CSI 4106 Introduction to Artificial Intelligence

Assignment 3: Neural Networks

Marcel Turcotte

Version: Oct 18, 2024 10:31

◎ Learning Objectives

1. **Implement** baseline and neural network models; justify model choices based on task suitability.
2. **Apply** hyperparameter tuning and regularization techniques to improve model performance.
3. **Evaluate** models using cross-validation and performance metrics; make informed model recommendations.

In 2024, the scientific community recognized the profound impact of machine learning, as evidenced by the awarding of the Nobel Prizes in Physics and Chemistry. The Physics prize was given for "foundational discoveries and inventions that advance **machine learning** through artificial **neural networks**," while the Chemistry prize honored the development of "an **AI model** that addresses a long-standing challenge in predicting the complex structures of proteins."

Assignment 3 is inspired by these accomplishments. The **protein structure prediction problem** has been a major scientific challenge for over five decades. A team at Google DeepMind, led by Demis Hassabis and John M. Jumper, developed an AI model called AlphaFold. This model is the first to predict the intricate structures of proteins with an accuracy comparable to experimental methods.

Proteins are chains of amino acids that fold into specific shapes, essential for their functions. These shapes are organized into four structural levels: 1) The primary structure is the sequence of amino acids. 2) The secondary structure includes local patterns like α-helices and β-sheets. 3) The tertiary structure is the overall three-dimensional shape of a single protein chain. 4) The quaternary structure forms when multiple protein chains combine into a larger complex.

Prior to the breakthrough achieved by AlphaFold, protein secondary structure prediction was regarded as an intermediate step towards the comprehensive solution of the protein structure prediction problem.

Protein secondary structure prediction involves identifying the local structures (α-helix, β-sheet, or coil) that a protein sequence will adopt, based on its amino acid sequence. This sequence is represented as a string using a 20-letter alphabet, each letter corresponding to an amino acid. The objective is to map this sequence to a 3-letter alphabet, where 'H' denotes α-helix, 'E' denotes β-sheet, and 'C' denotes coil, thereby describing the secondary structure at each position.

Machine learning practitioners have been drawn to this problem due to its accessibility to those without formal scientific training. The task involves designing a computer program that takes an input string over a 20-letter alphabet (the amino acid sequence) and outputs a string of the same length using a 3-letter alphabet (the local structure). The problem can be conceptualized as a cipher: translating a message encoded with 20 letters into one encoded with three letters.

The following example demonstrates the transformation of a 46-character input sequence, where each character signifies a unique amino acid, into an output structure of identical length. This output employs three specific characters: C, E, and H.

```
input_sequence   = "TTCCPSIVARSNFNVCRLPGTPEAICATYTGCIIIPGATCPGDYAN"
output_structure = "CEECCCHHHHHHHHHHHHCCCCHHHHHHHHCCEECCCCCCCCCCCC"
```

Similar to natural languages, where the meaning of a word depends on its context, the local structure (H, E, or C) adopted at a specific position in a protein sequence is presumed to depend on the surrounding amino acids. This principle underlies our approach.

A detailed description of the dataset is available in the Appendix. The dataset was generated by applying a sliding window of length 21 across the sequences within a protein set. With each shift of the window by one position, a new example was generated, accompanied by its true target, i.e. a categorical variable indicating the secondary structure: 0 for α-helix, 1 for β-sheet, and 2 for coil. Each example consists of 462 attributes, which are numerical values ranging between 0 and 1. These numbers represent the probability of observing a specific amino acid at a given position within the context window, where each of the 21 positions is encoded with 22 values (21 positions $\times$ 22 values $=$ 462 attributes).

We provide three datasets: the training set includes 58,291 examples, the validation set contains 7,409 examples, and the test set comprises 7,432 examples. The **target variable**, located in the **first column**, can take on one of three values: 0, 1, or 2. The remaining 462 columns represent attributes, which are numerical values ranging from 0 to 1. Understanding this one paragraph is all you need to successfully complete the assignment.

Important Note: Managing expectations for this assignment is paramount due to the complexity of the protein secondary structure prediction problem. Historically, even the most advanced algorithms achieved accuracies below 70% (Rost and Sander 1993). Publications often highlighted improvements of just one percentage point. In our evaluations using those datasets, baseline methods like logistic regression achieved accuracies between 66% and 68%, while our neural networks approached 72% [1]. We provide these figures to prevent undue efforts in pursuing exceptionally high accuracy. Your grade is based on the quality of your work rather than solely on the performance of your models. This is an inherently challenging problem, as expected for one of Nobel Prize significance!

Submission

- **Deadline:**
  - Submit your notebook by November 10, 11 PM.
- **Individual or Group Assignment:**
  - This assignment may be undertaken either individually (group of 1 student) or collaboratively in pairs (group of 2 students)
  - A group must submit a single joint submission.
  - Prior to submitting, it is necessary to **register your group** on Brightspace.
  - To accommodate changes in group membership for each assignment, a new registration of groups is required for every assignment.

[1] Improved results require a larger dataset and advanced neural network architectures. Understanding our current traditional architecture is essential for grasping the principles of more complex models.

- **Submission Platform:**
  - Upload your submission to Brightspace under the Assignment section (Assignment 3).
- **Submission Format:**
  - Submit a copy of your notebook on Brightspace.

Important Notice: If the corrector cannot run your code, your submission will receive a mark of zero. It is your responsibility to ensure that your submission works from a different computer than your own and that all cells in your notebook are executable.

Requirements

1. Exploratory Analysis

Loading the dataset

A custom dataset has been created for this assignment. It has been made available on a public GitHub repository:

- github.com/turcotte/csi4106-f24/tree/main/assignments-data/a3

You can access and read the dataset directly from this GitHub repository in your Jupyter notebook.

(1) **Load the Dataset**:

- Write code to load the three datasets.

Data Pre-Processing

(2) **Shuffling the Rows**:

- Since examples are generated by sliding a window across each protein sequence, most adjacent examples originate from the same protein and share 20 positions. To mitigate the potential negative impact on model training, the initial step involves shuffling the **rows** of the data matrix.

(3) **Scaling of Numerical Features**:

- Since all 462 features are proportions represented as values between 0 and 1, scaling may not be necessary. In our evaluations, using StandardScaler actually degraded model performance. Within your pipeline, compare the effects of not scaling the data versus applying MinMaxScaler. In the interest of time, a single experiment will suffice. It is important to note that when scaling is applied, a uniform method should be used across all columns, given their homogeneous nature.

(4) **Isolating the Target and the Data**:

- In the CSV files, the target and data are combined. To prepare for our machine learning experiments, separate the training data X and the target vector y for each of the three datasets.

Model Development & Evaluation

(5) **Model Development**:

- **Dummy Model**: Implement a model utilizing the DummyClassifier. This model disregards the input data and predicts the majority class. Such model is sometimes called a straw man model.
- **Basline Model**: As a baseline model, select one of the previously studied machine learning algorithms: Decision Trees, K-Nearest Neighbors (KNN), or Logistic Regression. Use the default parameters provided by scikit-learn to train each model as a baseline. Why did you choose this particular classifier? Why do you think it should be appropriate for this specific task?
- **Neural Network Model**: Utilizing Keras and TensorFlow[2], construct a sequential model comprising an input layer, a hidden layer, and an output layer. The input layer should consist of 462 nodes, reflecting the 462 attributes of each example. The hidden layer should include 8 nodes and employ the default activation function. The output layer should contain three nodes, corresponding to the three classes: helix (0), sheet (1), and coil (2). Apply the softmax activation function to the output layer to ensure that the outputs are treated as probabilities, with their sum equaling 1 for each training example.

We therefore have three models: dummy, baseline, and neural network.

(6) **Model Evaluation**:

- Employ cross-validation to assess the performance of the baseline model. Select a small number of folds to prevent excessive computational demands.
- **Training neural networks can be time-consuming.** Consequently, their performance is typically assessed once using a validation set. Make sure to not use the test set until the very end of the assignment.
- Assess the models using metrics such as precision, recall, and F1-score.

Hyperparameter Optimization

(7) **Baseline Model:**

- To ensure a fair comparison for our baseline model, we will examine how varying hyperparameter values affect its performance. This prevents the erroneous conclusion that neural networks inherently perform better, when in fact, appropriate hyperparameter tuning could enhance the baseline model's performance.
- Focus on the following relevant hyperparameters for each model:
  - DecisionTreeClassifier: `criterion` and `max_depth`.
  - LogisticRegression: `penalty`, `max_iter`, and `tol`.
  - KNeighborsClassifier: `n_neighbors` and `weights`.

[2]Keras and TensorFlow are pre-installed in the Google Colab environment. If you are conducting experiments on your personal computer, installing the latest release of TensorFlow, version 2.17 at the time of writing, will also include Keras.

- Employ a grid search strategy or utilize scikit-learn's built-in methods GridSearchCV to thoroughly evaluate all combinations of hyperparameter values. Cross-validation should be used to assess each combination.
- Quantify the performance of each hyperparameter configuration using precision, recall, and F1-score as metrics.
- Analyze the findings and offer insights into which hyperparameter configurations achieved optimal performance for each model.

(8) **Neural Network:**

In our exploration and tuning of neural networks, we focus on the following hyperparameters:

- **Single hidden layer, varying the number of nodes**.
  - Start with a single node in the hidden layer. Use a graph to depict the progression of loss and accuracy for both the training and validation sets, with the horizontal axis representing the number of training epochs and the vertical axis showing loss and accuracy. Training this network should be relatively fast, so let's conduct training for 50 epochs. Observing the graph, what do you conclude? Is the network underfitting or overfitting? Why?
  - Repeat the above process using 2 and 4 nodes in the hidden layer. Use the same type of graph to document your observations regarding loss and accuracy [3].
  - Start with 8 nodes in the hidden layer and progressively double the number of nodes until it surpasses the number of nodes in the input layer. This results in seven experiments and corresponding graphs for the following configurations: 8, 16, 32, 64, 128, 256, and 512 nodes. Document your observations throughout the process.
  - Ensure that the **number of training epochs** is adequate for **observing an increase in validation loss**. **Tip**: During model development, start with a small number of epochs, such as 5 or 10. Once the model appears to perform well, test with larger values, like 40 or 80 epochs, which proved reasonable in our tests. Based on your observations, consider conducting further experiments, if needed. How many epochs were ultimately necessary?
- **Varying the number of layers**.
  - Conduct similar experiments as described above, but this time vary the number of layers from 1 to 4. Document your findings.
  - How many nodes should each layer contain? Test at least two scenarios. Traditionally, a common strategy involved decreasing the number of nodes from the input layer to the output layer, often by halving, to create a pyramid-like structure. However, recent experience suggests that maintaining a constant number of nodes across all layers can perform equally well. Describe your observations. It is acceptable if both strategies yield similar performance results.
  - Select one your models that exemplifies overfitting. In our experiments, we easily constructed a model achieving nearly 100% accuracy on the training data, yet showing no similar improvement on the validation set. Present this neural network along with its accuracy and loss graphs. Explain the reasoning for concluding that the model is overfitting.

[3] In this assignment, whenever you are asked to describe or document your observations, we expect you to generate a graph that displays loss and accuracy for both the training and validation sets as a function of the number of epochs. Use this graph to support your explanation.

- **Activation function**.
  - Present results for one of the configurations mentioned above by varying the activation function. Test at least `relu` (the default) and `sigmoid`. The choice of the specific model, including the number of layers and nodes, is at your discretion. Document your observations accordingly.
- **Regularization** in neural networks is a technique used to prevent overfitting.
  - One technique involves adding a penalty to the loss function to discourage excessively complex models. Apply an `l2` penalty to some or all layers. Exercise caution, as overly aggressive penalties have been problematic in our experiments. Begin with the default `l2` value of 0.01, then reduce it to 0.001 and 1e-4. Select a specific model from the above experiments and present a case where you successfully reduced overfitting. Include a pair of graphs comparing results with and without regularization. Explain your rationale to conclude that overfitting has been reduced. Do not expect to completely eliminate overfitting. Again, this is a challenging dataset to work with.
  - Dropout layers are a regularization technique in neural networks where a random subset of neurons is temporarily removed during training. This helps prevent overfitting by promoting redundancy and improving the network's ability to generalize to new data. Select a specific model from the above experiments where you have muliple layers and experiment adding one or of few dropout layers into your network. Experiment with two different rates, say 0.25 and 0.5. Document your observations.
  - Summarize your experiments with using a graphical representation such as Figure 6.15 on this page.
  - Early stopping is a regularization technique in neural network training wherein the process is halted when validation set performance starts to decline, thus preventing overfitting by avoiding the learning of noise in the training data. From all the experiments conducted thus far, choose **one** configuration (the number of layers, number of nodes, activation function, L2 penalty, and dropout layers) that yielded the best performance. Use a graph of loss and accuracy to determine the optimal number of training iterations for this network. What is the optimal number of epochs for this network configuration and why?

Test

(9) **Model Comparison**:

- Evaluate the baseline model on the test set, using the optimal parameter set identified through grid search. Additionally, apply your best-performing neural network configuration to the test set.
- Quantify the performance of the baseline model (best hyperparameter configuration) and your neural network (best configuration) using precision, recall, and F1-score as metrics. How do these two models compare to the dummy model?
- Provide recommendations on which model(s) to choose for this task and justify your choices based on the analysis results.

2. Documentation of Exploratory Analysis

The report should comprehensively document the entire process followed during this assignment. While grading Assignment 1, we observed that some students submitted code without adequate descriptions or clarifications. Please be aware that a lack of comments will result in mark deductions. Utilize the structure of Jupyter notebooks effectively by dividing lengthy programs into multiple code cells, interspersed with clear explanations in markdown cells. The Jupyter Notebook must include the following:

- Your name(s), student number(s), and a report title.
- If you worked in pairs during the assignment, explain how the tasks have been split between the members. How did you make sure that both students achieve the learning outcomes?
- A section for each step of the exploratory analysis, containing the relevant Python code and explanations or results.
  - For sections requiring Python code, include the code in a cell.
  - For sections requiring explanations or results, include these in a separate cell or in combination with code cells.
- Ensure logical separation of code into different cells. For example, the definition of a function should be in one cell and its execution in another. Avoid placing too much code in a single cell to maintain clarity and readability.
- The notebook you submit must include the results of the execution, complete with graphics, ensuring that the teaching assistant can grade the notebook without needing to execute the code.

Evaluation

- **Overall Effort in the Report (5%)**
- **Loading and Pre-processing (5%)**
- **Model Development (5%)**
- **Model Evaluation (10%)**
- **Parameter Optimization for the Baseline Model (10%)**
- **Parameter Optimization for the Neural Network Model (50%)**
- **Analysis of Results (10%)**
- **Resources and References (5%)**

Resources

As previously mentioned, ensure that you cite any parts of your code that are derived from websites, textbooks, or other external resources. Even if you were already familiar with the concepts, be sure to cite the libraries you have utilized.

Currently, many programmers leverage artificial intelligence to enhance their productivity, a trend that is likely to continue growing. To better prepare you for the job market, it is plausible to utilize these technologies. However, it is imperative that you fully understand the concepts upon which you are evaluated, as these tools will not be available during in-person evaluations.

If you do use AI assistance, thoroughly document your interactions. Include the tools and their versions in your report, along with a transcript of all interactions. Most AI assistants keep a record of your conversations. The recommended practice is to create a new conversation specifically for the assignment and consistently reuse this conversation throughout your work on the assignment. Ensure that this conversation is solely dedicated to the assignment. Submit this conversation transcript in the reference section of your Jupyter Notebook.

Questions

- You may ask your questions in the Assignment topic of the discussion forum on Brightspace.
- Alternatively, you can email one of the four teaching assistants. However, using the forum is strongly preferred, as it allows your fellow students to benefit from the questions and the corresponding answers provided by the teaching assistants.

Dataset Description

What follows is an outline of the process used to produce the dataset. This dataset is derived from the widely utilized CB513 dataset (Cuff and Barton 1999). As indicated by its name, CB513 consists of 513 protein sequences, with lengths varying from 21 to 755 amino acids.

Adhering to the principle that the local structure at a specific position in a protein sequence depends on its surrounding amino acids, each example comprises a context window of length 21 and a target variable. This target variable indicates the local structure of the central amino acid in the window (position 11), taking one of three values: 0 for helix, 1 for sheet, and 2 for coil. The context encompasses 10 positions before and 10 positions after the central position, along with the central position itself ($(10 + 1 + 10 = 21)$). Conveniently, each example is encoded such that the target variable is in the first column, followed by the encoded attributes in the remaining columns.

We could have used the one-hot-encoding. Each amino acid would have been transformed into a unique vector where one element is "1" (hot) and the rest are "0" (cold).

```python
import numpy as np
from sklearn.preprocessing import OneHotEncoder

# One letter code for the 20 amino acids

aa = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']

onehot_encoder = OneHotEncoder(categories=[aa], sparse_output=False)

# input_sequence was defined in the introduction

X = np.array(list(input_sequence)).reshape(-1, 1)

X_encoded = onehot_encoder.fit_transform(X)

# First five encoded amino acids of input_sequence
```

```python
print(X_encoded[:5])
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

However, we can leverage the fact that CBS513 includes evolutionarily related sequences for each of the 513 input sequences, which are known to adopt the same structure. Consequently, each position within the context window is represented by a vector of length 22. This vector accounts for the 20 naturally occurring amino acids, one symbol for unknowns, and one for gaps. Our representation accommodates unknowns or missing values, a standard procedure in bioinformatics that will not adversely affect your results. The gap symbol indicates that some evolutionary related proteins may have experienced deletions. Each position in this 22-length vector is a real number representing the proportion of a specific amino acid at that position. For example, if the first position corresponds to 'A' and its value is 0.2, this indicates that 20% of the related proteins have an 'A' at that position. This encoding approach is known to yield improved results. The segment from 1m 12s to 1m 30s in the video What Is AlphaFold? illustrates this process. Specifically, our dataset is based on the MSA representation introduced at 1m 30s in the video.

To prevent data leakage, three distinct datasets were constructed at the protein level. As examples are generated by sliding a window across each protein sequence, multiple examples originate from the same sequence. To mitigate leakage, proteins were divided into three groups—training, validation, and test, ensuring that the total number of amino acids in each group aligns with a predetermined ratio of the overall amino acid count.

Videos

- Protein folding explained by Google DeepMind. Posted on November 30, 2020. (1m 51s)
- What Is AlphaFold? by the New England Journal of Medicine (NEJM) Group. Posted on September 21, 2023. (5m 14s)
- AlphaFold: The making of a scientific breakthrough by Google DeepMind. Posted on November 30, 2020. (7m 54s)

References

Cuff, James A., and Geoffrey J. Barton. 1999. "Evaluation and Improvement of Multiple Sequence Methods for Protein Secondary Structure Prediction." *Proteins: Structure, Function, and Bioinformatics* 34 (4): 508–19.

Rost, B., and C. Sander. 1993. "Improved Prediction of Protein Secondary Structure by Use of Sequence Profiles and Neural Networks." *Proc. Natl. Acad. Sci. U.S.A.* 90 (16): 7558–62.