Un ringraziamento speciale alle mie relatrici, per avermi guidato nei miei primi passi in un nuovo campo, e per avermi trasmesso il valore di una ricerca biomedica utile e d'impatto.

Dedico questa tesi ai miei nonni e a tutta la mia famiglia.

Contents

Chapter 1

Introduction

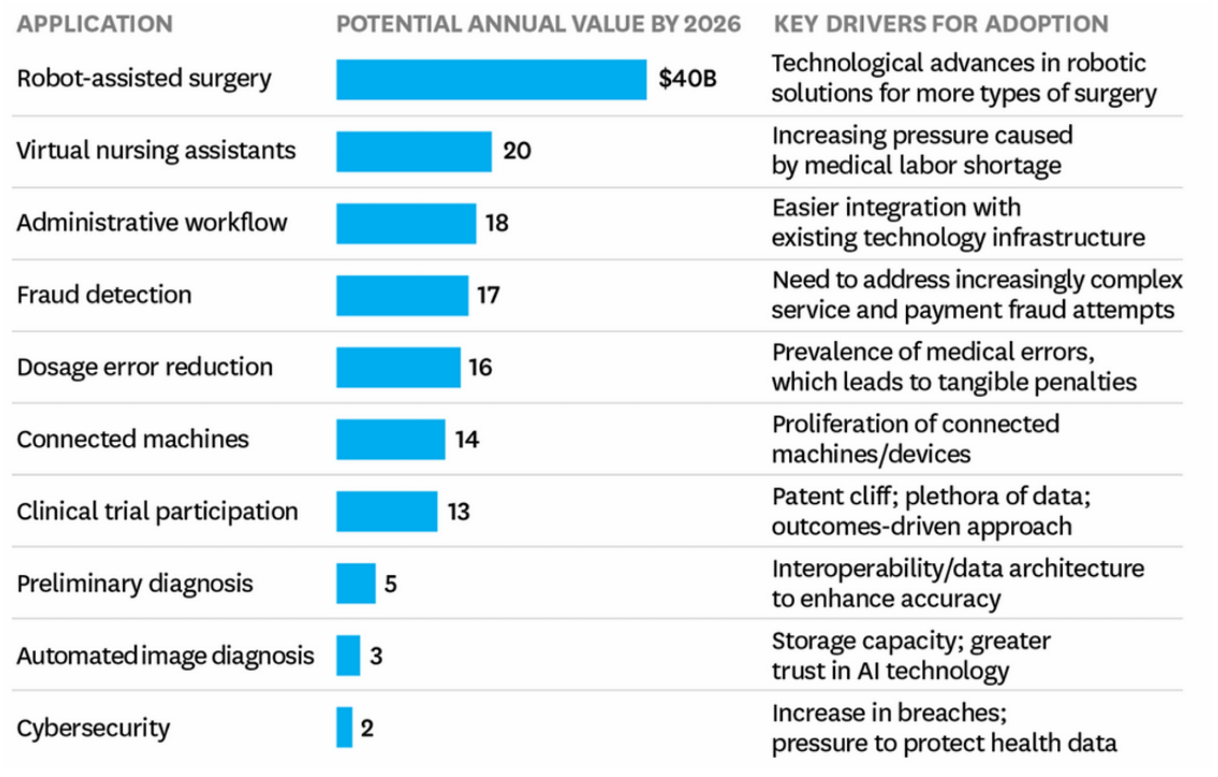
1.1 The rise of bioinformatics

Bioinformatics is an interdisciplinary field that employs techniques from computer science, mathematics, and statistics to analyze and interpret phenomena of a biological nature.

This discipline emerged in response to the need to manage and analyze the vast amounts of data produced by genomics research. Indeed, new technologies drastically increased the quantity, quality, and diversity of available biological data. Today, the field of genomics generates 40 thousand petabytes of data per year, which, for comparison, is orders of magnitude above YouTube's 2000 PB, astronomy's 1000 PB, and Twitter's 17 PB [2]. These numbers highlight that the amount of information produced yearly by experiments is too vast to be processed without resorting to computational tools, and that this field is fertile ground for machine learning algorithms.

The completion of the Human Genome Project in 2003 marked a significant milestone, highlighting the importance and impact of computational tools in biology. Since then, advancement in sequencing technologies, computer power, and algorithmic tools, have established bioinformatics as a fundamental part of biological and medical research.

The potential of AI in the medical field is immense, with implications to personalized treatment, automatic diagnosis, drug discovery, medical imaging, multi-omics integration, and robotic surgery.



APPLICATION	POTENTIAL ANNUAL VALUE BY 2026	KEY DRIVERS FOR ADOPTION
Robot-assisted surgery	$40B	Technological advances in robotic solutions for more types of surgery
Virtual nursing assistants	20	Increasing pressure caused by medical labor shortage
Administrative workflow	18	Easier integration with existing technology infrastructure
Fraud detection	17	Need to address increasingly complex service and payment fraud attempts
Dosage error reduction	16	Prevalence of medical errors, which leads to tangible penalties
Connected machines	14	Proliferation of connected machines/devices
Clinical trial participation	13	Patent cliff; plethora of data; outcomes-driven approach
Preliminary diagnosis	5	Interoperability/data architecture to enhance accuracy
Automated image diagnosis	3	Storage capacity; greater trust in AI technology
Cybersecurity	2	Increase in breaches; pressure to protect health data

Figure 1.1: *The potential economic impact of AI in health, source: Accenture*

A particularly successful example of modern deep learning tools employed in a biological context was *DeepMind*'s *AlphaFold* [3], released in 2021. This model profoundly impacted the field of proteomics by practically solving the long standing *protein folding problem*, which consists in predicting the three-dimensional spatial arrangement of a protein from its amino acid sequence. This is a crucial task because a protein's shape dictates its function in biological processes. It's worth noting that the general setting is NP-hard [4] and thus intractable efficiently unless P=NP. Fortunately, as AlphaFold showed, it is feasible for the amino acid sequences that appear in the real world, for which a machine learning algorithm can approximate the exact solution. The model's predictions were comparable in quality to experimentally determined structures.

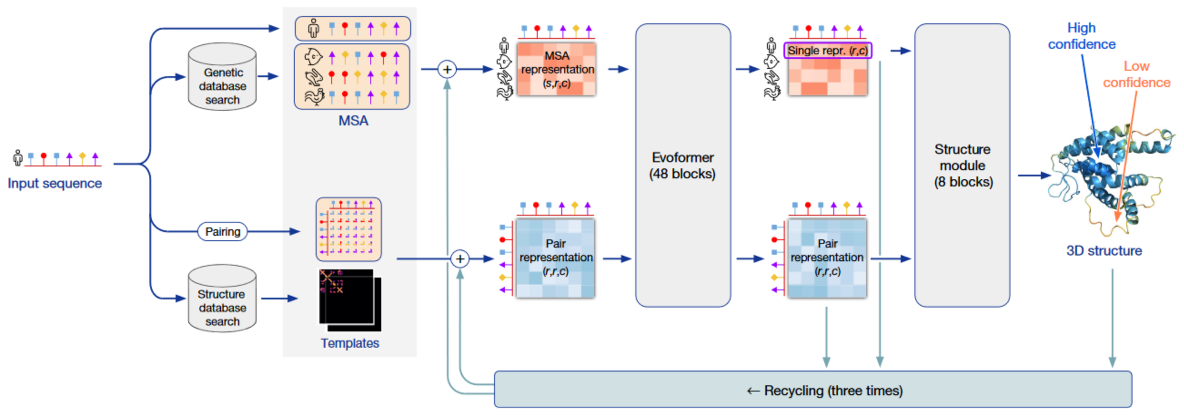

Input sequence
Genetic database search
MSA
Pairing
Structure database search
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Figure 1.2: *Schematization of AlphaFold's architecture, from the original DeepMind paper* [3].

1.2 Motivation

Proteins are fundamental agents in our bodies that carry out a wide range of critical cell functions. They provide the structural support needed for the cell to maintain its shape, catalyze biochemical reactions, and enable the complex metabolic processes that are necessary to life. Additionally, they act as receptors and messengers, allowing the cell to receive and send information from and to its environment.

Understanding how diseases affect the protein landscape can reveal important insights into the biological mechanisms implicated in such diseases, and it can help in designing effective treatments.

However, diseases are often better understood at the level of mRNA, which is easier to quantify. Several studies have reported discordance between mRNA and protein levels. Moreover, the relationship between the two is complex, noisy, and highly non-linear.

Bridging this gap would provide a more comprehensive understanding of cellular function and regulatory mechanisms.

While the mRNA molecules that are analyzed in this work code for protein synthesis, some proteins bind to specific regions of DNA and can, in turn, promote or inhibit the transcription of certain genes. These proteins are called transcription factors (TF). The study of how such systems evolve is generally based on gene regulatory networks (GRNs), and can greatly benefit from models that accurately predict and quantify the presence of regulators in unseen, new environments, and more in general by an improved understanding of the link between mRNA levels and TFs.

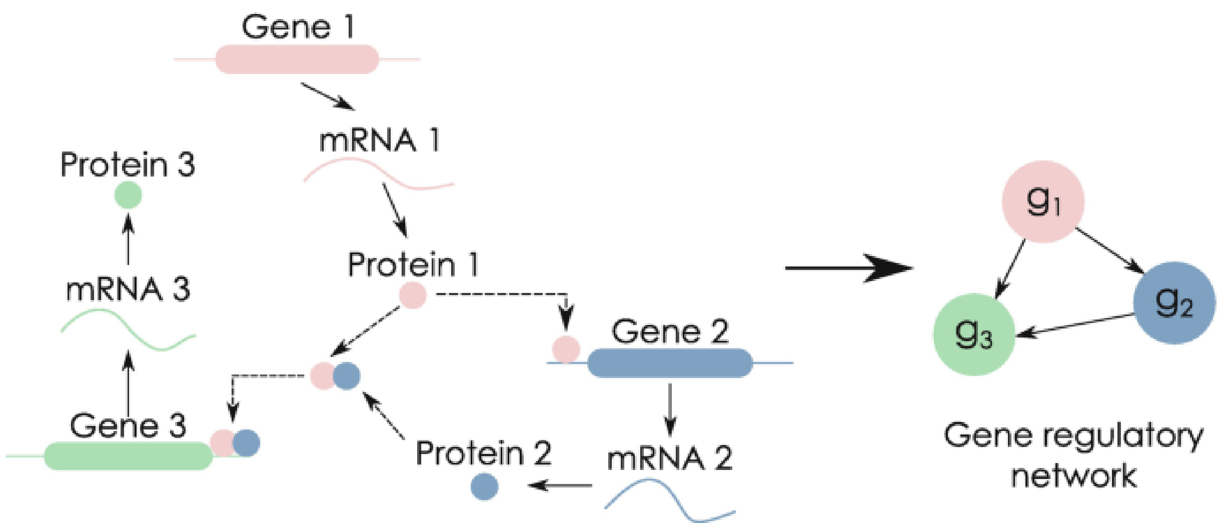


Figure 1.3: *In a GRN, two genes are linked if the first one codes for a TF that regulates the transcription of the second. Source: [5].*

Finally, this type of predictive model can provide other classifiers with useful additional information when the proteomics data is unavailable, for instance whenever a certain property (such as the presence of a specific disease) must be detected in samples using RNA levels. In that case, the transcriptomics can be used to generate synthetic protein abundances, and a new model can be trained to more accurately predict the presence of the property, leveraging both omics.

1.3 Research question

In an attempt to analyze at large scale the non-linear relationships between mRNA expression and protein abundance in cancer cell lines, the goal of this thesis is to investigate their correlation, as well as building and comparing different predictive models of protein levels based on transcriptomic profiles.

Regarding the latter part, this work focuses on two popular classes of non-linear machine learning algorithms: boosting and deep learning. These techniques capture the hidden relationships between variables, elucidating the complex dynamics governing protein synthesis and regulation in cancer. Once the models are trained, their predictions will be evaluated and compared, and their quality will determine whether or not the models are adequate for accurately inferring proteomic profiles.

The next chapters contain a description of the dataset and of the processing steps that were applied to it, a theoretical overview of the methods employed and of the metrics used for evaluation or comparison, an outline of the results, and a conclusive discussion.

1.4 Literature review

Many studies report that the levels of mRNA and the corresponding protein present an overall weak positive correlation, that varies a lot from gene to gene [6–11], while other works claim a general discordance between the two [12–14].

According to Upadhya et al. [10], reproducibly measured proteins display higher correlation with the corresponding transcript, suggesting that measurement errors limits the correlation between the two. Koussounadis et al. [7] suggest that differentially expressed genes are better correlated with their protein counterparts, emphasizing the informative power of this type of data. Other studies [11, 15] report that the correlation is highly dependent on gene function and cancer type. Some publications show that there are complex post-transcriptional effects that play a role in disrupting the link between RNA and proteins [12, 16]. Other interfering factors are the local availability of resources for protein biosynthesis, and the spatial and temporal variations in mRNA abundance. Correlations are higher at steady state, highlighting the complexity of regulation during dynamic transitions [12].

Furthermore, recent research introduced deep learning as an attempt to flexibly fill the gap between mRNA and proteins by building predictive models of the latter [17, 18]. The results strengthened trust in this approach, justifying an attempt to employ deep neural networks to infer protein abundance in an oncological setting.

Chapter 2

Materials

2.1 Data overview

The Cancer Cell Line Encyclopedia project was born in 2008 from a collaboration between the Broad Institute and the Novartis Research Foundation.

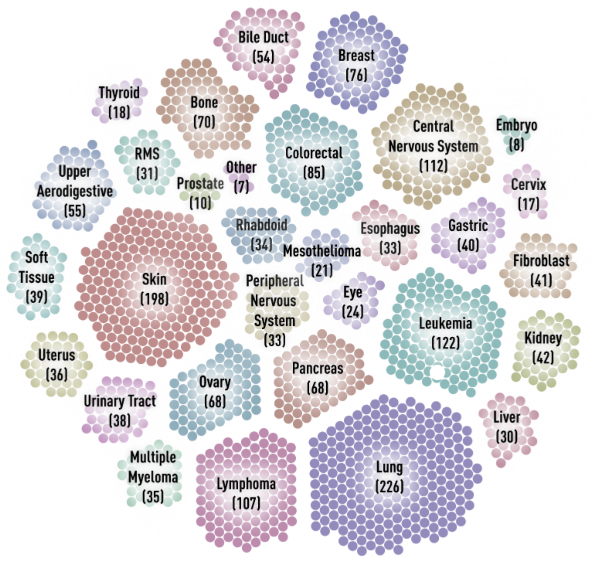


Figure 2.1: *The cell lines in CCLE, grouped by tissue. Source: DepMap portal*

The data can be found by consulting the key resources table in Upadhya et al. [10]. The transcriptomics come from Gandhi et al. [19] and can be downloaded from the DepMap portal, while the proteomics are from Nusinow et al. [20] and can be found on the website of the Gygi Lab, a laboratory of proteomics within Harvard Medical School.

In particular, throughout this study, I use `CCLE_expression_full.csv`, which includes all 52,004 genes, rather than `CCLE_expression.csv`, that only contains the 19,183 protein-coding genes. This is due to the potential presence of valuable information in non-coding RNA. Both files contain $\log_2(x+1)$-transformed expression values. In regards to the proteins, I opted for the normalized abundances: `Table_S2_Protein_Quant_Normalized.xlsx`.

Expression values are quantified from bulk RNA-seq using the GTEx pipelines, for a total of 1376 unique cell lines. The protein data includes proteomic profiles for 375 cell lines, obtained from mass spectrometry analysis, with 12,755 proteins being considered in the study.

2.2 Pre-processing

The data is initially processed as in the `1c_process_non_tumour_studies.ipynb` notebook from Upadhya et al.'s GitHub repository.

In particular, exploiting metadata from `sample_info.csv`, the dataframes are reindexed and filtered such that cell lines are identified by the CCLE name instead of DepMap ID, and genes/proteins by HGNC symbol rather than Ensembl/Entrez ID.

Then, the function `standardised_pipeline_utils.process` is applied to both datasets, which performed several favorable transformations:

1. rows (genes/proteins) missing more than 20% of values are dropped;
2. rows with the same index are collapsed into one row of average values;
3. rows that are indexed by a datetime object are dropped;

4. rows containing ":" are dropped.

This terminates the part of Upadhya's preprocessing that I decided to apply.

Subsequently, the sample *TT_OESOPHAGUS*, containing 4020 missing values in the expression data was removed. 2259 proteins containing missing values were dropped from the proteomics table. The result of this step were a 51,989 by 1375 transcriptomics table and a 5113 by 375 proteomics one, with neither containing any missing entries.

Finally, for the purpose of facilitating supervised learning tasks, the datasets were filtered to only include the 369 cell lines shared by both tables.

Chapter 3

Methods

For clarification purposes, and in order to guarantee accessibility to people of any background, this chapter gives a brief introduction to all the main methodologies employed throughout the thesis, starting from the very basics, and gradually progressing to more advanced topics.

3.1 Correlation measures

The first step in this project was to explore the correlation between RNA and protein. Three correlation measures were chosen. The scores were computed in Python using the Pandas method `corrwith`.

Pearson's correlation

Pearson's correlation coefficient measures the linear correlation between two variables. It is proportional to the covariance, but normalized to fit into $[-1, 1]$. Given random variables X and Y, it is defined as

$$r_{X,Y} := \frac{\mathrm{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Spearman's correlation

Spearman's ρ is the simplest measure of rank correlation. It is equivalent to Pearson's correlation coefficient between the rank variables $R(X)$ and $R(Y)$ obtained from X and Y

$$\rho_{X,Y} = r_{R(X),R(Y)} = \frac{\text{cov}(R(X), R(Y))}{\sigma_R(X)\sigma_R(Y)}$$

If the ranks are all distinct, and defining $d_i = R(X_i) - R(Y_i)$ as the difference between the ranks for an observation i, it can be characterized as

$$\rho_{X,Y} = 1 - \frac{6}{n(n^2-1)} \sum_i d_i^2$$

Kendall's correlation

Kendall's τ is another coefficient that measures the rank correlation between two variables. Considering observations $x_1, ..., x_n$ for variable X and $y_1, ..., y_n$ for variable Y, define a pair (i, j) with $x_i > x_j$ to be concordant if $y_i > y_j$ and discordant otherwise. Let the number of concordant pairs be n_c and the number of discordant observations be n_d. Define the coefficient as

$$\tau_{X,Y} = \frac{n_c - n_d}{n_c + n_d}$$

where $n_c + n_d = \frac{n(n-1)}{2}$.

3.2 Accuracy metrics

I will use several metrics to evaluate the quality of the predictions. I denote with y the true target value and with $\hat{y}(x)$ the prediction from the model.

- The mean squared error (MSE) is used throughout this thesis as a performance evaluator, as well as a loss function for training most of the regression models.

$$\text{MSE}(y, \hat{y}(x)) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} (y_i - \hat{y}(x_i))^2$$

- The mean absolute percentage error (MAPE) is more convenient to evaluate the precision of the estimates, and it is utilized extensively in the rest of the thesis.

$$\text{MAPE}(y, \hat{y}(x)) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \frac{|y_i - \hat{y}(x_i)|}{\max(y_i, \epsilon)}$$

  where ϵ is chosen to be 0.1, as an error of that size is negligible for our purpose.

- Throughout this work, I will use the term accuracy to refer to the share of test set samples with a $MAPE$ term below some value t, which is typically set to 10% unless another threshold is specified.

$$\text{Acc}(y, \hat{y}(x)) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{1} \left(\frac{|y_i - \hat{y}(x_i)|}{\max(y_i, \epsilon)} \leq t \right)$$

  This is similar to a typical accuracy measure for classification, where an estimate is classified correctly if it is within 10% from the true value.

- R-Squared is also used to determine the proportion of the variance that can be explained by the model. Define, with $\bar{y}$ being the mean observed target, the residual sum of squares SS_{res} and the total sum of squares SS_{tot} as follows

$$SS_{res} = \sum_{i \in \mathcal{D}} (y_i - \hat{y}(x_i))^2$$

$$SS_{tot} = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2$$

  Then R-Squared is defined as

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

3.3 Validation

For different predicting models, different validation approaches were used.

For boosting, *leave one out* was used to exclude one sample from the training data, and then *k-fold cross validation*, with $k = 5$ was used to determine the best configuration of parameters, which was then re-trained on the whole train set, and tested on the leave one out sample. This entire procedure was repeated for five different leave one out samples, obtaining five different predictions from the model, for each protein.

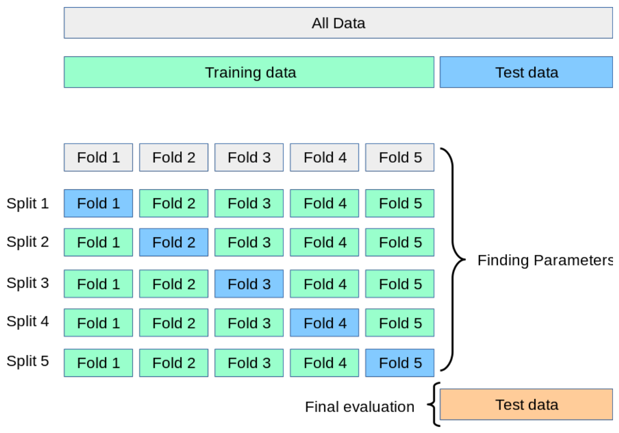


Figure 3.1: *The k-fold cross validation scheme, illustration from sci-kit learn*

For deep learning, a 90/10 train-test split was performed, and no automatic hyper-parameter tuning was implemented.

3.4 Feature selection

To reduce the computational cost of some of the algorithms, boosted regressors in particular, the F-statistic was used to obtain a selection of features for training. With $\bar{y}_j$ being the mean of the group to which observation y_i belongs, n the total number of observations, and k the number of groups, let the F-value be

$$F = \frac{SS_{between}/(k-1)}{SS_{within}/(n-k)}$$

where $SS_{between}$ and SS_{within} are defined as

$$SS_{between} = \sum_{j=1}^{k} n_j(\bar{y}_j - \bar{y})^2$$

$$SS_{within} = \sum_{i=1}^{K}\sum_{j=1}^{n}(y_i - \bar{y}_j)^2.$$

For each protein, the F-value was computed for every feature, and the 20 best features by F-value were selected for training the AdaBoost and XGBoost regressors.

3.5 Boosting

3.5.1 Ensemble learning

Ensemble methods are a vast class of machine learning algorithms that combine the predictions of multiple weak learners into a more precise and robust final prediction. The primary goal is to reduce the bias and variance that a single model might have.

In the general context of supervised learning, one often fixes an hypothesis class, and searches for an hypothesis that performs well in predicting some specific target variable. This is typically done via parametrization of the hypothesis space, and the search for a good hypothesis becomes a search for a good configuration of parameters. But even when the hypothesis space contains good functions, they can be particularly tricky to find. Ensemble methods combine different hypotheses into a single one, which will supposedly be more accurate.

3.5.2 Boosting methods

Boosting is a subclass of ensemble learning techniques, based on the idea of sequentially adding weak learners to make a strong one, with each weak model focusing on the errors made by the previous predictors. In this context, a weak learner is defined as a model whose predictions are only slightly better than random guesses, whereas a strong learner makes arbitrarily good predictions.

A boosted regressor $\hat{f}$ is a weighted sum of weak learners, where each weak learner fixes predictions about the data. Each weak regressor $\hat{\phi}_m$ is assigned a coefficient α_m such that for some error function E, it minimizes

$$\sum_{i=1}^{N} E(\hat{f}_{m-1}(x_i) + \alpha_m \hat{\phi}_m(x_i)).$$

3.5.3 AdaBoost

AdaBoost, which stands for Adaptive Boosting, was the first practical boosting algorithm [21]. Its creators, Freund and Schapire, were awarded the Turing prize for its invention.

The idea of AdaBoost is to sequentially train weak learners such as decision trees, focusing each time on the samples that were poorly predicted in previous iterations. This is done by keeping track of a collection of weights w_i, one for each sample x_i in the dataset. These values serve as weights in the decision tree's loss function. Once the decision tree is trained, it is used to compute errors that are, in turn, used to increase the weights of poorly-predicted samples. The algorithm also assigns each weak learner a coefficient α_m based on the overall weighted error of the model.

Algorithm 1: AdaBoost Regression

```
   Input  : Dataset $(x_i, y_i)_{i=1}^{N}$, number of iterations $M$
   Output: Final model $\hat{f}(x)$
 1 Initialize weights $w_i = \frac{1}{N}$ for all $i$;
 2 for $m = 1$ to $M$ do
 3     Train weak learner $\hat{\phi}_m$, using weights $w_i$;
 4     Predict $\hat{\phi}_m(x_i)$ for all $i$;
 5     Compute weighted error $e_m = \frac{\sum_{i=1}^{N} w_i \cdot |y_i - \hat{\phi}_m(x_i)|}{\sum_{i=1}^{N} w_i}$;
 6     Compute model coefficient $\alpha_m = \frac{1}{2} \ln\left(\frac{1-e_m}{e_m}\right)$;
 7     Update weights $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot |y_i - \hat{\phi}_m(x_i)|)$ for all $i$;
 8     Normalize weights $w_i \leftarrow \frac{w_i}{\sum_{j=1}^{N} w_j}$ for all $i$;
 9 Final model $\hat{f}(x) = \sum_{m=1}^{M} \alpha_m \hat{\phi}_m(x)$;
10 return $\hat{f}(x)$;
```

3.5.4 XGBoost

XGBoost, short for eXtreme Gradient Boosting, is another effective boosting algorithm which has gained popularity over the last decade.

Algorithm 2: XGBoost Regression

```
Input : Dataset $\{(x_i, y_i)\}_{i=1}^{N}$, number of iterations $M$, learning rate $\alpha$, loss
        function $L$
Output: Final model $\hat{f}(x)$
1  Initialize model with constant value: $\hat{f}_0(x) = \arg\min_\theta \sum_{i=1}^{N} L(y_i, \theta)$;
2  for $m = 1$ to $M$ do
3      Compute the gradient $\hat{g}_m(x_i) = \frac{\partial L(y_i,y)}{\partial y}$ at $y = \hat{f}_{m-1}(x_i)$;
4      Compute the hessian $\hat{h}_m(x_i) = \frac{\partial^2 L(y_i,y)}{\partial y^2}$ at $y = \hat{f}_{m-1}(x_i)$;
5      Fit a regression tree $\hat{\phi}_m = \arg\max_{\phi\in\Phi} \sum_{i=1}^{N} \frac{1}{2}\hat{h}_m(x_i)\left(\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right)$;
6      Update the model: $\hat{f}_m(x) = \hat{f}_{m-1}(x) - \alpha\hat{\phi}_m(x)$;
7  return Final model $\hat{f}_M(x)$;
```

Algorithm 3: XGB pipeline

```
1  Initialize table;
2  Set parameter grid;
3  for each protein do
4      Get $k$ best genes by F-value;
5      for each leave-one-out sample do
6          Perform 5-fold cross validation;
7          Obtain the best parameter configuration;
8          Train the final model;
9          Get the prediction on the leave-one-out sample;
10         Add the results to the table;
11 return Final table;
```

XGBoost computes, given a loss function L, the gradient and hessian

$$\hat{g}_m(x_i) = \frac{\partial L(y_i, y)}{\partial y}$$

$$\hat{h}_m(x_i) = \frac{\partial^2 L(y_i, y)}{\partial y^2}$$

for each sample i, evaluated at $y = \hat{f}_{m-1}(x_i)$, where $\hat{f}_{m-1}$ is the previous week learner. Then, the idea is to update the predictions similarly to what is done in Newton's method, where the step would be

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) - \alpha \frac{\hat{g}_m(x)}{\hat{h}_m(x)}.$$

However, instead of using $\frac{\hat{g}_m(x)}{\hat{h}_m(x)}$ directly, we train a decision tree $\hat{\phi}_m$ to obtain an estimate of this quantity, where the loss function of the decision tree is weighted according to the hessians

$$\hat{\phi}_m = \arg\max_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left(\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} \right).$$

This leads to an estimate of $\frac{\hat{g}_m(x)}{\hat{h}_m(x)}$ that is more precise for points x where the hessian of the loss is large. The motivation behind this is that the hessian reflects the curvature of the loss function, and when it is large, it indicates that the model is sensitive to changes in prediction for that point x_i, and that it deserves to be treated more carefully.

3.6 Deep learning

Deep learning is a statistical philosophy that suggests that overparametrization, paired with the right inductive bias, can be a powerful tool for learning complex patterns in data. The idea is to consider a very large hypothesis class, namely the class of deep neural networks with a certain fixed structure, and to deal with the issue of overfitting by introducing noise in the learning process, rather than by decreasing the model size. This section gives a brief overview of the most important aspects of this approach.

3.6.1 Introduction to neural networks

Artificial neural networks are biology-inspired non-linear models in which information is propagated from neuron to neuron and assembled into a final prediction. In their simplest form, they consist in a concatenation of affine transformations and non-linear activation functions. The data is fed into an input layer and manipulated through one or more hidden layers before producing a prediction.

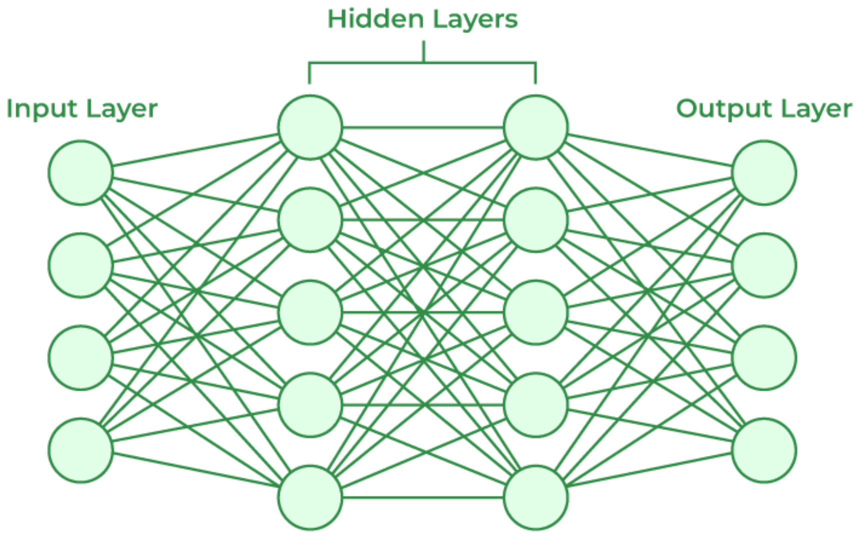


Figure 3.2: *An illustration of a simple neural network with two hidden layers, from an article on geeksforgeeks.org*

Intuitively, neurons will learn to activate if they detect some specific feature in the data. Neurons in the first layer only detect simple features that can be represented as the activation function of some linear transformation of the input data. Deeper layers will combine those simple features into more complex ones, until the level of complexity required by the final prediction is reached. Neural networks are universal classifiers, meaning that they can learn any arbitrarilly complex function, given that the model size is adequate.

Given $x_1^{l-1}, ..., x_{N_{l-1}}^{l-1}$ in layer $l-1$, the values of $x_1^l, ..., x_{N_l}^l$ in the next layer can be computed as the activation function of an affine combination of the previous layer. Matrix notation can be used for compactness

$$x_i^l = \sigma\left(\sum_{j=1}^{N_{l-1}} W_{ij}^l x_j^{l-1} + b_i^l\right) \forall i \in \{1, ..., N_l\} \implies \bar{x}^l = \sigma(W^l \bar{x}^{l-1} + \bar{b}^l).$$

If we include the bias $\bar{b}^l$ into the matrix W^l, which can be done by attaching a 1 to the end of each $\bar{x}^l$, the final prediction of the model boils down to

$$\hat{y}(\bar{x}) = W^l \sigma(W^{l-1} \sigma(...W^1(x))).$$

The activation function σ is applied element wise to the vectors and the most common choice for it is called ReLU, defined as $\max(0, x)$.

3.6.2 Learning in neural networks

The coefficients in W are parameters that need to be chosen and adjusted by the network. Learning is the process of finding a configuration of parameters that leads to good predictions.

In order to find such good configurations, we need three ingredients: a loss function, an algorithm for updating the parameter configuration, and a way to compute the gradient of the loss, which is needed by the updating algorithm.

In this thesis, which is mostly concerned with regression tasks, the mean square error was the loss function of choice, although the cross entropy loss is also utilized in the context of classifying binned data

$$\mathcal{L}_{\mathcal{D}}^{XEnt}(y, \hat{y}) = -\sum_{i=1}^{|\mathcal{D}|} y_i \log \hat{y}_i.$$

The most common algorithm for navigating parameter configurations is called gradient descent, and it simply states that we should move in the opposite direction to the gradient and proportionally to it, with a fixed multiplier called learning rate

Algorithm 4: Gradient Descent

```
Input  : Loss function $\mathcal{L}_\mathcal{D}$, number of iterations $M$, learning rate $\eta$
Output: Final configuration $\theta^*$
1 Initialize configuration $\theta_0$;
2 for $m = 1$ to $M$ do
3     Compute the gradient $\nabla_\theta \mathcal{L}_\mathcal{D}(\theta_{m-1})$;
4     $\theta_m := \theta_{m-1} - \eta \nabla_\theta \mathcal{L}_\mathcal{D}$;
5 return $\theta^* := \theta_M$;
```

The loss is a function of y and $\hat{y}(x)$. As previously discussed, the network is a concatenation of linear layers and non-linearities, meaning that the loss itself is a concatenation of functions. As for other concatenations, its gradient can be computed using the chain rule, which is what the *backpropagation* algorithm does. Although ReLU does not have a derivative at 0, this is not an issue, as the loss is differentiable almost everywhere and we can choose any of its subgradients with negligible practical consequences.

Throughout the thesis, I used PyTorch to build and train neural networks. The optimizer of choice was AdamW with stochastic gradients, with learning rates between 10^{-4} and 10^{-6}. Regularization techniques such as dropout and layernorm were also used extensively.

Stochastic Gradient Descent

To reduce the computational cost of calculating the gradient, we can consider an estimate of it by defining the loss function on a subset of the data called $mini - batch$. Since we are only moving according to an estimate of the gradient rather than its precise value, this will add random noise to the learning process, which turns out to be a good thing as it introduces the possibility of escaping local minima in the loss landscape.

Adam

The Adam optimization algorithm was introduced in a 2015 paper by Kingma and Ba [22]. Without delving too much into the details, Adam keeps a rolling average and rolling variance of the last gradients, using them to adapt its step at each iteration.

Denoting the gradient as g and the rolling average as $\langle \cdot \rangle$, Adam's step looks like

$$-\alpha \frac{\langle g \rangle}{\sqrt{\langle g^2 \rangle}}$$

and thus its step size depends positively on the weighted mean of the last gradients, and negatively on their weighted variance. This is because, intuitively, a higher variance means that there is uncertainty about what the optimal step may be, and that we should proceed carefully. Moreover, using a gradient average rather than its pure value can avoid oscillating effects which slow down training.

Adam is often paired with stochastic gradients for better performance.

AdamW

AdamW is just Adam with a weight decay. Analytically, it is equivalent to adding a ridge term to the loss. A weight decay behaves like a wind trying to slightly push the parameter configuration towards the origin, which is good for regularization as it prevents the parameters of the model from exploding or diverging.

Dropout

Dropout is a regularization technique that consists in turning off some neurons at random during training, to help the network learn more robust features and avoid overfitting.

Algorithm 5: Adam Optimization Algorithm

```
Input  : Loss function $\mathcal{L}_\mathcal{D}$, number of iterations $T$, learning rate $\alpha$, decay rates
         $\beta_1, \beta_2$, small constant $\epsilon$
Output: Final configuration $\theta^*$
1 Initialize parameters $\theta_0$, $m_0 = 0$, $v_0 = 0$, and $t = 0$;
2 for $t = 1$ to $T$ do
3     Compute the gradient $g_t = \nabla_\theta \mathcal{L}_\mathcal{D}(\theta_{t-1})$;
4     Update biased first moment estimate $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$;
5     Update biased second moment estimate $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$;
6     Compute bias-corrected first moment estimate $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$;
7     Compute bias-corrected second moment estimate $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$;
8     Update parameters $\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$;
9 return $\theta^* := \theta_T$;
```

Dropout, with a dropping probability of 0.5, was used extensively throughout the thesis. A dropout of 0.1 was also employed in all the Transformer regressors.

LayerNorm

LayerNorm essentially applies a layer-wise standard scaler, which avoids exploding or vanishing gradients due to the initialization of the network.

With low data and small batch sizes, as in our case, it is a more principled choice than the BatchNorm alternative. For this reason, it was used very often throughout this work.

3.6.3 The deep learning approach

Ultimately, the goal is to learn a parameter configuration that works well on unseen new data, rather than just on the training set. If our model is very complicated, meaning that it is chosen from a large class of functions, or, equivalently, that it is overly parametrized, it is possible that it will learn to perfectly fit the training set, while behaving terribly outside of it. This issue is called overfitting. In classical statistics, a solution is found by limiting the complexity of the model and disallowing the model from exactly learning the training data. The model will find the parameters that work best on the training set, but hopefully it will generalize well to new data as it was only able to learn a summary of the data. Naturally, however, this comes at the cost of limiting the complexity and diversity of learnable patterns.

Deep Learning is an approach that goes in the opposite direction. It is based on the idea of over-parametrizing the model, applying the right inductive bias, and adding some noise in the learning process such that the model will not converge to the parameters that work best on the training set, but to some configuration of parameters that lays in a large valley in the loss landscape, and which will likely generalize well.

3.7 Autoencoders and latent spaces

3.7.1 Autoencoders

An autoencoder is a neural network that learns a compact and informative representation of the data by trying to reconstruct the inputs after they are passed through a bottleneck. It can be interpreted as an encoder-decoder architecture, where the encoder's goal is to learn a short summary of the input, called latent representation, and feeding it to the decoder, that will try to reconstruct the original input from the encoding. Mathematically, assuming input space $\mathbb{R}^N$ and latent space $\mathbb{R}^M$, the encoder and the decoder are parametric

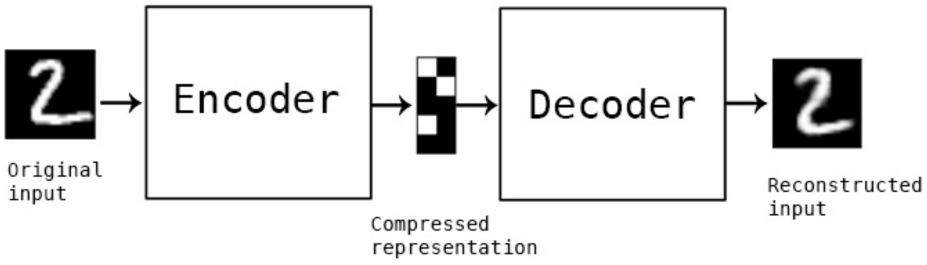


Figure 3.3: *An illustration of an image autoencoder for the MNIST dataset, from an article by Francois Chollet on the Keras blog*

functions of the form

$$E_\theta : \mathbb{R}^N \to \mathbb{R}^M,\ D_\phi : \mathbb{R}^M \to \mathbb{R}^N.$$

For each $x \in \mathbb{R}^N$, we obtain its latent representation z and reconstruction $\hat{x}$ as follows

$$z = E_\theta(x) \in \mathbb{R}^M$$

$$\hat{x} = D_\phi(z) = D_\phi(E_\theta(x)) \in \mathbb{R}^N.$$

Of course, it is important that the dimension of the latent space M is strictly smaller than the dimension of the input space N, in order to disallow the encoder from communicating the entire information to the encoder, which will then easily retrieve the original input, leading to a useless interpolation of the identity function.

One key consideration about autoencoders is that they are typically able to learn efficient representations because the input space is not fully exploited by the data. Points often all lie close to a lower-dimensional *data manifold* within the input space. In this interpretation, the latent representation can be seen as a parametrization of the manifold. The reconstructed input can then roughly be interpreted as the projection of the input on the manifold.

The autoencoder's loss function is called *reconstruction error*, and it is the mean squared error between the original and reconstructed inputs over the whole dataset $\mathcal{D}$

$$\mathcal{L}_{rec}^{\mathcal{D}}(\theta, \phi) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} (x - D_\phi(E_\theta(x)))^2.$$

3.7.2 Variational Autoencoders

In variational autoencoders, or VAEs, the encoder maps each data point to a distribution in latent space, rather than a fixed latent representation. This distribution is often assumed to be a Gaussian with some learned mean $\mu(x)$ and variance $\sigma^2(x)$, whereas the prior of z is a standard normal

$$z|x \sim \mathcal{N}(\mu(x), \sigma^2(x)),\ z \sim \mathcal{N}(0, 1).$$

The encoder of a VAE learns a vector that has twice the size of the latent space. This vector gets split into two, the first part being the mean, μ, and the second being the logarithm of the variance, $2 \log \sigma$, as taking directly the variance would lead to issues with dealing with a learned negative variance and numerical problems. After retrieving μ and σ, the VAE samples ε at random from a standard normal, and $z = \mu + \sigma\varepsilon$ is fed to the decoder.

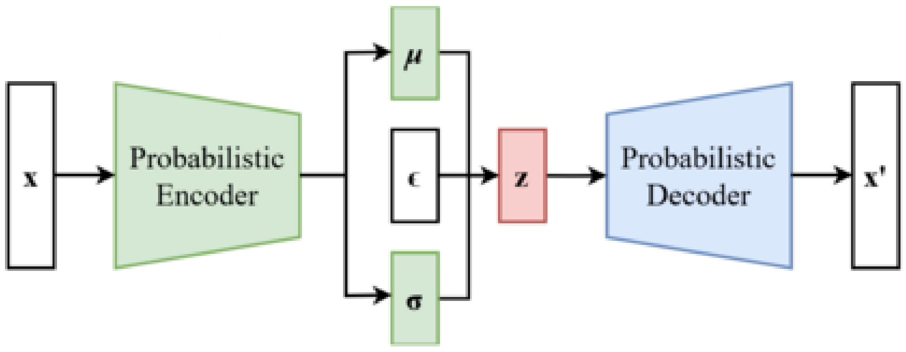


Figure 3.4: *Schematic illustration of a variational autoencoder from Wikipedia*

The VAE's objective function is called Evidence Lower Bound, or ELBO, and it jointly works to reduce the reconstruction error and to make sure that the distribution of the latent representations resembles the prior

$$\mathcal{L}_{ELBO} = \mathbb{E}_{z\sim q(\cdot|x)}[\log p(x|z)] - D_{KL}(q(z|x)||p(z))$$

In this case, $q(z|x)$ is the approximate posterior distribution, modelled by the encoder network as a Gaussian with learned mean and variance. $p(x|z)$ is the likelihood of the data given the latent representation, and it is modelled by the decoder. $p(z)$ is the prior on the latent variable, typically a standard normal.

In this context, D_{KL} is the Kullback-Leibler divergence, and it is defined as

$$D_{KL}(q||p) := \sum_{y\in\mathcal{Y}} q(y)\log\left(\frac{q(y)}{p(y)}\right)$$

$$D_{KL}(q||p) := \int_{\mathcal{Y}} q(y)\log\left(\frac{q(y)}{p(y)}\right)dy$$

in the discrete and continuous cases respectively. Here q and p are distributions over some domain $\mathcal{Y}$.

Under the mentioned Gaussian assumptions, maximizing the ELBO corresponds to minimizing the following loss

$$\mathcal{L}_{VAE} = \mathcal{L}_{rec} + D_{KL} = ||x - \hat{x}(z)||^2 + \frac{1}{2}\sum_{i=1}^{|\mathcal{D}|}(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2).$$

By ensuring that the latent variable is roughly normally distributed, VAE's yield more robust and informative representations than classical autoencoders. Although they were recently surpassed by diffusion models, VAEs have been at the forefront of generative AI for the majority of the last decade, and they still find wide applicability in biology.

3.7.3 β-Variational Autoencoders

A β-VAE is simply a VAE with an adjustable coefficient to the Kullback-Leibler term in the loss

$$\mathcal{L}_{VAE} = \mathcal{L}_{rec} + \beta D_{KL} = ||x - \hat{x}(z)||^2 + \frac{\beta}{2}\sum_{i=1}^{|\mathcal{D}|}(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2)$$

The coefficient β is a hyperparameter that can be chosen through cross validation.

In this thesis, I used a β-VAE with $\beta = \frac{1}{2}$, whose loss is a midpoint between a standard VAE and a classical autoencoder's loss. The autoencoders were used to obtain short representations of the transcriptomics data, leveraging the information contained in the spare unlabelled RNA data to obtain more revealing latent vectors. The size of the latent space was chosen to be 128, with the encoder and the decoder consisting of 3 linear layers each. No dropout was employed, as the risk of overfitting is null. ReLU activations and layer normalization were inserted between each linear transformation, and AdamW was the optimizer of choice.

3.8 Attention and Transformers

Transformers are feed-forward encoder-decoder neural networks based on multi-head attention [23]. They were developed in the context of machine translation, but they have been applied successfully to a wide variety of other domains, including transcriptomics [24].

Initially, tokens (e.g. words) are embedded into a *semantic space* where orthogonal directions encode high-level properties that a word could carry, such as its masculinity/femininity. Once a good embedding is learned, our space should roughly satisfy rela-

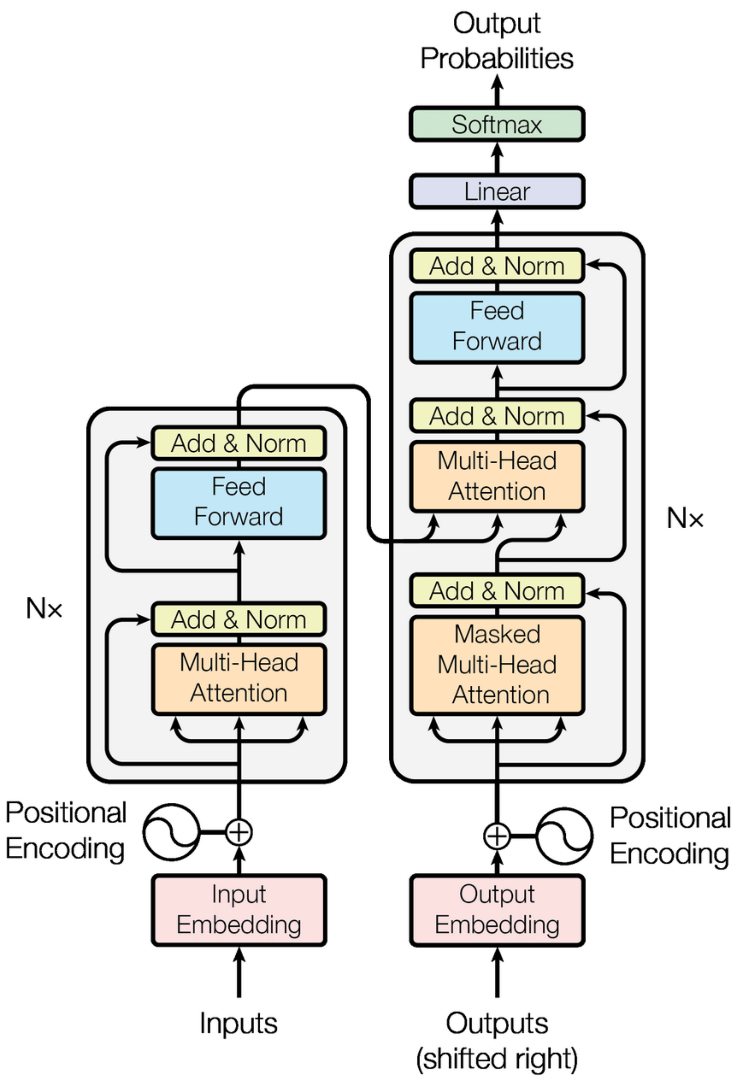


Figure 3.5: *Illustration of the Transformer architecture from the original "Attention is All You Need" paper*

tions of the type

$$E_{king} - E_{man} + E_{woman} \approx E_{queen}$$

as the vectors $E_{king} - E_{queen}$ and $E_{man} - E_{woman}$ are both expected to point in a *male direction*, by approximately the same amount.

The encoder part of a transformer takes an input sequence, such as an English sentence, and, through a mechanism called self-attention, updates the representation of each token using the rest of the sentence. For example, depending on the context, the word *model* could mean very different things, but the presence of nearby words such as *mathematics* or *fashion* reveals its semantic meaning, and this information can be used to update E_{model} in the embedding space. Similarly, we can imagine that the expression values of some group of genes might shed light on the abundance of some other gene, leading to an enriched, more complete picture.

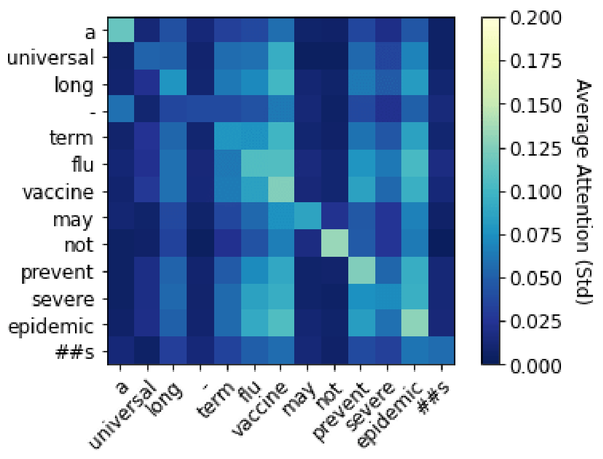


Figure 3.6: *Visualization of self-attention weights [25]*

In the decoder part of a classic transformer, the goal is to predict the next token given the tokens generated so far, and the input sentence from the encoder. In the context of translation, the decoder produces, word by word, the Italian translation of the English sentence that was fed to the encoder. In each decoder layer, similarly to what was seen earlier, self-attention within the beginning of the target sentence helps the model to get a better grasp of the ongoing semantics. Finally, cross-attention is computed between the input and output sentence, which helps the model understand which tokens of the input sentence it should focus more on for predicting the next one.

Similarly to language translation, biological translation, the cellular process that leads to the production of new peptides from mRNA templates, is complex and irregular. Transformers could be suitable for learning contextually rich representations of the tran-

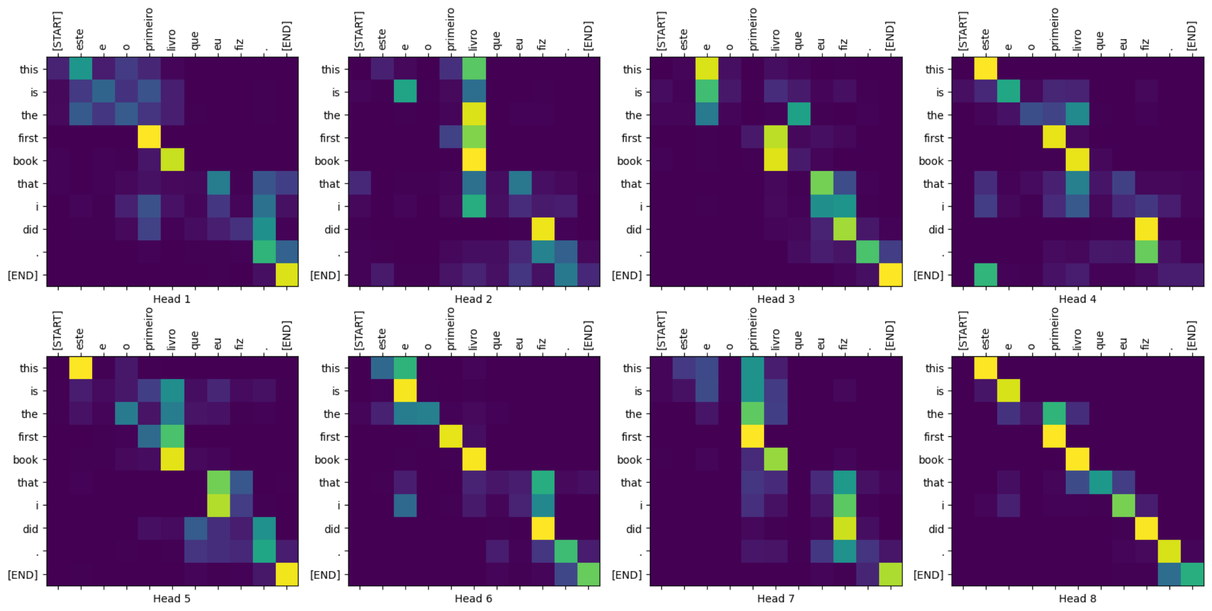


Figure 3.7: *Example of cross-attention maps in English-Portuguese translation, from a tutorial on kerasflow.org*

scriptomic and proteomic profiles, and for understanding and exploiting the relationship between the two.

Without delving too much into the details, in self attention, the input data gets linearly mapped into query, key, and value matrices, named this way due to an analogy with the classical dictionary data structure. Attention scores are then computed as

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

passed through a linear layer, and added to the original data, which then gets normalized layer-wise. The result of this is the updated, attention-enriched representation in the embedding space. This procedure gets iterated multiple times and alternated to pure feed forward layers.

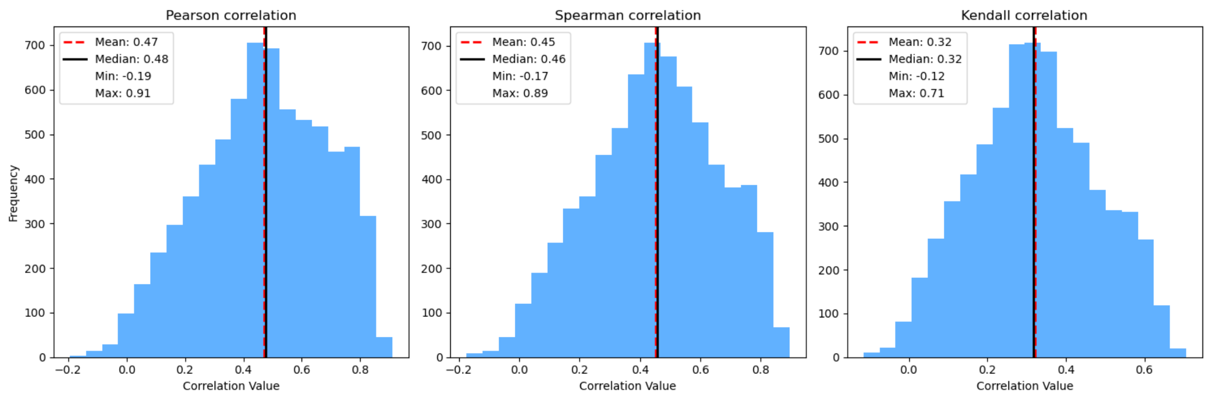
In this thesis, I opted for an encoder-only transformer with 4 encoder layers, 8 attention heads, and a 512-dimensional embedding space. I trained one version on pure transcriptomics data, and one on VAE latent representations. Finally, I built an encoder-decoder model with 4 encoder and 4 decoder layers, in an attempt to initiate an iterative data-generation procedure.

Chapter 4

Results and Discussion

4.1 Gene-protein correlations

Similarly to what was done in Upadhya et al. [10], correlations between associated gene-protein pairs were computed and analyzed. While that paper only considers Pearson and Spearman correlations, my analysis also includes Kendall's rank correlation, which is another measure of rank correlation. Pearson, Spearman, and Kendall correlations are calculated on processed, matched data, with 7372 proteins and 369 cell lines taken into considerations.



Plotting the frequencies of the computed correlation coefficients, we can see that the correlation metrics present a similarly shaped bell distribution.

The data is correlated, with only a few proteins exhibiting negative scores. This suggests that it is reasonable to construct predictive models of protein levels based on mRNA abundance.

Distribution of the correlation scores			
Coefficient	Pearson	Spearman	Kendall
Mean	0.4703	0.4507	0.3207
Standard deviation	0.2161	0.2109	0.1592
0%	-0.1945	-0.1746	-0.1154
25%	0.3180	0.3031	0.2069
50%	0.4772	0.4564	0.3172
75%	0.6369	0.6071	0.4340
100%	0.9106	0.8958	0.7064

As further confirmation of what can be found in the literature, gene-protein correlation varies a lot from protein to protein, with standard deviations above 0.2 for Pearson and Spearman coefficients.

4.2 Boosting-based protein inference

XGBoost and AdaBoost produced similar median MAPEs. However, XGBoost seemed to be significantly more precise in the worst case, leading to better mean MAPEs. For this reason, and due to lack of time and computing power, XGBoost was considered the better alternative, and it was trained on 5 different leave one out splits.

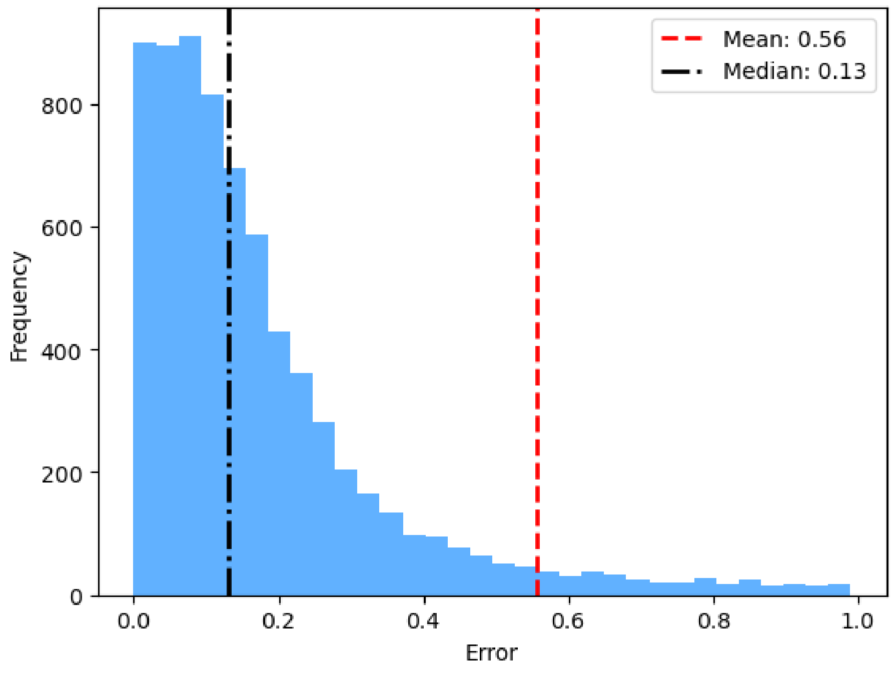


Figure 4.1: *Distributions of AdaBoost MAPE of different proteins, tested on one single LOO sample*

Its overall mean MAPE across proteins was 0.25, against AdaBoost's 0.56. The higher median of XGBoost in the overall case, 0.17, is just an effect of it being tested on multiple leave one out samples, and it shouldn't be seen as a negative result in comparison to Ada's 0.13, as can be seen by looking at XGB's single-sample distributions.

For the XGBoost predictions on each single leave one out sample, the accuracies (MAPE under 10%) were 0.36, 0.34, 0.48, 0.42 and 0.38 percent. However in the aggregated results, where the mean MAPE was considered for each unique protein, the accuracy became 19%. This may suggest that the validation procedure used in this thesis may be positively biased towards boosting methods, as in the deep learning section, a simple train-test split was applied, leading to reduced variability and fewer low-error proteins.

XGBoost's success can also be explained by the versatility provided by its large number of hyperparameters. In my analysis, again due to lack of time, I picked a fixed selection of

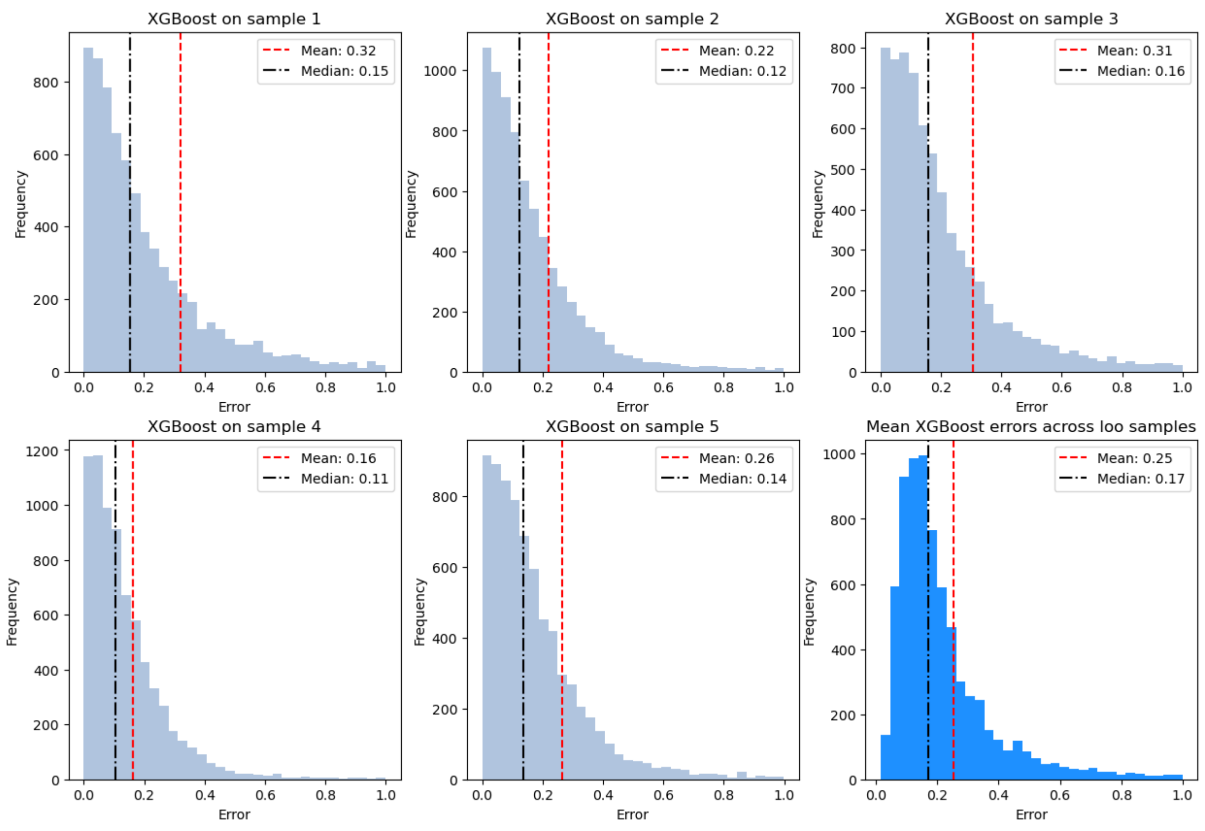


Figure 4.2: *Distributions of XGBoost MAPE of different proteins, for 5 different LOO samples and the protein-wise means*

hyperparameters by observing which configurations worked well on the first 500 proteins. The only hyperparameter tuned during cross-validation was the maximum depth of trees, letting the procedure choose between 3 and 6. Intuitively, this allowed the model to be more or less conservative depending on the protein, providing additional flexibility.

4.3 VAE performances

For the β-VAE with $\beta = \frac{1}{2}$ and latent dimension of 128, the reconstruction accuracy was 37%, meaning that only 37% of genes had a MAPE below 10%. Even though a VAE's reconstruction loss is generally worse than what can be achieved by a plain autoencoder of the same size, the latent representations are expected to be more robust, as the Gaussian assumptions usually confer VAE representations better generalization capabilities.

Latent representations were useful in allowing to speed up expensive procedures, as their dimension is 128 against the original transcriptomics data's 52 thousands. However, they did not lead to better accuracy when fed to regression models, with the latter mostly converging quickly to a slightly worse minimum than what plain-RNA models achieved.

4.4 Transformer-based protein inference

An encoder-only transformer with a 512-dimensional embedding space, 4 encoder layers, and 8 attention heads was built and trained on plain-RNA and latent VAE data. The model trained on latent data performed significantly worse, and was thus discarded.

In an attempt to obtain better predictions, an encoder-decoder transformer was trained by taking the predicted abundances of the first model as input to the decoder, while still using the true abundances as targets. After the first iteration, the quality of the predictions on a separate test set did not improve, suggesting that it would be pointless to continue iterating. This negative result was likely due to the relatively poor quality in the predictions of the first model, which is normal in biological settings but unsuitable to initiate such recursive semi-supervised procedures.

Accuracy scores for different precision thresholds	
Threshold	Encoder-only
1%	0.0525
5%	0.2408
10%	0.4360
25%	0.7637
50%	0.9090

The transformer encoder performed well for this task, even in comparison to the boosting algorithms, even though it was penalized by the validation procedure employed, as discussed in subsection 4.2. The following table displays the share of proteins that were predicted with MAPE below a certain threshold.

Chapter 5

Conclusion

To summarize, gene-protein correlations were computed, and turned out to be in line with the existing literature, with a moderate mean positive correlation and a lot of protein-to-protein variability.

XGBoost, AdaBoost, and Transformers were trained and tested for the task of inferring protein abundance from mRNA levels. Transformers performed slightly better than the boosting counterparts, with roughly 43% of proteins being predicted on average within 10% from the true value, and 90% of them with a mean error below 50%.

The main contribution of this thesis was to test the performance of modern machine learning algorithms in the context of cancer transcriptomics and proteomics. The analysis revealed that the large gene-specific variability in gene-protein correlation extends to predictive models.

The present work was limited by lack of computing power and time. Future work will focus on carrying out more complete validation and hyperparameter tuning procedures, leveraging high-performance computing and parallelization.

Other deep learning methods such as convolutional networks and graph neural networks have shown promising results in recent omics research, and they could be adapted to this task. Finally, it could be interesting to combine mechanistic models based on simulations or differential equations to the statistical approach employed in this thesis, either through ensemble learning or by incorporating biological knowledge into the loss function of a machine learning model.

Bibliography

[1] Edoardo Calderoni. *Nonlinear inference of protein abundance and regulation in cancer cell lines: boosting methods and deep learning.* July 2024. URL: https://github.com/caldeyy/pRNA.

[2] Zachary D. Stephens et al. "Big Data: Astronomical or Genomical?" In: *PLOS Biology* 13.7 (July 2015), pp. 1–11. DOI: 10.1371/journal.pbio.1002195. URL: https://doi.org/10.1371/journal.pbio.1002195.

[3] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (Aug. 2021), pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: https://doi.org/10.1038/s41586-021-03819-2.

[4] Aviezri S. Fraenkel. "Complexity of protein folding". In: *Bulletin of Mathematical Biology* 55.6 (1993), pp. 1199–1210. ISSN: 0092-8240. DOI: https://doi.org/10.1016/S0092-8240(05)80170-3. URL: https://www.sciencedirect.com/science/article/pii/S0092824005801703.

[5] Vân Anh Huynh-Thu and Guido Sanguinetti. "Gene Regulatory Network Inference: An Introductory Survey". In: *Gene Regulatory Networks: Methods and Protocols.* Ed. by Guido Sanguinetti and Vân Anh Huynh-Thu. New York, NY: Springer New York, 2019, pp. 1–23. ISBN: 978-1-4939-8882-2. DOI: 10.1007/978-1-4939-8882-2_1. URL: https://doi.org/10.1007/978-1-4939-8882-2_1.

[6] Marcus Gry et al. "Correlations between RNA and protein expression profiles in 23 human cell lines". In: *BMC Genomics* 10.1 (Aug. 2009), p. 365. ISSN: 1471-2164.

DOI: 10.1186/1471-2164-10-365. URL: https://doi.org/10.1186/1471-2164-10-365.

[7] Antonis Koussounadis et al. “Relationship between differentially expressed mRNA and mRNA-protein correlations in a xenograft model system”. In: *Scientific Reports* 5.1 (June 2015), p. 10775. ISSN: 2045-2322. DOI: 10.1038/srep10775. URL: https://doi.org/10.1038/srep10775.

[8] Fredrik Edfors et al. “Gene-specific correlation of RNA and protein levels in human cells and tissues”. en. In: *Mol Syst Biol* 12.10 (Oct. 2016), p. 883.

[9] Yu Han et al. “Transcriptome features of striated muscle aging and predictability of protein level changes”. In: *Mol. Omics* 17 (5 2021), pp. 796–808. DOI: 10.1039/D1MO00178G. URL: http://dx.doi.org/10.1039/D1MO00178G.

[10] Swathi Ramachandra Upadhya and Colm J. Ryan. “Experimental reproducibility limits the correlation between mRNA and protein abundances in tumor proteomic profiles”. In: *Cell Reports Methods* 2.9 (2022), p. 100288. ISSN: 2667-2375. DOI: https://doi.org/10.1016/j.crmeth.2022.100288. URL: https://www.sciencedirect.com/science/article/pii/S2667237522001709.

[11] Benoît P. Nicolet and Monika C. Wolkers. “The relationship of mRNA with protein expression in CD8+ T cells associates with gene class and gene characteristics”. In: *PLOS ONE* 17.10 (Oct. 2022), pp. 1–17. DOI: 10.1371/journal.pone.0276294. URL: https://doi.org/10.1371/journal.pone.0276294.

[12] Yansheng Liu, Andreas Beyer, and Ruedi Aebersold. “On the Dependency of Cellular Protein Levels on mRNA Abundance”. en. In: *Cell* 165.3 (Apr. 2016), pp. 535–550.

[13] Dongxue Wang et al. “A deep proteome and transcriptome abundance atlas of 29 healthy human tissues”. In: *Molecular Systems Biology* 15.2 (2019), e8503. DOI: https://doi.org/10.15252/msb.20188503. eprint: https://www.embopress.

org/doi/pdf/10.15252/msb.20188503. URL: https://www.embopress.org/doi/abs/10.15252/msb.20188503.

[14] Christine Wegler et al. “Global variability analysis of mRNA and protein concentrations across and within human tissues”. en. In: *NAR Genom Bioinform* 2.1 (Oct. 2019), lqz010.

[15] Gali Arad and Tamar Geiger. “Functional Impact of Protein-RNA Variation in Clinical Cancer Analyses”. en. In: *Mol Cell Proteomics* 22.7 (June 2023), p. 100587.

[16] Christian Brion, Sheila M Lutz, and Frank Wolfgang Albert. “Simultaneous quantification of mRNA and protein in single cells reveals post-transcriptional effects of genetic variation”. en. In: *Elife* 9 (Nov. 2020).

[17] Shinya Tasaki et al. “Inferring protein expression changes from mRNA in Alzheimer’s dementia using deep neural networks”. In: *Nature Communications* 13.1 (Feb. 2022), p. 655. ISSN: 2041-1723. DOI: 10.1038/s41467-022-28280-1. URL: https://doi.org/10.1038/s41467-022-28280-1.

[18] Mitra Parissa Barzine et al. “Using Deep Learning to Extrapolate Protein Expression Measurements”. en. In: *Proteomics* 20.21-22 (Oct. 2020), e2000009.

[19] Mahmoud Ghandi et al. “Next-generation characterization of the Cancer Cell Line Encyclopedia”. In: *Nature* 569.7757 (May 2019), pp. 503–508. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1186-3. URL: https://doi.org/10.1038/s41586-019-1186-3.

[20] David P. Nusinow et al. “Quantitative Proteomics of the Cancer Cell Line Encyclopedia”. In: *Cell* 180.2 (2020), 387–402.e16. ISSN: 0092-8674. DOI: https://doi.org/10.1016/j.cell.2019.12.023. URL: https://www.sciencedirect.com/science/article/pii/S0092867419313856.

[21] Robert E. Schapire. “A brief introduction to boosting”. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 1401–1406.

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[23] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: `1706.03762`. URL: `http://arxiv.org/abs/1706.03762`.

[24] Haotian Cui et al. “scGPT: Towards Building a Foundation Model for Single-Cell Multi-omics Using Generative AI”. In: *bioRxiv* (2023). DOI: `10.1101/2023.04.30.538439`. eprint: `https://www.biorxiv.org/content/early/2023/07/02/2023.04.30.538439.full.pdf`. URL: `https://www.biorxiv.org/content/early/2023/07/02/2023.04.30.538439`.

[25] Andrés García-Silva and Jose Manuel Gomez-Perez. “Classifying Scientific Publications with BERT – Is Self-Attention a Feature Selection Method?” In: (Jan. 2021).