IDL Module—Final Report

DeepNovoV2, Better *De Novo* Peptide Sequencing By Deep Learning

Simon Chardin & Samuel Diebolt, 136th Graduating Class, ESPCI Paris-PSL May 18, 2020

Introduction

Mass spectrometry (MS) for proteomics has become a well-known and widely used technique for protein identification, with applications in a wide variety of fields, including drug development, biochemical screening and toxicology. MS relies on the understanding of peptide fragmentation behaviors. In bottom-up proteomics, proteins in the samples of interest are digested by an enzyme into smaller, easier to handle peptides. These peptides are then separated using chromatography and analyzed by tandem mass spectrometry (MS/MS). Finally, *de novo* methods utilize computational approaches to deduce the sequence of these peptides directly from the experimental MS/MS spectra. These methods enable identification and quantification of the proteins in the original sample, but require solving complex combinatorial problems. Recently, a study proposed a deep learning model, DeepNovo, designed to perform *de novo* sequencing from MS/MS data [1]. On 16 low- and high-resolution MS/MS datasets, the tool outperformed current state of the art *de novo* peptide sequencing software such as PEAKS [2], PepNovo [3] and Novor [4]. In this report, we studied an improved version of this model, DeepNovoV2, currently described in a preprint by the same authors [5]. Given the lack of documentation in the original source code, we dismantled DeepNovoV2 into its main components and reconstructed it into a Python notebook, so as to provide better understanding of the model structure. The tool was then improved to handle more post-translational modifications (PTMs) and to parse MS/MS data annotated using software other than PEAKS. Finally, we developed a Python module for the *Spectrométrie de Masse Biologique et Protéomique (SMBP)* laboratory at ESPCI Paris-PSL to facilitate input files generation, features extraction and spectra annotations. Using this tool, we trained the model on MS/MS data from the SMBP laboratory and compared the results with those from DeepNovoV2's authors.

1 Background

1.1 Mass Spectrometry for Proteomics

MS is a technique that measures the mass-to-charge ratio (m/z) of gas-phase ions. A mass spectrometer consists of three main modules: an ion source converting molecules in the sample of interest into gas-phase ions, a mass analyzer separating these ions according to their m/z ratio and a detector measuring the ion counts for each m/z value. Numerous technology advances in each module have made MS a fundamental technique of proteomics. Ionization techniques such as electrospray ionization (ESI) [6] and matrix-assisted laser desorption/ionization (MALDI) [7] enable the ionization of proteins and peptides and are regularly coupled with quadrupole, ion-trap, time-of-flight or Fourier-transform ion cyclotron resonance mass analyzers.

De novo peptide sequencing is performed using spectra obtained with MS/MS, a technique where two mass analyzers are coupled with an additional fragmentation step to enhance the analysis of the peptides and proteins. Figure 1 shows a diagram of a typical MS/MS workflow. After an eventual sample preprocessing step—e.g. protein digestion—, the molecules of interest are ionized and introduced in the first mass analyzer, resulting in a first MS spectrum. Precursors ions corresponding to a given m/z ratio are then selected by the first mass analyzer and further fragmented. Collision-induced dissociation (CID) [8] is the most widely used MS/MS technique: the gas-phase ions are heated by collisions with rare gas atoms, leading to fragmentation of the peptide backbone. These new ion fragments are introduced in the second mass analyzer, resulting in the MS/MS spectrum. MS/MS has become a key technique for peptide sequencing, as it made possible the identification of ions with similar m/z ratios that couldn't be differentiated using a single mass analyzer.

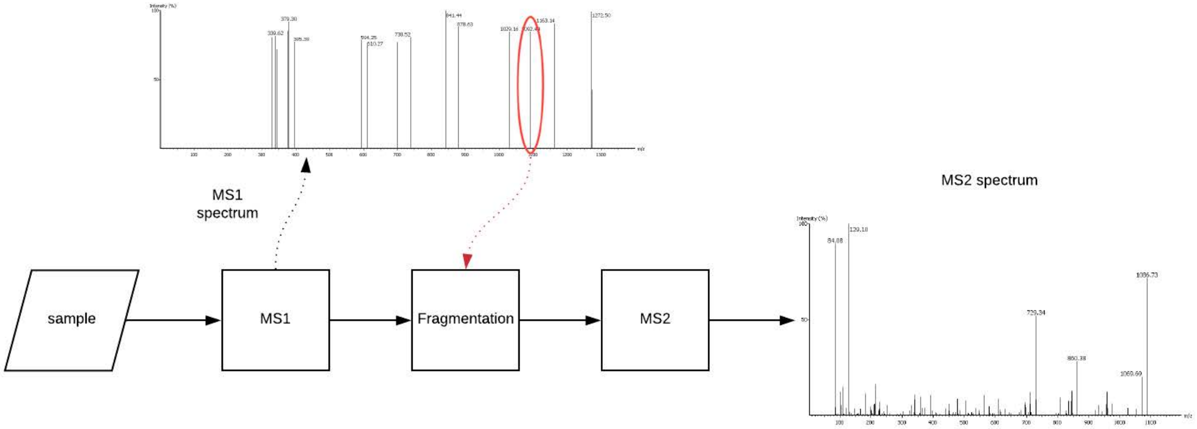

Figure 1: Diagram of a typical MS/MS workflow [5].

1.2 Database Searching—Current Methods

Currently, the most common method for peptide sequencing from MS/MS data involves searching against databases of experimental or theoretical sources (e.g. Uniprot [9], GenBank [10], etc.). MS/MS spectra can either be directly compared to experimental spectral libraries, or they can be used in search engines that will match them with candidate peptides obtained by *in silico* fragmentation and spectrum analysis. Widely used search engines include Mascot [11], SEQUEST [12] and OMSSA [13].

However, database searching for peptide sequencing has some limitations. The aforementioned databases can be incomplete, due for example to the lack of information on some biological species. Moreover, modifications to the protein sequences due to single-nucleotide polymorphisms (SNPs) or PTMs result in huge combinatorial problems that can hinder search engines performances. Finally, search engines are limited to using the species-specific databases and PTMs parameters selected by the user. This requires prior information on the samples and prevents identification of proteins and peptides not represented by the selected parameters.

1.3 *De Novo* Peptide Sequencing

As opposed to the previously discussed search engines, *de novo* peptide sequencing methods do not rely on databases. Modern *de novo* software use techniques such as machine learning, probabilistic modeling and dynamic programming to deduct the peptide sequences from MS/MS spectra. This task, similar to the image captioning problem in computer vision, allows the recovery of the exact amino acid sequence of the original peptide. Knowing the amino acid sequence of peptides from a protein digest is essential for study the biological function of the protein. De novo sequencing is an assignment of fragment ions from a mass spectrum. By integrating two fundamental types of neural networks, a convolutional neural network (CNN), and a long short-term memory (LSTM), Qiao et al. [5] try to exploit image recognition and natural language processing. They use the CNN to analyze the content of a spectrum as an image, and then with the LSTM, it allows for an interpretation of the image. The most widely used fragmentation methods today are CID and Electron-Transfer Dissociation (ETD). CID produces mostly b and y-ions; and ETD produces mostly c and z-ions.

1.4 DeepNovoV2 Input Files

In DeepNovoV2, input MS/MS spectra are represented as a set of $(m/z, \text{intensity})$ pairs, keeping only the top 500 most intense peaks in each spectrum by default. Spectra must be provided as mascot generic format (MGF) files [15], a plain-text format containing metadata on each spectrum—e.g. title, precursor ion m/z ratio and charge, chromatography retention time, peptide sequence, etc.—along with a centroid representation of the spectrum itself, i.e. $(m/z, \text{intensity})$ pairs where the intensity represents the area of the original profile peak.

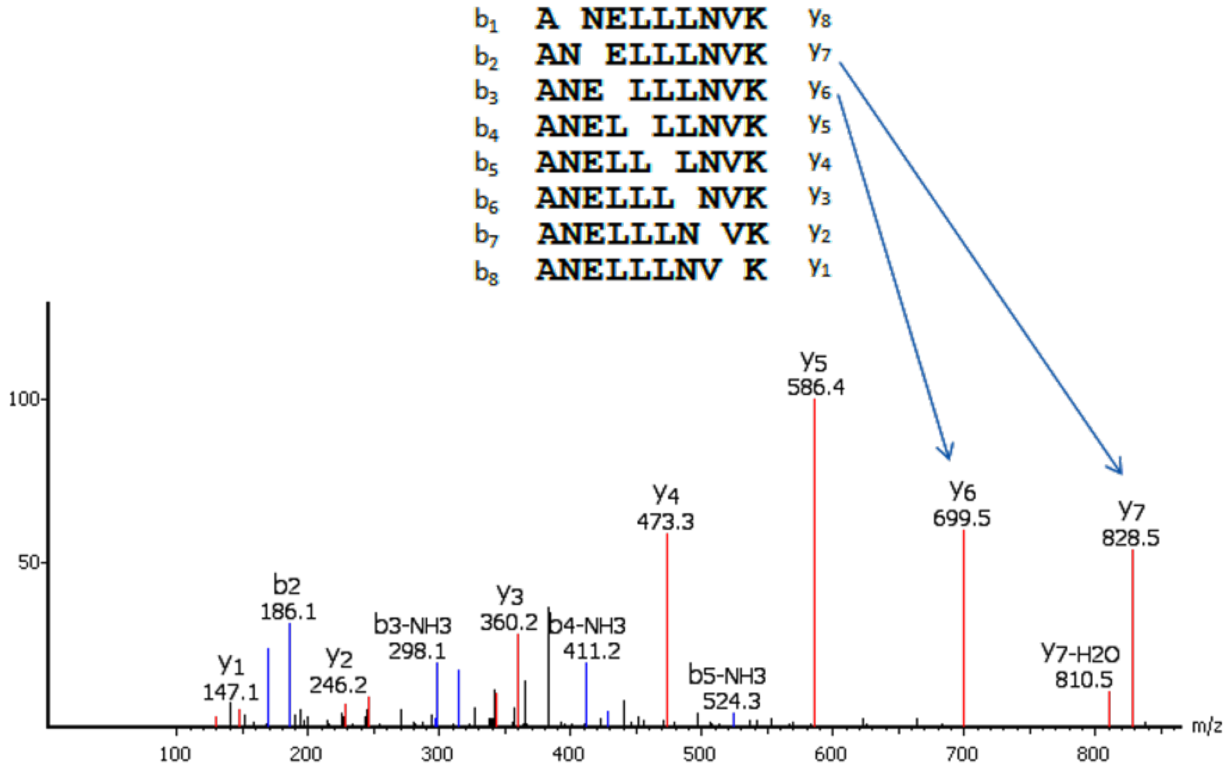

Figure 2: In a CID MS, many copies of the same peptide are fragmented at the peptide backbone to form b and y ions. The spectrum consists of peaks at the m/z (mass to charge) values of the corresponding fragment ions. A good quality spectrum often contains many (but not necessarily all) of the theoretical fragment ions. [14]

Along with the MGF spectrum file, DeepNovoV2 requires a comma-separated values (CSV) file containing information on each spectrum. In particular, the predicted peptide sequence obtained using other *de novo* software, when available, is stored in the features file.

The authors provided functions to convert and merge input files from the first DeepNovo model to DeepNovoV2 on a GitHub repository [16]. However, these functions lack documentation and aren't reliable with MGF annotated with software other than PEAKS. As the MS/MS data offered by the SMBP laboratory came with annotation results obtained using the Mascot software [11], we developed a Python module that offers functions for DeepNovoV2 input files generation using software-independent MGF files and Mascot extensible markup language (XML) result files. The Python module along with an explanatory Jupyter notebook are available on our GitHub repository: https://github.com/sdiebolt/espci-idl-homework.

Finally, the model is configured using the `deepnovo_config.py` file. This file contains the amino acids vocabulary and their respective masses, paths to the input files and the usual model parameters—e.g. batch size, initial learning rate, etc. The user must modify the `vocab_reverse` and `mass_AA` variables to add or remove their desired PTMs. Note however than the unmodified cysteine isn't present in the vocabulary by default, and that there are errors in the modified cysteines mass shifts available in the source comments.

1.5 DeepNovoV2 Outputs

The outputs of DeepNovoV2 are tab-separated values (TSV) files with predicted sequences and score, along with prediction accuracy for the testing step. Precision and recall at the peptide and amino acid levels are also printed during testing, but aren't saved by the software. The scores provided by DeepNovoV2 correspond to the output of the beam search (see Section 2.4), that is a $\mathrm{LogSoftMax}$ function applied to the last layer of the model. Therefore, scores range from 0 for a perfect prediction to $-\infty$.

2 Methods

2.1 Data Preprocessing

As explained in Section 1.4, two files are needed to use DeepNovo: a spectrum MGF file and a features CSV file. The spectrum file is provided by the mass spectrometer. MS vendors either offer a direct export to MGF or software to convert from a proprietary format—e.g. ThermoFisher's RAW format—to MGF. The available parameters and their order for each spectrum in the MGF file can vary: this can cause errors in DeepNovoV2, as the `data_reader.py` script expects a specific set of ordered parameters. The function `format_mgf_deepnovo`, available on our GitHub repository, will take any MGF file containing at least the parameters required by DeepNovoV2—i.e. `'TITLE'`, `'PEPMASS'`, `'CHARGE'`, `'SCANS'`, `'RTINSECONDS'`—and will reformat it to comply with DeepNovoV2's parser.

The features file, containing information on each spectrum, must be created by the user. We provide the function `extract_features` in our Python module to generate the features file from a formatted MGF spectrum file and an optional Mascot XML result file. If the latter is provided, `extract_features` will retrieve peptide sequences associated to each spectrum and populate the features file. In that case, any spectrum without an associated peptide sequence will be removed, as it won't be usable in the model training.

Finally, the functions `merge_mgf` and `merge_features` are provided in our Python module to merge input files. These functions re-number the scan IDs by prepending a tag corresponding to the order of the files to merge. Therefore, the user must be careful to have the same file order in the MGF list and in the features list. A Jupyter notebook is provided on our GitHub repository to facilitate the merging and generation of DeepNovoV2's input files.

2.2 Point Net – Use of Image recognition CNN

A spectrum is a set of (m/z, intensity) pairs, which means the order of peaks should be irrelevant. Thus, in this algorithm we suppose that the prediction has an order invariant property with respect to the first dimension of the feature file. Following the footsteps of PointNet, the model of *de novo* conserve the following rules:

- Permutation Invariance: given the unstructured nature of the spectrums, the data processing has to be invariant to the different representations.
- Transformation Invariance.
- Point Interactions: the interaction between neighboring pairs carries useful information.

Introducing the transformation network (T-Net) [17]. This network is a transformation of the input into a canonical space to approximately align point clouds before any processing is done. It can be seen as a generalization of PCAs to higher dimensions. Tensor decompositions have broad-ranging uses, including the analysis of a wide variety of probabilistic latent-variable models. The T-Net is shown in Fig.3 and consists of a multilayer perceptron (MLP), a max pooling operator, and two fully connected layers. This network predicts an affine transformation to ensure that the semantic labelling of a point cloud is invariant with geometric transformations.

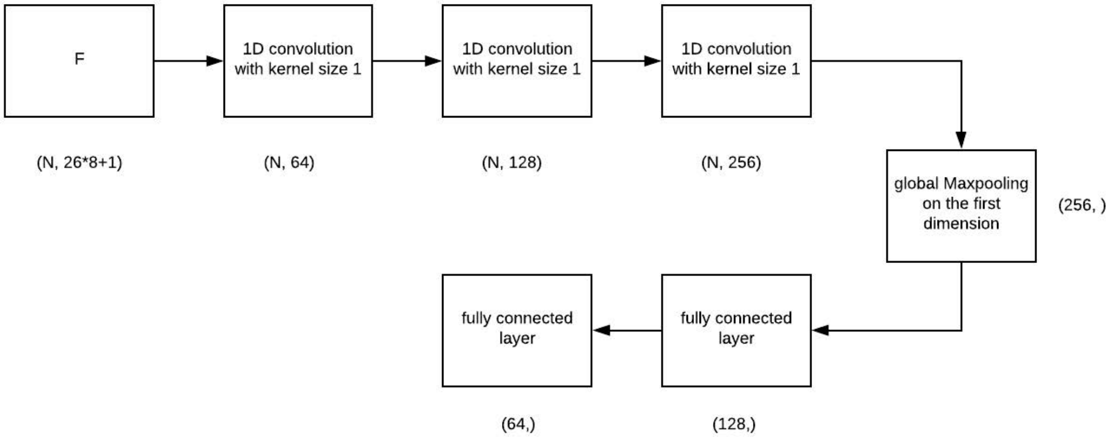
F
(N, 26*8+1)
1D convolution with kernel size 1
(N, 64)
1D convolution with kernel size 1
(N, 128)
1D convolution with kernel size 1
(N, 256)
global Maxpooling on the first dimension
(256,)
fully connected layer
(128,)
fully connected layer
(64,)

Figure 3: **T-Net** Module Organisation [5].

Thus, its predicted transformation matrix has a dimension of 64×64. This contributes to the difficulty of achieving optimization. To address this problem, a regularization term is added to the soft-max loss to constrain the 64×64 feature transformation matrix to be close to an orthogonal matrix [18]. Yet the imbalanced training set problem is more important in target detection tasks. Focal loss is very useful for training imbalanced datasets, especially for object detection tasks. Therefore, it has been applied instead of cross entropy loss when practicing the model in DeepNovoV2.

2.3 LSTM for peptide recombination

The algorithm uses a LSTM module to capture the “language model” of peptides. The LSTM model coupled to the T-Net is designed to learn sequence patterns of amino acids of the peptide in association with the corresponding spectrum. If we consider the spectrum intensity vector as an image, and the peptide sequence as a caption, the two models allows for the encoding of the intensity vector and decode the amino acids. LSTMs are a special kind of RNN, capable of learning long-term dependencies. LSTM processes data passing on information as it propagates forward. The operations of the LSTM cell are used to allow the LSTM to keep or forget information. The core concept of LSTM’s are the cell state, and it’s various gates represented in Fig.4.

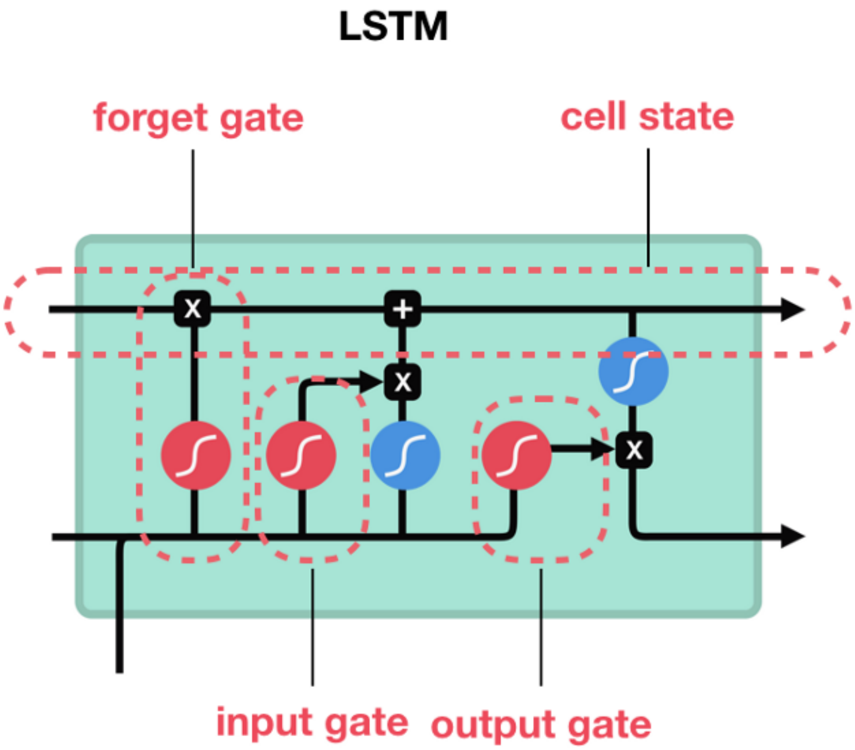

Figure 4: **LSTM Cell** and its operations [19].

The cell state relative information all the way down the sequence chain. It serves as a “memory” for the network. The cell state hold the necessary information during the processing of the sequence. As a result, information from early time steps will find its way to later time steps, reducing the impact of short-term memory. As the cell state goes through the process, information get’s added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state using sigmoid and tanh functions. The gates can learn what information is relevant to keep or forget during training. To sum up, the Forget Gate determines what is important to hold out of the previous phases. The input gate determines what information needs to be applied from the current stage. The output gate decides what the next hidden state is to be [20]. For the LSTM model, we use embedding vectors of size 512 to represent each of 26 symbols, as words in a vocabulary. Thus, the input to the LSTM model at each iteration is a vector of size 512. The initialization of the model is done using a sinusoidal m/z positional embedding, following the equations:

$$\text{PE}_{(loc,2k)} = \sin(\frac{loc}{10000^{\frac{2k}{d_{lstm}}}}) \tag{1}$$

$$\text{PE}_{(loc,2k+1)} = \cos(\frac{loc}{10000^{\frac{2k}{d_{lstm}}}}) \tag{2}$$

loc is the m/z location after discretization, and d is the dimension of the LSTM module. As the differences observed between peaks contains useful information. The sinusoidal positional embedding was therefore used to initialize the hidden states of the LSTM module. Finally, the LSTM architecture is composed of one layer of 512 neuron units and dropout layers with probability 0.25 for input and output.

2.4 *De Novo* prediction with multidimensional Knapsack Problem

Knapsack problems are well-studied combination optimization problems. In the context of a set of items, each with a weight and a value, the objective is to determine the number of each item to be included in the collection, so that the total weight is less than or equal to the limit and the total value is as large as possible. This takes its name from the dilemma posed by someone who is limited by a fixed-size knapsack and has to fill it with the most desirable objects. In our case, each predicted sequence is made up of a series of amino acids each with a projected likelihood. To solve this problem, Qiao et al. choose to use the binary (0–1) knapsack problem, where the decision-maker is allowed to select (1) or not to select (0) the object, in other words, the amino acids are not divisible [21] [22].

The version built in the *de novo* algorithm (called IonCNNDenovo) is not a CNN, but kept its name from the previous version of DeepNovo. The function uses the knapsack, to take values and weight arrays, number of items, and the capacity input as input. The matrix is set to the value “0”, such that if no objects are chosen, no value is generated. The solution is produced in polynomial time and space complexity. But the Knapsack dynamic programming algorithm provides a definite solution. Of this reason, this principle is widely used to find an optimal solution to problems, such as those illustrated by peptides. [23].

Beam Search (BS) is a search tree algorithm that uses a constructive approach, as define by Ponte *et al.* [25], there is a decision on the insertion or not of a solution component at each tree level. Typically, BS crosses the search tree by adopting the breadth-first approach and uses the knowledge given by the upper and lower bounds to pick the best solutions for the next tree stage. The α parameter is the number of nodes at each stage of the tree scan, typically referred to as beam width or beam size. Selecting only the right α solutions ensures that BS conducts a limited search in the list. For this reason, the optimal solution can not be found in general. Nonetheless, good solution consistency has been documented in a variety of hard optimization problems [26].

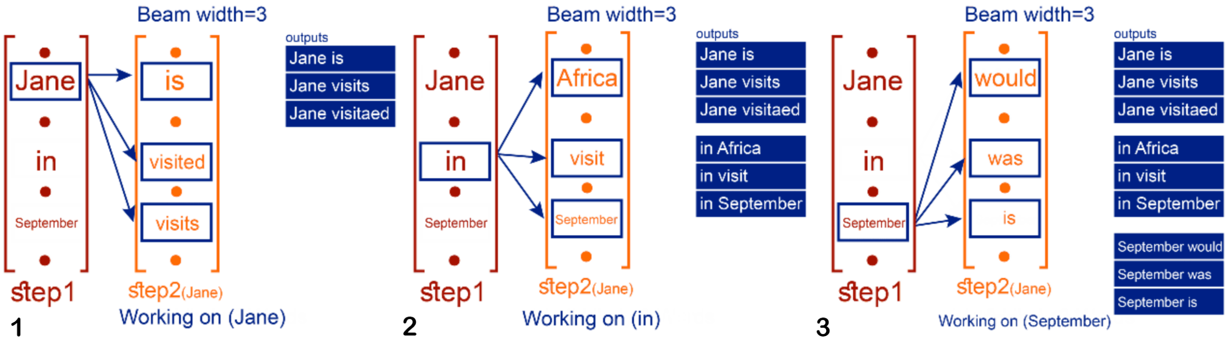

Figure 5: **Beam Search** with Beam Width=3 [24]

In their algorithm, Qiao *et al.* [5] use a "local" version of this dynamic programming, to simply filter amino acids not suitable for the suffix mass, without performing backtracking. This means that another algorithm can be implemented for better performance, but also that the algorithm learns better features that are not subject to be an issue for this problem-solving. The knapsack is generated if not present in the source of the algorithm. It is only dependent on the vocabulary set of the ions present in the data set used, and the masses. If these parameters do not change, the knapsack stays the same.

2.5 Data size and handling (Data Class)

To train the algorithm it can be interesting to use all the data available (all the peptides produced by a human for example, or a mouse, which can be sequenced with MS/MS) for example. This amount for a huge quantity of data. Thus, it is necessary that the entire data generation process does not become a bottleneck in the training procedure. In this deeplearning algorithm, using Python class Dataset, *Qiao et al.* are able to characterize the key features of the `dataset` they want to generate. They use the properties of `torch.utils.data.Dataset` in order to leverage functionalities such as multiprocessing. The dataset class is used to contain data systematically, but for the training loop, it is important to have the ability to slice and index the `dataset`. PyTorch offers this functionality with the function `DataLoader`, which act as a data feeder for an `dataset` object. It is possible to shuffle the data, determine the batch size and the number of workers to load the data on a GPU, and all that in a parallel way [27]. During the training phase, data is generated in parallel by the CPU, which can then be fed to the GPU for neural network computations.

The authors also use the utility function `to_one_hot` that they programmed. It allows for the conversion of a list of values into a list of integers. The `one_hot` vector is a row vector with the value of 1 at the index of the matrix encoded. This processing is used for the focal loss, to down-weights the loss assigned to well-classified examples.

2.6 Modifications to DeepNovoV2

Not only does the DeepNovoV2 source code lacks comments and documentation, but some of its functions are also designed to handle very specific cases. In particular, the raw sequence parsers present in `data_reader.py` and `deepnovo_worker_test.py` are designed to parse a hardcoded set of PTMs, with a format specific to the PEAKS software, that isn't modifiable by the user. Therefore, we provide a fork of the DeepNovoV2 repository, available at https://github.com/sdiebolt/DeepNovoV2, where we bypassed these parsers so that the set of allowed PTMs matches the vocabulary defined in `deepnovo_config.py`. This repository also contains the configuration needed to train the model on the data offered by the SMBP laboratory.

Furthermore, we dismantled DeepNovoV2's source code and created a Jupyter notebook implementation that can be run in a Google Colab environment with moderately sized datasets. Thus, we took advantage of the virtual GPUs offered by the platform: training the model using the data offered by the SMBP laboratory takes a maximum of two hours using our notebook, whereas it took more than ten hours on a server running two 12 cores Intel Xeon CPUs and 16GB of RAM.

The notebook follows a linear structure, with comments in each part of the model. All global variables previously available in `deepnovo_config.py` are declared at the beginning of the notebook. Finally, the Cython module available in `deepnovo_cython_modules.pyx` has been converted to a simple Python module, since using Cython on Jupyter notebooks isn't always reliable. This impacts performance, but the model still runs in adequate times on Google Colab.

2.7 Training Parameters & Predictions

DeepNovo V2 algorithm uses Adam Optimizer with an initial learning rate of 10^3. This learning rate can be adjusted during training to converge towards the best validation loss. The `adjust_learning_rate` function divides the learning rate after a selected number of training steps or epoch. The loss on the validation set is computed at a step frequency that can be set by the user. You can try different parameters depending on the size of your training data set. For small training data set, we tried computing the valid loss every 10 steps and thus divide the learning rate by 10 if the validation loss has not achieved a new low in the most recent ten evaluations.

The training of the algorithm can be saved and restarted. In fact, after each training, if it is satisfactory, the parameters of all the neural networks are saved in a path defined in the variables. Thus, launching a new training with saved parameters will load those previous parameters and adapt them, learning from the newly added dataset. This feature is very interesting, as Qiao *et al.* highlighted [1], to use a model pre-trained on a huge dataset very diverse, then to train it again on a smaller more specific dataset, to give it a focused edge on those specificities. But the ability to retrain the model is dependant on the parameters of the training. If the initial training used 5 layers of LSTM, the new training has to have the same parameters (which can be not possible is the computer used for retraining is less powerful then the one used for training).

3 Results

3.1 Outputs Interpretation

To evaluate the performance of *de novo* sequencing models, Qiao *et al.* [5] built a testing algorithm. They define a predicted amino "matched" with a real amino acid if their mass difference is less than 0.1 Da and if the prefix masses before them are different by less than 0.5 Da. The precision is defined as the amino acid level recall, calculated with the ratio of the total number of matched amino acids over the total number of amino acids in real peptide sequences. Similarly, the peptide level recall denotes the fraction of real peptide sequences that are fully correctly predicted. During training, the perplexity is calculated by the average focal-loss error. Using the ideas of perplexity, the average perplexity is displayed at the validation step, higher values mean more error. It gives a good idea of the error that the training model does on data with which it was not trained on. Qiao *et al.* also built a function inside the `WorkerTest` module that allows for the use of a file produce by PEAK DB, to compare the *de novo* sequence predicted by the model, with the ones that the software PEAK DB outputs. This function is in our case depricated, due to the fact that the predicted sequences will be compared to the output of MASCOT, initaly added to the feature file and initial spectrum. It should be taken into account that DeepNovoV2, will learn therefore either form the MASCOT method or the PEAK DB method, and thus we can assume it will take on their biases. That is why when trainning it is important to select reel reference data in order not to brough in biaises linked to the method of labeling.

3.2 Training with SMBP Data

The data offered by the SMBP laboratory consists of ten MGF spectrum files and their respective Mascot XML results. After features extraction using our Python module, we found that these spectra contained significantly less features than the dataset provided by DeepNovoV2's authors [5]—6,428 features after sequence annotation in the SMBP spectra, compared to 114,641 features in the authors' dataset. Moreover, the model couldn't be retrained since the PTMs used in the Mascot search were different than the default set of DeepNovoV2. Fortunately, the size of this dataset proved to be sufficient, as we achieved satisfactory results in the testing step.

After preprocessing the data using our Python module and splitting the data using a 80/10/10 ratio, the training was performed on a computer running a GTX 1080 with 8GB of VRAM and 16GB of RAM and took approximately 27 minutes. The vocabulary was adjusted with the desired PTMs for the SMBP dataset and the batch size was reduced from 32 to 24 to prevent any memory issues. Other model parameters were left to their default values. Figure 6 shows the evolution of the training and validation perplexity across 10 epochs, with 169 steps per epoch.

The best model was obtained at epoch 0, step 119. This is a bit worrying, as this could mean that the training data contains a lot of redundant information. However, this phenomenon was also observed when training the model using DeepNovoV2's authors dataset, with an optimal model

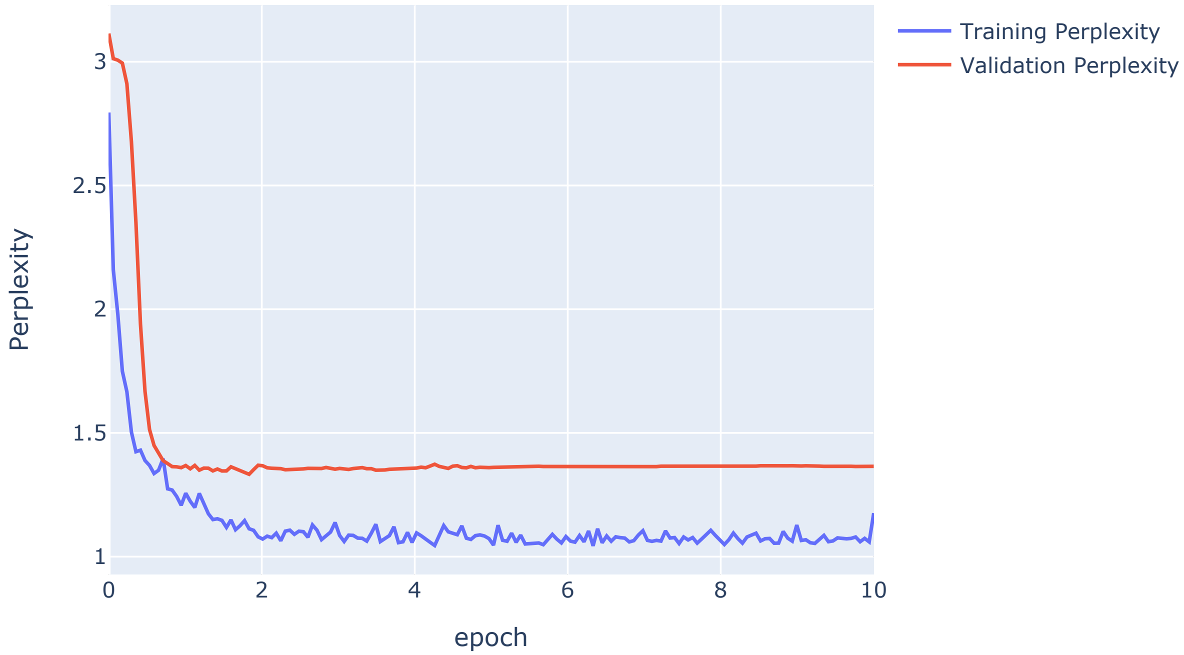

Figure 6: Evolution of the training and validation perplexity on SMBP data. The model used 169 steps per epoch and ran through 10 epochs.

achieved at epoch 4 out of 20. To test if this didn't lead to extensive overfitting, the model should be tested further using spectra coming from different sources (keeping in mind that the PTMs should be the same).

At the amino acid level, the testing resulted in a precision of 77% and a recall of 51%. At the peptide level, the precision was 51% and the recall was 35%. These results are consistent with those from the authors, but should be interpreted with caution, as we couldn't verify the presence of overfitting, given the possible redundancy in the data.

Conclusion

Qiao *et al.* proposed DeepNovoV2, a neural network-based *de novo* peptide sequencing model, and they claimed to outperform the previous state-of-the-art model. We tried to implement it on data from the SMBP, and so it could be used by the laboratory in the long run. Unfortunately, due to a lack of comment, the algorithm was very hard to read and interpret. This took a very long time to analyze each piece, and understand each function. Using a notebook to layout all the code and be able to run it on a smaller dataset was very useful, but time-consuming. One of the biggest downfalls was the computing power necessity that this model asks. Even the server provided by the SMBP was not enough and we had to run it on an adequate GPU, or vGPU with Colab. Nonetheless, this experience allowed us to discover a lot of good practice in Pytorch and Deep-Learning in general. We experimented with a lot of different new nothing like the knapsack problem and beam search. But also the implementation of LSTM to tune an image recognition neural network with an MLP. But also, the initial data preprocessing, from the result of MASCOT to the input of the model. And the analysis of the prediction of the model. As we saw in this experiment, training such a huge model is challenging and requires intensive attention to detail before each run. The results observed on the data from the SMBP are very encouraging, and the evolution of the focal-loss is coherent to a good training of the model.

The application of this model is very wide and with a more furnished dataset to train on, we believe the model can be well adapted to the needs of the SMBP. Using the same vocabulary and same model parameters, they will be able to continue the training with new data that they generate each day.

This work is very valuable to us and we would love to guide it and see it flourish. We want to thank the people of SMBP for their help and guidance in adapting this model.

Acronyms

CID collision-induced dissociation. 2–4

CNN convolutional neural network. 3

CSV comma-separated values. 3, 5

ESI electrospray ionization. 2

ETD Electron-Transfer Dissociation. 3

LSTM long short-term memory. 3, 13

MALDI matrix-assisted laser desorption/ionization. 2

MGF mascot generic format. 3–5, 11

MS mass spectrometry. 1, 2, 4, 5

MS/MS tandem mass spectrometry. 1–4, 9

PTM post-translational modification. 1, 3, 4, 10–12

SMBP Spectrométrie de Masse Biologique et Protéomique. 1, 4, 10–13

SNP single-nucleotide polymorphism. 3

T-Net transformation network. 6

TSV tab-separated values. 5

XML extensible markup language. 4, 5, 11

References

[1] N. H. Tran, X. Zhang, L. Xin, B. Shan, and M. Li, “De novo peptide sequencing by deep learning,” *Proceedings of the National Academy of Sciences*, vol. 114, pp. 8247–8252, Aug. 2017.

[2] M. B, Z. K, H. C, L. C, L. M, D.-K. A, and L. G, “Peaks: powerful software for peptide de novo sequencing by tandem mass spectrometry,” *Rapid communications in mass spectrometry : RCM*, 2003.

[3] F. A and P. P, “Pepnovo: de novo peptide sequencing via probabilistic network modeling,” Feb. 2005.

[4] M. B, “Novor: real-time peptide de novo sequencing software,” *Journal of the American Society for Mass Spectrometry*, Nov. 2015.

[5] R. Qiao, N. H. Tran, L. Xin, B. Shan, M. Li, and A. Ghodsi, “DeepNovoV2: Better de novo peptide sequencing with deep learning,” *arXiv:1904.08514 [cs, q-bio, stat]*, May 2019. arXiv: 1904.08514.

[6] J. Fenn, M. Mann, C. Meng, S. Wong, and C. Whitehouse, “Electrospray ionization for mass spectrometry of large biomolecules,” *Science*, vol. 246, pp. 64–71, Oct. 1989.

[7] M. Karas and F. Hillenkamp, “Laser desorption ionization of proteins with molecular masses exceeding 10,000 daltons,” *Analytical Chemistry*, vol. 60, pp. 2299–2301, Oct. 1988.

[8] A. K. Shukla and J. H. Futrell, “Tandem mass spectrometry: dissociation of ions by collisional activation,” *Journal of Mass Spectrometry*, vol. 35, no. 9, pp. 1069–1090, 2000.

[9] The UniProt Consortium, “UniProt: a hub for protein information,” *Nucleic Acids Research*, vol. 43, pp. D204–D212, Jan. 2015.

[10] D. A. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, “Genbank,” *Nucleic Acids Research*, vol. 41, pp. D36–D42, Jan. 2013.

[11] D. N. Perkins, D. J. C. Pappin, D. M. Creasy, and J. S. Cottrell, “Probability-based protein identification by searching sequence databases using mass spectrometry data,” *ELECTROPHORESIS*, vol. 20, no. 18, pp. 3551–3567, 1999.

[12] J. K. Eng, A. L. McCormack, and J. R. Yates, “An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database,” *Journal of the American Society for Mass Spectrometry*, vol. 5, pp. 976–989, Nov. 1994.

[13] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. Xu, D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant, “Open mass spectrometry search algorithm,” *Journal of Proteome Research*, vol. 3, pp. 958–964, Oct. 2004.

[14] B. Ma and R. Johnson, “De novo sequencing and homology searching,” *Molecular & cellular proteomics*, vol. 11, no. 2, 2012.

[15] “MGF file format.” http://www.matrixscience.com/help/data_file_help.html. Accessed: 2020-05-19.

[16] “DeepNovoV2 GitHub repository.” https://github.com/volpato30/DeepNovoV2. Accessed: 2020-05-19.

[17] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Honolulu, HI), pp. 77–85, IEEE, July 2017.

[18] J. Kossaifi, A. Bulat, G. Tzimiropoulos, and M. Pantic, “T-Net: Parametrizing Fully Convolutional Nets With a Single High-Order Tensor,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Long Beach, CA, USA), pp. 7814–7823, IEEE, June 2019.

[19] “Understanding LSTM Networks – colah’s blog.”

[20] M. Phi, “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation.” https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. Accessed: 2020-05-19.

[21] G. B. Mathews, “On the partition of numbers,” *Proceedings of the London Mathematical Society*, vol. 1, no. 1, pp. 486–490, 1896.

[22] C. Blum and G. Raidl, “Dynamic Programming,” in *Hybrid metaheuristics: powerful tools for optimization*, pp. 17 – 19, Springer, 2016. OCLC: 953510818.

[23] T. Sawik, “Selection of optimal countermeasure portfolio in IT security planning,” *Decision Support Systems*, vol. 55, pp. 156–164, Apr. 2013.

[24] A. Z. On, “Beam Search & Attention for text summarization made easy (Tutorial 5) | Hacker Noon.”

[25] A. Ponte, L. Paquete, and J. R. Figueira, “On beam search for multicriteria combinatorial optimization problems,” in *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pp. 307–321, Springer, 2012.

[26] L. N. Kolechkina and E. A. Rodionova, “Multicriteria combinatorial optimization problems on a set of polypermutations,” *Cybernetics and Systems Analysis*, vol. 44, pp. 276–282, Mar. 2008.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative

style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.