MULTI-PLATFORM GENOMIC DATA FUSION WITH INTEGRATIVE DEEP LEARNING

MULTI-PLATFORM GENOMIC DATA FUSION WITH INTEGRATIVE DEEP LEARNING

By Olatunji Oni, B.Sc.

A Thesis Submitted to the School of Graduate Studies in the Partial Fulfillment of the Requirements for the Degree Master of Science

McMaster University
Master of Science (2019)
Hamilton, Ontario (School of Computational Science and Engineering)

TITLE: Multi-Platform Genomic Data Fusion with Integrative Deep Learning
AUTHOR: Olatunji Oni (McMaster University)
SUPERVISOR: Dr. Sanzheng Qiao
NUMBER OF PAGES: xi, 63

Abstract

The abundance of next-generation sequencing (NGS) data has encouraged the adoption of machine learning methods to aid in the diagnosis and treatment of human disease. In particular, the last decade has shown the extensive use of predictive analytics in cancer research due to the prevalence of rich cellular descriptions of genetic and transcriptomic profiles of cancer cells. Despite the availability of wide-ranging forms of genomic data, few predictive models are designed to leverage multidimensional data sources. In this paper, we introduce a deep learning approach using neural network based information fusion to facilitate the integration of multi-platform genomic data, and the prediction of cancer cell sub-class. We propose the dGMU (deep gated multimodal unit), a series of multiplicative gates that can learn intermediate representations between multi-platform genomic data and improve cancer cell stratification. We also provide a framework for interpretable dimensionality reduction and assess several methods that visualize and explain the decisions of the underlying model. Experimental results on nine cancer types and four forms of NGS data (copy number variation, simple nucleotide variation, RNA expression, and miRNA expression) showed that the dGMU model improved the classification agreement of unimodal approaches and outperformed other fusion strategies in class accuracy. The results indicate that deep learning architectures based on multiplicative gates have the potential to expedite representation learning and knowledge integration in the study of cancer pathogenesis.

Acknowledgments

I would first like to thank Dr. Sanzheng Qiao for his supervision and guidance throughout each stage of the project. As my research advisor, his expertise was invaluable in ensuring the technical proficiency of my work. I am very grateful to all those with whom I have had the pleasure of working with at McMaster University. This includes my fellow graduate students, and the members of my thesis committee, whose personal and professional counsel stayed with me for the duration of the project. Lastly, I must express my profound gratitude to my family and friends for the unwavering encouragement and continual support.

Contents

List of Tables

List of Figures

Symbol	Description
X	A matrix.
$\hat{y}$	The model approximation from a neural network.
θ	The variable weights in a neural network.
b	The bias terms in a neural network.
Θ	The set of all parameters in a neural network.
$J(\cdot)$	A neural network cost function.
$\psi(\cdot)$	An activation function.
α	An optimization learning rate.
$\tilde{x}$	The noised version of input data x.
x'	The reconstruction of input data x from an autoencoder.
$g_{\theta}(\cdot)$	The encoding function of an autoencoder.
$f_{\theta^T}(\cdot)$	The decoding function of an autoencoder.
$\mathbb{Z}_{\geq 0}$	The set of all non-negative integers.

Table 1: Summary of Mathematical Notations

Chapter 1

Introduction

Recent advances in biotechnology have enabled a multidimensional approach for exploring human disease. New high-throughput technologies can quantify and characterize the biomolecules that define the architecture, behaviour, and dynamics of a biological system. Research in the last decade has introduced a multifaceted exploration of cancer biology at an unprecedented scale [24]. Cancer research projects are using next-generation sequencing (NGS) data to characterize the genome, epigenome, and transcriptome to capture the complexity and phenotypic heterogeneity of cancer cells [11].

Despite the availability of wide-ranging forms of genomic data, few predictive models are designed to leverage multidimensional data sources. The unilateral approach has been effective in the identification of single-cell expression profiles, epigenomic states, and previously uncharacterized sequence variants [21, 28]. However, individual data types are unable to capture the systems-wide view required to understand the complexity of cancer pathology. This has resulted in the need for integrative methods designed to combine multi-platform data [50]. In the most recent biomedical literature, there have been several attempts to utilize multi-platform biomedical data, however, these existing methods rely on linearly fusing data sources, resulting in high data sparsity and an inability to capture both intra-modality and cross-modality correlations [29, 31, 39]. In this work, we introduce a deep learning approach based on the Gated Multimodal Unit (GMU) to facilitate the integration of multi-platform genomic data and predict cancer cell tissue sub-class. GMUs are neural networks that utilize multiplicative gates to learn intermediate representations between diverse sources of information. Here we show that a series of deeply connected GMUs can be used to extract a biologically relevant latent space from multi-platform genomic data.

The presence of diverse forms of genomic data has encouraged the use of machine learning methods in clinical decision support. Clinicians are increasingly adopting coupled frameworks of NGS and predictive models to support cancer diagnosis and patient stratification. Rapid developments in machine learning are enabling opportunities for improved clinical decision making in the healthcare industry, however, several key challenges hinder its utility by clinicians and researchers. The application of deep learning for medical predictions often results in a hindered ability to interpret the decision made by the classifier. Healthcare professionals require informative tools that can explain their predictions. Domain experts need to ensure a level of trust in predictive models by evaluating the usefulness, reliability, and internal logic of the system. In this work, we assess several methods that leverage interpretable dimensionality reduction and model agnostic explanations to help understand the behaviour of the underlying model.

The remaining content of this thesis is organized as follows. Chapter 2 provides a general background of machine learning, and in particular reviews the basic concepts of deep learning, autoencoders, and model generalization. Chapter 3 describes the most recent academic literature pertaining to neural network based dimensionality reduction and information fusion, and model interpretation in biomedical research. Chapter 4 describes the main components and architecture of our approach, the dGMU (deep gated multimodal unit) for integrative information fusion and representation learning. In chapter 5, we provide a description of the methodologies utilized to evaluate our approach, and in chapter 6 we report the results obtained across a variety of experiments. Lastly, the thesis is concluded in chapter 7 with a summary of the experimental results and a discussion of potential future research directions.

Chapter 2

Background

The background material in this chapter provides an overview of the theoretical concepts and fundamental technologies on which our deep learning approach is built. A few main concepts must be understood to appreciate the role of deep learning in data integration and predictive models. The first, and most fundamental to the discussion is an introduction to Artificial Neural Networks (ANNs), and their extension into the field of deep learning and information fusion.

2.1 Artificial Neural Networks

Introduction

Early attempts in designing computational systems to exhibit intelligent behaviour were conducted through formal rule-based programs referred to as Expert Systems [45]. In these systems, inference engines were used to apply a knowledge base of curated rules to deduce new rules and make predictions on future behaviour. It soon became evident that the sheer number of hard-coded rules required to simulate predictive behaviour was several orders of magnitude higher than what was capable to write or store for even moderately complicated tasks. This resulted in a shift from rigid and deductive systems towards inductive systems that learn to extract information from observed data.

Naturally, complex phenomena generally have dynamic correlations and nonlinear relationships. Accordingly, various techniques were developed to elucidate nonlinear relationships and emulate intelligent behaviour from observed training data. For instance, in statistics, parametric models were developed so data could be described using finite-dimensional classes of nonlinear functions such as exponential, polynomial or power functions. However, these kinds of finite-dimensional models are limited in application to data adequately described by a bounded array of parametric functions. An additional approach, kernel methods, are based on non-linear projections of observed data into a latent space that can measure the distance between observations. New regression values or classifications are then predicted based on distances in the latent space. Unfortunately, the construction of the kernel matrix in kernel-based methods becomes oppressive as the size of the data set increases, rendering these algorithms unfeasible for large data sets. Another class of models, ANNs, were discovered to elucidate many of the concerns of prior nonlinear models. ANNs thrive with large data sets and can learn to approximate any nonlinear function.

ANNs are loosely based on the function of biological neurons. In biology, neurons facilitate the flow of nerve impulses through networks that process and transmit information. The processing of inputs and outputs in neural structures allows biological neurons to adaptively learn and react based on previously observed patterns. ANNs implement this idea mathematically allowing them to act as nonlinear universal approximators. They perform this task by aggregating a cascade of simple nonlinear computations to form robust and complex nonlinear functions. Recently, ANNs have been particularly successful at solving large fundamental problems in natural language processing, voice recognition, and image classification [8, 10, 20].

2.1.1 The Multilayer Perceptron

The simplest form of a multilayer perceptron (MLP) is a feedforward and fully connected network with a single hidden layer, as shown in Fig. 2.1a. A supervised learning problem for an MLP involves approximating a nonlinear function $f(x)$. The problem is considered supervised because each training example is associated with a label y. Consider data set $X \in \mathbb{R}^{m \times n}$ composed of m observations on n variables, where the ith observation is $x_i = (x_1, x_2, ..., x_n)$. The computation in a single node of a neural network is simply the linear combination of vector x_i with respective weights $\theta_i = (\theta_1, \theta_2, ..., \theta_n)$ and an added bias term b:

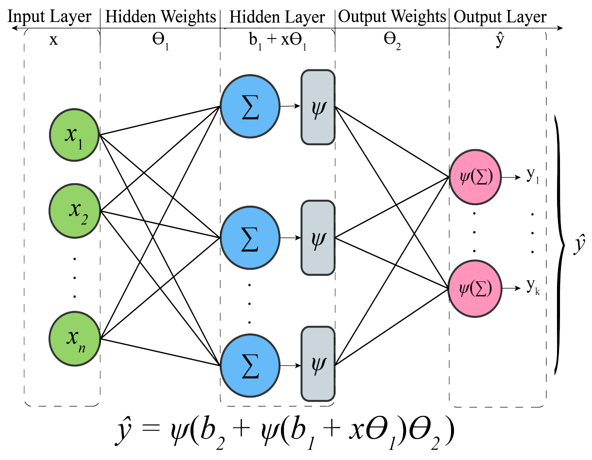
$$z = b + \sum_{i=1}^{n} x_i \theta_i \tag{2.1}$$

The linear combination is then transformed by applying a nonlinear activation function $\psi(z)$, which maps the weighted inputs to the scalar output of the node. This simple computation within a single node is the basic building block of an MLP. MLPs contain at least two layers of these processing nodes (hidden and output layers), along with an input layer for training data. The hidden and output layers contain parallel processing nodes, each receiving input from the previous layer. This cascade of information from the input layer towards the output layer is why MLPs are classically defined as feed forward neural networks.

A nonlinear activation function ψ allows the compositional output function $\hat{y}$ to map inputs non-linearly to outputs. Deeper MLPs will contain many more hidden layers then displayed in Fig. 2.1a and a neural network with n layers can be defined recursively as:

$$\hat{y} = \psi(\theta_n(\psi(\theta_{n-1}(\psi(\theta_{n-2}(\ldots\psi(\theta_1 x + b_1)\ldots)) + b_{n-2})) + b_{n-1}) + b_n), \tag{2.2}$$

where the structure of $\hat{y}$ depends on the desired task (e.g nonlinear regression or classifi-

(a)


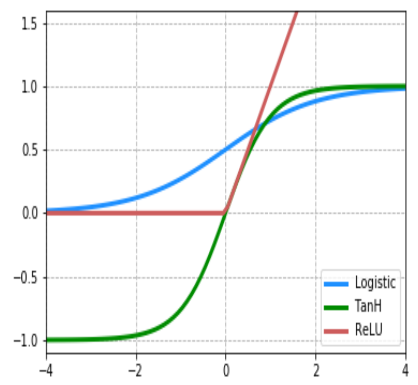
(b)


Figure 2.1: a) Structure of a feed forward MLP with three layers. b) The output for Logistic, Tanh and ReLU activation functions for input value range [-4,4].

cation). For regression problems, $\hat{y}$ is a real value, and for classification problems, $\hat{y}$ is a k dimensional vector of real values. So although it is generally advantageous for hidden units to have nonlinear activation functions, the choice of activation functions for output layers will largely depend on the desired task. For nonlinear regression, a linear activation function is generally adequate. However, in classification problems, it can be useful to view the k dimensional output vector $\hat{y} = (y_1, y_2, ..., y_k)$ as providing the probabilities a given input example resides in each respective class. This produces a probability distribution over the k classes, where entries fall between $(0, 1)$, and the sum of the vector $\hat{y}$ is one. This behaviour can be accomplished by the softmax function:

$$\text{softmax}(z) = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}} \quad \text{for} \quad j = \{1 \dots k\}, \tag{2.3}$$

where softmax(z) represents the categorical distribution of an arbitrary k dimensional vector z. Furthermore, with a formally defined arbitrary output function $\hat{y}$, the next step requires the artificial neural network to learn weights and biases in order to produce desired outputs given input data.

The Cost Function

Training an artificial neural network relies on determining the network parameters that minimize the error between outputs $\hat{y}$ and true values y. This entails producing a model function, that given inputs, can produce the output to the closest degree. In machine learning, the notion of a good model is explicitly defined using some cost function $J(\hat{y}, y; \Theta)$, where $\Theta = \{\theta, b\}$ is the set of all network weights and biases. The cost function keeps track of the model's prediction error. Finding better models equates to finding better network parameters that minimize the cost function. For nonlinear regression, a commonly used measure of cost is simply the mean squared error between the output of the neural network and the true values:

$$J(\hat{y}, y; \Theta) = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{2.4}$$

For classification problems, it is often beneficial to represent categorical true labels y as binary vectors. This is referred to as one hot encoding, where the ith position of class i is one, and every other term is zero. For example, in a five-class problem, a class of three would be coded as $y = [0, 0, 1, 0, 0]$. In this way, the error must be calculated for each potential class k over all examples in the sample set. Accordingly, the most commonly used cost function for classification problems is the cross-entropy loss:

$$J(\hat{y}, y; \Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{k} [y_{i,j} \log(\hat{y}_{i,j}) + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j})], \tag{2.5}$$

where the true labels and model predictions are defined as $y_{i,j} \in \{0, 1\}$ and $\hat{y}_{i,j} \in (0, 1)$, respectively. In the computation of cross-entropy loss, k error terms are generated for every training example. The cross-entropy loss represents the log probability of classes given the model - that is, maximizing the likely hood of a training example belonging to a specific class is equivalent to minimizing the cross-entropy loss between $\hat{y}$ and y.

The Optimization Algorithm

The cost, $J(\hat{y}, y; \Theta)$, is conveniently a function of the training examples and model parameters Θ. In order to effectively minimize the cost function, it is useful to observe how the cost changes with respect to weights θ. Formally, this is expressed with the partial derivative $\frac{\partial J}{\partial \theta}$. With this expression, we can search for weights in the direction that cost J decreases. This technique, called gradient descent, minimize cost by iteratively updating θ in the opposite direction of the gradient:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\hat{y}, y; \Theta) \quad \text{for} \quad j = \{1, \ldots, n\}, \tag{2.6}$$

where α, the learning rate, controls the size of steps made in the direction of the negative gradient. Furthermore, in order to compute the gradients of the cost with respect to model weights, an algorithm called backpropagation is commonly used. The cost function, as one large nested composite function, contains all of the computations in the neural network. With this, the backpropagation algorithm cleverly applies the chain rule of calculus to recursively compute the gradients of each weight, as shown in Fig. 2.2.

Fig. 2.2 illustrates the mechanism of a feedforward multilayer perceptron with an input layer X_{mn}, a hidden layer $a^{(2)}_{mj}$, and an output layer $\hat{y}_{mk}$. The input layer, considered the first activation $a^{(1)}_{mn}$, is used to form the first linear combination $z^{(1)}_{mf}$. In the hidden layer, a nonlinear activation function $\psi(z^{(1)}_{mj})$ is used to produce the hidden activations $a^{(2)}_{mj}$. In deeper networks, the preceding layers outputs are continually utilized in a series of linear combinations, following by nonlinear activation functions until the output layer is reached. At this point, the model predictions are compared to the true values using cost function $J(\hat{y}_{mk})$. In order to train the neural network, or minimize the cost function, the gradient of the cost function is computed to determine the direction in the parameter space required to lower network error. The gradient of the cost function is computed efficiently propagating

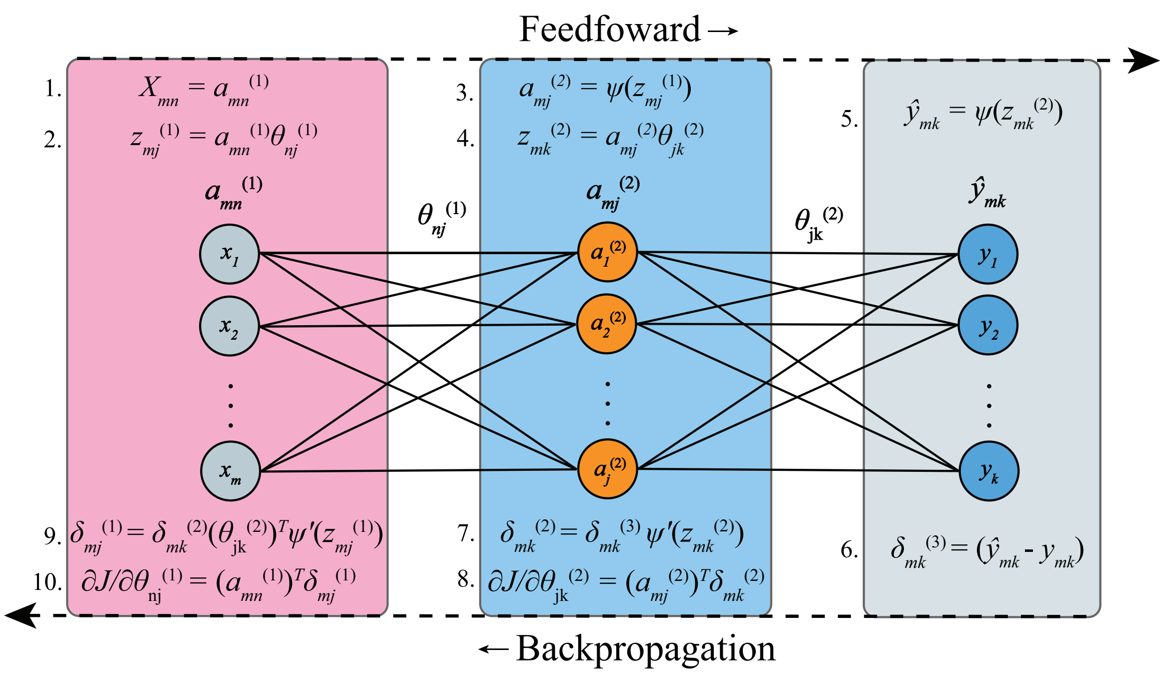

Feedfoward →
1. $X_{mn} = a_{mn}^{(1)}$
2. $z_{mj}^{(1)} = a_{mn}^{(1)}\theta_{nj}^{(1)}$
$a_{mn}^{(1)}$
x_1
x_2
x_m
$\theta_{nj}^{(1)}$
3. $a_{mj}^{(2)} = \psi(z_{mj}^{(1)})$
4. $z_{mk}^{(2)} = a_{mj}^{(2)}\theta_{jk}^{(2)}$
$a_{mj}^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_j^{(2)}$
$\theta_{jk}^{(2)}$
5. $\hat{y}_{mk} = \psi(z_{mk}^{(2)})$
$\hat{y}_{mk}$
y_1
y_2
y_k
9. $\delta_{mj}^{(1)} = \delta_{mk}^{(2)}(\theta_{jk}^{(2)})^T\psi'(z_{mj}^{(1)})$
10. $\partial J/\partial\theta_{nj}^{(1)} = (a_{mn}^{(1)})^T\delta_{mj}^{(1)}$
7. $\delta_{mk}^{(2)} = \delta_{mk}^{(3)}\psi'(z_{mk}^{(2)})$
8. $\partial J/\partial\theta_{jk}^{(2)} = (a_{mj}^{(2)})^T\delta_{mk}^{(2)}$
6. $\delta_{mk}^{(3)} = (\hat{y}_{mk} - y_{mk})$
← Backpropagation

Figure 2.2: Feedforward and Backpropagation Schematic.

the error back through the network using the derivate chain rule. In Fig. 2.2, backpropagation begins with the output *backpropagation-error* $\delta_{mk}^{(3)} = (\hat{y_{mk}} - y_{mk})$ shown in step 10. *Backpropagation-error* is the layer specific approximation error, and the hidden and input layer *backpropagation-errors* are represented as $\delta_{mk}^{(2)}$ and $\delta_{mf}^{(1)}$, respectively. In Eq. 2.6, we see that the partial derivative of the cost function with respect to each weight is required in order to compute a gradient update. In this example, the weights of the model are spread across two matrices, $\theta_{nf}^{(1)}$ and $\theta_{jk}^{(2)}$. Accordingly, we will calculate gradient matrices of the same dimension, $\partial J/\partial\theta_{nf}^{(1)}$ and $\partial J/\partial\theta_{jk}^{(2)}$, to store the gradient of the cost function with respect to each weight in the model. Starting with $\partial J/\partial\theta_{jk}^{(2)}$, we get the following expression using squared error as the cost function:

$$\frac{\partial J}{\partial\theta_{jk}^{(2)}} = \frac{\partial\frac{1}{2}\sum(y_{mk} - \hat{y}_{mk})^2}{\partial\theta_{jk}^{(2)}}$$

If we remove the summation from this expression we can compute the gradient for a single example. Evaluating the resultant derivative produces:

$$\frac{\partial J}{\partial \theta^{(2)}_{jk}} = -(y_{mk} - \hat{y}_{mk})\frac{\partial \hat{y}_{mk}}{\partial \theta^{(2)}_{jk}}$$

From step 5 in Fig. 2.2, we see that $\hat{y}_{mk} = \psi(z^{(2)}_{mk})$. Accordingly, we can now apply the chain rule to further evaluate the expression:

$$\frac{\partial J}{\partial \theta^{(2)}_{jk}} = -(y_{mk} - \hat{y}_{mk})\frac{\partial \hat{y}_{mk}}{\partial z^{(2)}_{mk}}\frac{\partial z^{(2)}_{mk}}{\partial \theta^{(2)}_{jk}}$$

We can now replace $\partial \hat{y}_{mk}/\partial z^{(2)}_{mk}$ with $z^{(2)}_{mk}$ evaluated by the derivative of the activation function, $\psi'(z^{(2)}_{mk})$. As well, from step 4 in Fig. 2.2, we see that $z_{mk} = a^{(2)}_{mj}\theta^{(2)}_{jk}$. The derivative of this linear relationship is simply the slope $a^{(2)}_{mj}$. We can now simplify our expression:

$$\frac{\partial J}{\partial \theta^{(2)}_{jk}} = -(y_{mk} - \hat{y}_{mk})\psi'(z^{(2)}_{mk})a^{(2)}_{mj}$$

The hidden layer *backpropagation-error* is given by $\delta^{(2)}_{mk} = -(y_{mk} - \hat{y}_{mk})\psi'(z^{(2)}_{mk})$. Notice that if we transpose the activity matrix $a^{(2)}_{mj}$, we can perform matrix multiplication with $a^{(2)}_{mj}$ and $\delta^{(2)}_{mk}$ to sum across all examples. This effectively takes care of the earlier omission of the summation, so that the gradient can be calculated efficiently as shown in step 8 in Fig. 2.2. The simplified expressions becomes:

$$\frac{\partial J}{\partial \theta^{(2)}_{jk}} = (a^{(2)}_{mj})^T \delta^{(2)}_{mk}$$

The derivation for $\partial J/\partial \theta^{(1)}_{nj}$ is similar to the above:

$$
\begin{aligned}
\frac{\partial J}{\partial \theta^{(1)}_{nj}} &= \delta^{(2)}_{mk} \frac{\partial z^{(2)}_{mk}}{\partial \theta^{(1)}_{nj}} \\
&= \delta^{(2)}_{mk} \frac{\partial z^{(2)}_{mk}}{\partial a^{(2)}_{mj}} \frac{\partial a^{(2)}_{mj}}{\partial \theta^{(1)}_{nj}} \\
&= \delta^{(2)}_{mk} (\theta^{(2)}_{jk})^T \frac{\partial a^{(2)}_{mj}}{\partial \theta^{(1)}_{nj}} \\
&= \delta^{(2)}_{mk} (\theta^{(2)}_{jk})^T \frac{\partial a^{(2)}_{mj}}{\partial z^{(1)}_{mf}} \frac{\partial z^{(1)}_{mf}}{\partial \theta^{(1)}_{nj}} \\
&= \delta^{(2)}_{mk} (\theta^{(2)}_{jk})^T \psi'(z^{(1)}_{mf}) \frac{\partial z^{(1)}_{mf}}{\partial \theta^{(1)}_{nj}} \\
&= (a^{(1)}_{mn})^T \delta^{(2)}_{mk} (\theta^{(2)}_{jk})^T \psi'(z^{(1)}_{mf}) \\
&= (a^{(1)}_{mn})^T \delta^{(1)}_{mj}
\end{aligned}
$$

The simplified expressions for both gradients are closely related. We see that the gradients depend on the layer activity and the *backpropagation-error*. In turn, the calculations for *backpropagation-error* are dependent on the error terms in the next layer. Thus, the computation of error proceeds backwards from the output layer towards the input layer. This is where backpropagation, or backwards propagation of errors, gets its name. The error δ^{ℓ} at layer ℓ is dependent on the errors $\delta^{\ell+1}$ at the next layer $\ell + 1$.

2.2 Deep Learning

2.2.1 Deep Neural Networks

Deep learning is a subset of machine learning based on artificial neural networks. Deep learning employs architectures such as deep neural networks (DNN) that typically have multiple layers between the input and the output layers. DNNs learn hierarchical representations of the data through the layers of the network in order to map complex relationships between the input and the desired outputs. Recently, these networks have made state-of-the-art breakthroughs in fields such as natural language processing, computer vision, and speech recognition.

DNNs are especially useful for learning intermediate representations of input data. The representations are formed with the composition of non-linear transformations through multiple layers in the network. The output of each layer forms a hierarchy of distributed representations that have an increasing level of abstraction as an input flows through the network. The performance of a DNN depends largely on the representations it learns to output. This is because the distribution of the input data is generated by a combination of underlying features and a model that learns to compactly represent the features can generalize for more variation without requiring as many examples. In cases where the transformation results in data compression the learning task equates to developing output representations that map to the naturally occurring input data distribution but in a low dimensional manifold.

2.2.2 Autoencoders

Autoencoders represent a family of unsupervised artificial neural networks that learn latent data codings for input data. Here, training examples are utilized without labels, and autoencoders are trained to generate outputs identical to the inputs. The role of an autoencoder is split into two tasks, encoding and decoding. The process of encoding maps the input to lower dimension features, and decoding maps the encoded data back into the original space. The mapping of an input layer produced by function, $g_\theta(\cdot)$, can be expressed as:

$$z = g_\theta(x) = \psi(\theta x + b) \tag{2.7}$$

This latent layer, of potentially reduced dimensionality, can then be decoded back to the original space with function $f_{\theta^T}(\cdot)$:

$$x' = f_{\theta^T}(z) = f_{\theta^T}(g_\theta(x)) = (\theta^T z + b), \tag{2.8}$$

where x' is the reconstructed input data, and θ^T is the transpose of weights used for encoding. Furthermore, to train the model, the reconstructed output is compared to the original input using the mean squared error to calculate reconstruction error:

$$J(x_i', x_i; \Theta) = \frac{1}{2m}\sum_{i=1}^{m} \|x_i - x_i'\|^2, \tag{2.9}$$

where $\Theta = \{\theta, b_\theta, \theta^T, b_{\theta^T}\}$ is the set of network parameters used for encoding and decoding operations.

Denoising Autoencoders

Training an autoencoder with partially corrupted data while comparing the reconstruction to the original input is a commonly used technique to increase the robustness of an autoencoder. This modification produces a variant of the basic autoencoder called a denoising autoencoder. By adding corruption to the input data, $\tilde{x}_i \sim \mu_D(\tilde{x}_i|x_i)$, the autoencoder must learn parameters that can overcome stochastic noise. The operation μ_D defines the form of noise used to corrupt the input data in order to increase robustness, and the reconstruction error is calculated using cost $J(\tilde{x}_i', x_i; \Theta)$.

Furthermore, deeper frameworks of denoising autoencoders, called stacked denoising autoencoders (SDAEs), are employed to increase the number of latent abstractions. SDAEs are composed of multiple layers of incrementally stacked denoising autoencoders that are trained one layer at a time [43]. In this way, once the kth hidden layer is trained, layer $k + 1$ can be trained using the kth layer as the input data.

2.3 Improving Machine Learning Models

2.3.1 Regularization

A key aspect of producing a good machine learning model is to avoid overfitting the training data. The problem of overfitting arises when the model variables and parameters aggressively capture both the underlying pattern and stochastic noise of the training data. The model is learning information that does not represent the true properties of the underlying relationship when it captures too much random noise. In this case, the model will perform well on the test set, but will likely have a poor prediction and generalization power on examples it has never seen before.

This problem gets worse as the model complexity increases, and the model variables and parameters have the flexibility to increasingly capture background noise. A way to avoid overfitting is by using cross-validation. Cross-validation is beneficial because the model variables and parameters are determined by estimating the error of the model over a subset of the data the model has not observed. In k-fold cross-validation, this is performed by dividing the data into k subsets. Then each of the k subsets is used as a validation set, and the other $k - 1$ subsets are combined as the training set. Accordingly, the error estimation gets averaged over the k trials to compute the total effectiveness of the model.

Furthermore, another commonly used way to avoid overfitting is through model regularization. Regularization is a form of regression that applies a constraint to the model complexity. Ridge regression involves penalizing the cost function during the training procedure by adding a multiple of the squared magnitude of the model coefficients. Accordingly, ridge regression employs weighted L2 regularization producing the following loss equation:

$$\text{Loss} = J(\hat{y}, y; \Theta) + \frac{\lambda}{2m} \sum_{j=1}^{m} \sum_{k=1}^{p} \Theta_{j,k}^2 \quad , \tag{2.10}$$

where m is the number of training examples, p is the number of model weights, and λ is the regularization parameter. Here, λ decides how much the model flexibility should be penalized. When $\lambda = 0$, the penalty term is ignored, and the trained model is susceptible to overfitting. However, as $\lambda \to \infty$, the model weights are increasingly penalized and will approach zero. This will result in under-fitting, where the model loses the ability to fit the data entirely. Therefore, selecting a good regularization parameter is essential for optimizing the performance of the machine learning model.

2.3.2 Model Averaging

The dropout technique is a relatively simple way to regularize a neural network using the concept of model averaging. Dropout entails randomly setting a fraction of the neurons in the network to zero during forward propagation. At each training step, a neuron is either kept with a probability of p, or *dropped out* with probability $1 - p$ to produce a network of reduced size. During backpropagation, the weights of dropped out neurons will not be updated. Accordingly, dropout is training a subsample of the whole neural network on every training iteration. In this way, dropout can be seen as an ensemble of randomly sampled models that share parameters. As well, dropout is a useful technique for addressing co-adapting behaviour in machine learning models. Co-adaptation occurs when neurons learn to fix the mistakes of other neurons in a fully connected neural network. This is a problem because co-adapted neurons tend to result in overfitting because they do not generalize well to new data. With dropout, neurons are forced to learn more robust features that are independent of the other neurons.

Furthermore, an activation function called maxout was proposed to leverage the model

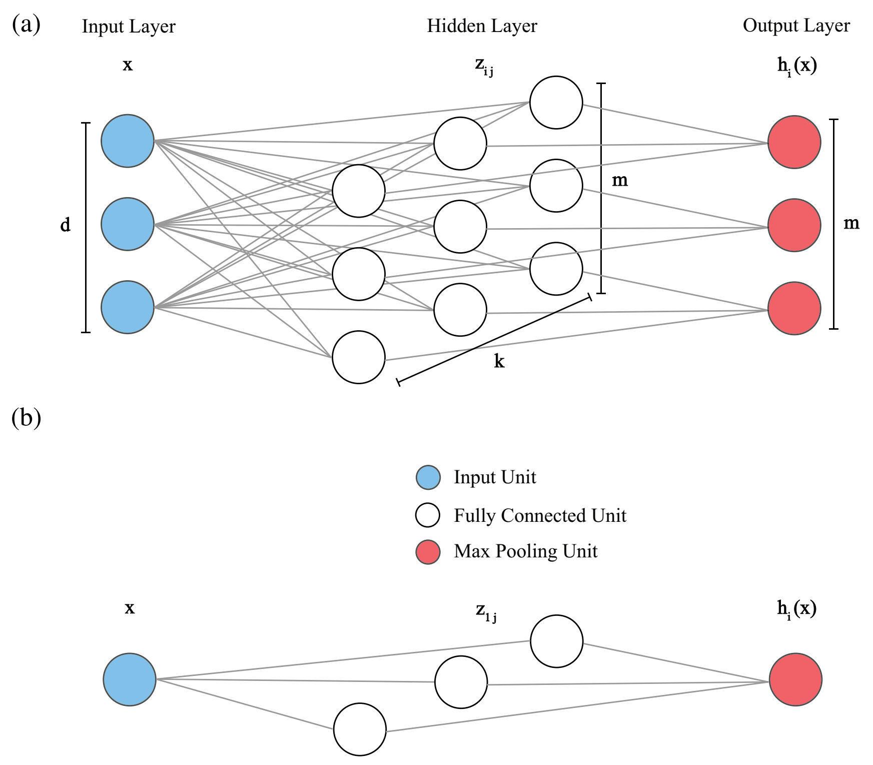


Figure 2.3: a) Structure of a maxout neural network b) Maxout neural network with d = 1, m = 1, and k =3

averaging performed by dropout [17]. The maxout activation function was shown to improve approximate model averaging in deep models over non-linear activations such as the Tanh function [17]. Formally, the maxout activation function is defined as:

$$h_i(x) = \max_{j\in[1,k]} z_{i,j}, \tag{2.11}$$

where $x \in \mathbb{R}^n$ is an input vector, $z_{i,j} = x^T W_{\dots ij} v + b_{ij}$ is output for the j-th linear transformation of the i-th hidden unit, and $W \in \mathbb{R}^{d\times m\times k}$ and $b \in \mathbb{R}^{m\times k}$ are learned parameters. In simple terms, maxout accepts an input of dimensionality d and computes k linear transformations and returns the largest unit for each of the m linear feature extractors. The maxout activation function is applied by using a small differentiable sub-network as shown in Fig.

2.3.

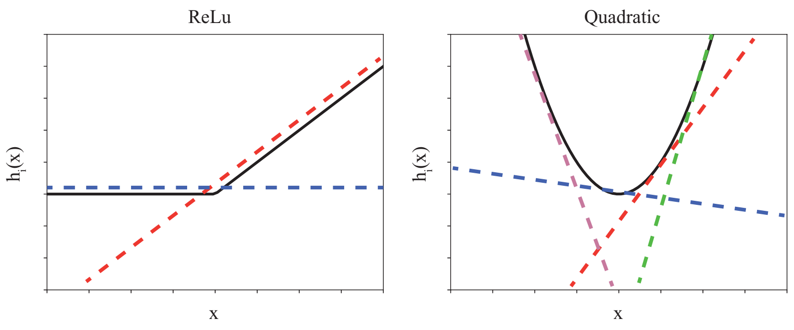


Figure 2.4: Maxout approximation of ReLu and quadratic activation functions.

In Fig. 2.3a, the hidden layer implements the weighted sum of all inputs. In Eq. 2.11, this is represented by $z_{i,j} = x^T W_{...ij} + b_{ij}$. The three-dimensional tensor $W_{...ij}$, represents the weight vector for the unit in row i and column j of the fully connected units. The max pooling units simply take the maximum output from the neurons of each row. From Fig. 2.4, we can see how the maxout activation function can approximate a ReLu and quadratic activation function. In this way, maxout can produce a piecewise linear approximation of an arbitrary convex function.

Chapter 3

Related Work

3.1 Dimensionality Reduction

Multi-platform biomedical datasets present numerous challenges for traditional machine learning and statistical approaches. Biological data are often high-dimensional, noisy and sparse. High-throughput transcriptome sequencing and genome-wide genotyping arrays can produce tens of thousands to millions of features, making the identification of biomarkers a central issue in cancer research [42]. Representation learning for regularized and data-driven feature identification has thus emerged as a critical component of the dimensionality reduction paradigm. Various unsupervised methods have been used for dimensionality reduction and classification of sequencing data. Techniques such as stacked denoising autoencoders (SDAEs) have been used to acquire low dimension non-linear feature sets from breast cancer RNA expression data [40]. Transformative autoencoders have achieved some success, but these techniques result in encodings that lack direct interpretability. Recently, various efforts have produced deeply connected genes using an SDAE designed to extract meaningful subsets of genes that are useful in informing the strategies of clinicians [13, 40].

3.2 Information Fusion

Given the complexity of biomedical systems, various strategies have been used to integrate diverse sources of biological information. Data integration methods generally rely on one of the following two strategies for combining information from multiple modalities. Feature fusion (early fusion) involves the concatenation of data sources into a feature-based table for the input into a classifier or predictive model. In decision fusion (late fusion) an independent model is designed for each modality in order to derive a classification or prediction. The outputs from the independent models are then combined to produce a consensus, typically derived by model averaging, taking a majority vote, learning gate parameters, or using Bayesian frameworks [48].

3.2.1 Mixture of Experts

A commonly used technique in neural network based information fusion involves the use of probabilistic gating functions to fuse the results of multiple subnetworks. In machine learning, this approach is referred to as mixture of experts (MoE). This method employs the use of individualized expert modules that are tailored for subsets of the training data [22]. The gating network learns to determine how the experts are used for each input example. The learning scheme thus consists of learning the parameters for each expert, and learning the parameters of the gate function. MoE models leverage the strategy of divide and conquer by dividing one large complex problem into simpler problems whose solutions can be combined [22]. As well, these models are of interest due to their decreased model complexity, wide applicability, and advantages of faster deep learning. Recently, several [14, 19, 29] have proposed MoE variations for general information fusion and biomedical classification. In the biomedical domain, these more recent efforts have involved processing sequencing and clinical data in individual deep learning modalities and using a gating network to weight the contribution of an expert for a given example. In these applications, the MoE formulation combines a set of expert classifiers, $f_1^E, \ldots, f_C^E$, using a gating function g_c that returns a generalized linear function activated by a generic softmax function:

$$g_c(x_p; \theta_c^G) = \text{softmax}(f_c^G(x_p; \theta_c^G)), \tag{3.1}$$

where $f_c^G(x_p; \theta_c)$ is a gating function with input x_p and unknown parameters θ_c for the c-th expert. The final output for MoE is the weighted sum of predictions by the experts:

$$f(\hat{y}|x; \Theta) = \sum_{c=1}^{C} g_c(x; \theta_c^G) f_c^E(y|x; \theta_c^E), \tag{3.2}$$

where $f_c^E(y|x; \theta_c^E)$ is the output prediction of the c-th expert. The general structure of an MoE network is shown in Fig. 3.1

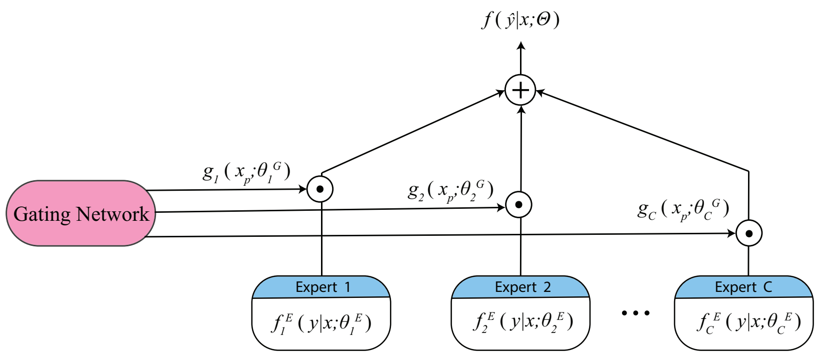


Figure 3.1: Structure of an MoE Network.

3.2.2 Gated Multimodal Units

More recently, several strategies have been successfully employed to increase the accuracy of joint model classifiers. These methods avoid developing individual models for each modality or directly combining data sources, but rather incorporate data integration into the architecture of the classifier or predictive model. In machine learning, gated neural networks have shown superior classification performance over traditional fusion methods [3, 4]. Recent work has shown that neural networks with multiplicative gates can be trained to relate inputs, their fusion, and their classification into a single model. Accordingly, these models are uniquely equipped for learning fusion transformation by governing how each modality contributes to the activation of the network [3]. Fig. 3.2 shows the structure of a generic GMU. The equations that govern the fusion are described in Chapter 4.

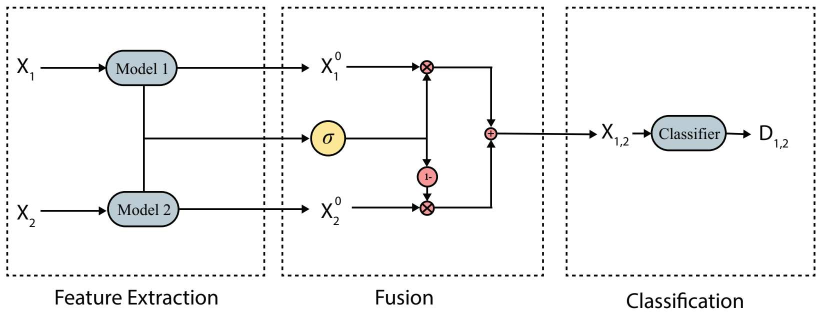


Figure 3.2: Structure of a GMU Network. The input modalities 1 and 2 are represented by X_1 and X_2, respectively. The features from each modality, X_1^0 and X_2^0, are fed into the fusion gates, and $X_{1,2}$ is the fused joint representation. The decision produced by the classifier is represented by $D_{1,2}$.

3.3 Model Interpretation

The ability to interpret the behaviour of a machine learning model can provide valuable insight into the internal logic of the classifier and the structural importance of the features. As the applications of predictive systems are integrated deeper into the industrial and scientific domains, it is becoming increasingly important to be able to explain the basis of their decisions. Certain models benefit from an inherent transparency in interpretation. These techniques provide a direct link to the features used to make a prediction. Unfortunately, transparent models such as decision trees, sparse linear models and rule-based systems have inferior predictive performance compared to more complex model abstractions such as random forest classifiers, support vector machines and deep neural networks. This has encouraged the development of techniques to understand the internal logic of complex predictive models.

Model agnostic interpretation methods are used to explain the behaviour of models where the internal logic of the system is not directly available for inspection. Model agnostic methods are flexible in that they can derive explanations from any underlying model. A widely utilized technique to perform model agnostic interpretation employs the use of a global surrogate model. Global surrogates approximate the behaviour of a complex model by using an interpretable model [12]. Interpretable models, such as generalized regression models or decision trees, are trained to approximate the predictions of an underlying model, and global explanations are derived from analyzing the surrogate. Another technique used is permutation feature sensitivity analysis. These methods employ permuting the input and observing the variation to the model output. Local sensitivity analysis allows the determination of the specific output variance caused by permuting the elements of the input for a training example. Recently, combinations of these approaches were developed to extend the utility of surrogate models with sensitivity analysis, producing an algorithm referred to as local interpretable model-agnostic explanations (LIME) [36]. LIME generates interpretable explanations by approximating the prediction of any classifier locally for a given training example. Local explanations of the underlying model are captured by training an interpretable model on perturbations of the input data. LIME generates a sample set of perturbed examples in the neighborhood of the local instance and uses an interpretable model to draw a decision boundary. Explanations are derived from analyzing the parameters of the decision boundary. Formally, a local surrogate model g is defined through the following expression:

$$\min_{g \in G} J(f, g, \mu_{x_i}) + \lambda(g), \tag{3.3}$$

where $J(f, g, \mu_{x_i})$ is a cost function that measures how closely the surrogate model g approximates the underlying model f while keeping the model complexity $\lambda(g)$ low. The proximity measure μ_{x_i} determines the size of the neighborhood around an example x_i. For a training example, the probability that a classifier maps the input to a class label k is denoted by $y_k = f(x_i)$. Accordingly, LIME works to optimize the expression in (3.3) to interpret why f maps feature vector x_i to a class label k.

In order to produce an explanation, LIME first builds a dataset of perturbed instances $\tilde{x}$ by adding noise Z_i to the mass center of the training data. The noise, $Z_i \sim N(0, \sigma^2)$, is drawn from a zero-mean normal distribution with variance σ^2. The underlying model can then be used to generate a sample set that is weighted by their proximity to the selected instance. The surrogate model can then be trained using a cost function of the mean squared error:

$$J(f, g, \mu_{x_i}) = \sum_i \mu_{x_i}(f(\tilde{x}_i) - g(\tilde{x}_i))^2, \tag{3.4}$$

The learned weights of the trained model g form an n dimensional vector where each weight corresponds to a feature in training vector x_i. The magnitude of the n-th weight, $|w_n|$, defines the importance of that features on the prediction. The feature effect is defined by the polarity of the weight, where $w_n > 0$ or $w_n < 0$ suggests that the feature has a positive or negative influence on the prediction of the given class, respectively. The LIME procedure is illustrated in Figure 3.3.

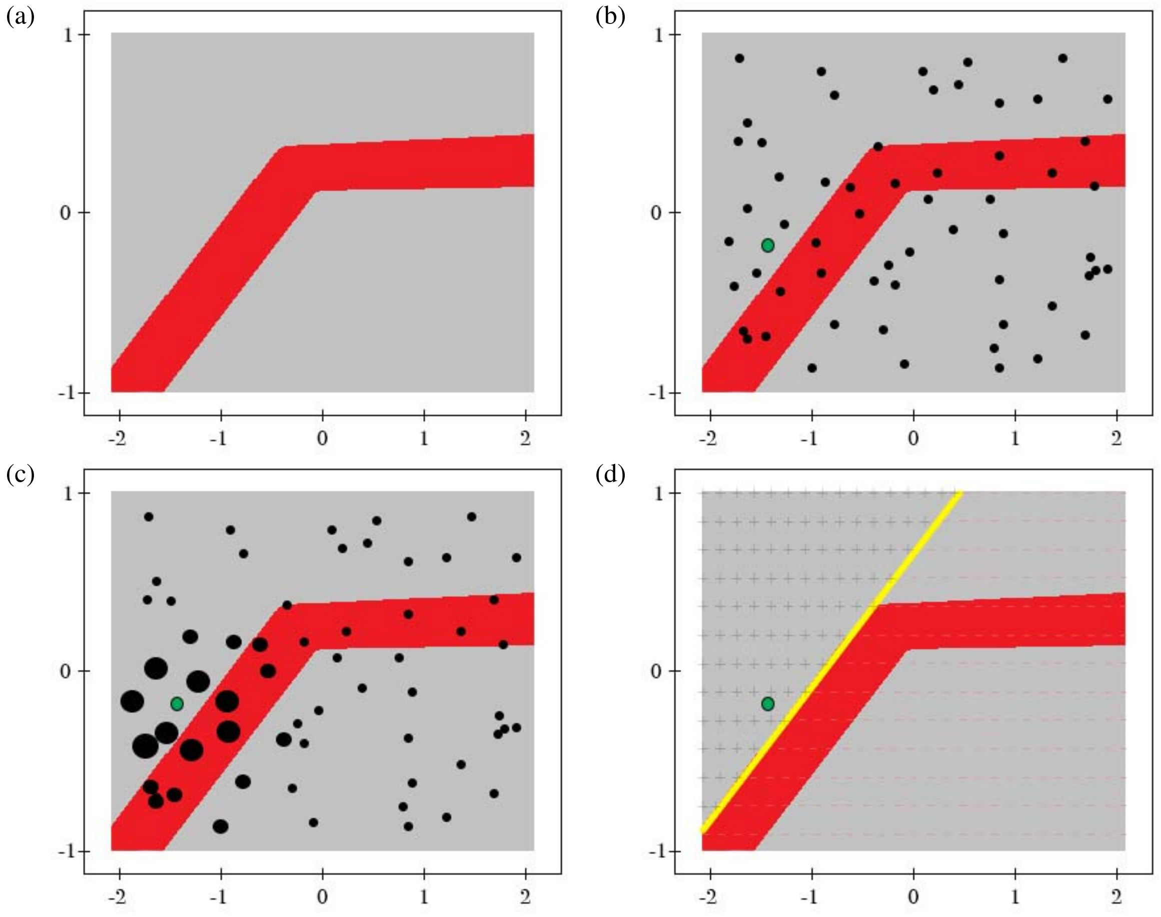


Figure 3.3: Generalized LIME procedure. (a) The red (class 0) and grey (class 1) background represents the decision function of an underlying model with two variables. (b) The instance to be explained is shown as a green dot, and the perturbed instances are shown as black dots. (c) Perturbed instances are given higher weight (shown by size) based on their proximity to the instance being explained. (d) The yellow line is the local explanation for the selected instance.

Chapter 4

Deep Gated Multimodal Units

In this chapter, we propose a novel multimodal fusion approach to integrate information from multiple genomic sources. While most methods have solely relied on data level fusion (early fusion) or decision level fusion (late fusion), our approach utilizes a series of cascading gated multimodal units to deeply connect the integration of data fusion and decision fusion.

4.1 Architecture

The deep gated multimodal unit (dGMU) first contains multiplicative gates designed to construct an intermediate representation of data from multiple modalities. The input modalities along with the intermediate representation are then fed to a decision network that fuses the predictions using an additional gate. These two processes can be subdivided into the function of a representation network and a decision network. This structure is illustrated in Fig. 4.1.

Representation Network

In this network, the input modalities learn a latent representation of the combined input

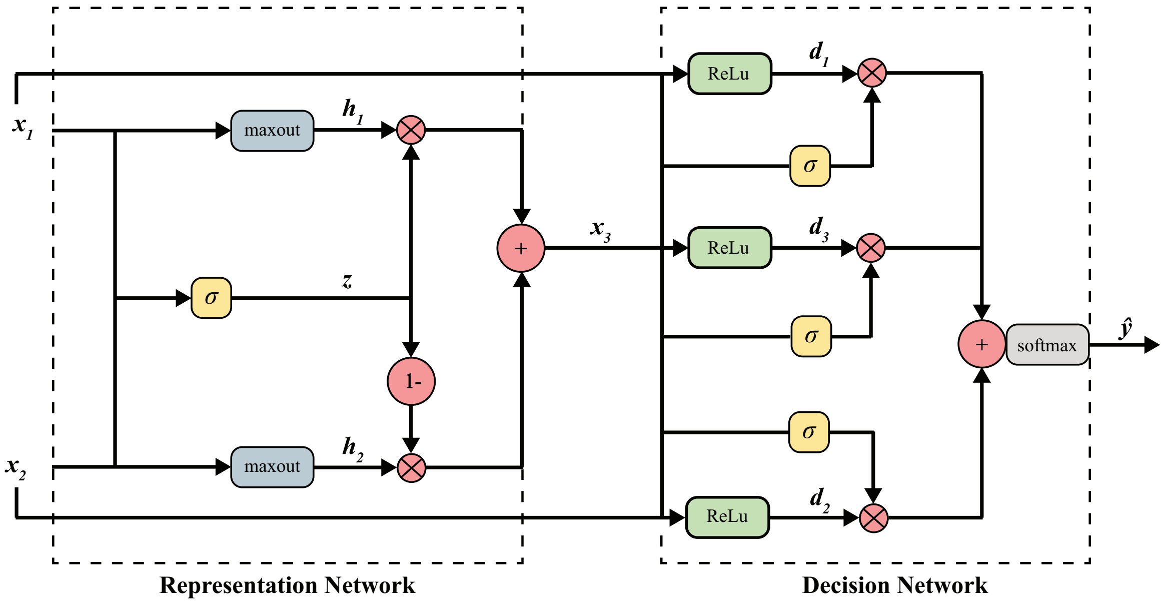


Figure 4.1: Deep Gated Multimodal Unit.

data. Each modality becomes the input for a multilayer perceptron (MLP) with a maxout activation function, maxout(·) [17]. In Fig. 4.1, this produces $h_1 = \text{maxout}(\theta_{h1} \cdot x_1)$ and $h_2 = \text{maxout}(\theta_{h2} \cdot x_2)$, for modalities x_1 and x_2, respectively. Activated by the sigmoid activation function, $\sigma(\cdot)$, the gating neuron, $z = \sigma(\theta_z \cdot [x_1, x_2])$, ties both modalities and controls their contribution to the output of the unit. The output of the representation network is governed by the following equations:

$$
\begin{aligned}
h_1 &= \text{maxout}(\theta_{h1} \cdot x_1) \\
h_2 &= \text{maxout}(\theta_{h2} \cdot x_2) \\
z &= \sigma(\theta_z \cdot [x_1, x_2]) \\
x_3(x_1, x_2; \Theta_R) &= z * h_1 + (1 - z) * h_2, \qquad (4.1)
\end{aligned}
$$

where latent space, $x_3(x_1, x_2; \Theta_R)$, depends on inputs x_1 and x_2, and $\Theta_R = \{\theta_{h1}, \theta_{h2}, \theta_{x3}\}$ is the set of parameters used for encoding the latent space.

Decision Network

This network makes predictions based on all representations and learns to decide how decisions influence the activation of the output unit. Each representation from the representation network becomes the input to an MLP with a rectified linear unit (ReLU) activation function. Here, gating neuron $\sigma(\cdot)$ controls the untied contributions of decision gates d_1, d_2, and d_3. The decision network is governed by the following equation:

$$\hat{y}(x_1, x_2, x_3; \Theta_D) = \sum_{i=1}^{3} \text{ReLu}(\theta_{di} \cdot x_i)\sigma(\theta_{di} \cdot [x_1, x_2, x_3]), \tag{4.2}$$

where the network output, $\hat{y}(x_1, x_2, x_3; \Theta_D)$, depends on inputs x_1, x_2, and x_3, and $\Theta_D = \left\{\theta_{d1}, \theta_{d2}, \theta_{d3}, \theta_{g1}, \theta_{g2}, \theta_{g3}\right\}$ is the set of network parameters used across the untied gates in the decision network.

4.2 Training

The dGMU model parameters were learned with batch stochastic gradient descent with ADAM optimization [27]. The training complexity was reduced by using a supervised pre-training scheme on the decision network [33]. This method is used to initialize the parameters of the decision network to ease the training of the larger model, reducing computation time and increasing model robustness [9]. The complete network was optimized using supervised fine-tuning with the connected sub-networks. During the training process, overfitting was controlled using dropout and L_2 regularization. For classification problems, the global loss is computed using the softmax cross entropy loss as in Equation (2.5). With regularization, this results in the following global loss:

$$\text{Loss} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}[y_{i,j}\log(\hat{y_{i,j}}) + (1 - y_{i,j})\log(1 - \hat{y_{i,j}})] + \frac{\lambda}{2m}\sum_{j=1}^{p}\sum_{k=1}^{q}\Theta_{j,k}^{2} \tag{4.3}$$

4.3 Implementation

The dGMU model was implemented with original code in Tensorflow version 1.11.0 on an Nvidia Tesla K80 GPU. With a high level of parallelization and batch training used during pretraining and finetuning, the model training takes a few minutes, and validation and testing are conducted in a matter of seconds.

The dGMU model benefits from the modularity of gated multimodal units. Accordingly, this architecture can be adapted using varying models for each modality depending on the application. As well, the decision network can be modified to accept inputs from more than two modalities while only resulting in a linear increase in the number of training weights. Furthermore, this model generates a latent space in between the representation network and the decision network. This offers an interesting avenue into investigating the biological significance of the fused latent representation.

Chapter 5

Materials and Methods

5.1 Genomic Data Preprocessing

In this report, all genomic data was acquired from the National Cancer Institutes (NCI) genomic data portal [18]. Healthy and tumorous cell mass RNA expression, microRNA expression, copy number variation, and simple nucleotide variation data was acquired for nine different forms of cancer and solid tissue normal (STN) samples. The nine cancer types included were head and neck squamous cell carcinoma (HNSC), kidney renal clear cell carcinoma (KIRC), kidney renal papillary (KIRP), liver hepatocellular carcinoma (LIHC), lung adenocarcinoma (LUAD), lung squamous cell carcinoma (LUSC), prostate adenocarcinoma (PRAD), thyroid carcinoma (THCA). The following subsections detail the preprocessing steps required to transform and extract features from the raw genomic data.

5.1.1 Copy Number Variation

Preprocessing

The copy number variation (CNV) data were derived from somatic and germline genotyping array (Affymetric Genome-Wide Human SNP Array 6.0). The raw CNV data is given as segmented genomic regions that have the same DNA copy number. This form provides the number of bound probes and the binary logarithm of the mean intensity (segmented mean) for each segmented genomic region as shown in Table 5.1.

GDC Aliquot [a]	Chr	Start	End	Probes	Segment Mean
00e5b006-6afc-4ea4-90e3-f29741560020	1	62920	814954	31	0.4742
00e5b006-6afc-4ea4-90e3-f29741560020	1	817186	3303537	710	-0.0539
00e5b006-6afc-4ea4-90e3-f29741560020	1	3303596	16477281	7873	0.0117
00e5b006-6afc-4ea4-90e3-f29741560020	1	16477846	16935737	127	0.3408
00e5b006-6afc-4ea4-90e3-f29741560020	1	16935752	30261189	7664	0.0229

[a] Aliquot cooresponds to KIRC primary tumour UUID 0063a6fa-9ebd-4b71-83c0-aeb17b97eb6.

Table 5.1: Raw CNV Data from Genome Wide SNP Segmentation

To extract features that can be shared between all nine cell mass types, the chromosomal regions were mapped to genes. Using the BioMart community portal we acquired the start and end positions of every gene in the human genome assembly GRCh38 (hg38) [38]. The human genes were then mapped to the CNV regions for each sample type. An example of the resulting process is shown in Table 5.2.

Ensembl ID	Chr	Abberration	Segment Mean	CNV Region	Gene Region
ENSG00000237763	1	DEL	-1.361	103620877-103717410	103655290-103664554
ENSG00000244057	1	DEL	-1.9195	152583230-152613762	152600662-152601086
ENSG00000198502	6	DUP	1.9859	32488906-32533522	32517343-32530287
ENSG00000264892	17	DEL	-2.8409	16806233-16815664	16812447-16812651
ENSG00000279442	22	DUP	2.0311	15294547-15315221	15298378-15304556

Table 5.2: Significant CNV Abberations Mapped to Gene Ensembl ID in KIRC

Dataset

For the i-th cell mass, a set of g_i genes were mapped to aberrant regions within the genome. The set of common genes between all cell mass types were defined as:

$$\bigcap_{i=1}^{n} g_i, \tag{5.1}$$

Where n is the number of cell mass samples. As a result, the CNV data contained the segmented mean of 11479 genes for each cell mass sample, resulting in a processed data matrix $C \in \mathbb{R}^{11479 \times n}$.

5.1.2 Transcriptome Expression

Preprocessing

This study utilized RNA sequence (RNA-seq), and microRNA sequence (miRNA-seq) transcriptome expression profiling. The miRNA-seq data is a form of transcriptome profiling that provides miRNA molecule quantification. The miRNA-seq data used in this study was derived using the BCGSC miRNA profiling pipeline [6]. Furthermore, RNA-seq is a form of transcriptome profiling that provides gene expression quantification. The RNA-seq data used in this study was derived from HTSeq-Counts framework [2].

All expression profiles were organized in relation to cancer subclass, individual case ID, and sequence ID. An example of this is shown in Fig. 5.1.

Dataset

The expression profiles of the transcriptome expression data had the input feature dimensionality of all assayed genes and miRNA molecules. The miRNA-seq expression data contained the normalized expression counts of 1881 miRNA molecules, while the RNA-

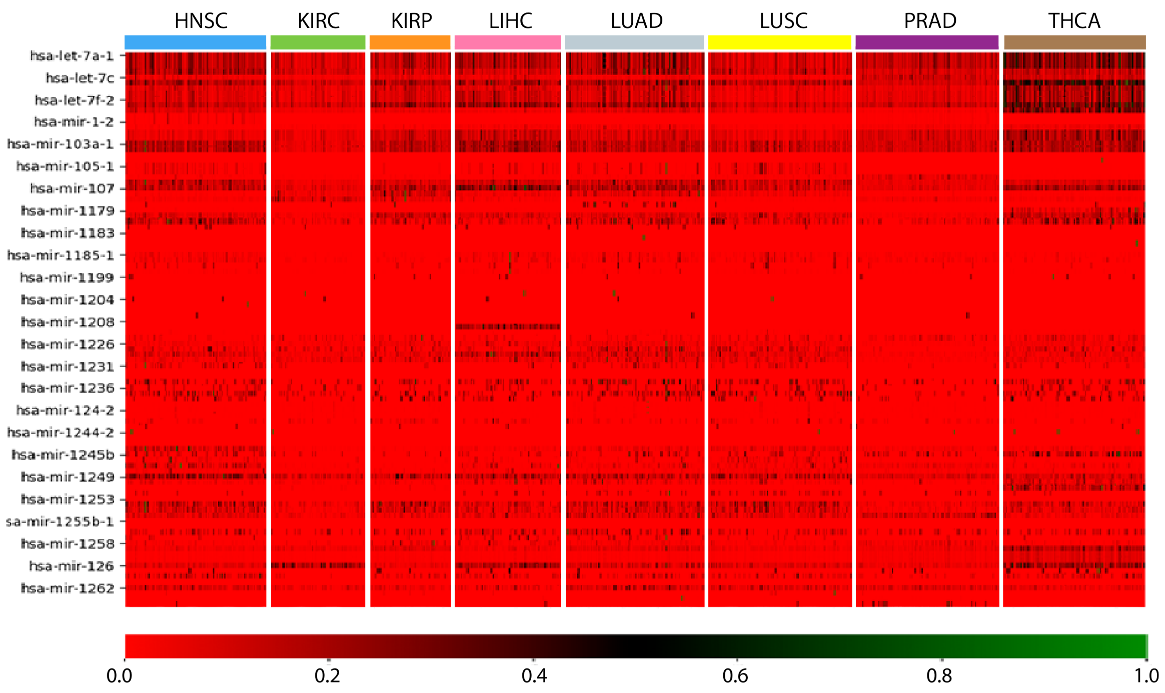


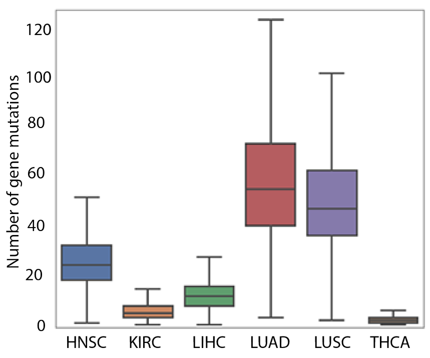
Figure 5.1: Normalized miRNA expression profiles.

seq expression data contained the normalized expression counts of 60484 genes. As a result, the processed miRNA-seq data produced data matrix $M \in \mathbb{R}^{1881 \times n}$, and the processed RNA-seq data produced data matrix $R \in \mathbb{R}^{60484 \times n}$.

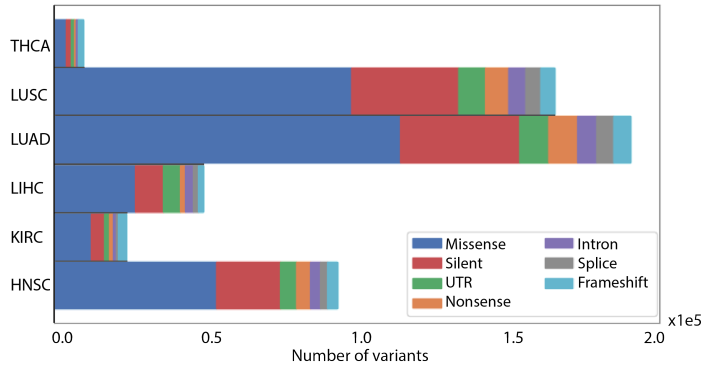
5.1.3 Simple Nucleotide Variation

Preprocessing

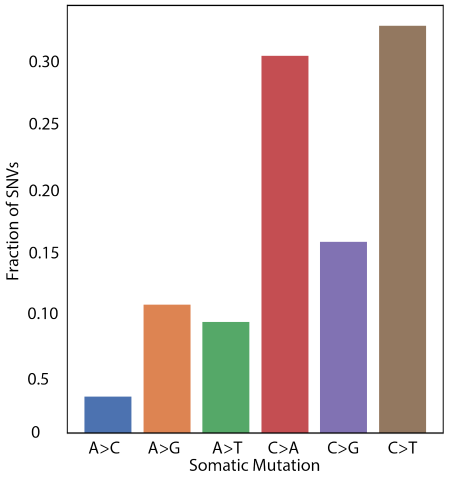
The simple nucleotide variation (SNV) data was obtained in the form of masked somatic mutations, derived from a MuTect2 Variant Aggregation and Masking workflow [7]. The SNV data is summarized in Fig. 5.2. In 5.2a, a boxplot of the accumulated gene mutations is shown for each cancer class. Fig. 5.2b shows a stacked barplot of the distribution of genetic variations for each cancer class. Fig 5.2c shows a barplot of the somatic mutations. Somatic mutations are represented using the (>) symbol to denote an alteration from nucleotide X to nucleotide Y as X>Y. Lastly, in Fig. 5.2d, the variant distribution of the top ten mutated genes are illustrated as a series of stacked barplots organized in ascending order by the most frequently mutated genes.



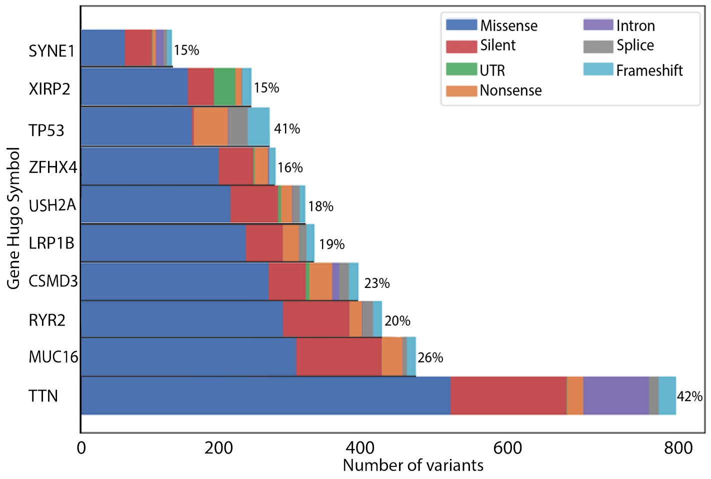
a) Accumulated gene mutations



b) Variant classification distribution



c) Fraction of somatic muations



d) Variant distribution of top 10 mutated genes

Figure 5.2: a) Box plot of accumulated gene mutations for each cancer type. b) Stacked bar plot showing the distribution of variant classification for each cancer type. c) Bar plot showing the fraction of all somatic mutations. d) Stacked bar plot detailing the distribution of the top 10 mutated genes.

In this study, the analysis of the raw SNV data was based on the variant occurrence frequency of the genetic data. Variation occurrence was mapped to every listed gene for all available cell samples. This was performed by mapping mutated genes to cell sample IDs in the raw SNV data, and accumulating the number of mutations for each respective cell sample.

Dataset

After preprocessing the SNV data, the variant occurrence frequency was obtained for 20516 human genes for each cell mass sample. The variant occurrence frequency was recorded in matrix $S \in \{\mathbb{Z}_{\geq 0}\}^{20516 \times m}$, where the 20516 rows correspond to genes, and the m columns correspond to the m cell samples per gene. Accordingly, an entry of the matrix S indicates the number of mutations observed for a cell on a given gene.

5.2 Dimensionality Reduction

In the following, we describe the various forms of dimensionality reduction used to deal with the high dimensionality of the genomic data and the selection of relevant features.

5.2.1 Stacked Denoising Autoencoder

An SDAE was used to acquire compressed feature vectors from all genomic data sources. A two-layer SDAE with dimensions 1000, and 500 was trained using a designated training set. Optimal model parameters were selected based on model performance during 10-fold cross-validation. Post training, a layer with reduced dimensionality and a low cross-validation error was selected. The defined objective here was to acquire a reduced mapping that encodes the original data with minimal loss of meaningful patterns.

5.2.2 Deeply Connected Genes

In our experiments, the weights of the trained SDAE were used to extract the raw features most strongly connected to the reduced subspace for CNV, RNA-seq and miRNA-seq data sources. These features were extracted from the SDAE by computing the product of the weight matrices for each layer [13]. The product of the weights for each layer in the trained and optimally parameterized SDAEs were observed to be highly normally distributed as shown in Fig. 5.3.

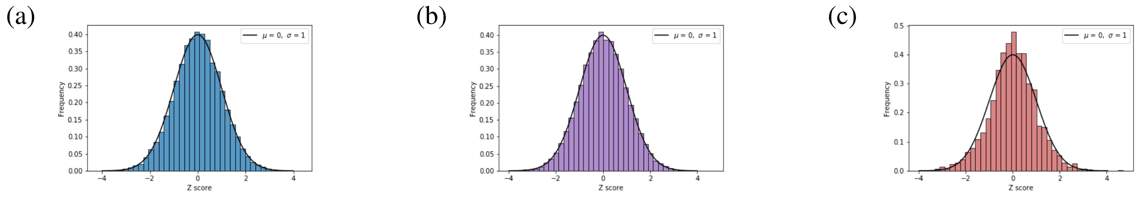


Figure 5.3: Histogram of z-scores from dot product of SDAE weight matrices for (a) CNV (b) RNA-seq and (c) miRNA-seq.

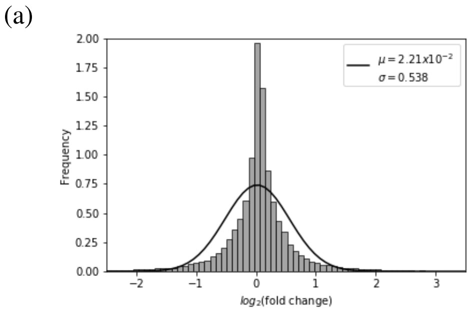
The most statistically significant features were identified by fitting the weight matrices to a normal distribution, and computing a p-value to select features that match the preselected experimental dimensions.

5.2.3 Differential Expression

For the transcriptome expression data, deferentially expressed genes were identified, and utilized as features. The $\log_2$(fold change) was computed between the median tumour cell mass expression and healthy cell mass expression. The most statistically significant features were identified by fitting the differential expression to a Gaussian distribution and computing a two-tailed p-value. Features that match the preselected experimental dimensions were acquired by selecting the top most significant deferentially expressed genes using the two-tailed p-values.

5.2.4 Clustered Gene Filtering

Due to the sparse nature of the discrete point mutation SNV data, clustered gene filtering (CGF) was used to select a subset of the most discriminatory genes based on the variant occurrence frequency matrix S [49]. The procedure involves filtering the genes into groups based on a similarity criteria, and then selecting a subset of genes from each group that have



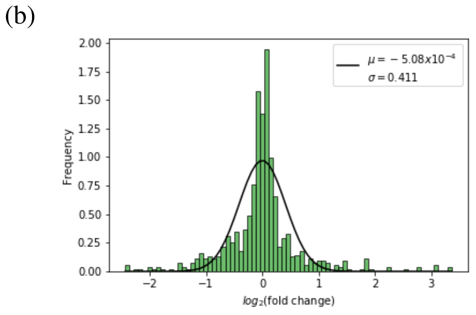


Figure 5.4: Differential expression log_2(fold change) and Gaussian fit for (a) RNA-seq and (b) miRNA-seq.

the highest mutation frequency because these are likely of more interest. Dimensionality reduction can be controlled algorithmically through the modulation of distance threshold d_{cgf}, and the group element threshold n_{cgf}. The distance threshold dictates how similar the mutation profiles of two genes need to be grouped together. The group element threshold is the number of genes kept from each group. The CGF algorithm used is summarized in Algorithm 1.

In line 2, we sum matrix S by its second dimension, such that $S_{sum} = \sum_j S_{ij}$ is a one dimensional vector of length p. We then produce S^*_{sum} by sorting matrix S_{sum} in descending order. The matrix S is then reindexed to match the sorted array so that the genes with the highest mutation frequency are listed first. Line 5 initializes a p-dimensional array to store the clustered group numbers for the genes. In steps 7 to 14, the genes of matrix S are clustered by inter-gene similarity. The similarity metric between two genes A and B is calculated using the cosine similarity:

$$d(A, B) = \frac{AB^T}{||A||\ ||B||} \tag{5.2}$$

In line 7, the index i iterates through the p genes, starting with the gene with the highest mutation frequency S_1. The similarity between this gene and all the remaining genes are

Algorithm 1 Clustered Gene Filtering

```
Require: Data matrix $S \in \{\mathbb{Z}_{\geq 0}\}^{p \times m}$
Require: Distance threshold $d_{cgf} > 0$, and group element threshold $n_{cgf} > 0$
 1: procedure CGF($S, d_{cgf}, n_{cgf}$)
 2:     $S_{sum} \leftarrow sum(S, axis = 2)$
 3:     $S^*_{sum} \leftarrow sort(S_{sum}, \text{order} = descending)$
 4:     $S \leftarrow reindex(S, \text{index} = S^*_{sum})$
 5:     $g = 0_{1,p}$
 6:     groupNum $\leftarrow 0$
 7:     for $i \in \{1 \ldots p\}$ do
 8:         if $g_i = 0$ then
 9:             groupNum $\leftarrow$ groupNum $+ 1$
10:             $g_i$ = groupNum
11:             for $j \in \{2 \ldots p\}$ do
12:                 if $j \neq i$ and $g_j = 0$ then
13:                     if $d(S_{i\ldots}, S_{j\ldots}) > d_{cgf}$ then
14:                         $g_j$ = groupNum
15:     $g_{out} \leftarrow \emptyset$
16:     for $k \in \{1 \ldots max(g)\}$ do
17:         for all $g_c = k$ do
18:             if $g_c > n_{cgf}$ then
19:                 $g_{out} \leftarrow g_{out} \cup g_c[1 \ldots n_{cgf}]$
20:     $S_{cgf} \leftarrow S[g_{out}, :]$
21:     return $S_{cgf}$
```

calculated, and if the similarity is larger than the threshold d_{cgf}, the respective gene is assigned to the group of S_1. After the similarity between S_1 with all genes are computed, the inter-sample similarity calculations and gene group assignments are repeated for the next ungrouped element in S. The last step forms the discriminatory subset by selecting the top n_{cgf} in each group. The indices for the discriminatory subset are stored in the variable g_{out}, which is initialized as an empty set in line 15. The following two for loops iterate through all the genes g_c for a specific group k, and stores the top n_{cgf} genes as long as the group does not have fewer than n_{cgf} elements.

5.3 Model Interpretation

5.3.1 Gene-wise Interpretable Explanations

To find a gene-wise explanation, the LIME procedure is used to approximate the dGMU model with a linear model of class G, such that $g(\tilde{x}) = {w_g}^T \tilde{x}$. Perturbed instance $\tilde{x}$ is generated by individually noising each feature by drawing from a normal distribution. The mean and standard deviation is taken from each feature in the original dataset X. The perturbed instance is weighted using an exponential kernel learned over a Euclidean distance by letting $\mu_{x_i}(\tilde{x}) = exp(-\sqrt{\sum_{i=1}^{n}(x - \tilde{x})^2})/\sigma)$. The kernel width σ is defined as 0.75 times the square root of the number of training instances (default value for σ is used as established in [36]). With a locally weighted squared error J, as defined in Eq. (3.3), we learn the weights w_g of the sparse linear model via least squares. Each trained model provides interpretable explanations through the learned weights. The magnitude of a coefficient relates to the importance of the respective gene in sample x_i. Furthermore, genes with a positive weight coefficient are positively correlated with the prediction of the dGMU model and genes with a negative weight coefficient are negatively correlated. Accordingly, the explanation of a single prediction provides an interpretable framework by indicating the genes that are most influential. Specifically, a single LIME explanation can explain how the RNA-seq expression or SNV of the gene correlates with the model prediction.

Algorithm 2 Gene-Wise Global Importance with LIME

```
Require: Data matrix $X$, Perturbed data $\tilde{X}$
Require: Decision function $f$, True labels $y$
Require: Number of samples $M$, Class $k$, Kernel width $\sigma$
 1: procedure GeneLIME($X, \tilde{X}, f, \sigma, N, k$)
 2:     $\mathcal{X}_k \leftarrow \{\}$
 3:     for $i \in \{1 \ldots M\}$ do
 4:         if $f(X^{(i)}) = k$ and $y^{(i)} = k$ then
 5:             $\mathcal{X}_k \leftarrow \mathcal{X}_k \cup X^{(i)}$
 6:     for all $x^{(i)} \in \mathcal{X}_k$ do
 7:         Initialize $w_g^{(i)}$
 8:         $g^{(i)} \leftarrow (w_g^{(i)})^T \tilde{x}^{(i)}$
 9:         $\mu_x^{(i)} \leftarrow exp\left(-\sqrt{\sum_{i=1}^{n}(x^{(i)} - \tilde{x}^{(i)})^2}/\sigma\right)$
10:     $J(f, g, \mu_x) = \sum_i \mu_x^{(i)}(f(\tilde{x}^{(i)}) - g(\tilde{x}^{(i)}))^2$
11:     $\mathcal{W} \leftarrow \min_{g \in G} J(f, g, \mu_x) + \lambda(g)$    ▷ Minimize cost function, and store trained weights
12:     $\mathcal{G}_{i,j} \leftarrow \sum_{j=1}^{p} \mathcal{W}_{i,j}$
13:     return $\mathcal{G}_{i,j}$
```

The gene-wise explanations for a single prediction provides locally faithful insight into the logic of the classifier. In order to assess the global fidelity of the model, gene-wise explanations are pooled to evaluate the reliability of the predictions as a whole. A procedure for generating gene-wise LIME explanations is summarized in Algorithm 2. Gene-wise explanations are extended to understand the set of individual instances associated with correctly labelled predictions. Explanations for a set of correctly labelled instances are relevant in understanding the reliability of the classifier and assessing how the model behaves globally. For a given cancer class k, we can denote the dataset of correctly labelled instances as $\mathcal{X}_k$. The process of producing the matrix $\mathcal{X}_k$ is shown in lines 2 to 5 in Algorithm 2. Furthermore, we can denote the process of deriving an explanation from a subset of samples with a function $\xi(\cdot)$. Applying the function $\xi(\cdot)$ is equivalent to performing lines 6 to 11 in Algorithm 2. We now construct an $n \times p$ dimensional explanation matrix by setting $\mathcal{W}_{i,j} = \xi(\mathcal{X}_k)$. The matrix $\mathcal{W}_{i,j}$ represents the local importance of all n genes for each of the p correctly labelled instances for a given class. The gene-wise global weights can then be pooled in an n dimensional vector $\mathcal{G} = \sum_{j=1}^{p} \mathcal{W}_{ij}$. Accordingly, genes that explain more instances will be ranked with higher importance.

Chapter 6

Experimental Results

6.1 Experiment setup

6.1.1 Datasets

In order to validate the experimental methods discussed in this thesis, we utilized various datasets as shown in Table 6.1.

Experiment	Bimodal dataset	Method	Samples	Classes	Feature type
Sec. 6.2.1	CNV + miRNA-seq	DCF	3988	9	real
Sec. 6.2.1	CNV + miRNA-seq	SDAE	3988	9	real
Sec. 6.2.2	RNA-seq + SNV	DE + CGF	3375	6	real + integer

Table 6.1: Summary of experimental datasets

6.1.2 Metrics

In these experiments, all models performed multinomial classification. In order to use binary performance metrics, a one-vs-all approach was taken with the model predictions. This approach involves treating each individual prediction as a binary classification where samples of the correct class are positive and all other samples are negative. This strategy works because for each input example the models output a k-dimensional vector of real-valued confidence scores for each of the k classes. The predicted output is then designated as the label k for which the classifier reports the largest confidence score:

$$\hat{y} = \underset{k \in \{1 \ldots K\}}{\operatorname{argmax}} f_k(x)$$

This strategy creates a particular problem related to class distribution in the training set. Treating each classification as a binary prediction results in an unbalanced distribution. The set of negatives examples will be much larger than the set of positive examples [5]. As well, the distribution of labels between the cell types are not equal. This can lead to issues when evaluating models if the accuracy is the only measure of performance. In order to avoid issues related to class imbalance, the precision, recall, and F_1 score were also used as performance metrics. The F_1 score is the harmonic mean of the recall and precision:

$$F_1 = \left(\frac{recall^{-1} + precision^{-1}}{2} \right)^{-1},$$

where precision and recall are defined as:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \qquad recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

Furthermore, in order to visualize the performance of the multinomial cell type classifications we used area under the curve (AUC) receiver operating characteristics (ROC). AUC-ROC plots provide a performance measurement at various statistical thresholds. These measurements indicate how much the model is capable of discerning between classes. The higher the AUC, the better the model is at classifying true positives as positive and true negatives as negative. ROC curves are produced by plotting the true positive rate (TPR) on the y-axis, and the false positive rate (FPR) on the x-axis. The TPR is equivalent to *recall*, and the FPR is equal to $1 - specificity$:

$$specificty = \frac{true\ negatives}{true\ negatives + falsepositives} \qquad FPR = \frac{false\ positives}{true\ negatives + false\ positives}$$

A perfect model has an AUC of 1. This means that the model can perfectly separate $truenegatives$ and $truepositives$ samples 100% of the time. This results in a ROC curve where $TPR = 1$ for $FPR \in 0, 1$. The closer the top left corner of the ROC plot is to the point (0,1), the better the model. An AUC of 0.5 means the model does not have the ability of separating the classes. In this case, the model predictions are essentially assigned by random chance. On the ROC plot, an AUC of 0.5 results in a straight diagonal line.

In order to extend the ROC for multi-class classification, a pairwise comparison was performed to binarize the output of the classification. Here a one versus all approach was taken to compute the ROC for individual cancer cell tissue types [1]. Accordingly, the models were evaluated for their ability to classify an individual cell tissue type against all other cell tissue types. ROC curves were also drawn using micro-averaging, where each element of the model classification was considered as a binary prediction, and macro-averaging, which finds the unweighted mean of all class ROCs.

6.2 Results

In the following, we report the results from all experiments. In all cases, experiments were constructed using the following experimental settings and constraints:

- The datasets were divided into a 60/20/20 split for training, validation, and testing, respectively.

- Hyperparameter tuning was performed on the cross-validation set.
- Experiments were repeated 10 times using 10 fold cross-validation on the training set.

6.2.1 Classification Agreement and Model Performance

Comparison of Classification Agreement

In order to evaluate the effectiveness of the dGMU model, we compared single modality, and bimodality classification agreements for SDAE and DCF reduced miRNA-seq and CNV features. The results for each feature type along with their respective models are summarized in Table 6.2. The DCF features were observed to result in higher classification scores for most feature and model type combinations. The only exception was the single modality CNV with SDAE features, which had a higher accuracy, recall, and macro f1-score for MLP and SVM models. Independent of the evaluated feature types, the dGMU model achieved the highest classification metrics amongst the utilized methods.

Comparison of Model Performance

A more exhaustive characterization of the classification performance was achieved with the multi-class receiver operator characteristic (ROC) curves shown in Fig. 6.1. The false positive rate versus the true positive rate was plotted for a subset of the best performing models trained with bimodal DCFs reduced to a dimensionality of 500 from miRNA-seq and CNV data sources. In all models, as the true positive rate increased, the false positive rate increased exponentially. In order to compare model performance, it was ideal to observe the rate in which the true positive rate approaches one as the false positive rate increases. The

Experiment			Classification Metric			
Modality	Feature	Model	Accuracy	Precision	Recall	F1-Score
Bimodal	SDAE	dGMU	0.8764	0.8739	0.8749	0.8737
		GMU	0.8722	0.8700	0.8661	0.8675
		MoE	0.7611	0.7617	0.7566	0.7556
		MLP	0.8622	0.8613	0.8542	0.8570
		SVM	0.8605	0.8567	0.8542	0.8550
	DCF	dGMU	**0.9373**	**0.9357**	**0.9327**	**0.9341**
		GMU	0.9273	0.9292	0.9228	0.9255
		MoE	0.9156	0.9109	0.9111	0.9109
		MLP	0.9189	0.9199	0.9123	0.9154
		SVM	0.9172	0.9175	0.9150	0.9136
CNV	SDAE	MLP	0.7586	0.7622	0.7534	0.7526
		SVM	0.7343	0.7459	0.7275	0.7321
	DCF	MLP	0.7468	0.7795	0.7270	0.7200
		SVM	0.7335	0.7460	0.7256	0.7310
miRNA-seq	SDAE	MLP	0.8530	0.8560	0.8425	0.8477
		SVM	0.8429	0.8388	0.8347	0.8362
	DCF	MLP	0.9097	0.9118	0.9022	0.9052
		SVM	0.9072	0.9104	0.9045	0.9004

Table 6.2: Summary of classification agreement for CNV and miRNA-seq reduced to 500 features.

ROC shows the trade-off between sensitivity (true positive rate) and the specificity (1 - false positive rate) for each model. Model types that have curves closer to the top left corner indicated a better performance. In Fig. 6.1a and 6.1b, we see that the dGMU and GMU ROC curves cluster more tightly into the top left corner of their plots as opposed to the MLP and MoE ROC plots. In Fig. 6.1c and 6.1d, the MLP and MoE plots showed a decrease in model performance as the cluster of curves have decreased slopes. This is especially evident by observing the significantly lowered benign ROC curves in the MLP and MoE plots. In this study, it was imperative to maximize the true positive rate while keeping the false positive rate to a minimum. Accordingly, the area under the curve (AUC) was computed for each cell tissue type. The models that utilized GMUs were specifically more effective at classifying benign cell tissue types. The MLP and MoE models struggled with this classification problem resulting in a benign AUC of 0.90 and 0.87, respectively. The GMU based models rectified this deficiency, both resulting in a benign AUC of 0.98. Furthermore, utilizing the ROC averages as a holistic measure of model classification performance, the dGMU model was observed to have the largest macro-average AUC of 0.98 and shared an identical micro-average AUC of 0.98 with the single GMU model. This indicates that the deeper configuration of the GMU improves cell tissue differentiation.

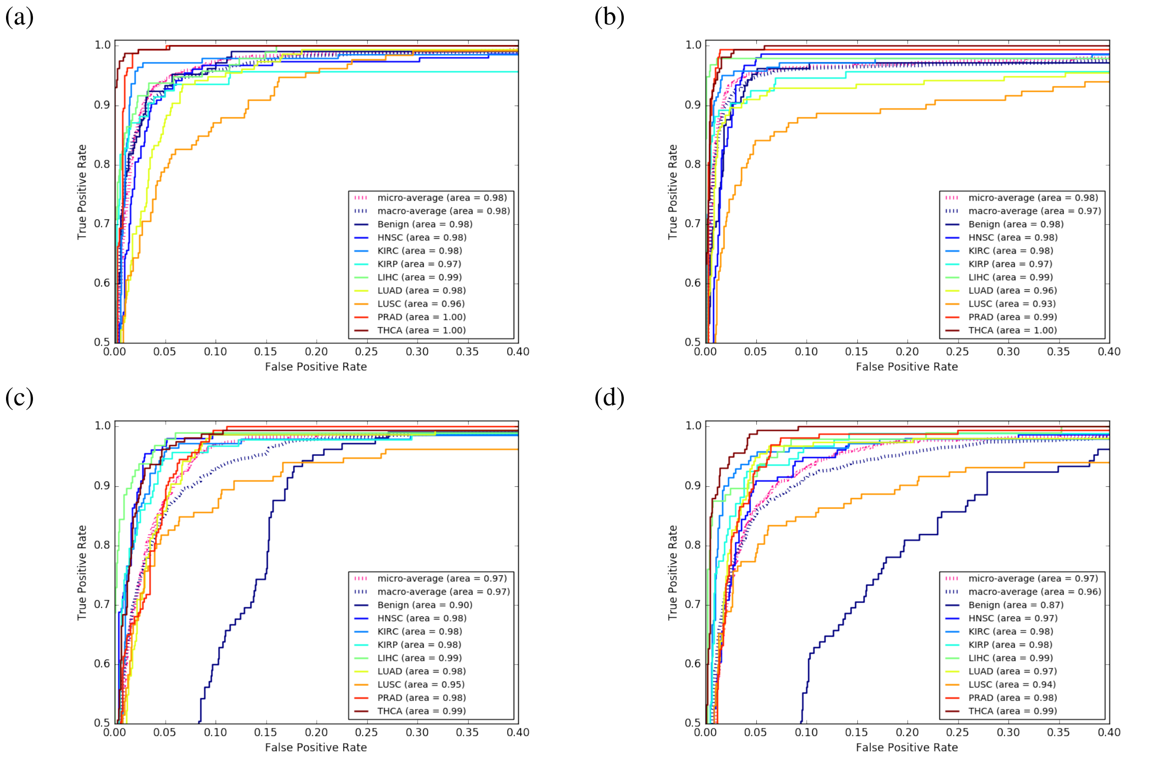


Figure 6.1: CNV and miRNA-seq DCF bimodal model ROC plots for (a) dGMU (b) GMU (c) MLP and (d) MoE models.

In order to evaluate the impact of feature dimensionality on the performance of the bimodal models, classification experiments were conducted with input dimensionalities ranging from 10 to 1500 features per modality. Fig. 6.2 shows the error rates observed for miRNA-seq and CNV DCF inputs at varying sizes. In all models, as the feature dimensionality increased, the error rate decreased. Though, the error rate stops improving rapidly after an input dimensionality of approximately 500. The dGMU model obtained the lowest error rate at 5.93%. The dGMU model also demonstrates the highest resistance to decreasing dimensionality, except for in feature sizes ranging from 100 to 200, where the GMU model obtained lower error rates.

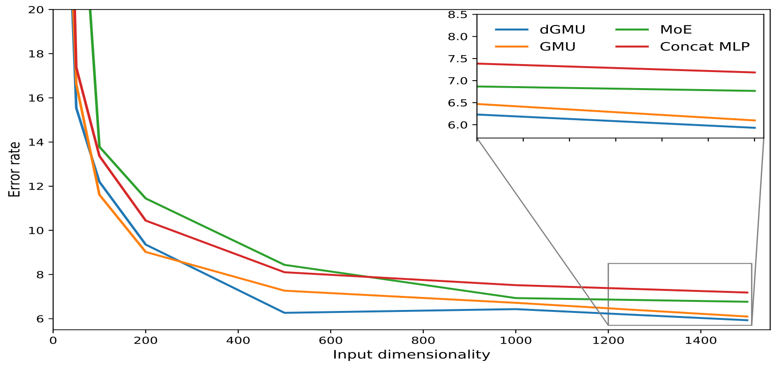


Figure 6.2: CNV and miRNA-seq DCF model error rates over a range of input dimensionalities.

Visualization of dGMU Latent Space

The latent space produced by the dGMU model was visualized with t-distributed stochastic neighbor embedding (t-SNE) [32]. In Fig. 6.3a, the dGMU model was shown to have a highly discriminatory latent space. The cell tissue types were largely clustered into the original nine classifications with a perceived level of geometric preservation between related cancers. These relationships include the kidney renal carcinoma's KIRC and KIRP, and the adenocarcinoma's LUAD and PRAD.

In order to evaluate inherit characteristics of the latent space, we considered the dispersion and the mean of the fusion encodings. In Fig. 6.3b, the dispersion was plotted against the log mean encoded value. The dissemination of the latent encoding was stratified in relation to the $\log_2$(Dispersion) and the $\log_2$(Mean). Distinct clusters were observed with labeled red and green markers for the samples derived from primary tumours and normal tissue, respectively. The dispersion values scatter with a degree of variance, which was ex-

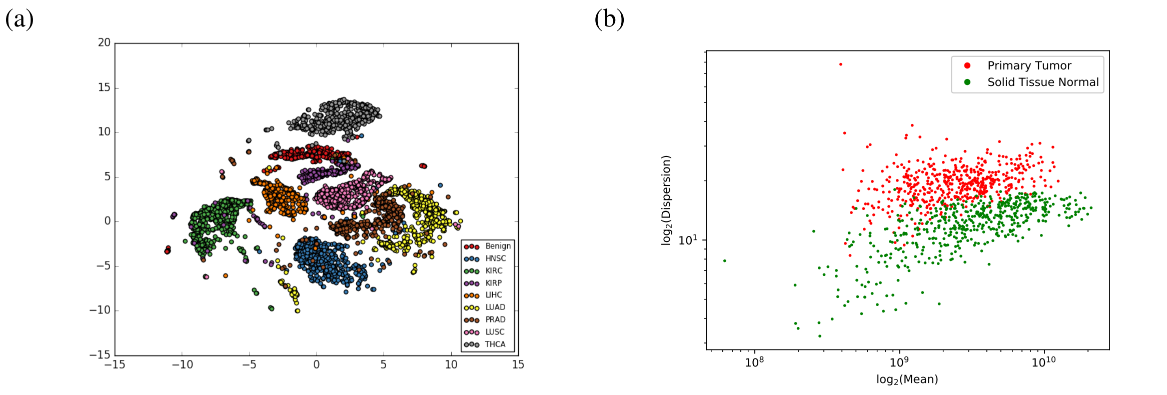


Figure 6.3: (a) CNV and miRNA-seq DCF dGMU model latent space clustered with t-distributed stochastic neighbor embedding (t-SNE). (b) Fold change and dispersion of the dGMU latent space.

pected given the sample size between the primary tumour and solid tissue normal samples [23].

6.2.2 dGMU Model Interpretation

We examined the functional enrichment of the top 400 interpretable gene components through a GO term and KEGG pathway analysis. The top 400 genes that promote positive explanations for the eight cancer types were identified as having significantly enriched GO terms and related pathways. The biological process related GO terms with a p-value smaller than 10^{-10} and the related KEGG pathways with p-value smaller than 10^{-3} are presented in Table 6.3. Many of the statistically significant pathways and terms are related to DNA replication, DNA repair, and cell cycle processes. This suggests that the genes most attributed to explaining the cancer classifications are related to cell proliferation and tumor growth. Furthermore, an additional review of literature was used to identify relationships between the significantly enriched pathways and the cancer types. The enrichment analysis of LIHC identified the carbon metabolism (hsa01200) KEGG pathway, and the response to insulin (GO:0032868), response to activity (GO:0014823), and fatty acid metabolic process

(GO:0006631) GO terms. The identification of these biological processes supports significant research describing the pathophysiological link between the human bodies response to insulin and the incidence of LIHC [30, 37]. Insulin stimulates the liver to store glucose, and the liver is the primary site for converting excess carbohydrates into fatty acids. Dysregulated cellular metabolism, where aberrant oncogenic signals alter the expression of metabolic enzymes, is a reoccurring theme in cancer cells. Currently, there is substantial evidence supporting dysregulated fatty acid metabolism and lipid metabolic reprogramming in LIHC [35, 44]. Through the application of LIME, we identified that the dGMU model is using biologically relevant information to stratify cancer classifications. These results suggest that a domain expert can use the interpretable gene components to understand why the dGMU model correctly classifies true positive cancer instances.

The cell division (GO:0051301) GO term was found to be significantly enriched in four cancer types. For HNSC, the pathway related genes were BUB1, LIG1, BIM, CIB1, SAC3D1, SPC24, BORA, BIRC, ECT2, KIF14, BUB3, and NCAPG. For KIRP, the genes were MAD2L2, ZWINT, CDCA3, CDK5, CDK7, KIRF2C, PARD3B, PRKCE, CDT1, BUB1, and TACC1. For LUAD, the genes were ATAD3B, BUB1B, BUB1, DSN1, NEK2, BIRC5, CDC25C, CDC6, CHEK2, KIF18B, NCAPG, RCC1, SGO1, UBE2C. Lastly, for LUSC the genes were ATAD3B, BUB1, LIG1, DSN1, MAD2L2, SPC25, ZWINT, MCM5, PRKCE, RCC2, TACC1, UBE2C. The greatest overlapping similarities were shared between the two lung cancers LUAD and LUSC, where four genes were shown to be shared. Despite representing the same biological process related to cell division, between the four cancer types, the gene sets were observed to be quite heterogeneous. This suggests that the genes identified as interpretable components have potential applications as biomarkers.

The intuition of the dGMU model was further investigated by examining the top three explanatory genes derived from LIME analysis for each cancer class. The weighted con-

Cancer Name	Enriched GO term and Related Pathway			
	ID	Name	Enrichment	P-value
HNSC	hsa04110	Cell cycle	5.2	4.1E-5
	hsa03030	DNA replication	9.8	3.1E-4
	hsa04914	Progesterone-mediated oocyte maturation	5.4	6.2E-4
	GO:0051301	Cell division	4.7	3.3E-11
	GO:0007062	Sister chromatid cohesion	9.2	8.3E-11
	GO:0008283	Cell population proliferation	4.4	4.3E-15
KIRC	GO:0007162	Negative regulation of cell adhesion	16.5	1.8E-13
	GO:0001666	Response to hypoxia	4.4	5.3E-13
KIRP	hsa04210	Apoptotic process	18.2	1.3E-5
	hsa04914	Progesterone-mediated oocyte maturation	5.4	6.2E-4
	GO:0051301	Cell division	4.7	3.3E-11
	GO:0007062	Sister chromatid cohesion	9.2	8.3E-11
LIHC	hsa01200	Carbon metabolism	6.4	7.04E-4
	GO:0032868	Response to insulin	8.6	2.6E-13
	GO:0014823	Response to activity	10.8	5.9E-13
	GO:0006631	Fatty acid metabolic process	8.9	1.0E-12
LUAD	hsa00630	Glyoxylate and dicarboxylate metabolism	11	9.7E-4
	GO:0001525	Angiogenesis	4.1	1.9E-14
	GO:0031568	G1/S transition of mitotic cell cycle	5.9	4.1E-14
	GO:0007062	Sister chromatid cohesion	6.4	2.5E-15
	GO:0051301	Cell division	6.0	8.2E-15
LUSC	hsa03440	Homologous recombination	28.2	8.6E-6
	hsa03030	DNA replication	13.6	1.9E-4
	GO:0051301	Cell division	5.4	2.8E-17
	GO:0000724	Double-strand break repair via homologous recombination	16.4	2.3E-14
PRAD	hsa04530	Tight junction	6.5	2.2E-4
THCA	GO:0006260	DNA replication	10.2	3.3E-12
	GO:0006974	Cellular response to DNA damage stimulus	4.4	5.03E-13
	GO:0006915	Apoptotic process	2.5	1.1E-12

Table 6.3: Summary of enriched gene ontology terms and related pathways.

tribution of these genes along with their respective $\log_2$FC from the differential expression analysis are shown in Fig. 6.4. For HNSC, CDC25A was identified as one of the explanatory genes. CDC25A is a protein-coding gene that performs an integral role in cell cycle progression. In literature, CDC25A is a known oncogene that is overexpressed in head and neck cancers [16]. This validates the explanation derived from the model that found CDC25A as a key explanatory gene with an overexpressed $\log_2$FC of 5.9. For THCA, LIME identified PRKCQ and BMP1 as the top two explanatory genes for the dGMU model. PRKCQ has been identified as having a potential role in the progression of thyroid cancer, and BMP1 is a known oncogene with potential gene interactions that are influential in the

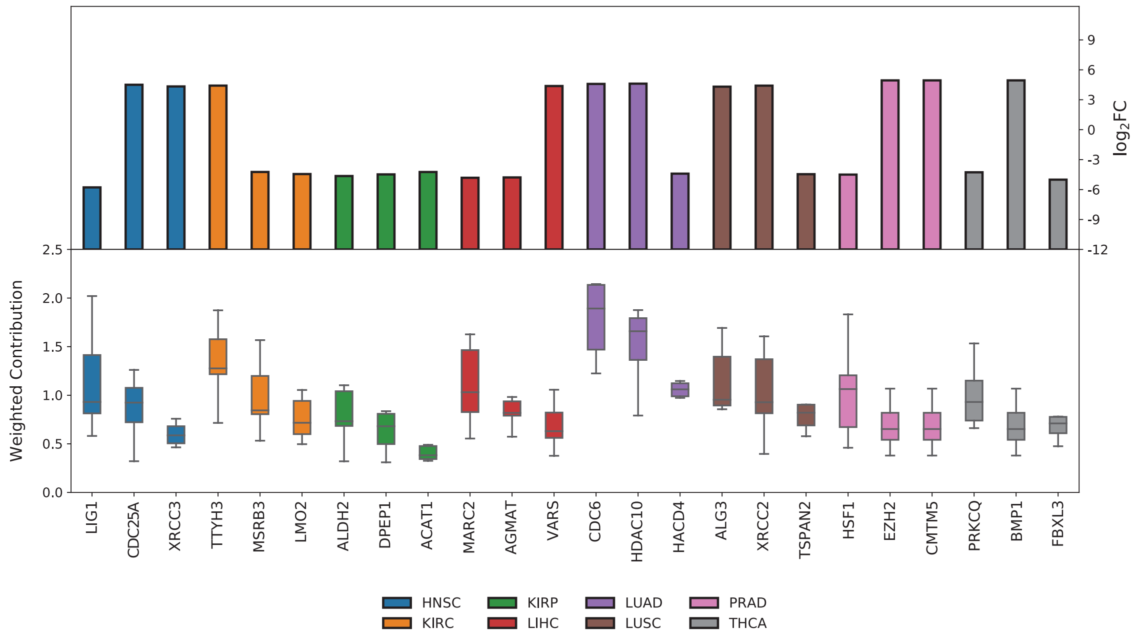


Figure 6.4: Top 3 explanatory genes for each cancer class.

carcinogenesis of thyroid cancer [15, 47]. For both KIRC and KIRP the top explanatory genes, TTYH3 and ALDH2, were identified as prognostic markers for kidney cancer [41]. Accordingly, through the application of LIME explanations, the dGMU model has shown a substantial utility of biologically relevant information for predicting cancer type class.

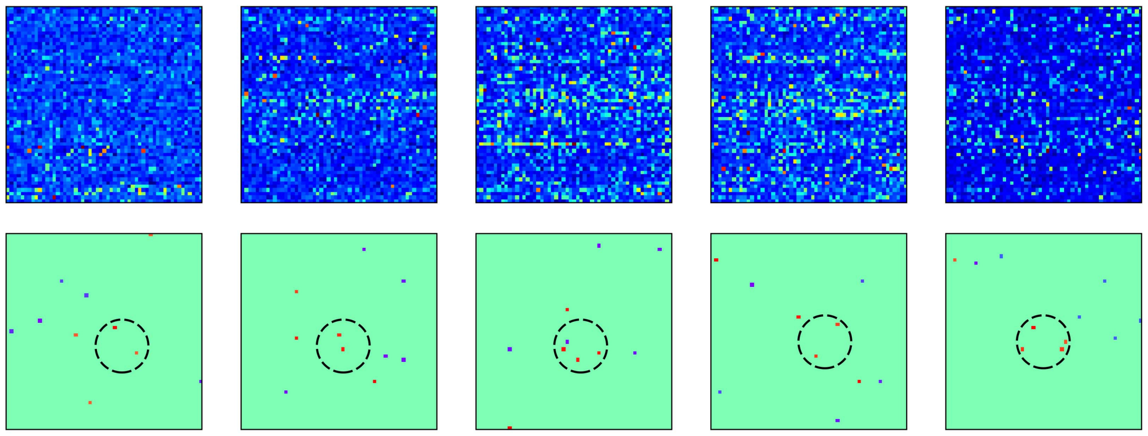


Figure 6.5: 2D embedding of RNA-seq and explanation heatmap with a localization of persistent explanations.

The location and variability of explanations were visualized using 2D embeddings of the RNA-seq input data. The RNA-seq data were embedded into 2D images by ordering the genes based on gene function and then reshaping the 3025 $\times$ 1 arrays into 55 $\times$ 55 images. An example is shown for five correctly classified HNSC RNA-seq profiles in Fig. 6.5. The first row shows the 2D embedding of the RNA-seq instances and the second row shows the respective top five positive and negative gene explanations. On the second row, the positive gene explanations that encouraged the prediction of the correct class were labeled in red, and the negative gene explanations were labeled in blue. The circled regions indicate a cluster with a high density of explanatory genes between examples. Although the explanatory genes were determined locally for a given instance, a general consistency in positive explanations remained between RNA-seq data input.

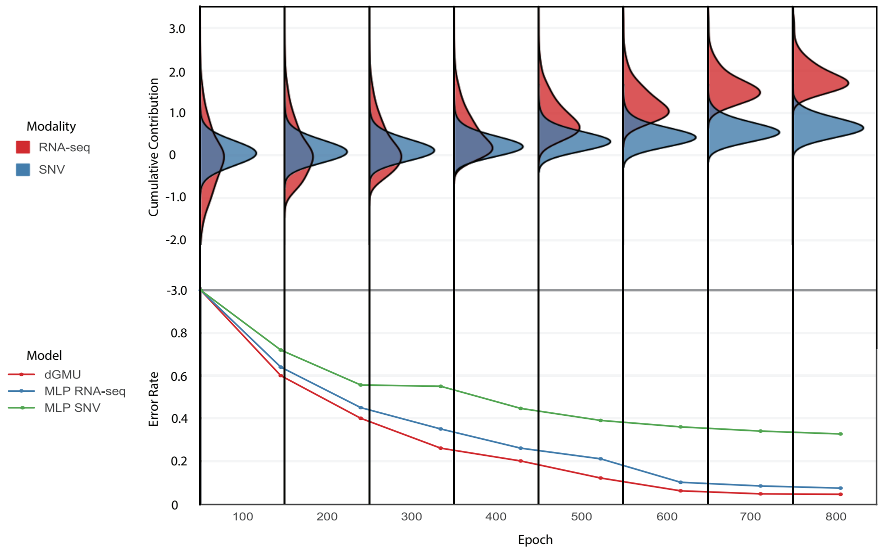


Figure 6.6: Distribution of cumulative contribution over a range of training epochs.

During the dGMU model training scheme, the cumulative weighted contributions for the RNA-seq and SNV features were examined. We found that as the model increased in efficacy, the influence of the RNA-seq modality increasingly dominated in weighted contribution as shown in the top part of Fig. 6.6. The red kernel density function for the RNA-seq modality progressively separates and settles at a larger average value than the blue kernel density function for the SNV modality. This suggests that the RNA-seq modality contributes stronger explanatory information on average than the SNV modality. This makes sense as the single modality RNA-seq model obtained a higher accuracy than the SNV modality as shown by the associated line chart of error rates illustrated alongside the labelled training epochs on the bottom half of Fig. 6.6.

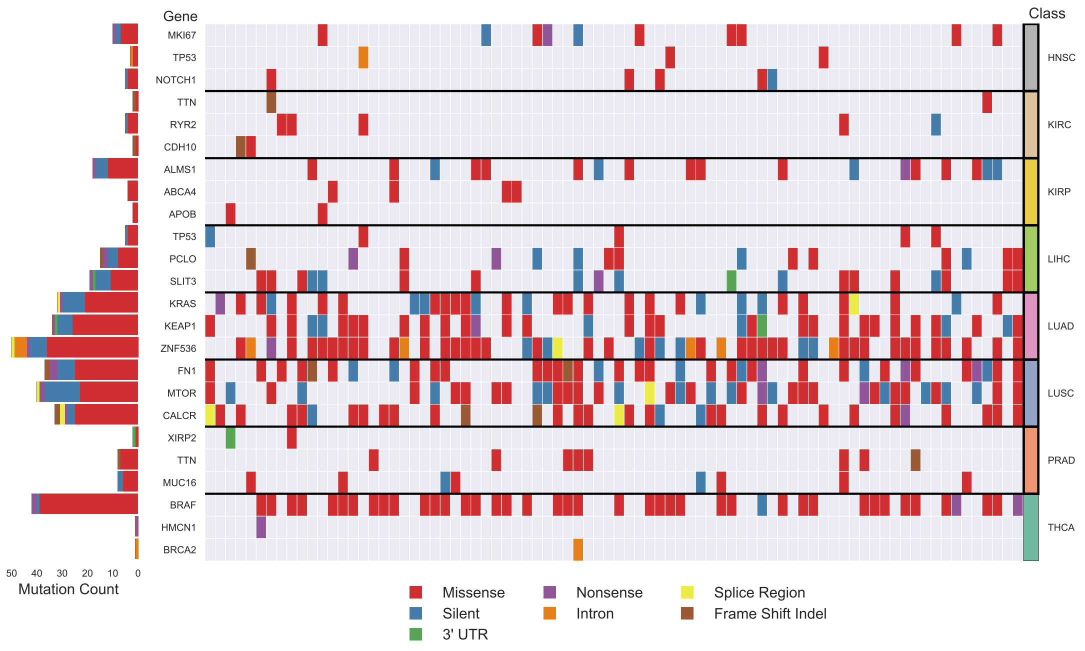


Figure 6.7: Top 3 explanatory single nucleotide variations for each cancer class.

The genetic variations of the top 3 explanatory SNV gene regions were visualized across the cell samples for each cancer class in Fig. 6.7. Each column represents a sample and each row a different gene. The fields are labeled to indicate the category of SNV that is present in the region of the respective gene. The left barplot shows the frequency of variations for each gene, and the associated cancer class is labeled on the right. The mutation frequency of the top 3 SNV gene regions varied widely across the cancer types. The lung cancers LUAD and LUSC had the highest presence of SNVs and the remaining cancers had very sparse variations across cell samples. An exception to this appeared for the BRAF gene in the THCA cancer cell samples. The genetic aberrations of all top three genes have been implicated in the development of various cancers, but genetic variations in HMCN1 and BRCA2 were only found in one case each [25, 26, 34]. Genetic variations in BRAF were found in more than half of the cell samples, the majority of which were missense mutations. BRAF is a protein-coding gene involved in the regulation of signalling pathways that influence cell division, differentiation, and secretion. Mutations in this gene are recurrent in THCA and the missense point mutation in which a single nucleotide change results in valine 600 to glutamic acid (V600E) is the most prevalent [46]. Although THCA has a low mortality rate, the presence of the V600E mutation is associated with faster cancer growth and a higher death rate [46]. Accordingly, the interpretable local explanations derived from LIME indicate that the dGMU model draws from clinically relevant information. This trend is found across the different cancer types. The LIME algorithm indicated SNVs in cancer-related genes in all cancer types which provides reasonable explanations that a domain expert can use to understand the prediction of the dGMU model.

Chapter 7

Conclusion

The dGMU model offers a promising framework for learning fusion representations from multi-platform genomic data. Our model is devised as a new component in the biomedical representation learning scheme, making it independent from past methods by leveraging the paradigms of *early* and *late fusion*. The main objective of this work is to utilize dimensionality reduction and interpretable features for multimodal cancer phenotype prediction. In our experiments, the dGMU model was able to learn a biologically relevant latent space using RNA expression and copy number variation from eight cancer types, and it outperformed both unimodal features and various common fusion strategies in classification agreement. The results indicate that deep learning architectures based on GMUs have the potential to expedite representation learning and knowledge integration in the study of cancer pathogenesis. Additional evaluations must be made to test further if the dGMU latent space can produce features that can generalize associations between increasingly diverse biomedical data. This effort could include the identification of potential cross cancer biomarkers by integrating heterogeneous cancer data in ensembles. In future work, we expect that these kinds of models support integrative and interpretable deep learning methods. As integrative machine learning methods become more common, we believe that multiplicative gating systems will provide clinicians with viable models to personalize patient care to their unique genomic profile.

The LIME algorithm was extended to facilitate the interpretation of multi-platform genomic data. We demonstrated the use of this algorithm on a multimodal neural network to generate gene-wise RNA-seq and SNV explanations for the classification of correctly labelled instances. We found that gene-wise explanations are useful for revealing clinically relevant genes used by the machine learning model to make accurate predictions. We also demonstrated that the explanations derived from multi-platform genomic data are helpful for identifying potential biomarkers and validating the predictive influence of known oncogenes. The additional insight gained by examining the explanations is helpful to gain trust in the predictions of the dGMU model. For a given classification, a domain expert can obtain the relative contributions of the modalities and the top explanatory RNA-seq expression and SNV gene regions. In the future, we would like to evaluate enhanced interpretable representations that incorporates the interactions between modalities. This involves incorporating known pathways and gene-gene relationships as a part of the model. We believe that correlating deeper biological relationships will help facilitate a greater insight into the underlying machine learning model.

Bibliography

[1] Erin L Allwein, Robert E Schapire, and Yoram Singer. “Reducing multiclass to binary: A unifying approach for margin classifiers”. In: *Journal of machine learning research* 1.Dec (2000), pp. 113–141.

[2] Simon Anders, Paul Theodor Pyl, and Wolfgang Huber. “HTSeq—a Python framework to work with high-throughput sequencing data”. In: *Bioinformatics* 31.2 (2015), pp. 166–169.

[3] John Arevalo et al. “Gated Multimodal Units for Information Fusion”. In: *arXiv preprint arXiv:1702.01992* (2017).

[4] Xiang Bai et al. “Integrating scene text and visual appearance for fine-grained image classification”. In: *IEEE Access* 6 (2018), pp. 66322–66335.

[5] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[6] Andy Chu et al. “Large-scale profiling of microRNAs for the cancer genome atlas”. In: *Nucleic acids research* 44.1 (2015), e3–e3.

[7] Kristian Cibulskis et al. “Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples”. In: *Nature biotechnology* 31.3 (2013), p. 213.

[8] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. “Multi-column deep neural networks for image classification”. In: *arXiv preprint arXiv:1202.2745* (2012).

[9] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. “The evolutionary origins of modularity”. In: *Proc. R. Soc. B* 280.1755 (2013), p. 20122863.

[10] Ronan Collobert et al. “Natural language processing (almost) from scratch”. In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.

[11] ENCODE Project Consortium et al. “An integrated encyclopedia of DNA elements in the human genome”. In: *Nature* 489.7414 (2012), p. 57.

[12] Mark Craven and Jude W Shavlik. “Extracting tree-structured representations of trained networks”. In: *Advances in neural information processing systems*. 1996, pp. 24–30.

[13] Padideh Danaee, Reza Ghaeini, and David A Hendrix. “A deep learning approach for cancer detection and relevant gene identification”. In: *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2017*. World Scientific. 2017, pp. 219–229.

[14] Elias Ebrahimzadeh et al. “Prediction of paroxysmal Atrial Fibrillation: a machine learning based approach using combined feature vector and mixture of expert classification on HRV signal”. In: *Computer methods and programs in biomedicine* 165 (2018), pp. 53–67.

[15] Anthony A Firek et al. “Pathologic significance of a novel oncoprotein in thyroid cancer progression”. In: *Head & neck* 39.12 (2017), pp. 2459–2469.

[16] Daniela Gasparotto et al. “Overexpression of CDC25A and CDC25B in head and neck cancers”. In: *Cancer Research* 57.12 (1997), pp. 2366–2368.

[17] Ian J Goodfellow et al. “Maxout networks”. In: *arXiv preprint arXiv:1302.4389* (2013).

[18] Robert L Grossman et al. “Toward a shared vision for cancer genomic data”. In: *New England Journal of Medicine* 375.12 (2016), pp. 1109–1112.

[19] Rahul Gupta, Kartik Audhkhasi, and Shrikanth Narayanan. “A mixture of experts approach towards intelligibility classification of pathological speech”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 1986–1990.

[20] Geoffrey Hinton et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal processing magazine* 29.6 (2012), pp. 82–97.

[21] Yi-Fei Huang, Brad Gulko, and Adam Siepel. “Fast, scalable prediction of deleterious noncoding variants from functional and population genomic data”. In: *Nature genetics* 49.4 (2017), p. 618.

[22] Robert A Jacobs et al. “Adaptive mixtures of local experts”. In: *Neural computation* 3.1 (1991), pp. 79–87.

[23] Tingting Jiang et al. “Statistical measures of transcriptional diversity capture genomic heterogeneity of cancer”. In: *BMC genomics* 15.1 (2014), p. 876.

[24] Hirak Kashyap et al. “Big data analytics in bioinformatics: A machine learning perspective”. In: *arXiv preprint arXiv:1506.05101* (2015).

[25] Electron Kebebew et al. “The prevalence and prognostic value of BRAF mutation in thyroid cancer”. In: *Annals of surgery* 246.3 (2007), p. 466.

[26] Chie Kikutake et al. “Intratumor heterogeneity of HMCN1 mutant alleles associated with poor prognosis in patients with breast cancer”. In: *Oncotarget* 9.70 (2018), p. 33337.

[27] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[28] Anshul Kundaje et al. “Integrative analysis of 111 reference human epigenomes”. In: *Nature* 518.7539 (2015), p. 317.

[29] Kim-Anh Lê Cao, Emmanuelle Meugnier, and Geoffrey J McLachlan. “Integrative mixture of experts to combine clinical factors and gene markers”. In: *Bioinformatics* 26.9 (2010), pp. 1192–1198.

[30] Xu Li, Xiaocong Wang, and Pujun Gao. “Diabetes mellitus and risk of hepatocellular carcinoma”. In: *BioMed research international* 2017 (2017).

[31] Muxuan Liang et al. “Integrative data analysis of multi-platform cancer data with a multimodal deep learning approach”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 12.4 (2015), pp. 928–937.

[32] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[33] Dhruv Mahajan et al. “Exploring the Limits of Weakly Supervised Pretraining”. In: *arXiv preprint arXiv:1805.00932* (2018).

[34] Jacqueline Mersch et al. “Cancers associated with BRCA 1 and BRCA 2 mutations other than breast and ovarian”. In: *Cancer* 121.2 (2015), pp. 269–275.

[35] Hayato Nakagawa et al. “Lipid metabolic reprogramming in hepatocellular carcinoma”. In: *Cancers* 10.11 (2018), p. 447.

[36] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should i trust you?: Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2016, pp. 1135–1144.

[37] Mandeep Kumar Singh et al. “Diabetes and hepatocellular carcinoma: A pathophysiological link and pharmacological management”. In: *Biomedicine & Pharmacotherapy* 106 (2018), pp. 991–1002.

[38] Damian Smedley et al. “The BioMart community portal: an innovative alternative to large, centralized data repositories”. In: *Nucleic acids research* 43.W1 (2015), W589–W598.

[39] Dongdong Sun, Minghui Wang, and Ao Li. “A multimodal deep neural network for human breast cancer prognosis prediction by integrating multi-dimensional data”. In: *IEEE/ACM transactions on computational biology and bioinformatics* (2018).

[40] Vıtor Teixeira, Rui Camacho, and Pedro G Ferreira. “Learning influential genes on cancer gene expression data with stacked denoising autoencoders”. In: *Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1201–1205.

[41] Mathias Uhlen et al. “A pathology atlas of the human cancer transcriptome”. In: *Science* 357.6352 (2017), eaan2507.

[42] Charles J Vaske et al. “Inference of patient-specific pathway activities from multidimensional cancer genomics data using PARADIGM”. In: *Bioinformatics* 26.12 (2010), pp. i237–i245.

[43] Pascal Vincent et al. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion”. In: *Journal of machine learning research* 11.Dec (2010), pp. 3371–3408.

[44] Mingda Wang et al. “Dysregulated fatty acid metabolism in hepatocellular carcinoma”. In: *Hepatic oncology* 3.4 (2016), pp. 241–251.

[45] Sholom M Weiss and Casimir A Kulikowski. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc., 1991.

[46] Mingzhao Xing et al. "Association between BRAF V600E mutation and mortality in patients with papillary thyroid cancer". In: *Jama* 309.14 (2013), pp. 1493–1501.

[47] Yaping Xu et al. "Identification of thyroid carcinoma related genes with mRMR and shortest path approaches". In: *PLoS one* 9.4 (2014), e94022.

[48] Pengyi Yang et al. "A review of ensemble methods in bioinformatics". In: *Current Bioinformatics* 5.4 (2010), pp. 296–308.

[49] Yuchen Yuan et al. "DeepGene: an advanced cancer type classifier based on deep learning and somatic point mutations". In: *BMC bioinformatics* 17.17 (2016), p. 476.

[50] Marinka Zitnik et al. "Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities". In: *Information Fusion* 50 (2019), pp. 71–91.