HSLU

AI/ML

Ribonucleic Acid Molecule Structure Prediction

Predict the structure of any RNA molecule and the resulting chemical mapping profile

AICH Report
Team 3 HSLU StableConfusion

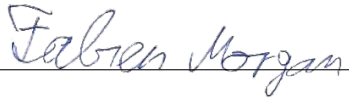
Morgan Fabien, Kluser Flavio,
Kessler Joël, Nyungmartsang Choekyel

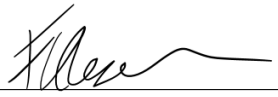
Module Coordinator: Prof. Dr. Jana Köhler
Supervisor: Dr. Andreas Streich

AI Challenge HS2023
Hochschule Luzern - Departement Informatik
January 14, 2024

Eidesstattliche Erklärung

Hiermit bestätigen wir, dass wir die vorliegende Arbeit eigenständig und ohne unerlaubte externe Unterstützung verfasst haben. Alle verwendeten Quellen, Literatur und andere Hilfsmittel, einschliesslich ChatGPT, wurden ordnungsgemäss zitiert. Passagen, die wortwörtlich oder inhaltlich aus anderen Quellen übernommen wurden, sind entsprechend gekennzeichnet. Wir verpflichten uns dazu, die Vertraulichkeit des Auftraggebers zu wahren und die Urheberrechtsbestimmungen der Fachhochschule Zentralschweiz (siehe das Dokument "Merkblatt Studentische Arbeiten" auf MyCampus) zu respektieren.

Rotkreuz, January 14, 2024 

Rotkreuz, January 14, 2024 

Rotkreuz, January 14, 2024 

Rotkreuz, January 14, 2024 

Acknowledgements

First of all, we thank our supervisor Dr. Andreas Streich, for his guidance and expertise throughout the challenge.

We thank the Hochschule Luzern for giving us the opportunity to participate in this challenge.

We thank all of our colleagues who helped us with their feedback and support.

We thank the Kaggle community for their support and feedback.

A special mention is also due to Luis Vieira, who is a MSc student in Biomedicine at UZH and who helped us with his expertise in the field of RNA folding.

Abstract

This report offers a detailed exploration and solution strategy for a competition centered on predicting RNA structures. The initial section introduces the competition's context, outlining its datasets, evaluation metrics, and key challenges. An overview of RNA and its fundamental structures is provided, highlighting the importance of understanding its primary, secondary, and tertiary forms. The subsequent sections focus on data quality assessment, feature engineering, and the tokenization and padding processes, crucial for preparing the data for machine learning models.

In the machine learning theory section, the report covers the foundational principles of modeling RNA structures. It discusses the adoption of libraries like arnie and forgi for visualizing RNA sequences, the tokenization of categorical features, and the critical step of padding to standardize sequence lengths. The creation of a unique evaluation metric, Reactivity Weighted Mean Absolute Error (rewMAE), tailored for the specificities of RNA data, is also highlighted. The report details the selection of models, the incorporation of additional features through feature engineering, and preparing the data for model training.

The modeling section provides an in-depth analysis of the models used, including the use of a transformer. It offers insights into hyperparameter tuning strategies unique to each model. The conclusion summarizes the findings and contributions of the report, highlighting the significance of accurate RNA structure prediction in biological research and potential applications. The reflective section shares insights into the challenges faced during teamwork, the decision-making process, and the overall workflow of preparing, modeling, and evaluating RNA data. In summary, this report serves as a comprehensive guide to predicting RNA structures through machine learning techniques, contributing valuable insights for future undertakings in this field.

Contents

List of Figures

List of Tables

Glossary

2A3	2-aminopyridine-3-carboxylic acid imidazolide.
A, C, G, U	Adenine, Cytosine, Guanine and Uracil.
AI	Artificial Intelligence.
AIML	Artificial Intelligence and Machine Learning.
BiRNN	Bidirectional Recurrent Neural Network.
BPPM	Base Pair Probability Matrix.
CNN	Convolutional Neural Network.
DMS	dimethyl sulfate.
LLM	Large Language Model.
LSTM	Long Short-Term Memory.
MaP	Mutational Profiling.
ML	Machine Learning.
NLP	Natural Language Processing.
ReLU	Rectified Linear Unit.
RNA	Ribonucleic Acid.
RNN	Recurrent Neural Network.
S, I, M, H, E	Stem, Internal loop, Multi-loop, Hairpin loop, Unpaired nucleotides.
tanh	tangent hyperbolicus.

1. Introduction

1.1. AI Challenge

The AI Challenge module is a must for all AIML students and is typically taken in the second-to-last semester, serving as preparation for the bachelor thesis. It involves applying knowledge gained from previous modules to real-world scenarios. Students can choose between participating in a Kaggle competition (focused on research or featured categories) or assisting the HSLU research team in the Robo Cup.

Our team, consisting of members Kessler Joel, Kluser Flavio, Morgan Fabien and Nyungmartsang Choekyel, chose a Kaggle competition over the Robo Cup. We had to select challenges that not only fell into our chosen category but were also currently active. After considering about 10 Kaggle challenges meeting these criteria, we decided to take on the research challenge called Stanford Ribonanza RNA Folding.

1.2. Kaggle Competition

This section first highlights general information about the scope of the AI Challenge. Furthermore, it provides a comprehensive overview of the scope and conditions of the chosen competition, including the evaluation metric, timeline, prize money, and number of participants.

1.2.1. Competition Host Information

The Stanford University, specifically the DAS Lab, hosts this competition. The DAS Lab focuses on understanding how RNAs work in living systems, using computational and chemical tools to model and design them precisely. This challenge is part of the EteRNA project, an initiative by the lab. (“DAS Lab”, n.d.)

EteRNA is an educational platform teaching about RNA and its various shapes. Participants control RNA bonds (ACGU) to influence the molecule’s shape. The EteRNA community has previously contributed to scientific advancements, including designing riboswitches and improving mRNA vaccine shelf life. Now, EteRNA is creating diverse sequences for complex structures, forming the basis of this Kaggle competition. (“Eterna”, n.d.)

1.2.2. Competition Introduction

Ribonucleic acid (RNA) plays an important role in biological functions, holding the potential for groundbreaking advancements in medicine, including treatments for diseases like pancreatic cancer and Alzheimer’s, as well as the development of antibiotics. To unlock this potential, a deeper comprehension of RNA structure is necessary, making it an ideal challenge for data science.

However, previous attempts to predict RNA structure faced challenges such as limited training data, computational constraints, and difficulties in separating training and test data. This competition addresses these issues by providing experimental measurements of chemical reactivity at

each position of an RNA molecule. These measurements, sensitive to the RNA's structure in the test tube, offer a unique opportunity for algorithms to 'understand' RNA structures.

A model that does well in this competition can be a helpful tool for scientists working on medicine. It can help them find targets for drugs based on RNA and make it easier to create medicines that use RNA, such as mRNA vaccines and CRISPR gene treatments. Besides being important for medicine, RNA molecules are essential for life processes. They have influenced life from the earliest forms on Earth to plants and marine organisms that play a key role in our planet's carbon cycle. Getting a good understanding of RNA is crucial for uncovering the mysteries of life. ("Kaggle Competition information", n.d.)

1.2.3. Competition Conditions

1.2.3.1. Evaluation Metric

A standardised evaluation metric is provided by the host of the competition. The metric is the mean absolute error (MAE) between the predicted and the actual reactivity values. The MAE is calculated for each of the two ground truth values (DMS and 2A3) separately. The MAE is calculated as follows:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} | \, y_i - \hat{y}_i \, | \tag{1.1}$$

where,
N : Number of scored ground truth values
y_i : Actual values
$\hat{y}_i$: Predicted values

The y_i values will be clipped between 0 and 1 before calculating MAE

$$y_i = \max(\min(y_i^{RAW}, 1.0), 0.0) \tag{1.2}$$

where,
y_i^{RAW} : Raw data values

Each RNA sequence has some number of nucleotides (positions). For each position, there will be two ground truth values, corresponding to reactivity determined from two kinds of chemical mapping experiments, DMS and 2A3.

1.2.3.2. Timeline

The timeline of the competition is as follows and all deadlines are at 11:59 PM UTC on the corresponding day of the competition:

- **07 September 2023:** Start of the competition - The competition has been published on Kaggle with all relevant information about its scope and conditions.
- **30 November 2023:** Entry deadline - One must accept the competition rules before this date in order to compete.

  Team Merger deadline - This is the last day participants may join or merge teams.
- **07 December 2023:** Final submission deadline.

1.2.3.3. Prize Money

The competition offered a total of 100,000$ in prize money. The breakdown of the prize money is shown below.

- 1st-Place: 40.000$
- 2nd-Place: 25.000$
- 3rd-Place: 15.000$
- 4th-Place: 8.000$
- 5th-Place: 7.000$
- 6th-Place: 5.000$

1.2.4. Number of Participants

755 teams participated in the competition. The number of participants in each team varied from 1 to 5.

2. Ribonucleic Acid Terminology

Before we can start to talk about the different types of Models, we need to define some basic terms. This chapter will give a short introduction to the terminology used in this competition. The first section will give a short introduction to *Ribonucleic Acid*. The second section will give a short introduction to the *function and structure* of RNA. The last section will give a short introduction to the *folding* of RNA.

2.1. Ribonucleic Acid (RNA)

RNA, or ribonucleic acid, serves as a crucial molecular messenger, relaying genetic information from DNA (deoxyribonucleic acid) to enable cellular functions. Imagine it as a vital courier transferring essential genetic instructions from your DNA to your cellular machinery. This process ensures the construction and operation of proteins, the functional workers within the cells. ("RNA - central molecule of life", n.d.)

In simpler terms, RNA acts as a translator, ensuring that cellular tasks proceed according to the genetic blueprints stored in the DNA.

2.2. Function and Structure of RNA

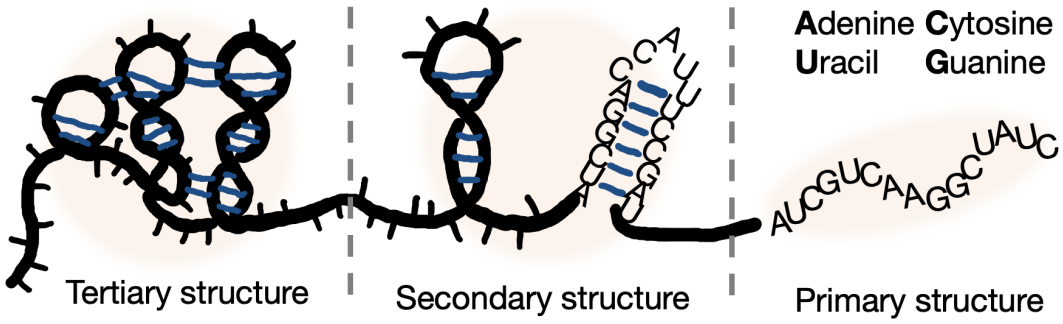


Figure 2.1.: Representation of Ribonucleic Acid (RNA) Structures

Figure 2.1 shows from right to left, RNA's primary, secondary, and tertiary structures. These structures play critical roles in storing genetic information, determining folding patterns, and influencing cellular functions. (Rogers, n.d.)

2.2.1. Primary Structure

The primary structure of RNA is a linear sequence of nucleotides, the molecular building blocks. Each nucleotide comprises three components: ribose (a sugar), a phosphate group, and a nitrogenous base (A=adenine, C=cytosine, G=guanine, or U=uracil). The sequence of these bases dictates the genetic information encoded in the RNA. (Minchin & Lodge, 2019)

2.2.2. Secondary Structure

RNA's secondary structure refers to how the molecule folds due to interactions between nucleotides. These interactions primarily involve base pairing: A pairs with U, and C pairs with G. This folding creates various shapes, such as stem-loop structures and internal loops, vital for its function within cells. (Dirks et al., 2004)

2.2.3. Tertiary Structure

The tertiary structure of RNA represents its complete three-dimensional form, shaped by intricate interactions among different segments of the molecule. This structure determines how RNA engages with other molecules such as proteins, thereby impacting its role and behavior within cells. ("Tertiary Structure of Protein", n.d.)

2.3. RNA Folding

RNA folding is the intricate process where RNA assumes its 3D structure, involving its primary, secondary, and tertiary structures. It undergoes bending and arranging to create a specific 3D pattern, a process dynamic and responsive to environmental cues or molecule interactions. (Rogers, n.d.)

Predicting RNA folding and tertiary structure computationally remains a challenging task. Researchers use algorithms and experimental data to model and comprehend RNA's 3D shapes. This understanding is crucial in developing accurate models, a fundamental aspect of this competition.

3. Research on Chemical Reactivity

In this chapter, we explore the role of chemical reactivity in understanding the structure of RNA molecules, specifically through DMS (dimethyl sulfate) and 2A3 (2-aminopyridine-3-carboxylic acid imidazolide) mapping experiments. The unique 3D structure of an RNA molecule leads to some regions being more exposed, while others are more hidden. Chemicals like DMS and 2A3 react differently based on this exposure, offering insights into the molecule's structure.

3.1. Dimethyl Sulfate (DMS)

DMS interacts with specific nucleobases in RNA, offering insights into their exposure or pairing. (Mitchell et al., 2023)

3.2. 2-Aminopyridine-3-Carboxylic Acid Imidazolide (2A3)

In contrast, 2A3 targets RNA's sugar components, assesses their flexibility, and identifies exposed regions in the folded structure. (Marinus et al., 2021)

3.3. Similarities and Differences

Similarities

- Both DMS and 2A3 offer precise insights into RNA structures by measuring reactivity at different positions.
- These reagents are powerful tools for understanding RNA structure

Differences

- 2A3 excels in probing RNA structure in living cells, outperforming other SHAPE (Selective 2'-Hydroxyl Acylation analyzed by Primer Extension) reagents. (Marinus et al., 2021)
- DMS, traditionally probing adenine and cytosine, has evolved to cover all nucleotides, providing enhanced structural information. (Mitchell et al., 2023)
- DMS's four-base reactivities and improved MaP strategy contribute to accurate RNA structural modeling. (Mitchell et al., 2023)

In short, DMS and 2A3 are both helpful for understanding RNA structure, but they work in different ways. When used together, they give us better information about RNA than when used separately. This improves our understanding of the Kaggle competition where we predict RNA structure.

4. Machine Learning Theory

This chapter contains all the theoretical aspects of the machine learning approaches we used in this competition including the workflow, frameworks, models and hyperparameter tuning. This chapter is written with the help of ChatGPT.

4.1. Machine Learning Workflow

"A machine learning workflow involves a streamlined process to ensure efficient model development and reproducibility." (Chat GPT, 2023f) "Versioning and experiment tracking form the backbone of a robust ML workflow. Versioning ensures that every change in data, models, and code is tracked and reversible, while experiment tracking records the comprehensive details of each experiment for reproducibility and comparison. These practices are integral to developing effective and reliable ML models in a systematic and collaborative manner." (Chat GPT, 2023g)

4.1.1. Versioning

"Versioning in machine learning and software development is crucial for tracking changes, maintaining historical records, and ensuring transparency in project evolution. It facilitates collaboration by coordinating work among team members, managing concurrent updates, and preventing conflicts. Additionally, versioning is vital for experimentation in machine learning, allowing for testing new ideas without disrupting the main project, thereby balancing innovation with the stability of the primary codebase." (Chat GPT, 2023o)

4.1.1.1. Git

"Git, our chosen tool for version control, specifically addresses several needs in our project. Its features of branching and merging are fundamental to our versioning strategy. Branching allows team members to create separate streams of work, such as developing new features or testing, in parallel with the main project. This approach facilitates experimentation and development without affecting the core project. Once these branches are finalized, they can be merged back

Git's distributed nature means that every team member has a local copy of the entire project history, enabling work even when offline and providing an additional layer of backup. This feature is particularly beneficial for large teams or teams working remotely." (Chat GPT, 2023o)

4.1.1.2. GitHub

"The project's codebase on GitHub streamlines versioning and collaboration. As a cloud-based platform, GitHub allows easy access and sharing of the project's code from anywhere. It serves as a central repository for team members to update the latest project version, ensuring synchronized development. GitHub's user-friendly interface aids in reviewing code, discussing changes, and managing pull requests. Additionally, it offers free access for students and is reliable, allowing the team to concentrate on the machine learning aspect of the project." (Chat GPT, 2023p)

4.1.2. Experiment Tracking

"Experiment tracking in machine learning is a systematic approach to record and manage all aspects of ML experiments, vital for maintaining efficiency and effectiveness. Its primary purpose is ensuring reproducibility by documenting data, models, hyperparameters, and environments. This facilitates model comparison, enhances team collaboration, and monitors progress to understand successful or unsuccessful approaches. Integrated into the ML workflow, it includes automated logging compatible with ML frameworks, user annotations for context, visualization tools for metric comparison, and alerts based on specific criteria like target metrics achievement." (Chat GPT, 2023a)

4.1.2.1. Weights & Biases

We decided to use Weights & Biases for Experiment Tracking since it offers a wide range of features and excellent collaboration functionality. The API is well-documented and available for virtually every machine learning framework, including PyTorch and PyTorch Lightning.

In terms of features, we want to use the following:

- Collaboration and Sharing
- Visualization of metrics
- Record of all Metrics, Hyperparameters and Run Metadata
- Attachment of models

The Weight & Biases module has been integrated in the training class to track all training runs by default. All data is accessible under https://wandb.ai/hslu-stableconfusion/hslu-stableconfusion for the participants.

4.2. Machine Learning Frameworks

"Machine learning frameworks are vital in developing and implementing models, particularly in neural networks and data preprocessing. They provide a structured, efficient approach with prebuilt algorithms and essential components like layers and optimizers, making it easier to construct both simple and advanced models. These frameworks also assist in data preprocessing, offering tools for normalization, handling missing values, and converting data into a uniform format.

Utilization of these frameworks streamlines model development by reducing coding requirements and enforcing standardized practices, crucial in collaborative environments. Built for scalability, they can handle diverse data sizes and complexities. Additionally, extensive community support offers resources, tutorials, and forums, ensuring compatibility with various data sources, hardware, and software tools." (Chat GPT, 2023m)

4.2.1. Deep Learning Frameworks

4.2.1.1. PyTorch

"PyTorch is a popular open-source machine learning library known for its flexibility, ease of use, and dynamic computational graph feature. It has become a go-to tool for developers and researchers working in the field of deep learning and artificial intelligence. PyTorch's dynamic computation graph, known as Autograd, is a core feature that sets it apart from other frameworks. This means that the network behavior can be altered on-the-fly during runtime, offering exceptional flexibility and making it well-suited for research and experimentation.

The integration of PyTorch with Python is one of its most appealing aspects. This Pythonic nature makes PyTorch intuitive for those familiar with Python, easing the coding process and facilitating debugging. Moreover, PyTorch is equipped with robust support for CUDA, enabling efficient GPU acceleration for training complex models and processing large datasets." (Chat GPT, 2023h)

4.2.2. Preprocessing Frameworks

4.2.2.1. Pandas

"Pandas, a versatile and widely used open-source Python library, is essential in data science and analytics for its ease of use and powerful data handling capabilities. It's known for efficiently processing structured data with its two primary structures: Series, a one-dimensional array for single data arrays and indices, and DataFrame, a two-dimensional table with variable data types per column. Pandas excels in data manipulation, offering comprehensive features for filtering, grouping, aggregating, and cleaning data, as well as merging, joining, and concatenating datasets from various sources." (Chat GPT, 2023i)

4.2.2.2. Scikit learn

"Scikit-learn, a leading Python library in machine learning, is highly regarded for its comprehensive tools, especially in data preparation and manipulation. It offers an efficient method for splitting datasets into training and testing sets, a critical step in building and evaluating machine learning models. This process helps prevent overfitting and assesses a model's generalization capabilities. Scikit-learn ensures a random yet reproducible division of data, maintaining representativeness and avoiding biases in the training and testing sets." (Chat GPT, 2023j)

4.2.2.3. Forgi

"Forgi is a specialized Python library for analyzing and manipulating RNA secondary and tertiary structures, treating them as graph-like entities. It is adept at reading, storing, and outputting RNA structures, categorizing them into elements like stems, loops, and multiloop segments. This classification allows for detailed examination and manipulation of RNA structures, including interpreting dot-bracket notation used in secondary structure prediction.

The library excels in analyzing base pairing, identifying pairing patterns, and determining the longest stem in a structure, crucial for understanding RNA's structural aspects. It also offers sequence analysis capabilities, like finding sequences of RNA elements and their neighbors, and determining loop dimensions. Forgi includes iterators for various elements and features for multiloop analysis, facilitating iterative operations and providing insights into complex RNA junctions and spatial relationships." (Chat GPT, 2023q)

4.2.2.4. Arnie

"Arnie is a Python utility library designed for estimating, comparing and reweighting RNA energetics using various secondary structure algorithms. It stands out for its ability to compute RNA energetics and perform structure prediction across a wide range of secondary structure packages. This library supports several renowned packages including Vienna, NUPACK, RNAstructure, RNAsoft, CONTRAfold, and EternaFold, making it a versatile tool in the field of RNA research." (Chat GPT, 2023r) "Additionally, Arnie offers code for computing Maximum Expected Accuracy structures." (Chat GPT, 2023r)

4.3. Machine Learning Models

"Machine learning models are the cornerstone of artificial intelligence, providing the means for computers to learn from and make decisions based on data. These models, ranging from simple linear regression to complex deep neural networks, are built to identify patterns, make predictions, or generate insights from diverse datasets. They are trained using various algorithms on datasets, where they learn to make predictions or classify data based on input features. The effectiveness of these models depends on the quality of data, the appropriateness of the chosen algorithm and the tuning of model parameters." (Chat GPT, 2023b)

"In the context of sequential machine learning problems, models play a pivotal role in deciphering patterns, trends, and relationships within sequential data. These models are specifically designed to handle data where the sequence or order of elements is crucial.

Sequential machine learning models, unlike traditional models, take into account the temporal aspect of data. This consideration is key to accurately capturing the dynamics and dependencies that occur over sequences." (Chat GPT, 2023n)

4.3.1. Recurrent Neural Network (RNN)

Figure 4.1 introduces the RNN architecture where each vertical rectangular box is a hidden layer at a time-step, t. Each such layer holds a number of neurons, each of which performs a linear matrix operation on its inputs followed by a non-linear operation (e.g. tanh()). At each time-step, there are two inputs to the hidden layer: the output of the previous layer h_{t-1}, and the input at that timestep x_t. The former input is multiplied by a weight matrix $W^{(hh)}$ and the latter by a weight matrix $W^{(hx)}$ to produce output features h_t, which are multiplied with a weight matrix $W^{(S)}$ and run through a softmax over the vocabulary to obtain a prediction output $\hat{y}$ of the next word (Equations 4.1 and 4.2). The inputs and outputs of each single neuron are illustrated in Figure 4.2.

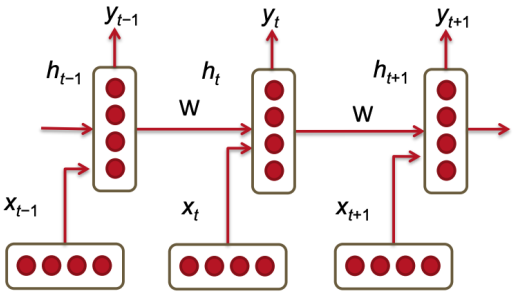


Figure 4.1.: A Recurrent Neural Network (RNN). Three time-steps are shown. Source: (Mohammadi et al., 2019a)

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}) \quad (4.1)$$

$$\hat{y}_t = softmax(W^{(S)}h_t) \quad (4.2)$$

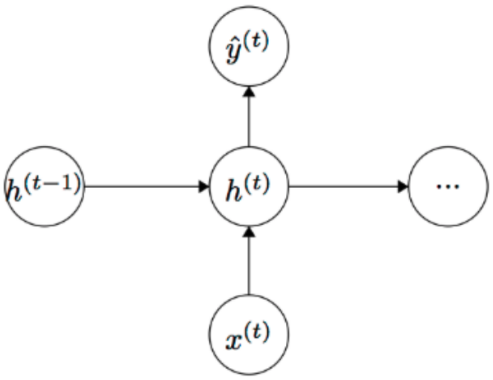


Figure 4.2.: The inputs and outputs to a neuron of a RNN. Source: (Mohammadi et al., 2019a)

What is interesting here is that the same weights $W^{(hh)}$ and $W^{(hx)}$ are applied repeatedly at each timestep. Thus, the number of parameters the model has to learn is less, and most importantly, is independent of the length of the input sequence - thus defeating the curse of dimensionality!
Below are the details associated with each parameter in the network:

- x_1, ..., x_{t-1}, x_t , x_{t+1}, ...x_T: the word vectors corresponding to a corpus with T words.
- $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$: the relationship to compute the hidden layer output features at each time-step t.
  - $x_t \in \mathbb{R}^d$: input word vector at time t.
  - $W^{hx} \in \mathbb{R}^{D_h \times d}$: weights matrix used to condition the input word vector, x_t
  - $W^{hh} \in \mathbb{R}^{D_h \times D_h}$: weights matrix used to condition the output of the previous time-step, h_{t-1}
  - $h_{t-1} \in \mathbb{R}^{D_h}$: output of the non-linear function at the previous time-step, $t-1$. $h_0 \in \mathbb{R}^{D_h}$ is an initialization vector for the hidden layer at time-step $t = 0$.
  - $\sigma()$: the non-linearity function (sigmoid here)
- $\hat{y}_t = softmax(W^{(S)}h_t)$: the output probability distribution over the vocabulary at each time-step t. Essentially, $\hat{y}_t$ is the next predicted word given the document context score so far (i.e. h_{t-1}) and the last observed word vector $x^{(t)}$. Here, $W^{(S)} \in \mathbb{R}^{|V| \times D_h}$ and $\hat{y} \in \mathbb{R}^{|V|}$ where $|V|$ is the vocabulary.

(Mohammadi et al., 2019b)

4.3.2. Long Short-Term Memory (LSTM)

"Long Short-Term Memory (LSTM) is a recurrent neural network architecture." (Calzone, 2019a) "The LSTM architecture consists of one unit, the memory unit (also known as LSTM unit). The LSTM unit is made up of four feedforward neural networks. Each of these neural networks consists of an input layer and an output layer. In each of these neural networks, input neurons are connected to all output neurons. As a result, the LSTM unit has four fully connected layers.

Three of the four feedforward neural networks are responsible for selecting information. They are the forget gate, the input gate, and the output gate. These three gates are used to perform the three typical memory management operations: the deletion of information from memory (the forget gate), the insertion of new information in memory (the input gate), and the use of information present in memory (the output gate). The fourth neural network, the candidate memory, is used to create new candidate information to be inserted into the memory." (Calzone, 2019a)

"An LSTM unit receives three vectors (three lists of numbers) as input. Two vectors come from the LSTM itself and were generated by the LSTM at the previous instant (instant $t-1$). These are the cell state (C) and the hidden state (H). The third vector comes from outside. This is the vector X (called input vector) submitted to the LSTM at instant t. Given the three input vectors (C, H, X), the LSTM regulates, through the gates, the internal flow of information and transforms the values of the cell state and hidden state vectors. Vectors that will be part of the LSTM input set in the next instant (instant $t+1$). Information flow control is done so that the cell state acts as a long-term memory, while the hidden state acts as a short-term memory. In practice, the LSTM unit uses recent past information (the short-term memory, H) and new information coming from the outside (the input vector, X) to update the long-term memory (cell state, C). Finally, it uses the long-term memory (the cell state, C) to update the short-term memory (the hidden state, H). The hidden state determined in instant t is also the output of the LSTM unit in instant t. It is what the LSTM provides to the outside for the performance of a specific task. In other words, it is the behavior on which the performance of the LSTM is assessed." (Calzone, 2019a)

"All three gates are neural networks that use the sigmoid function as the activation function in the output layer. The sigmoid function is used to have, as output, a vector composed of values between zero and one and near these two extremes. All three gates use the input vector (X) and the hidden state vector coming from the previous instant (H_{t-1}) concatenated together in a single vector. This vector is the input of all three gates." (Calzone, 2019a)

$F_t = \sigma(W_f * [H_{t-1}, X_t] + B_f)$
$i_t = \sigma(W_i * [H_{t-1}, X_t] + B_i)$
$\tilde{C}_t = tanh(W_C * [H_{t-1}, X_t] + B_C)$
$C_t = F_t * C_{t-1} + I_t * \tilde{C}_t$
$o_t = \sigma(W_o[H_{t-1}, X_t] + B_o)$
$H_t = O_t * tanh(C_t)$
W: Weight
B: Bias
H_t: Hidden state at timestep t
C_t: Cell state at timestep t
X_t: Input at timestep t
F_t: Forget Gate at timestep t
I_t: Input Gate at timestep t
O_t: Output Gate at timestep t
$\tilde{C}_t$: Candidate Cell state at timestep t

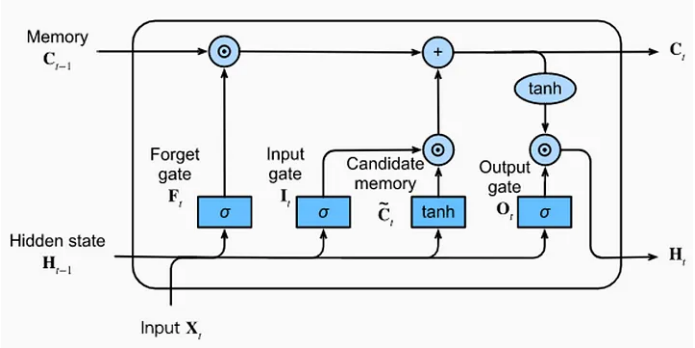


Figure 4.3.: Architecture of a LSTM Unit. Source: (Calzone, 2019b)

4.3.3. Transformer

"The Transformer architecture represents a notable development in machine learning for processing sequential data. Its key innovation is parallel processing, which enables it to handle entire data sequences simultaneously." (Chat GPT, 2023e)

"Central to the Transformer's effectiveness is its self-attention mechanism. This feature enables the model to dynamically focus on different parts of the input data, assigning varying levels of importance to each part. This ability to weigh the significance of different components in a sequence allows the Transformer to capture intricate relationships within the data, which is particularly beneficial for complex tasks in natural language processing." (Chat GPT, 2023c)

"The Transformer's ability to process all parts of the input concurrently allows it to effectively manage long-range dependencies in the data. This is crucial for accurately understanding sequential data.

Scalability and efficiency in training are significant advantages of the Transformer architecture. It's designed for handling large datasets and long sequences, speeding up training while maintaining high accuracy. The architecture's parallel processing enhances its learning capabilities from extensive data." (Chat GPT, 2023e)

"The Transformer architecture utilizes embeddings as a crucial initial step to convert input data, especially textual data, into a format that can be processed by the model. Embeddings are essentially numerical representations of data, where each word or token in the text is mapped to a vector in a high-dimensional space. This mapping allows the Transformer to capture the semantic and syntactic nuances of each word or token, facilitating a deeper understanding of the input data. Unlike simpler representations that might treat tokens as isolated entities, embeddings encapsulate relationships between tokens." (Chat GPT, 2023d)

"Positional encoding in the Transformer adds information to each element's embedding, indicating its position in the sequence. This is crucial for the model to understand sequential context and relationships, as it lacks inherent sequential processing like RNNs or LSTMs.

Multi-head attention is a key feature of the Transformer, dividing the model's attention into several 'heads' to focus on different parts of the input simultaneously. Each head captures unique aspects of the data, and their outputs are combined for a more comprehensive understanding. This allows the Transformer to process complex tasks requiring a contextual understanding of the input data." (Chat GPT, 2023e)

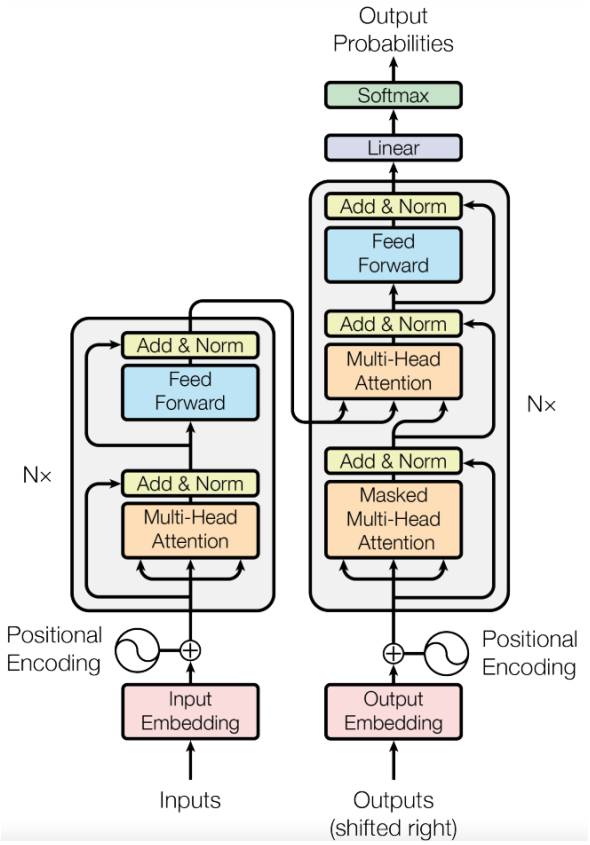


Figure 4.4.: Transformer architecture Source: (Vaswani et al., 2017)

4.3.4. Pretrained Model

"Pretraining in machine learning, particularly for deep neural networks, is a process where a model is first trained on a large, general dataset before fine-tuning for a specific task. This approach leverages the idea that certain features are transferable across similar tasks, allowing the model to apply learned patterns to specific applications. During pretraining, models are exposed to diverse datasets, ranging from text corpora to varied images, to learn broad features like word associations or visual elements. This step significantly enhances performance, especially when task-specific data is limited, and improves efficiency by reducing training time and computational resources for the specific task, as the model has already learned basic patterns.

However, pretraining is not without challenges. The relevance of the pretraining dataset to the target task is crucial; the more aligned the pretraining data is with the task, the more effective the transfer learning process. Additionally, there's a risk of overfitting the model to the pretraining data, potentially hindering its adaptability to the specific task. Pretraining is a fundamental aspect of transfer learning, often used in deep learning, especially for large neural networks, enabling models to start from an advanced point for a new task." (Chat GPT, 2023k)

4.4. Hyperparameter Tuning

"Hyperparameter tuning is a crucial step in machine learning, focusing on optimizing the parameters that control the model's training process. These hyperparameters, set before training, include elements like learning rate, neural network layers, and regularization parameters, and significantly influence the model's learning speed, quality, and ability to generalize from training to unseen data. The tuning process involves searching a predefined space of hyperparameter values using methods like grid search, random search, or Bayesian optimization. This search aims to find the optimal combination that yields the best model performance.

The challenge in hyperparameter tuning lies in the trade-off between exploration of a wide range of values and exploitation of the most promising ones, balancing the need to discover optimal regions while not over-focusing on less promising areas. Additionally, hyperparameter tuning can be computationally intensive, especially with large models and datasets. Techniques such as early stopping can help manage this by halting the training process when performance ceases to improve, thereby conserving computational resources." (Chat GPT, 2023l)

5. Data Curation

This chapter dives into the exploration and evaluation of the provided datasets in the competition, focusing on critical attributes and techniques applied to ensure data reliability and model accuracy. It covers a detailed analysis of data sets, data quality assessment, data splitting strategies, feature engineering insights, tokenization approaches, and error handling techniques, showing essential steps in processing and understanding RNA sequence data.

5.1. Data Sets

In the competition, all teams were provided with three main datasets, namely train, submission and test. The data sets are described below and their respective information has been taken from the dedicated Kaggle competition page ("Kaggle Competition information", n.d.).

5.1.1. train_data.csv

This training file contains the following features:

- `sequence_id`: Identifier for each RNA sequence.
- `sequence`: String of Adenine (A), Uracil (U), Cytosine (C), and Guanine (G) representing RNA sequence.
- `experiment_type`: DMS or 2A3.
- `dataset_name`: Name of high throughput sequencing dataset.
- `reads`: Number of reads in the sequencing experiment.
- `signal_to_noise`: Signal/noise value for the profile.
- `SN_filter`: Boolean indicating profile's `signal_to_noise` and `reads`.
- `reactivity_0001,...,reactivity_0206`: Reactivity profile for RNA sequence.
- `reactivity_error_0001,...,reactivity_error_0206`: Errors in experimental values.

5.1.2. sample_submission.csv

The competition submission format requires entries to follow this structure:

- `id`: Integer identifying each sequence position.
- `reactivity_DMS_MaP`: Sample submission values.

5.1.3. test_sequences.csv

This file is dedicated to testing purposes, where predictions need to be made for submission.

- `id_min, id_max`: Minimum and maximum id values.
- `sequence_id`: Identifier for each test sequence.
- `sequence`: RNA sequence for prediction.
- `future`: Flag for sequences to be collected after the competition start.

Additionally, supplementary folders were provided containing extra data, such as RNA sequences in FASTA format and predicted RNA secondary structures. However, we did not use this information in our analysis as it did not play a role in shaping our model or predictions.

5.2. Data Quality Assessment

As we explore the dataset, we have discovered several observations, highlighting important patterns and inconsistencies. These include missing values, signal-to-noise filtering, sequence repetition, and varying sequence lengths. Understanding these patterns is important, as they significantly influence the trustworthiness and interpretation of our predictive models.

5.2.1. Identification of Missing Values

In the training dataset, there is a noticeable pattern showing that certain columns consist of NaN values, particularly in the `reactivity` and `reactivity_error` columns. These NaN values appear at the beginning and end positions of the sequences. However, other essential columns such as `sequence`, `experiment_type`, `dataset_name`, `reads`, `signal_to_noise`, and `SN_filter` show no missing data.

5.2.2. Signal-to-Noise Filter Analysis

The `SN_filter` column indicates whether the sequence meets specific criteria based on signal-to-noise and read counts. Approximately 26.6% of sequences pass this filter, implying higher reliability in these measurements. For the evaluation process, only sequences with `SN_filter` = 1 are utilized.

5.2.3. Sequence Repetition and Length Variation

Surprisingly, not all RNA sequences in the dataset are unique, as over 20'000 repetitions are observed across both testing types. However, this repetition of sequences does not imply identical reactivity values. So, there is no duplication across the entire dataset. Sequence lengths display limited variation, with the majority (hovering) around 177 bases, constituting over 95% of the dataset. Only a few distinct lengths, such as 170, 115, 155, and 206 bases, are observed in a smaller proportion.

5.3. Data Splitting

We split our data into an 80/20 ratio for training and validation purposes, respectively. As only 26.6% of the data remained available after applying the `SN_filter`, we chose not to create an independent test set. Instead, we relied on the Kaggle evaluation as our test benchmark. However, to ensure robust model assessment and prevent data leakage between training and validation sets, we implemented k-fold cross-validation. This technique systematically rotated subsets of the data as validation sets while using the remainder for training, enhancing the reliability of our model evaluation.

5.4. Feature Engineering

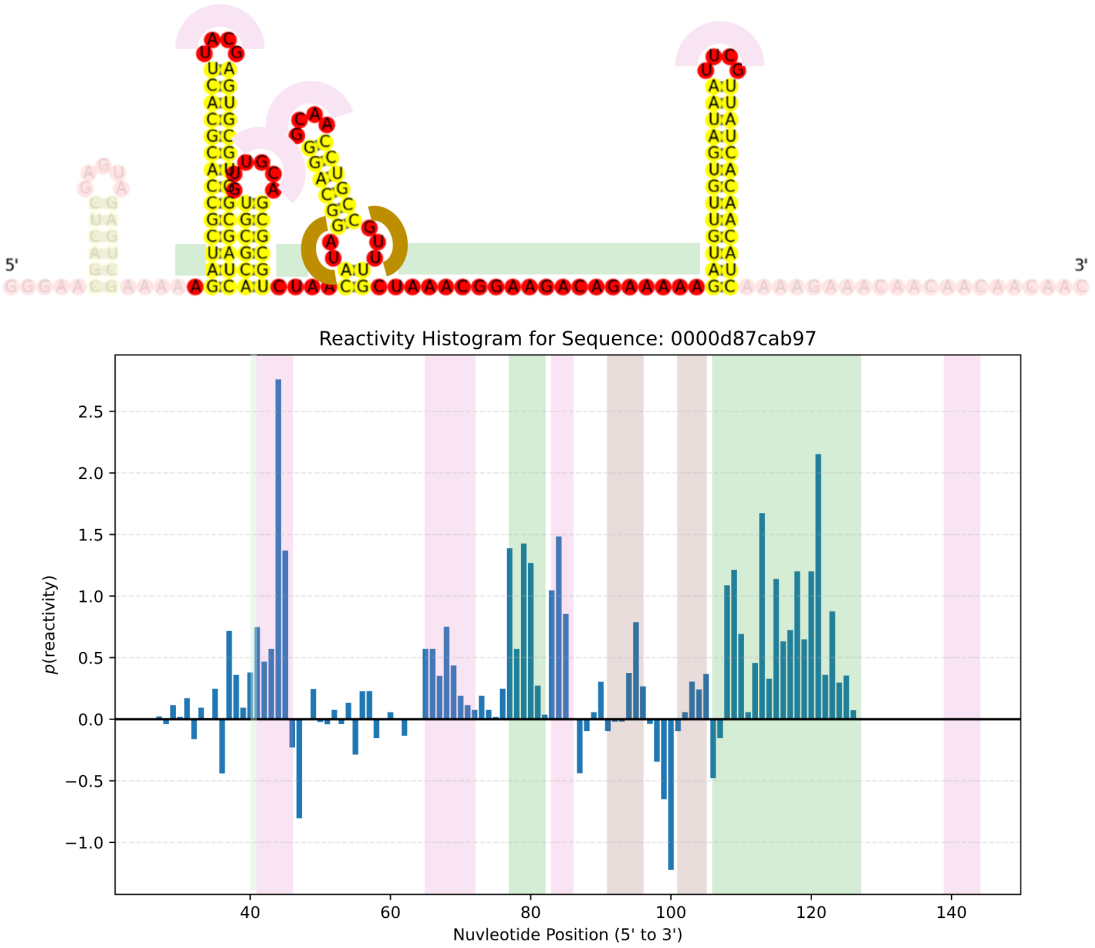


Figure 5.1.: Visualization of RNA Sequence Reactivity: Combining Secondary Structure and Histogram Plot

Figure 5.1 shows a secondary structure and a histogram plot of a randomly selected sequence from the 2A3 experiment type. The grayed-out parts at the beginning and end of the secondary structure indicate regions with NaN values, signaling no recorded reactivity. The red sections show different loop types, while the yellow parts indicate the sturdy stems within the structure.

Our approach to feature engineering came from exploring code on the Kaggle Forum, where we found Arnie and Forgi libraries providing insights into RNA sequence visualization. When visualizing RNA sequences with overlapped reactivity data, we noticed an interesting pattern: loop types showed spikes in reactivity, whereas strong base stems showed minimal or no reactivity. This emphasized the importance of loop types in prediction. Using tools such as Arnie and EternaFold, we calculated sequence structures for individual bases, represented as dot and round parentheses. Using Forgi, we predicted loop types for individual bases, categorizing them as S for stem, I for internal loop, M for multi-loop, H for hairpin loop, and E for unpaired nucleotides. With that, we introduced two new features to our dataset. These categorizations were taken from the Forgi documentation ("RNA Secondary Structure as a Graph Using the forgi Library — forgi 2.0.0 documentation", n.d.).

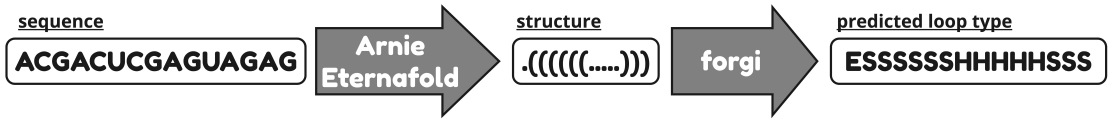


Figure 5.2.: From Raw String to Predicted Structure and Loop Types

Figure 5.2 shows the process of analyzing an arbitrary RNA sequence. At first, the sequence itself is displayed, providing a visual representation of the RNA string. Then, the structure of this sequence is computed using Arnie and EternaFold. Finally, Forgi is used to predict the loop types for individual bases within the RNA sequence.

5.5. Tokenization

The tokenization process involves encoding categorical features into numerical values. For instance, considering three features: 'sequence,' 'structure,' and 'predicted loop type.' The 'sequence' consists of letters like A, C, G, and U. The 'structure' uses symbols like dots and parentheses. Lastly, the 'predicted loop type' uses letters such as S, I, M, H, and E. To convert these categories into numbers, each unique element gets assigned a specific value. For instance, if the first base of the sequence is 'A', the structure is '('and the predicted loop type is 'E,' the tokenization could represent this combination with a numerical code like '125.'

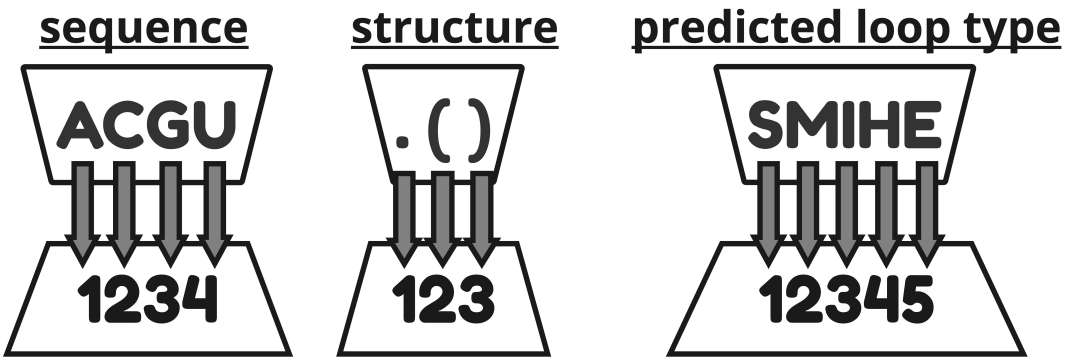


Figure 5.3.: Categorical Features and their Numerical Representations

Figure 5.3 shows the three features at the top: 'sequence,' 'structure,' and 'predicted loop type.'. Each of these features corresponds to specific values represented at the bottom of the figure.

This visualization shows the relationship between the categorical features and their respective numerical representations.

5.6. Padding

In our data preparation process, we use padding to standardize the length of our tokenized features. Since the maximum sequence length possible is 457, we extended sequences shorter than this length by adding zeros to reach the maximum length.

5.7. Reactivity Error Handling

The reactivity error represents the calculated uncertainties in the experimental reactivity values. While these error values are not part of the predictions in this competition, they play a role in evaluating our models. After applying the signal-to-noise filter, most significant errors were filtered out. However, some bases still have high reactivity error. We considered these errors in our evaluation metric, Mean Absolute Error (MAE) and created a new metric: Reactivity Weighted Mean Absolute Error (rewMAE).

The rewMAE uses a clipping function that limits a value between a lower bound min and an upper bound of 1.

$$clip(x) = \begin{cases} min, & \text{if } x < min \\ x, & \text{if } min \leq x \leq 1 \\ 1, & \text{if } x > 1 \end{cases} \tag{5.1}$$

min : Floating point variable $[0.1, 0.3, 0.5, 0.7, 0.9]$

Calculating the rewMAE multiplies each absolute difference with a weight. This weight ranges from a min value to 1 if a is 0 or from $min + 1$ to 2. The to be clipped value is the reactivity accuracy or the inverse of the reactivity error. Additive value a is a boolean where 0 means we want to decrease the impact of noisy samples and 1 means we want to increase the impact of high-quality samples.

$$rewMAE = \frac{1}{N} \sum_{i=1}^{N} \mid y_i - \hat{y}_i \mid * [clip(1 - e_i) + a] \tag{5.2}$$

N : Number of scored ground truth values
y_i : Actual value
$\hat{y}_i$: Predicted value
e_i : Reactivity error
a : Additive weight $[0, 1]$

6. Modeling

6.1. Model Modality

Throughout this project, two model modalities have been used. The modality is a general design feature of our prediction mechanism and is independent of underlying model technologies. Modularity and loose coupling were a core consideration in our codebase. Modality can be changed at any time at a higher level without significant changes in the model definition.

For each feature X_n (e.g. a base character in an RNA sequence) two target values $[y_{n1}, y_{n2}]$ (e.g. the corresponding DMS and 2A3 reactivities) are predicted. The modality describes, whether two separate models are employed for individual prediction of DMS and 2A3 or one single model is used to predict both.

From a computational perspective, a single-model solution seems more reasonable since training one instead of two models requires less time and resources. For use in a production environment, it is also advantageous to be able to perform inference with one single model. Because of the competition characteristic, we want to identify the best modality for our models with the goal of reaching the maximum score.

6.1.1. Dual Model

With Dual Model, two separate models are trained to predict DMS and 2A3 reactivities. In our workflow, both models are trained with the same hyperparameters and architecture sequentially.
$m_1 \colon X_1 \longmapsto y_1$
$m_2 \colon X_2 \longmapsto y_2$
Assuming that 1 *and* 2 *denote the reactivity reagents, DMS and 2A3 respectively.* m *: model.*

6.1.2. Single Model

With Single Model, one model is trained to predict both DMS and 2A3 reactivities.
$m \colon X \longmapsto [y_1, y_2]$
Assuming that 1 *and* 2 *denote the reactivity reagents, DMS and 2A3 respectively.* m *: model.*

6.1.3. Evaluation

Dual Model was utilized in the project's initial stage. However, this came with the inherent disadvantage of having to train two separate models every time. It was decided to implement a Single Model mode for training and inference, which would halve training times. However, it was unclear if the switch to Single Model would decrease model performance.

To determine whether Single Model can achieve similar performance as Dual Model, a benchmark run was conducted with identical hyperparameters. The results are shown in the following table. 6.1

RunID(s)	Modality	Training Time	Private Score	Public Score
astral-field-158;logical-lake-154	Dual Model	1'561 min	0.217	0.165
dry-glitter-200	Single Model	597 min	0.203	0.170

Table 6.1.: Comparison of Kaggle Scores for Dual Model and Single Model

The experiment confirmed, that Single Model can achieve similar performance as Dual Model. The required computation time was lower than expected. Single model finished in only 38% instead of the expected 50% of time. The experiment couldn't determine a clear winner in terms of Public/Private Score. However, this doesn't matter since small fluctuations in scores are expected.

Based on these findings, it was decided to employ Single Model for all further experiments.

6.2. Preprocessing

To potentially increase performance in the models, some additional preprocessing steps were taken. To evaluate the performance impact of the different configuration settings, three different models were used.

Model name	Epochs	Configuration
RNN	20	3 Bidirectional RNN layers with a hidden_size of 64
LSTM	20	3 Bidirectional LSTM layers with a hidden_size of 64
Transformer	5	3 Transformer encoder only layers with dim_feedforward 800, nhead 5, ntokens 500, d_model 200, dropout 0.2

Table 6.2.: Used models for preprocessing

The RNN was preliminarily used for training speed, to get as fast as possible results and was replaced by the LSTM later on. All tests were run on our final choice of model type, the transformer model. As training even a small transformer model can take quite a while, we decided to go with 5 epochs per run for each transformer model rather than 20, which were previously used by the RNN and LSTM.

Note that after each test, the best value resulting from the tested setting is used for all subsequent tests.

6.2.1. Noise Filtration

Some models train better on larger quantities rather than on lower quantities with higher-quality data. The SN_filter column provided us with a way to remove noisy samples with a simple flag. The following performance in 6.3 shows the impact of the flag set to $True$ meaning the noisy samples got removed or $False$ where no samples were removed.

RunIDs (RNN; LSTM; Transformer)	filter_noise	Validation Loss
laced-puddle-205; graceful-pond-320; easy-night-330	False	0.444; 0.391; 0.414
dulcet-totem-204; flowing-spaceship-319; deep-glitter-329	True	0.281; 0.269; 0.306

Table 6.3.: Validation Loss comparison for filter_noise

The validation metric clearly shows that an approach with the $filter\_noise$ enabled increases performance in the models. This leaves about a quarter of the original training data which massively increases training speed.

6.2.2. Data Split and Cross-Validation

There were two different possible approaches to train the models. A classical training and validation split or a cross-validation approach. While we initially used the classical approach we quickly removed it from our consideration, as this removes a large chunk of data our model could be used to better generalize and help us increase performance in the Kaggle challenge.

RunIDs (RNN; Transformer)	k_fold	validation_split	Validation Loss
hopeful-voice-203; balmy-frost-328	5	-	0.281; 0.309
visionary-puddle-202; astral-microwave-327	10	-	0.314; 0.311
stellar-universe-201; vivid-waterfall-326	-	0.2	0.276; 0.291
super-dust-199; zesty-waterfall-325	-	0.1	0.274; 0.289

Table 6.4.: Validation Loss comparison for k_folds and validation_split

Both approaches are displayed above in 6.4. Where rows with k_fold set to any number means the cross-validation approach was used and $validation\_split$ otherwise. Cross-validation and the validation split approach cannot be directly compared. Cross-validation validates on each fold at least once and after 5 epochs times $n - folds$ has seen every sample multiple times. As for our specific use case where another test set is available on Kaggle we decided for the cross-validation approach and chose $k\_fold = 5$, which means each fold is 20% of all available training data.

6.2.3. Clipping

The competition metric only considers reactivity values in the range [0,1], values outside this range will be strictly penalized. We considered the possibility that the performance increases when clipping the training data before the model is trained. The idea is that the model learns to predict more reasonable ranges that don't differ between negative infinity and positive infinity but rather closer to 0 and 1. Instead of our previously used evaluation metric where we just calculated the absolute difference between the prediction and true value, we chose another metric. This metric clips all predictions and true between the range of 0 and 1. This removes the possibility that the validation loss is only smaller with the clipped values as outliers aren't predicted properly.

RunIDs (RNN; LSTM; Transformer)	clip	Clipped Validation Loss
super-lion-334; dazzling-eon-318; likely-galaxy-324	False	0.183; 0.161; 0.220
misunderstood-waterfall-333; lyric-thunder-317; stilted-bird-323	True	0.182; 0.161; 0.221

Table 6.5.: Clipped Validation Loss comparison for clipping

After evaluating the three different model types there is no clear winner. The impact of clipping the values before training is negligible.

RunIDs (LSTM; Transformer)	pred_min	pred_max	difference
dazzling-eon-318; likely-galaxy-324	-0.225; -0.150	3.150; 2.086	3.375; 2.236
lyric-thunder-317; stilted-bird-323	-0.511; -0.062	1.704; 1.168	2.215; 1.230

Table 6.6.: Prediction minima and maxima comparison for clipping

As expected the range of the predicted minima to maxima is reduced by introducing clipping. As the challenge doesn't require us to predict any reactivity above 1 and below 0 the decision was taken to set $clip$ to $True$.

6.2.4. Structural Encoding

The general approach until now was to encode each base as a single token. We can tokenize the sequence where each base is a single integer, this would be $structure = False$. Another approach is to use additional structural information as defined in the chapter 5.5, this would set the flag $structure = True$.

RunIDs (LSTM; Transformer)	structure	Validation Loss
lyric-thunder-317; stilted-bird-323	False	0.162; 0.222
noble-durian-315; sunny-sound-321	True	0.146; 0.172

Table 6.7.: Validation Loss comparison for structure

After adding the structural encoding, we can see a significant increase of nearly 0.05 in the transformer and a less but still significant change of 0.016 in the LSTM. We therefore decided to set the $structure$ flag to $True$.

6.2.5. Weighted Loss

Until now the error of each reactivity was not factored in. Using a custom weighted loss function the idea was to increase the model performance by rating noisy samples lower when using the weighted $additive$ loss set to $False$ or increasing the weight for good quality samples with the weighted $additive$ loss set to $True$. If both $weight$ and $additive$ are set to False no weighted loss was used.

RunIDs (LSTM; Transformer)	weight	additive	Validation Loss
rose-pine-246; fanciful-firefly-258	False	False	0.145; 0.167
warm-water-251; wise-spaceship-263	0.9	False	0.144; 0.166
lilac-oath-250; driven-silence-262	0.7	False	0.144; 0.167
fresh-eon-249; pretty-snow-261	0.5	False	0.144; 0.168
upbeat-resonance-248; lemon-music-260	0.3	False	0.146; 0.169
comfy-plasma-247; winter-water-259	0.1	False	0.150; 0.172
flowing-hill-257; fallen-river-269	0.9	True	0.144; 0.165
honest-silence-256; silvery-salad-268	0.7	True	0.143; 0.164
lemon-sea-255; glorious-haze-267	0.5	True	0.144; 0.165
logical-plant-254; feasible-capybara-266	0.3	True	0.144; 0.167
brisk-aardvark-253; smooth-sound-265	0.1	True	0.144; 0.165

Table 6.8.: Validation Loss comparison for LSTM weighted loss

Even though the difference is only marginal we can see that in both the LSTM and Transformer the $weighted\ loss$ and $additive$ set to $True$ slightly increase the performance of the model. We decided to continue with the parameters $weight = 0.7$ and $additive = True$ because they yielded the best result.

6.2.6. Final Configuration

As in the previous subsections, the following final configuration for the preprocessing was chosen.

Setting	Value
filter_noise	True
validation_split	-
k_fold	5
clip	True
structure	True
weighted_loss	0.7
additive_weight	True

Table 6.9.: Final preprocessing configuration

These settings yielded the best overall performance.

6.3. Models

This section provides an overview of the models used in this competition.

6.3.1. Bidirectional RNN

6.3.1.1. Rationale

Recurrent Neural Networks (RNNs) work by using hidden states and feedback loops to process a sequence of data. This type of neural network is often used for language models or time-series data analysis. (Das et al., 2023)

Because of the simplicity and efficiency aspects of an RNN, it was decided to start with an Elman RNN (also called Simple Recurrent Network or SRN) (“7 The Simple Recurrent Network: A Simple Model that Captures the Structure in Sequences”, n.d.).

6.3.1.2. Architecture

It was decided to use a bidirectional RNN since it is expected to perform better than a unidirectional RNN since information from both directions of a sequence is used, which provides richer information for the prediction. It is proven, that the Memory Capacity of a bidirectional RNN is strictly twice as large as that of a unidirectional RNN. (Hassan, 2021)

The PyTorch implementation of an RNN has two options for the activation function, that is used to calculate the hidden state h_t: tanh and ReLU. The tanh function is the default setting. (“RNN — PyTorch 2.1 documentation”, n.d.)

The purpose of this activation is to learn non-linear dependencies. It was decided to use ReLU instead of tanh since it is computationally less expensive. (Brownlee, 2019) The use of ReLU in Recurrent Neural Networks poses a challenge since outputs have no upper bound and might explode over timesteps. Exploding gradients would pose an additional challenge in addition to vanishing gradients, which is a general problem with RNNs. However, there has been research into RNNs with ReLU activation that suggests, that these problems can be solved with careful initialization of the weights. (Le et al., 2015)

In our experiments, problems with exploding gradients did only pose a challenge for larger RNN networks. This was solved by using gradient clipping.

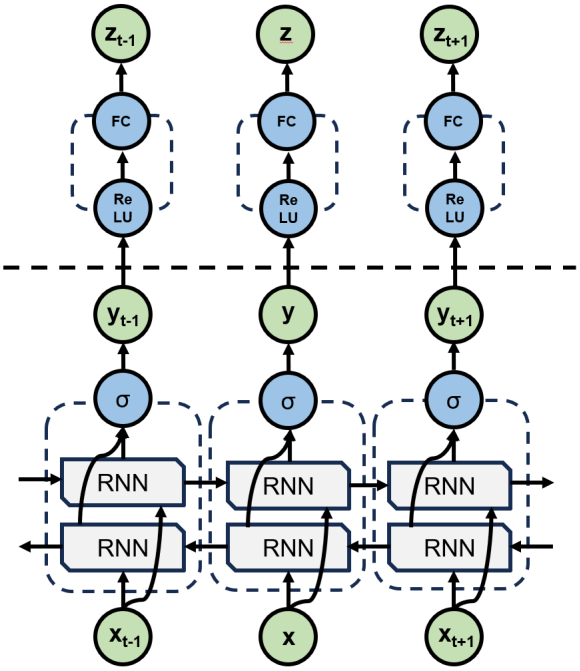


Figure 6.1.: Architecture of Bidirectional RNN over multiple timesteps

6.3.2. Bidirectional LSTM

6.3.2.1. Rationale

LSTM (Long short-term memory) is a type of neural network that works with recurrence and processes data in timesteps. In addition to its predecessor, the classic RNN, it can deal with the vanishing gradient problem. The decision to set up an LSTM model for this challenge was evident since they have demonstrated remarkable performance in sequential tasks like language translation, financial modeling or speech recognition. Given the sequential structure of RNA sequences, it is reasonable to anticipate good performance with neural networks that incorporate recurrence.

6.3.2.2. Architecture

Developing the first LSTM architecture was an incremental process since there are many parameters. It was decided to start with the simplest possible network, evaluate and add more complexity iteratively.

Since the BiRNN showed good performance, a Bidirectional LSTM was used as a starting point. During experimentation, mainly *hidden_size* and *num_layers* were changed in order to find a suitable architecture.
The final LSTM architecture consisted of a 3-layer Bidirectional LSTM with a *hidden_size* of 64 (per cell). This increase in layers came with the cost of more computational expense, which slowed down training.

The ReLU function is applied on the output of the LSTM, a tensor of hidden state h_t for each time step t. This only affects negative values. A linear layer projects this tensor to the desired output size of 1 for Dual Model respectively 2 for Single Model. The activation functions for the LSTM Cell gates have not been changed.

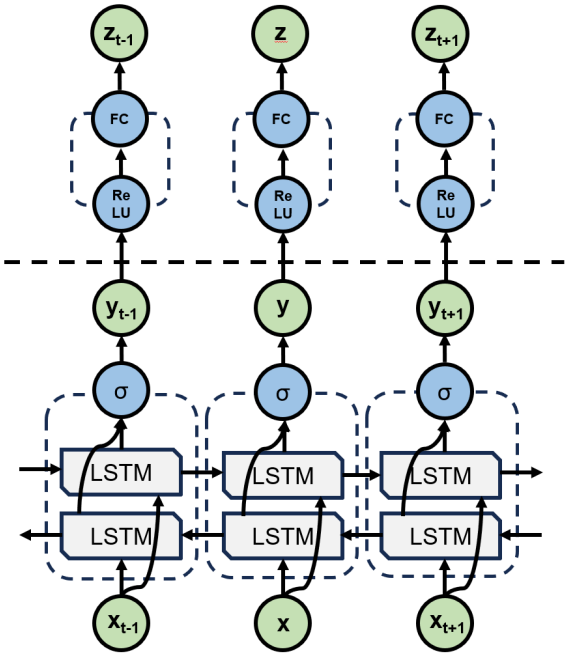


Figure 6.2.: Architecture of Bidirectional LSTM over multiple timesteps

6.3.3. Transformer

6.3.3.1. Rationale

There are mainly two reasons to continue with transformer-based models. First, this is by far the most popular choice in public Kaggle notebooks for this challenge. This provides confidence, that Transformer is a reasonable choice for this task. The second reason is the versatility and success of transformer-based networks. The Transformer architecture has shown superior performance to RNNs and Convolutional Neural Networks (CNNs) in a wide range of areas including text translation-/processing and image classification. (Katrompas et al., 2022)

While the transformer architecture brings many new concepts, our main focus is on the attention mechanism. While it is possible to implement the attention mechanism into RNNs, we have not done that for our recurrent models.

6.3.3.2. Architecture

The Transformer architecture introduced in "Attention is all you need" (Vaswani et al., 2023) consists of two components: The Encoder and the Decoder. They have different roles. The Encoder has the goal of capturing the essence of each token and understanding its interrelationships. Meanwhile, the Decoder is dedicated to the task of generating new sequences rather than understanding context. (Abaskohi, 2023)

Since our task is to build an understanding of sequence reactivities and predict them for each base in a sequence, we only need the Encoder part. There is no generative task involved.

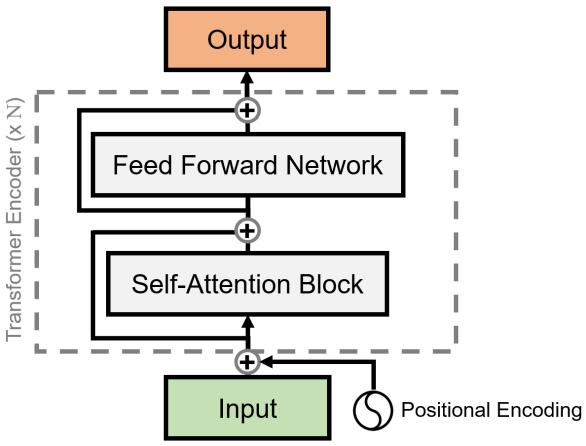


Figure 6.3.: Architecture of Transformer Encoder with N layers

Figure 6.3 shows the architecture of our TransformerEncoder, which is similar to the original variant. Unlike in the original variant, the normalization layer is not denoted in the diagram. This is because it is considered as an architecture parameter and not part of strict model specification. More about Architecture Parameters and their evaluation can be found in chapter 6.4.1.

The TransformerEncoder consists of N layers, which are stacked on top of each other. To process the inputs, the model will internally create Embeddings for the input tokens. The Embedding is added together with the Positional Encoding, which is implemented as specified in the paper "Attention is all you need". For the Transformer Encoder, the Torch implementations *torch.nn.TransformerEncoder* and *torch.nn.TransformerEncoderLayer* have been used. To project the outputs of the TransformerEncoder to the desired output size (2 for Single Model, 1 for Dual Model), a Linear Layer is used.

The TransformerEncoder Model Definition is attached in A.1 for reference.

6.3.4. Pretrained Model (BERT)

6.3.4.1. Rationale

All models up to this point were trained from scratch. The Transformer Encoder has shown the best performance for this task so far. In NLP, the use of pre-trained models has become a standard practice. Pretrained models are trained on large datasets and can be used as a starting point for transfer learning. (Qiu et al., 2020) In the case of NLP, it makes sense to rely on pre-trained models that understand fundamental features and complex workings of language and fine-tune them to a specific task. In the case of LLMs, pre-trained models save a lot of computational power while making the models widely accessible.

However, the efficacy of pre-trained models for this particular task is unclear. While in NLP, different applications share the same underlying structure of language, this is not necessarily the case for RNA sequences. There is a broad range of RNA / DNA specific tasks and while there are some general pre-trained models for RNA data, it remains to be determined if they are suitable for this particular task. (Wang et al., 2023)

6.4. Model Architecture Optimization

6.4.1. Transformer Architecture Parameters

This section describes the selection of Transformer encoder-specific architecture parameters. It was decided to not use automatic search algorithms (like grid search) to find optimal parameters for mainly two reasons.

First, the use of automatic search would skip a large part of model engineering, thus reducing the understanding of the model and complicating explainability. Second, Transformer experiment runs are considerably more time-consuming than runs with recurrent models (RNNs/LSTMs). Running an excessive amount of experiments in order to "brute force" the optimal architecture is not feasible due to constraints in time and computational resources.

6.4.1.1. d_model

d_model is an integer parameter in the Torch *TransformerEncoderLayer* Class that determines the dimensionality of the input. The paper "Attention is all you need" suggests a value of 512 for d_model. (Vaswani et al., 2023) It was decided to use 200 for d_model to ensure a sufficient number of dimensions even for higher nhead values. However, this value was not extensively tuned since we didn't notice significant changes for different dimensions.

6.4.1.2. dim_feedforward

dim_feedforward is the dimension of the feedforward network model. The paper "Attention is all you need" suggests a value of 2048 for dim_feedforward. (Vaswani et al., 2023) It was decided to use the same ratio between d_model and dim_feedforward as in the paper, which lead to a dim_feedforward value of $4 * d\_model = 4 * 200 = 800$.

6.4.1.3. dropout

Dropout is a powerful and effective regularization technique that can be used in deep neural networks of all types. By switching off a number of outputs in a layer, noise is added to the network, which makes it more robust and less prone to overfitting. (Brownlee, 2018)

While the default is 0.1 in the PyTorch *TransformerEncoderLayer* Implementation, we chose to increase the dropout rate to 0.2. ("TransformerEncoderLayer — PyTorch 2.1 documentation", n.d.) This was done to reduce overfitting and improve generalization capabilities.

6.4.1.4. n_layers

n_layers specifies, how many *TransformerEncoderLayer* units are part of a *TransformerEncoder*. The layers are stacked on top of each other. ("TransformerEncoder — PyTorch 2.1 documentation", n.d.) In the paper "Attention is all you need", $N = 6$ layers are used for the Encoder. (Vaswani et al., 2023)

We experimented in the range of $N = [3, 6, 12]$ layers. This was done to find out if a deviation from $N = 6$ would have advantages in our model. The amount of layers has a significant impact on the runtime of a single epoch. While the runs with $N = 3$ were finished in around half the time, they have shown a higher validation loss. Runs with $N = 12$ had a better validation loss than $N = 6$ but this came at the cost of a doubled runtime. Therefore we decided to stick with $N = 6$.

6.4.1.5. norm_first

norm_first is a boolean parameter in the Torch *TransformerEncoderLayer* Class that determines if the layer normalization is applied before attention and feedforward operations. ("TransformerEncoderLayer — PyTorch 2.1 documentation", n.d.) In default setting, norm_first is set to False.

The paper on Layer Normalization in the Transformer Architecture (Xiong et al., 2020) suggests that Pre-LN (Prev. Layer Normalization) can lead to improved performance by addressing gradient problems that occur in the original Transformer Architecture (Vaswani et al., 2023), which relies on Post-LN. (Raschka, 2023)

Since there is inconclusive empirical evidence for the superiority of Pre-LN, we decided to run experiments with both settings. These 2 runs were conducted with identical settings on a Transformer Encoder with 3 layers for 5 epochs.

RunID	norm_first	Validation Loss
apricot-microwave-226	False (Default)	0.265
colorful-leaf-215	True	0.259

Table 6.10.: Validation Loss comparison for norm_first

The experiment suggests using norm_first, since it leads to a lower validation loss. To rule out the possibility that the difference in validation loss is due to a random effect, the loss curves of both runs were compared. Both curves are almost identical in their form without overlaps.

6.4.1.6. nhead

nhead is an integer parameter in the Torch *TransformerEncoderLayer* Class that determines the number of heads in the multihead attention layer. The idea of Multi-head Attention was originally introduced in the paper "Attention is all you need" and is a core feature of the Transformer Architecture. Instead of using a single attention function with keys, values and queries in size d_{model}, a number of h individual parallel attention functions is used. In the paper, $h = 8$ attention layers were employed in parallel and $d_{model} = 512$ was used. Since d_{model} is divided over heads h, the total computational cost is similar to Single-head attention. (Vaswani et al., 2023)

Employing Multi-head attention gives the model multiple representation subspaces, which allows the model to focus on different positions in parallel. This mechanism has been shown to improve performance in many NLP tasks. Visualizations of attention heads show, that multiple attention heads focus on different structures of the input sequence in NLP tasks. (Alammar, n.d.)

To find an optimal number of heads for this task, an experiment was conducted with identical runs, only varying the number of heads. The experiment was conducted with a Transformer Encoder with 3 layers for 5 epochs.

RunID	nhead	Validation Loss	Runtime
pleasant-sun-308	1	0.1827	60 min
whole-donkey-307	2	0.1801	64 min
rich-pyramid-306	5	0.1768	75 min
autumn-sound-309	8	0.1763	91 min
sandy-gorge-310	20	0.1802	165 min
curious-sponge-314	50	0.1823	360 min

Table 6.11.: Validation Loss comparison among different nhead values

The loss curves of these runs have been consistent and without significant overlaps. All runtimes have been rounded to the minute. The experiment suggests that the optimal number of heads is 8. However, it is notable that there is a significant rise in computing time with higher nhead values. The run with 50 heads took 6 hours to complete whereas the run with 1 head took only 1 hour. A difference of this magnitude was not expected since Single-head attention uses the same number of total dimensions d_{model} as Multi-head attention with any number of heads. Based on the description of Multi-head attention in "Attention is all you need", a similar computational cost was expected. Apart from computing time, utilized memory was also affected. The run with nhead=1 utilized 4.6 GB whereas 14.2 GB was used for nhead=50. Apart from the optimal setting for nhead, this experiment also observed that Multi-Head attention has a major impact on the computational complexity of our model.

It was decided to continue with nhead=5. While nhead=8 showed slightly better performance, the difference was not significant enough to justify the additional computational cost.

6.4.2. Transformer Runtime Optimization

Besides finding optimal parameters, model efficiency is substantial since our computing resources are limited. The objective is to assess if there is room for optimization in our code. An optimized model allows us to perform more experiments.

The focus was the *.forward()* method in our Transformer Model. The *TransformerEncoder* model Definition is attached in A.1 for reference.

6.4.2.1. No datatype conversion (float32 -> int32)

For every call of the forward method, the input tensor from the dataloader is converted from float32 to int32, which is the required input format. The reason to use float32 in the dataloader is to ensure compatibility with our recurrent models.

We ran an experiment to assess if this datatype conversion has any meaningful impact on runtime. The preprocessing dataloader was changed to return int32 tensors, therefore the conversion in the forward method could be removed. The run without conversion was slightly faster. However, the difference was well below 1% and not significant enough to justify the change.

6.4.2.2. Source Key Padding Masking

src_key_padding_mask is an optional parameter in the *TransformerEncoder* Class and allows to mask out padding elements in the input sequence. Masked elements will be ignored by the attention. (“Transformer — PyTorch 2.1 documentation”, n.d.)

Since padding is applied to the input sequences to ensure equal length, we decided to use this parameter. Therefore a padding mask tensor is created for each forward call. This calculation of this mask is trivial but we still wanted to assess if there is any significant impact on runtime. We’ve conducted an experiment run without the masking instruction and the runtime savings were below 2%. While it would be possible to preprocess this padding mask, the savings are not significant enough to justify the additional complexity and memory usage.

6.5. Error Analysis

6.5.1. Additional Error Metrics

Metrics are an essential part of our experiment tracking and are fundamental to evaluate run performance. At the very start of this project, only Validation- and Training Loss were recorded as metrics. When working with RNNs, an often occurring problem was predictions converging to zero. The loss values were not insightful enough to distinguish between zero predictions and reasonable predictions with inaccuracies.

To address this problem, two new metrics were introduced. *pred_max* and *pred_min* record the maximum and minimum prediction value for each epoch, which gives some insight into the model's tendencies. Thanks to these metrics, runs with zero convergence issues could be recognized and terminated in the first few epochs of training, which saved a lot of time.

The loss function in our model was revised multiple times. The goal of this was to approximate the Kaggle challenge scoring function and improve model performance. For example, the Kaggle Scoring function (MAE) ignores predictions for prefix and suffix positions and penalizes predictions outside of the range [0,1]. Additionally to that, an error-dependent loss function was introduced to weigh errors according to base reading quality.

To evaluate these incremental changes in the loss function, multiple new metrics were introduced. This allowed us to evaluate whether newly introduced loss functions showed reasonable behavior. This is a list of all additional metrics:

- pred_min, pred_max
- train_loss_clipped, val_loss_clipped
- train_loss_weighted, val_loss_weighted

6.5.2. Sequence Prediction Error Analysis

The additional Error Metrics described in 6.5.1 gave some insight into the model's tendencies. However, to analyze the occurrence of errors in a sequence, a closer examination was needed. The goal of this is to spot patterns and potential influences on the error. In order to compare ground truth values with predictions, a small sample of sequences was used. It is important to note, that the sample does not consist of unseen data. Therefore this analysis is not an advisable measure of model generalization capabilities.

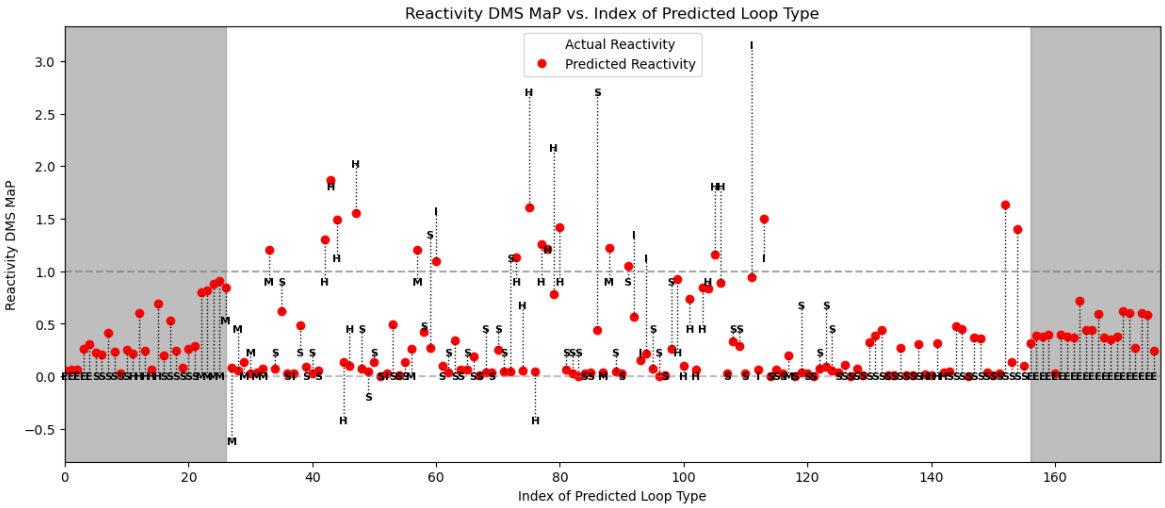


Figure 6.4.: Prediction Deviations over a sequence

Figure 6.4 shows a sample sequence of length 177 and the corresponding *actual reactivities* (denoted as loop type characters) and the *predicted reactivities* (denoted with red points). The vertical dotted line illustrates the magnitude of prediction errors. The greyed-out ranges left and right are prefix- and suffix parts of the sequence and are excluded from evaluation. The horizontal grey dotted lines on y=0.0 and y=1.0 denote the valid value range for the Kaggle Evaluation. All predictions above 1.0 or below 0.0 will strictly lead to an increase in error.
From a biological point of view, this chart gives some remarkable insights. The loop type of a base has a significant impact on its reactivity. Stem bases (**S**) generally have lower values than bases in a hairpin loop (**H**) or multiloop (**M**).

This behavior is in accordance with RNA theory. Stem bases (**S**) are already in a bond since a stem consists of at least 2 consecutive base pairs. Therefore, reactivites are usually low for stem segments. In the case of hairpin loops (**H**) and multiloops (**M**), the bases are usually not in a bond and are often structurally exposed. This makes them attractive candidates for bonding, hence the high reactivity values. (“RNA Secondary Structure as a Graph Using the forgi Library — forgi 2.0.0 documentation”, n.d.)

The model seems to have learned some structural tendencies. For this sequence, it seems to underestimate Stem Bases pretty consistently. Hairpin loop bases, in contrast, are often overestimated.

There are some valuable learnings from this analysis:

- Predictions outside of the range [0,1] are a significant source of error and have to be handled (Clipping of Input/Output Values)
- Reactivities are strongly influenced by structural properties like loop type. Structural data has to be incorporated for accurate predictions.
- Neighboring bases often show similar reactivities. Short-range dependencies are important.

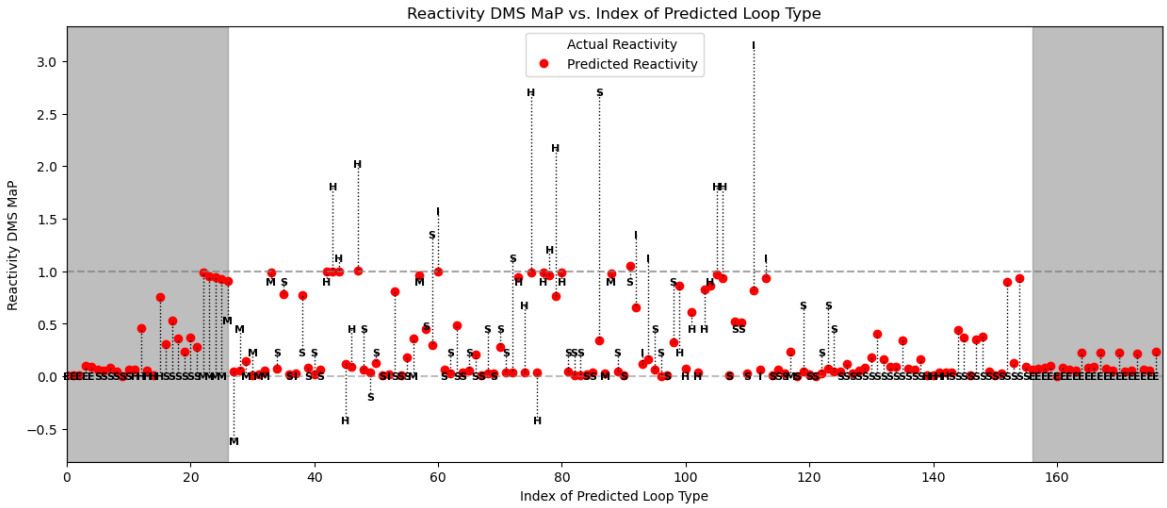


Figure 6.5.: Prediction Deviations over a sequence with Input Clipping

Figure 6.5 shows the same sequence with different predictions. This time, the model was trained with InputClipped Values. All reactivities in the train data have been clipped to [0,1]. This was the only change but had the immediate effect, that predictions were in this value range pretty consistently. Both this visualization and further experiments with InputClipping showed improved performance. Therefore it was enabled for all further experiments. Besides that, OutputClipping will also be applied by default because it will strictly lead to better scores.

6.6. Results

We decided to continue with the Transformer model because of two reasons. First, it has a more sophisticated and versatile architecture (refer to 6.3.3 Rationale). Also, this architecture is used by almost every public solution, which substantiates its capabilities. Secondly, it has achieved the best scores in our experiments (Public Scores). The section 6.7 elaborates on the different models and their scores.

The best model consists of preprocessing settings (refer to 6.2.6) and architecture settings in the table below.

Setting	Component	Value
d_model	TransformerEncoderLayer	200
nhead	TransformerEncoderLayer	5
dim_feedforward	TransformerEncoderLayer	800
dropout	TransformerEncoderLayer	0.2
ntokens	Embedding	500
nlayers	TransformerEncoder	6
optimizer	Optimizer	AdamW
learning_rate	Optimizer	0.001
weight_decay	Optimizer	0.05
scheduler	Scheduler	CosineAnnealingLR

Table 6.12.: Final architecture settings

6.7. Score Comparison

The following table 6.13 lists the best Kaggle submissions for each model. There are two leaderboards on Kaggle, public and private. While both have been evaluated for every submission, private scores have been available after December 8, 2023, when the competition closed. The differences between public and private scores can be explained by the different length ranges used. The private score is calculated on sequences with a higher length range of 207-457 whereas the public score is calculated on the same length range as in the training data. This should help to test the model's generalization capabilities. ("Stanford Ribonanza RNA Folding", n.d.)

Submission	Private Score	Public Score
Zero	0.3329	0.3397
Mean Values	0.2951	0.2994
LSTM	0.2125	0.2146
RNN	0.2024	0.2075
Transformer	0.2131	0.1648
DNABert	0.2810	0.2815
RoBERTa	0.2810	0.2807

Table 6.13.: Public and Private Kaggle Scores for each model type

The table 6.13 is in chronological order and shows the most important submissions. The best scores are marked in each category. The Public Score shows a gradual improvement over time. The last two submissions were both for pre-trained models (DNABert and RoBERTa). These models were still in an early experimentation phase by the time of submission and had not matured, hence the high error. We decided to include them nevertheless.

To offer a point of reference for scores, we included two baseline submissions: *Zero* and *Mean Values*. *Zero* is, as the name suggests, a submission with all predictions set to zero. The submission *Mean Values* consists of mean reactivities for the two reagents, which was calculated on the training data.

It is remarkable to see the Transformer achieving similar private scores as recurrent models despite the promising Public Score. This could be explained by a possible overfitting to the shorter training sequences. We also theorized that RNNs could have an inherent advantage of generalizing to any sequence length because of the recurrence, which focuses on short-range dependencies. This could explain the score similarities for recurrence-based models. However, this is just a theory and has not been conclusively confirmed.

The RNN model achieved the best private score. This model was extended with Cross-Validation and was developed along with the Transformer, mainly to test and evaluate new features. This allowed for more efficient testing since the RNN is computationally cheaper than the Transformer. This further development is the reason, why it is listed after LSTM and achieves better scores than the LSTM. However, the result with the best Private Score was unexpected since this submission was just an experiment to test Cross-Validation with RNN.

As for the next steps, we would either continue with the Transformer and focus on improving its generalization capabilities or continue developing the pre-trained models.

7. Conclusion

The Stanford Ribonanze RNA Folding Challenge closed on December 8, 2023. There have been 891 competitors in 755 teams with a combined total of 9'814 submissions. ("Stanford Ribonanza RNA Folding", n.d.)

Team	Rank	Public Score	Private Score
vigg	1	0.13499	0.13960
Hoyeol Sohn	2	0.13544	0.14015
ap eh ka	3	0.13503	0.14037
HSLU StableConfusion	615	0.19483	0.27650

Table 7.1.: Leaderboard Scores (sorted by Private Score)

For leaderboard evaluation, two submissions can be selected. This prevents participants from brute-forcing the leaderboard. By default, the two submissions with the highest public score are selected. We decided to not override this selection. However, it turned out that these did not generalize well and the error on the Private Score was unusually high (0.27650). As shown in Table 6.13, Private Scores of 0.2024 were achieved in other submissions. However, these submissions were not in the selection for leaderboard evaluation.

Our team (HSLU StableConfusion) was placed at rank 615 out of 755. The rank was not a core objective for us. It was much more important to come up with own ideas and approaches. If rank was the main objective, it would have been more efficient to copy a good-ranking public notebook and use it as a starting point. However, this would have come with the disadvantage of skipping a significant part of model engineering and testing. Still, the publicly available work of users like *iafoss* has been very helpful for us.

In this competition, a lot of submissions can be attributed to the submission of publicly available solutions. There have been 150 teams with a private score of 0.26885 and 78 teams with a private score of 0.23524. These scores could be achieved by using a public solution without any changes. Competitors with this score had an average submission count of 4.07 respectively 2.37 whereas the average submission count over all teams was 12.95. Our group counts 29 submissions. ("Stanford Ribonanza RNA Folding", n.d.)

All Prize Winners in the Leaderboard decided to release their solution to the public. There are some significant differences between the winning solution and our approach. All winner solutions calculated BPPM (Base Pair Probability Matrix) and used this as a feature. A BPPM represents the probabilities for every possible base pair in an RNA sequence and plays an essential role in RNA secondary structure predictions. (Hamada, 2012) BPPMs as a feature has proven to be highly beneficial. According to the winners, EternaFold has shown the best performance for BPPM prediction. (vyaltsevvaleriy, n.d.)

Another remarkable insight is, that all prize winners incorporated CNNs (Convolutional Neural Networks) in their solution, mostly in the form of *Conv2D* or *Conv1D* layers. In almost all cases, the architecture of the model was highly customized. For example, the winning group modified the Self-Attention block to include a Convolutional block, which calculates BPPM values. (vyaltsevvaleriy, n.d.)

In conclusion, highly customized and sophisticated architectures have been used to achieve the best results in this competition. There is definitely a good amount of knowledge and experience in ML research required to develop such solutions with novel approaches. In our view, all the knowledge, research and discourse that has been shared in this competition is not only highly beneficial for the participants, but moreover of great value for the entire field of RNA folding research.

8. Reflection

Overall, the project was a great experience for all of us. We got along well, which is key for a great team. It was the first time that we worked on an AI project developing our own AI solution, which gave us many learnings. The project was a great insight into how our future work life could look like and how to approach an AI project. The entire machine learning workflow and model evaluation on a real-world problem was a great new experience for the whole team.

8.1. Competition Selection

The whole team is happy about the project selection. Because of previous experiences with hackathons and other projects, we knew how to evaluate the different possible projects on Kaggle. It was important for us that we weren't constrained by the dataset, and that's why we looked for challenges with big datasets. Additionally, it was important for us to work on a challenge that had a clear path to a solution, but was still complex enough that we wouldn't get bored after a few weeks. The challenge that we selected exactly met that criteria. It was clear that we needed a model that is able to handle the context of the whole RNA sequence, a task we never worked on previously, but at the same time, we knew that when we decided to work on the challenge, the problem would be solvable with an implementation of an already existing model.

8.2. Teamwork

We decided to work together on this project long before the first lecture started. We knew each other before, and most of us worked together on countless projects before. This was essential to working efficiently together and leveraging each other's strengths. This made the work together enjoyable and efficient. We all agree that this project was the most efficient group project of our entire academic career, and we are all proud of the result.

8.3. Challenges

Morgan Fabien
For me, this project had the biggest scope of any project I ever worked on in my entire academic career. This obviously meant that I faced some major challenges during the project. The biggest challenge that I encountered was to implement the newly learned machine learning algorithms at the same time we were studying them. Before this semester, we only had the machine learning theory and the math behind it. I didn't implement any machine learning algorithms on my own before. Then in this semester I visited the NLP course in parallel to this project and all the models that we implemented in this project I gathered a deep understanding at the same time as we implemented them. This made it quite a challenge because I was forced to get a deep understanding for the models in a short time and couldn't just study them later in the semester. The same could be said about coding the models. Because the previous lectures did also not focus on the implementation I never worked with the library PyTorch before. This meant not only that I needed to get a deep understanding of the ML models in a short time, but also that I needed to learn a new library at the same time. Other than that I was used to working with tools like Git, Jupyter notebooks and all the other tools that were not machine learning specific which at least meant I didn't need to learn these tools.

Kluser Flavio
We've worked on many projects together before, even participated in hackathons. I'm grateful for our previous experiences since that helped us to work together efficiently and benefit from previous learnings. AI Challenge was our biggest endeavor yet. The Stanford Ribonanza RNA Folding Challenge posed two main challenges in my view. First of all, it required a comprehensive understanding of modern neural network architectures, particularly the Transformer. We've only delved into these topics this semester in the NLP course. The second challenge was the required domain knowledge. I only had surface knowledge of RNA and biology in general. To understand the problem and the type of data we were working with, I had to do quite a bit of research. Overall, a lot of experimentation and research was needed to figure things out.

Kessler Joel
During my participation in this course, I was able to deeply grasp the concept of experimental tracking. As each and every decision leading to a better-performing model required to do extensive experimentation which was rather challenging. Experiments could take up multiple days therefore it was necessary that each and every team member had knowledge of what the others were doing, so not to repeat the same experiments. Another important part was that we had limited resources available. While sometimes this could lead to crunches, it helped a lot in the long run as we focused on memory-efficient ways to implement the models. This limitation also gave us a realistic insight into what type of resources we could expect in a real project where money and time is a limiting factor.

Nyungmartsang Choekyel
The scope of the project itself presented a challenge. Although our team held regular meetings on Tuesdays and Thursdays, I found myself dedicating most of my time to the project. Challenges, such as gathering domain knowledge in RNA, comprehending its structure, understanding the behavior of knots, and making decisions about the model and its tuning, were successfully handled as a team. Personally, I found it challenging to efficiently divide the workload while still trying to understand all aspects of the implementations. Unlike the workplace, where time equals money, in the school setting, my goal was to learn, implement, make mistakes, and learn again on my own terms. Additionally, writing this project report using LaTeX was a new experience for me. However, once set up, the process became quite enjoyable.

8.4. Learnings

Morgan Fabien
My challenges and learnings tie well together. I learned to implement my first machine learning algorithms that weren't instructions that you could just blindly follow. I learned everything around implementing my first model like coding my own training loop, evaluating the model, saving the model and run inference on it. Furthermore, I can confidently say after this project that I would be able to work on a machine learning project and implement the demanded technologies. I probably would still need more time and experience to be proficient in that work, but I feel confident enough to start projects in the field of ML and would know what to look for and where to look for if I'm stuck. Something I couldn't say before this project.

Kluser Flavio
Throughout this project, I've learned an incredible amount. We have been working with a variety of ML implementations, ranging from Recurrent models over the Transformer to Pretrained Models. Exploring all of them and their inner workings has been incredibly insightful. Besides this, a lot of work has gone into preprocessing to introduce new parameters while maintaining compatibility. The workflows and interworkings in AI tasks became much clearer to me. Also, the Machine Learning Operations aspect was new to me. Setting up an Experiment Tracking environment to evaluate all the different training runs, some local and some in the cloud, was a valuable insight. Lastly, I've learned something about RNA biology and gained fascination for the topic. I'm confident that this project has been a valuable experience for upcoming projects.

Kessler Joel
Embarking on a project that requires knowledge in a previously for me unknown field of research, RNA was a very insightful although sometimes stressful procedure. Over the course of this module, I got many great insights into preprocessing data and how to expand on other parts evolving around AI and ML. Especially on data quality above quantity and customized loss functions. Overall I was able to greatly increase my knowledge in adopting a proper ML workflow and the steps it takes to create an environment where every participant, without prior knowledge of the topic, can work together to solve such an astonishingly difficult task.

Nyungmartsang Choekyel
Throughout the project and participating in a Kaggle competition, I discovered that it is a fantastic way to assess your knowledge on a global scale. It was very interesting to see the diverse knowledge base used by others compared to what we had learned throughout the semester. Like my teammates, I gained valuable insights into preprocessing techniques, introducing new features, and training a transformer model. The experiment tracking of the runs and the combination of biology and machine learning was particularly interesting, and we recognize its potential. This experience provided not only technical knowledge but also a deeper understanding of how to combine different fields to solve complex problems.

Bibliography

7 the simple recurrent network: A simple model that captures the structure in sequences. (n.d.). Retrieved January 8, 2024, from https://web.stanford.edu/group/pdplab/pdphandbook/handbookch8.html

Abaskohi, A. (2023). Navigating Transformers: A Comprehensive Exploration of Encoder-Only and Decoder-Only Models, Right... Retrieved January 10, 2024, from https://medium.com/@amirhossein.abaskohi/navigating-transformers-a-comprehensive-exploration-of-encoder-only-and-decoder-only-models-right-a0b46bdf6abe

Alammar, J. (n.d.). The Illustrated Transformer. Retrieved January 5, 2024, from https://jalammar.github.io/illustrated-transformer/

Brownlee, J. (2018). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. Retrieved January 5, 2024, from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Brownlee, J. (2019). A Gentle Introduction to the Rectified Linear Unit (ReLU). Retrieved January 5, 2024, from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Calzone, O. (2019a). An intuitive explanation of lstm [[Online; accessed December 5, 2023]]. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c

Calzone, O. (2019b). An intuitive explanation of lstm [[Online; accessed December 5, 2023]]. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c

Chat GPT. (2023a). Prompt: Can you describe me experiment tracking in the context of a machine learning workflow for a documentation i need to write
follow up prompt: Can you write the same text but without bullet points for my report?
follow up prompt: Can you shorten this text to one paragraph (one third of it's original length)? [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023b). Prompt: Can you just write one paragraph about machine learning models in general [[Online; accessed December 4, 2023]]. https://chat.openai.com/

Chat GPT. (2023c). Prompt: Can you just write one paragraph about machine learning models in general
follow up prompt: Can you write me a text in the same style as you just wrote about the machine learning architecture called transformer?
follow up prompt: Thanks for the text. can you rewrite it? can you not write about the invention. the text should purely focus on what transformer is and not mention models like bert or t5 [[Online; accessed December 4, 2023]]. https://chat.openai.com/

Chat GPT. (2023d). Prompt: Can you just write one paragraph about machine learning models in general
follow up prompt: Can you write me a text in the same style as you just wrote about the machine learning architecture called transformer?
follow up prompt: Thanks for the text. can you rewrite it? can you not write about the invention. the text should purely focus on what transformer is and not mention models like bert or t5
follow up prompt: Can you write one paragraph about how transformer uses embeddings [[Online; accessed December 4, 2023]]. https://chat.openai.com/

Chat GPT. (2023e). Prompt: Can you just write one paragraph about machine learning models in general follow up: Can you write me a text in the same style as you just wrote about the machine learning architecture called transformer? follow up: Thanks for the text. can

you rewrite it? can you not write about the invention. the text should purely focus on what transformer is and not mention models like bert or t5 follow up: Make the text shorter [[Online; accessed December 4, 2023]]. https://chat.openai.com/

Chat GPT. (2023f). Prompt: Can you write a text without bulletpoints about machine learning workflow? [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023g). Prompt: Can you write a text without bulletpoints about machine learning workflow?
follow up prompt: Can you shorten the text to ml workflow. it should mainly talk about versioning and experiment tracking [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023h). Prompt: Can you write me a chapter for my documentation about the python library pytorch?
follow up prompt: I think you missunderstood the task. i want a chapter in the context of machine learning frameworks that describes the funcitonality of pytorch. the machine learning frameworks chapter is a subchapter of the machine learning theory chapter
follow up prompt: Just write me a text without bullet point what pytorch is and for what it gets used. i don't need to know who developed it [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023i). Prompt: Can you write me a chapter for my documentation about the python library pytorch?
follow up prompt: I think you missunderstood the task. i want a chapter in the context of machine learning frameworks that describes the funcitonality of pytorch. the machine learning frameworks chapter is a subchapter of the machine learning theory chapter
follow up prompt: Just write me a text without bullet point what pytorch is and for what it gets used. i don't need to know who developed it
follow up prompt: Can you write me a text in the same stile as you just wrote about the python library pandas?
follow up prompt: Can you shorten the 3 paragraph text to one paragraph? [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023j). Prompt: Can you write me a chapter for my documentation about the python library pytorch?
follow up prompt: I think you missunderstood the task. i want a chapter in the context of machine learning frameworks that describes the funcitonality of pytorch. the machine learning frameworks chapter is a subchapter of the machine learning theory chapter
follow up prompt: Just write me a text without bullet point what pytorch is and for what it gets used. i don't need to know who developed it
follow up prompt: Can you write me a text in the same stile as you just wrote about the python library scikit-learn? scikit.-learn only got used for the train and test split. can you write the text knowing that fact?
follow up question: Can you shorten the text from two paragraphs to one paragraph? it also talks about how we used scikit learn in our project which shouldn't appear in the shortened text. [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023k). Prompt: Can you write me a subchapter about pretraining?
follow up prompt: Can you write me the same type of text just as one text without bullet points and titles?
follow up prompt: Can you shorten the four paragraph text into a two paragraph text? [[Online; accessed December 5, 2023]]. https://chat.openai.com/

Chat GPT. (2023l). Prompt: Can you write me a subchapter about pretraining?
follow up prompt: Can you write me the same type of text just as one text without bullet points and titles?
follow up prompt: Can you write a subchapter in a similar style as the previous output

about hyperparameter tuning
follow up prompt: Can you shorten this five paragraph text into two paragraphs? [[Online; accessed December 5, 2023]]. https://chat.openai.com/

Chat GPT. (2023m). Prompt: Can you write me a text for the subject machine learning frameworks, it should contain why what machine learning frameworks are for (in our case for neural networks and preprocessing) and why frameworks in general are used
follow up prompt: Can you write me a text without bullet points and don't talk about specific libraries like tensorflow, pytorch or keras. this will come up in a later stage of the documentation
follow up prompt: Can you shorten the text from five paragraphs to two? [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023n). Prompt: For an other subchapter i need a text about machine learning models. can you write me a text about that?
follow up prompt: You shouldn't write about machine learning in general. we are solving a sequential machine learning problem. i have an example text on how it should look like: Moreover, the library ensures that the split is conducted in a random yet reproducible manner. randomization is key in ensuring that the training and testing sets are representative of the overall dataset, avoiding biases that might skew the model's performance. [[Online; accessed December 4, 2023]]. https://chat.openai.com/

Chat GPT. (2023o). Prompt: I use git as a versioning tool for the project. write about what versioning is and that we use git. don't use lists or bullet points and only focusing at these subjects: Track changes, collaboration, experimentation, branching and mergeing
follow up prompt: Perfect. can you also add a paragrah that we hosted the code on github?
follow up prompt: Can you detach the text about git and versioning into two texts. one that generally talks about versioning and one about git in specific
follow up prompt: Can you shorten the text about versioning to one paragraph (a third of the length)? [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023p). Prompt: I use git as a versioning tool for the project. write about what versioning is and that we use git. don't use lists or bullet points and only focusing at these subjects: Track changes, collaboration, experimentation, branching and mergeing
follow up prompt: Perfect. can you also add a paragrah that we hosted the code on github?
follow up prompt: Perfect. can you also add a paragrah that we hosted the code on github?
follow up prompt: Can you shorten the text (half it's length) [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023q). Prompt: User i need to write a subchapter in my documentation about the python library forgie. can you summerize what the functionallity of that library are?
follow up prompt: Can you write me a similar text but without bulletpoints and not mentioning the subjects: Visualization integration, support for bpseq and fasta formats, pair table generation, element identification and analysis, subgraph and minimum spanning tree analysis, graph traversal
follow up prompt: Can you shorten the text from five paragraphs to two paragraphs? [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Chat GPT. (2023r). Prompt: User i need to write a subchapter in my documentation about the python library forgie. can you summerize what the functionallity of that library are?
follow up prompt: Can you write me a similar text but without bulletpoints and not mentioning the subjects: Visualization integration, support for bpseq and fasta formats, pair table generation, element identification and analysis, subgraph and minimum spanning tree analysis, graph traversal
follow up prompt: Can you write me a similar text as above about the python library arnie
follow up prompt: Can you write me a similar text but without bulletpoints [[Online; accessed December 3, 2023]]. https://chat.openai.com/

Das, S., Tariq, A., Santos, T., Kantareddy, S. S., & Banerjee, I. (2023). Recurrent neural networks (RNNs): Architectures, training tricks, and introduction to influential research. In O. Colliot (Ed.), *Machine learning for brain disorders* (pp. 117–138). Springer US. https://doi.org/10.1007/978-1-0716-3195-9_4

Das lab [[Online; accessed October 9, 2023]]. (n.d.). https://daslab.stanford.edu

Dirks, R. M., Lin, M., Winfree, E., & Pierce, N. A. (2004). Paradigms for computational nucleic acid design. [Place: England]. *Nucleic acids research*, *32*(4), 1392–1403. https://doi.org/10.1093/nar/gkh291

Eterna [[Online; accessed October 9, 2023]]. (n.d.). https://eternagame.org

Hamada, M. (2012). Direct Updating of an RNA Base-Pairing Probability Matrix with Marginal Probability Constraints. *Journal of Computational Biology*, *19*(12), 1265–1276. https://doi.org/10.1089/cmb.2012.0215

Hassan, A. (2021). Why bidirectional RNN is better than unidirectional RNN: A theoretical proof. https://www.deepwizai.com/simply-deep/a-mathematical-proof-of-why-bidirectional-rnns-are-better-than-unidirectional-rnns

Kaggle competition information [[Online; accessed October 9, 2023]]. (n.d.). https://www.kaggle.com/competitions/stanford-ribonanza-rna-folding

Katrompas, A., Ntakouris, T., & Metsis, V. (2022). Recurrence and self-attention vs the transformer for time-series classification: A comparative study. In M. Michalowski, S. S. R. Abidi, & S. Abidi (Eds.), *Artificial intelligence in medicine* (pp. 99–109). Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-031-09342-5_10

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). A Simple Way to Initialize Recurrent Networks of Rectified Linear Units [_eprint: 1504.00941]. https://arxiv.org/abs/1504.00941

Marinus, T., Fessler, A. B., Ogle, C. A., & Incarnato, D. (2021). A novel SHAPE reagent enables the analysis of RNA structure in living cells with unprecedented accuracy. *Nucleic Acids Research*, *49*(6), e34. https://doi.org/10.1093/nar/gkaa1255

Minchin, S., & Lodge, J. (2019). Understanding biochemistry: Structure and function of nucleic acids. [Place: England]. *Essays in biochemistry*, *63*(4), 433–456. https://doi.org/10.1042/EBC20180038

Mitchell, D., III, Cotter, J., Saleem, I., & Mustoe, A. M. (2023). Mutation signature filtering enables high-fidelity RNA structure probing at all four nucleobases with DMS. *Nucleic Acids Research*, *51*(16), 8744–8757. https://doi.org/10.1093/nar/gkad522

Mohammadi, M., Mundra, R., Socher, R., Wang, L., & Kamath, A. (2019a). Cs224n: Natural language processing with deep learning part v language models, rnn, gru and lstm [[Online; accessed December 3, 2023]]. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/readings/cs224n-2019-notes05-LM_RNN.pdf

Mohammadi, M., Mundra, R., Socher, R., Wang, L., & Kamath, A. (2019b). Cs224n: Natural language processing with deep learning part v language models, rnn, gru and lstm [[Online; accessed December 3, 2023]]. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/readings/cs224n-2019-notes05-LM_RNN.pdf

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, *63*(10), 1872–1897. https://doi.org/10.1007/s11431-020-1647-3

Raschka. (2023). About LayerNorm Variants in the Original Transformer Paper, and Some Other Interesting Historical Tidbits About LLMs. Retrieved January 5, 2024, from https://magazine.sebastianraschka.com/p/why-the-original-transformer-figure

RNA - central molecule of life. (n.d.). Retrieved January 13, 2024, from https://molecool.ch/en/rna-knowledge/detail/rna-zentrales-molekuel-des-lebens

RNA Secondary Structure as a Graph Using the forgi Library — forgi 2.0.0 documentation. (n.d.). Retrieved January 5, 2024, from https://viennarna.github.io/forgi/graph_tutorial.html

RNN — PyTorch 2.1 documentation. (n.d.). Retrieved January 5, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

Rogers, R. (n.d.). Rna folding. Retrieved January 13, 2024, from http://compbio.pbworks.com/w/page/16252918/RNA%20Folding

Stanford Ribonanza RNA Folding. (n.d.). Retrieved January 5, 2024, from https://kaggle.com/competitions/stanford-ribonanza-rna-folding

Tertiary Structure of Protein: Function & Techniques. (n.d.). Retrieved January 13, 2024, from https://www.studysmarter.co.uk/explanations/chemistry/organic-chemistry/tertiary-structure-of-protein/

Transformer — PyTorch 2.1 documentation. (n.d.). Retrieved January 9, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html

TransformerEncoder — PyTorch 2.1 documentation. (n.d.). Retrieved January 12, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html

TransformerEncoderLayer — PyTorch 2.1 documentation. (n.d.). Retrieved January 5, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention is all you need* [[Online; accessed December 4, 2023]]. arXiv:1706.03762v7. https://arxiv.org/abs/1706.03762

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention Is All You Need [_eprint: 1706.03762]. https://arxiv.org/abs/1706.03762

vyaltsevvaleriy. (n.d.). [1st place solution] transformer model with dynamic positional encoding + cnn for bppm features. Retrieved January 12, 2024, from https://kaggle.com/competitions/stanford-ribonanza-rna-folding

Wang, X., Gu, R., Chen, Z., Li, Y., Ji, X., Ke, G., & Wen, H. (2023). UNI-RNA: UNIVERSAL PRE-TRAINED MODELS REVOLUTIONIZE RNA RESEARCH [Publisher: Cold Spring Harbor Laboratory _eprint: https://www.biorxiv.org/content/early/2023/07/12/2023.07.11.548588.full.pdf]. *bioRxiv*. https://doi.org/10.1101/2023.07.11.548588

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., & Liu, T.-Y. (2020). On Layer Normalization in the Transformer Architecture [_eprint: 2002.04745]. https://arxiv.org/abs/2002.04745

A. Code

Listing A.1: TransformerEncoder Model Definition

```
class PositionalEncoding(torch.nn.Module):
    def __init__(self, d_model, dropout = 0.1, max_len = 5000):
        super().__init__()
        self.dropout = torch.nn.Dropout(p=dropout)

        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log
            ↪ (10000.0) / d_model))
        pe = torch.zeros(max_len, 1, d_model)
        pe[:, 0, 0::2] = torch.sin(position * div_term)
        pe[:, 0, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe)

    def forward(self, x):
        """
        Arguments:
            x: Tensor, shape ``[seq_len, batch_size, embedding_dim]``
        """
        x = x + self.pe[:x.size(0)]
        return self.dropout(x)

class Transformer_RNA(torch.nn.Module):
    def __init__(self, ntoken, d_model, nhead, d_hid, nlayers, out_dim,
        ↪ dropout = 0.5):
        super().__init__()
        self.model_config = {"ntoken":ntoken, "d_model":d_model, "nhead":
            ↪ nhead, "d_hid":d_hid, "nlayers":nlayers, "out_dim":out_dim,"
            ↪ dropout":dropout}
        self.model_type = 'Transformer'
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        encoder_layers = torch.nn.TransformerEncoderLayer(d_model, nhead,
            ↪ d_hid, dropout, norm_first=True)
        self.transformer_encoder = torch.nn.TransformerEncoder(
            ↪ encoder_layers, nlayers)
        self.embedding = torch.nn.Embedding(ntoken, d_model)
        self.d_model = d_model

        self.linear = torch.nn.Linear(d_model, out_dim)

        self.init_weights()

    def init_weights(self):
        initrange = 0.1
```

```
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.linear.weight.data.uniform_(-initrange, initrange)
        self.linear.bias.data.zero_()

    def forward(self, src):
        src = src.squeeze(dim=-1)
        mask = (src == 0)
        src = src.permute(1, 0)

        src = self.embedding(src.to(torch.int32)) * math.sqrt(self.d_model)
        src = self.pos_encoder(src)

        output = self.transformer_encoder(src, src_key_padding_mask=mask)

        # Apply final linear layer
        final_output = self.linear(output)

        final_output = final_output.permute(1, 0, 2)

        return final_output

def create_transformer(suffix, epochs):
    ntokens = 500 # size of vocabulary (Note: This has to be > ~500 if
        ↪ Structure is encoded)
    emsize = 200 # embedding dimension
    d_hid = 800 # dimension of the feedforward network model in ``nn.
        ↪ TransformerEncoder``
    nlayers = 6 # number of ``nn.TransformerEncoderLayer`` in ``nn.
        ↪ TransformerEncoder``
    nhead = 5 # number of heads in ``nn.MultiheadAttention``
    dropout = 0.2 # dropout probability
    out_dim = 2 # 1 if dual model, 2 if single model

    transformer = Transformer_RNA(ntokens, emsize, nhead, d_hid, nlayers,
        ↪ out_dim, dropout)
    optimizer = torch.optim.AdamW(transformer.parameters(), lr=0.001,
        ↪ weight_decay=0.05)
    scheduler = CosineAnnealingLR(optimizer, T_max=epochs)
    transformer = base_model.BaseModel(
        optimizer, transformer, f'TRANSFORMER-{suffix}.pth', scheduler=
            ↪ scheduler, enable_wandb=True)
    return transformer
```