 512 neuron units and dropout layers with probability 0.5 for input and output. The recurrent iterations of the LSTM model can be summarized as follows:

$$x_t = \text{CNN}_{\text{spectrum}}(I)$$

$$a_{t-1} = \text{Embedding}_{a_{t-1}, s_{t-1}}$$

$$s_t = \text{LSTM}(x_t, a_{t-1})$$

where $I$ is the spectrum intensity vector, $a_{t-1}$ is the symbol predicted at iteration $t − 1$, Embedding, $s_{t}$ is the row $i$ of the embedding array, and $s_{t}$ is the output of the LSTM model and will be used to predict the symbol at iteration $t$, $t = 1, 2, 3, \ldots$ Similar to the ion-CNN model, we also add a fully connected layer of 26 neuron units to perform a linear transformation of the LSTM 512 output units into signals of 26 symbols to predict.

Last but not least, LSTM networks often iterate from the beginning to the end of a sequence. However, to achieve a general model for diverse species, we found that it is better to apply LSTM on short $k$-mers. However, this topic requires additional analysis with more data to find an optimum solution.

**Integrating ion-CNN and LSTM models.** To combine the ion-CNN and LSTM models, we first concatenate the outputs of their second to last layers, each of size 512, to form a vector of size 1,024. Then, we add a fully connected layer of 1,024 neuron units with ReLU activation and dropout with probability 0.5 followed by another fully connected layer of 26 neuron units to perform a linear transformation into signals of 26 symbols to predict (Fig. 1C). Thus, the final output of DeepNovo neural networks is a vector of 26 signals, often called logits (unscaled log probabilities). This logits vector will be further used to calculate the loss function during training or calculate the prediction probabilities during testing.

In this section, we have completely described all details of DeepNovo model. All weight and bias parameters (i.e., $W$'s and $B$'s) of the CNNs, embedding vectors, and parameters of the LSTM will be estimated and optimized during the training process. In addition, DeepNovo performs bidirectional sequencing and uses two separate sets of parameters, forward and backward, except for the spectrum-CNN and the embedding vectors. The hyper-parameters, such as the numbers of layers, the numbers of neuron units, the size of embedding vector, the dropout probabilities, the number and types of fragment ions, etc., can be configured to define an instance of DeepNovo model.

**De Novo Peptides Identified by DeepNovo but Missed by Database Search.** DeepNovo is able to find high-quality matches that elude database search identification. To show this advantage, we performed an experiment on a conventional dataset Clinical Proteomic Tumor Analysis Consortium as follows.

A yeast lysate was spiked with a mixture of 48 human proteins (Sigma-Aldrich UPS1). The sample was then analyzed three times by the Thermo LTQ-Orbitrap instrument. We then used PEAKS DB to do a database search with a false discovery rate of 1%. We first searched this dataset against a combined database including both human and yeast proteins. As shown in Fig. S7A, the total number of identified peptide-spectrum matches (PSMs) is 18,306, including 16,617 from yeast and 1,689 from human. Next, we searched this dataset against the yeast database only and found 16,693 PSMs.

We used DeepNovo to perform de novo sequencing on the whole dataset. After excluding 16,693 spectra identified from the yeast database search and selecting the top 50% high-confidence results, we found 7,146 spectra identified by DeepNovo only. Among those 7,146 spectra, 1,524 matched to the human peptides were identified in the first round of database search and covered...
Thus, DeepNovo was able to identify human peptides that eluded the second round of database search. This result shows the importance of de novo sequencing when the database information is missing.

**Training DeepNovo with MS/MS Data.** Here, we would like to emphasize some important techniques for training DeepNovo. MS/MS data have a special property: the same peptide could appear multiple times in a dataset with different spectra. Such spectra may have different fragment ions, and even if they share some major ions, the intensities of those ions also vary from spectrum to spectrum. However, the model is able to learn some common features of different spectra that come from the same peptide, and those features are not generalized well to other peptides. This problem will lead to overfitting if we randomly partition a dataset into training, validation, and testing sets (a common technique in most model training tasks). The model will perform well on those three sets, but its performance gets worse on a new dataset. Thus, it is essential to make sure that the training, validation, and testing sets do not share common peptides. In addition, we found that it is preferable to collect more data from a wide variety of sources than increase data from the same source. This observation may be related to the one to many relationship between peptide and spectra mentioned earlier.

**GPUs and Big Data: Two Advantages of Neural Network Models.** Recent breakthroughs in neural networks and deep learning are driven by the two main engines: powerful GPUs and massive amount of datasets. These two also fit nicely into the problem of de novo peptide sequencing. De novo peptide sequencing is well-known as a computation-intensive optimization problem, and modern MS instruments often produce data faster than many sequencing software can analyze in real time. Recently, Novor has greatly improved the speed and is able to keep up with the rate of data acquisition. However, it is still highly desirable to make use of high-performance hardware, such as GPUs, instead of traditional central processing units (CPUs). DeepNovo is implemented on the Google TensorFlow platform and able to run on both GPUs and CPUs. Moreover, TensorFlow scales up easily to multiple GPUs, CPUs, and even different workstations, maximizing most computational resources.

In this study, we used only 50,000 spectra from each dataset for training (i.e., about 10% of the total data for training; for testing, we still used all data). However, even with that limited amount of training data, the accuracy of DeepNovo was already 7.7–22.9% higher than the current state of the art. Although it is not always a simple increase between the amount of training data and the model accuracy, we believe that neural network models, such as DeepNovo, are the ideal choice and can benefit the most from huge proteomics databases, such as PRIDE, MassIVE, and others.

**Fig. S1.** The precision–recall curves of the de novo sequencing results on the other four low-resolution datasets. (A) The precision–recall curves on *Mus musculus*. (B) The precision–recall curves on *Caenorhabditis elegans*. (C) The precision–recall curves on *Escherichia coli*. (D) The precision–recall curves on *Drosophila melanogaster*.
Fig. S2. The precision–recall curves and the AUCs of PepNovo, Novor, PEAKS, and DeepNovo on nine high-resolution datasets. (A) Precision–recall curves on Vigna mungo. (B) Precision–recall curves on Mus musculus. (C) Precision–recall curves on Methanosarcina mazei. (D) Precision–recall curves on Bacillus. (E) Precision–recall curves on Candidatus endoloripes. (F) Precision–recall curves on Solanum lycopersicum. (G) Precision–recall curves on Saccharomyces cerevisiae. (H) Precision–recall curves on Apis mellifera. (I) Precision–recall curves on Homo sapiens. (J) AUC of four sequencing tools on nine datasets.
Fig. S3. Total recall and precision of PepNovo, Novor, PEAKS, and DeepNovo on nine high-resolution datasets. (A) Recall at the amino acid level. (B) Precision at the amino acid level. (C) Recall at the peptide level. *A. mellifera*, *Apis mellifera*; *C. endoloripes*, *Candidatus endoloripes*; *M. mazei*, *Methanosarcina mazei*; *M. musculus*, *Mus musculus*; *S. cerevisiae*, *Saccharomyces cerevisiae*; *S. lycopersicum*, *Solanum lycopersicum*; *V. mungo*, *Vigna mungo*. 

Tran et al. www.pnas.org/cgi/content/short/1705691114
Fig. S4. The precision–recall curves and the AUCs of PEAKS, Novor, and DeepNovo on three public real datasets. (A) Precision–recall curves on Ubiquitin. (B) Precision–recall curves on UPS2. (C) Precision–recall curves on U2OS. (D) AUC of three sequencing tools on three real datasets.

Fig. S5. DeepNovo assembly result for the WigG1 light chain. (A) BLAST alignment of the full-length assembled contig against the target light chain. (B) Details of the alignment in A. The red bars indicate the mismatches between the assembled light-chain sequence and the target light-chain sequence.
Fig. S6. DeepNovo assembly result for the WlgG1 heavy chain. (A) BLAST alignment of the top-assembled contigs against the target heavy chain. (B) Details of the alignment in A. The red bars indicate the mismatches between the assembled heavy-chain sequence and the target heavy-chain sequence.

Fig. S7. Identification of spectra with de novo high scores but elude database search. (A) The number of spectra identified by searching the human–yeast database, searching the yeast database only, and using DeepNovo. (B) The Venn diagram of spectra matched with human peptides and DeepNovo only.
Table S1. Summary of seven low-resolution datasets used in our experiments

<table>
<thead>
<tr>
<th>Species</th>
<th>No. of RAW files</th>
<th>Total no. of spectra</th>
<th>No. of PSMs (1% FDR)</th>
<th>Error tolerance</th>
<th>Accession no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mus musculus</td>
<td>40</td>
<td>792,148</td>
<td>355,514</td>
<td>10</td>
<td>0.6</td>
</tr>
<tr>
<td>Caenorhabditis elegans</td>
<td>108</td>
<td>1,125,050</td>
<td>437,097</td>
<td>20</td>
<td>0.8</td>
</tr>
<tr>
<td>Escherichia coli</td>
<td>70</td>
<td>3,239,116</td>
<td>1,174,817</td>
<td>20</td>
<td>0.5</td>
</tr>
<tr>
<td>Drosophila melanogaster</td>
<td>18</td>
<td>681,968</td>
<td>178,853</td>
<td>20</td>
<td>0.5</td>
</tr>
<tr>
<td>H. sapiens</td>
<td>27</td>
<td>804,473</td>
<td>497,191</td>
<td>10</td>
<td>0.6</td>
</tr>
<tr>
<td>Saccharomyces cerevisiae</td>
<td>6</td>
<td>558,564</td>
<td>280,377</td>
<td>20</td>
<td>0.5</td>
</tr>
<tr>
<td>Pseudomonas aeruginosa</td>
<td>25</td>
<td>2,781,682</td>
<td>603,601</td>
<td>20</td>
<td>0.3</td>
</tr>
</tbody>
</table>

FDR, false discovery rate; RAW, a Thermo Scientific mass spectrometer binary data file format.

Table S2. Summary of nine high-resolution datasets used in our experiments

<table>
<thead>
<tr>
<th>Species</th>
<th>No. of RAW files</th>
<th>Total no. of spectra</th>
<th>No. of PSMs (1% FDR)</th>
<th>Error tolerance</th>
<th>Accession no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vigna mungo</td>
<td>19</td>
<td>735,618</td>
<td>37,775</td>
<td>20</td>
<td>0.05</td>
</tr>
<tr>
<td>Mus musculus</td>
<td>9</td>
<td>276,648</td>
<td>37,021</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td>Methanosarcina mazei</td>
<td>16</td>
<td>800,768</td>
<td>164,421</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td>Bacillus</td>
<td>14</td>
<td>571,615</td>
<td>291,783</td>
<td>30</td>
<td>0.05</td>
</tr>
<tr>
<td>Candidatus endoliripes</td>
<td>9</td>
<td>1,862,619</td>
<td>150,611</td>
<td>20</td>
<td>0.05</td>
</tr>
<tr>
<td>Solanum lycopersicum</td>
<td>5</td>
<td>603,506</td>
<td>290,050</td>
<td>15</td>
<td>0.05</td>
</tr>
<tr>
<td>Saccharomyces cerevisiae</td>
<td>17</td>
<td>277,077</td>
<td>111,312</td>
<td>20</td>
<td>0.05</td>
</tr>
<tr>
<td>Apis mellifera</td>
<td>26</td>
<td>684,821</td>
<td>130,583</td>
<td>20</td>
<td>0.02</td>
</tr>
</tbody>
</table>

FDR, false discovery rate; RAW, a Thermo Scientific mass spectrometer binary data file format.