Somatic Variant Calling in Human Cancer Genome Data with Deep Learning

Daniel Biro

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

This skeleton demonstrates how to use the `infthesis` style for undergraduate dissertations in the School of Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file `skeleton.tex` generates this document and can be used as a starting point for your thesis. The abstract should summarise your report and fit in the space on the first page.

Acknowledgements

Acknowledgements go here.

Table of Contents

Chapter 1

Introduction and Background

As sequencing technologies and bioinformatics tools develop, ever larger studies are possible to build our understanding of the complex mutational landscape of various types of cancers. Coordinated whole-genome scale studies distributed across continents are emerging to combat the second most frequent cause of death that is only set to increase in the coming decades [1].

This project is aimed at exploring Deep Learning for Somatic Variant Calling in cancer to develop a robust and accurate method for identifying mutations. This chapter is broken down into five major sections to demonstrate the background: explaining the basics of Cancer Genomics, introducing Somatic Variant Calling and a benchmarking study, showcasing a more classical and a Deep Learning based approach to the problem, and finally connecting this project back to the work done in my previous dissertation project titled Deep Learning for Variant Calling.

1.1 Cancer Genomics

The development of cancer tumors in humans is a long process taking up to 2-3 decades caused by sequential alterations, most frequently in epithelial cells, that is ”surface” cells covering blood vessels, organs, skin, etc. [2]

The first step in this process is a so called ”gatekeeping” mutation occuring that confers a selective growth advantage to a cell that allows it to outgrow the surrounding cells. Mutations that provide selective growth advantages are called driver mutations since they ”drive” cancer development. Other mutations occur as well without providing a direct or indirect advantage to the tumor, these are called passanger mutations.

The effect of driver mutations on the difference between cell birth and cell death is quantitatively rather small, amounting to around 0.4% increase, however over the years this gets compounded up to to two times a week that could result in large tumors containing billions of cells.

After an initially slow growth another gene is bound to mutate that leads to further expansion. Eventually the tumor turns malignant and starts to invade different tissues

and even spread to different organs that may be distant from the primary site through the bloodstream for example. This is called metastasis and once it happens it becomes very difficult to cure the cancer and patient survival rates drop significanltly, although treatment and slowing the progression could still be possible.

Most cancers go through two to eight such sequential alterations that function through a dozen signalling pathways to alter the core cellular processes of cell fate determination, cell survival and genome maintenance.

Cell fate refers to whether the cell is performing divison or differentiation. Cells usually divide to populate tissues and then differentiate into different types and eventually die or become inactive. Driver mutations may let tumor cells divide indefinitely and effectively become immortal.

Altering cell survival means the cancer cell may become able to thrive in environments where normal cells would perish due to the lack of sufficient nutrient concentrations.

Lastly a change in genome maintencance could mean that the cells of the tumor can bypass the mechanisms that force normal cells who make mistakes during DNA replication to slow down or commit suicide. These processes exist to remove damaged cells to protect the organism and since cancer cells are ”damaged” in a sense from the perspective of the host, a mutation letting them survive this weeding out can even speed up the acquisition of further mutations.

The genetic heterogeneity of cancer is substantial: no two tumors are the same and these differences can result in different responses to therapeutics. The four main types of heterogeneity are: intratumoral, intermatastatic, intrametastatic and interpatient.

Intratumoral heterogeneity is the heterogeneity that exists among the cells of one tumor, as cells divide they acquire mutations and diverge from their common ancestor. This happens with normal cells as well and in the case of tumors may not be too relevant as the goal of surgeries is to remove the whole tumor irrespective of the amount of heterogeneity in it. Nevertheless this sows the seeds for the next type of heterogeneity.

Intermetastatic heterogeneity refers to the differences in mutations between the different sites the cancer has metastasized to. There can be a lot of heterogeneity between the different sites which makes treatment exceedingly hard as the proliferation goes on. Without eradicating all of them, the long-term survival of the patient cannot be ensured.

Intrametastatic heterogeneity happens as the cells of each metastasis further divide, mutating further similarly to intratumoral heterogeneity. Unlike intratumoral mutations these may provide drug resistance eventually. Since generally only primary tumors can be removed, targeted therapies must be used that despite initial successes inevitably relapse if such resistance is present.

Interpatient heterogeneity means there is no two identical tumors, no two patients with cancer who can be cured or treated in the same exact way. These differences may come from the germline variants of the patient, the somatic mutations of the tumor, etc. Even the same driver gene mutations can occur from different point mutations. This means

that individualized treatments are needed that are based on the genome of the cancer patient.

1.2 Somatic Variant Calling

The cancer mutations refered to in the previous section are somatic variants, that is mutations occuring in an individual cell opposed to germline variants that are hereditary and can be found across the organism. They can occur in any cell of the body except for germ cells (sperm and egg cells) and are of utmost importance in cancer genesis, progressing and treatment.

Detecting somatic variants is harder than detecting germline variants as they have to be distinguished from the later. A normal and a mutated sample has to be compared to filter out the germline variants. Cross contamination of tumor and normal samples, high tumor heterogeneity and errors occuring during sequencing are all sources of difficulty in the process.

Many different methods exist developed by different organisations to perform somatic variant calling. To compare and contrast some of these a comprehensive benchmarking study is presented in [3]. For this project the same dataset will be used as in this study. This dataset was obtained by downloading samples from the Genome in a Bottle dataset [4] [5] (see section 1.5 for previous work) and then mixing the data with synthetically generated tumor samples. This is needed due to the difficulty of obtaining true positive and negative variants from real tumor samples.

Different versions of the data were created by varying an important statistic called tumor purity. Tumor purity refers to the percentage of cancer cells in a solid tumor sample. The four different versions prepared had 10, 20, 40 and 60% tumor purity mixed into a normal sample separately. A result of the benchmarking was the conclusion that higher tumor purity tends to lead to higher accuracy in identifying both single nucleotide variants (SNVs or SNPs) and insertion/deletion variants (INDELs), with worse performance on the later irrespective of tumor purity.

Six different somatic variant callers were benchmarked in the study: TNscope [6] and TNseq of Sentieon [7], MuTect2 [8] of GATK [9], NeuSomatic [10], VarScan2 [11] and Strelka2 [12]. The next section takes a closer look at one of the more classical somatic mutation detection approaches: MuTect2. The secition afterwards examines NeuSomatic, the only Deep Learning based method out of the ones evaluated in the study.

1.3 A Classical Approach: MuTect2

MuTect2 [8] is an somatic point mutation caller developed for the Genomic Analysis Toolkit (GATK) [9] of the Broad Institute. It is designed to meet the critical need for high sensitivty and specificty in the somatic variant calling process that the authors perceived was only inadequately met by previous approaches.

The method takes advantage of different types of information about local sequencing error rates, allelic fractions, base quality scores, etc to hand-craft filters that samples can be compared against. The aim is to eliminate possible categories of false positives at each successive stage and filter.

The pipeline is constructed as follows. First standard preprocessing steps are applied, such as marking duplicate reads, base quality score recalibration and local realignment. A normal and a tumor sample of reads are paired for each genomic position tested to serve as the input. Then the four key steps of the algorithm are performed in succession, these are: removal of low-quality sequence data, variant detection in the tumor sample, filtering to remove false positives and finally the designation of variants as somatic or germline.

Variant detection in the tumor sample happens via a Bayesian classifier that captures a reference and a variant model. The former assumes there is no variant at the site and any non-reference bases read are simply random sequencing errors. The later assumes there is a true variant allele with some fraction in addition to these errors. This fraction is estimated from the data. The sample is classified as a candidate variant if the log-likelihood ratio of the data under the variant and reference models is above a certain threshold that was also determined empirically.

Candidates are evaluated against six different filters aimed at catching various types of artifacts and germline events. As part of this steps a panel of normal samples (PON) filter is used to eliminate further false positives caused by rare error models that are only present in other samples.

Finally a second Bayesian classifier is used to label the sample as one of three classes: somatic, germline or variant (indeterminate due to insufficient data). To make this decision log odds score is used, similarly to the first classifier, to compare the likelihood of the data under models of the variant being present as a heterozygote or absent in the normal sample. The third option is chosen if the power to make the classification is insufficient.

Overall MuTect2 has great performance, it achieves both high specificity and sensitivity in the various tests the authors conduct. However, there are numerous assumptions going in to the creation of this model, e.g the type of sequencing errors that the filters are based upon. There are also empirically set threshold values present that may vary between data sources. These issues provide the main points of contrast with Deep Learning based approaches.

1.4 A Deep Learning Approach: NeuSomatic

NeuSomatic [10] is one of the first Deep Learning approach for somatic mutation detection. It is a Convolutional Neural Network (CNN), a type of neural network loosely inspired by the way the human visual cortex processes signals, which has gained popularity for various classification tasks since 2011, mostly in computer vision.

The attractiveness of Deep Learning for variant calling comes from its relatively simplicity and robustness compared to the traditional statistical and algorithmic approaches

used by tools such as the ones refered to in the previous sections. While working remarkably well, the heavy feature engineering involved in traditional approaches makes their generalization ability limited. In other words they may be suboptimal on data coming from sample types and sequencing technologies they were not designed for. The hope of using Deep Learning methods, and hence of this project too, is to create tools that are agnostic to these variations in data while retaining high accuracy.

Data samples are of the (kx5x32) dimensions and are formatted as follows. The reference sequence is taken in a seven base pair long window around a candidate SNP (fifteen base pair long sequences). This sequence is then augmented by adding gaps where insertions are detected in the reads. The same is done for all aligned reads from a tumor and a normal sample, with the reference then being summarised in a binary matrix and the tumor and normals reads in count matrices where position (i,j) represents the count (presence/absence in the reference matrix) of base (or gap) i at position j in the reads. Both count matrices are then normalised by the coverage resulting in base frequencies.

These matrices are then concatenated with additional ones of the same shape for tumor and normal coverages, positions, additional alignment features (e.g base quality), forming a multi-channeled input matrix. An ensemble mode can be selected as well to add extra matrices for the features of other the other methods NeuSomatic is ensembled with.

The prepared input matrices are fed into a network with nine convolutional layers that are divided into four bloks, each connected by residual skip connections. The ReLU activation function and polling is applied at alternating layers respectively. A final fully connected layer is applied at the end with its output fed into three separate classifiers, two softmaxes and one regressor. The first classifier determines whether the candidate is a variant, an insertion, a deletion or a non-somatic call. The second classifier identifies the insertion/deletion length and can output zero (not an INDEL), one, two or greater than two. Finally the regressor predicts the position of the somatic mutation.

NeuSomatic is compared to DeepVariant [13] and Clairvoyante [14], both of which served as the basis of my previous work described in the next section.

DeepVariant is the frst Deep Learning based germline variant calling method developed by DeepMind, also using a CNN architecture. It uses read pileups for inputs instead of the base frequency summaries used by NeuSomatic, which only allows a less efficient neural network structure. Variant calling on 30x coverage whole-genome sample takes around 1000 CPU-core hours. In contract NeuSomatic can perform variant calling on a paired tumor-normal whole-genome sample of the same coverage depth in roughly 156 CPU-core hours.

Clairvoyante was an improvement over DeepVariant, both running faster and performing better. It uses three channels in its inputs, summarising base counts, deletions and insertions in a window. Compared to this, NeuSomatic summarises all three types of information in a single frequeny matrix.

NeuSomatic serves as the main inspiration of this project and as a benchmark to com-

pare results to on the data presented in the benchmarking study of the previous section. Before concluding this chapter, my previous work that is the stating point is presented.

1.5 Previous Work

This project builds on my Master of Informatics thesis project part 1 titled Deep Leanning for Variant Calling. The main objective of this project was the development of a method to perform germline variant calling on a subset of the Genome in a Bottle (GIAB) dataset [4] [5].

GIAB is a consortium hosted by the US National Institute of Standards and Technology (NIST) containing seven whole human genomes sequenced with a few different next generation sequencing methods, along with corresponding concensus variant call sets for each. The model was trained on a part of the first chromosome data of one of these samples and then evaluated on a part of the firt chromosome data of a not genetically related sample, and first and second chromosme data of another genetically related sample.

The inputs were constructed by taking a context window of base pairs around a candidate location (similarly to Clairvoyante and NeuSomatic [14] [10]) with three different input channels corresponding to the fraction of reads showing the reference and alternate nucleotides, and a gap at the given position.

Different architectures were experimented with, all being various versions of Recurrent Neural Networks (RNNs), another popular twist on neural networks that are primarily associated with natural language processing applications. More specifically vanilla RNN, Long-Short Term Memory (LSTM) [15] and Gated Recurrent Unit (GRU) [16] was tried with the last one emerging as the best performing one after hyperparameter tuning, having two recurent layers and a fully connected final layer outputting a class. It was also shown that these architectures perform better than a simple perceptron with no hidden layer corresponding to a linear transformation.

Initially only a binary distinction was made between variant and not variant, which was later expanded to first include a class for INDELs and then separating SNP variants into homozygous and heterozygous classes.

Various other things were tested such as whether dropout [17] could improve the performance or how the context window length affects accuracy.

All networks were implemented via the Pytorch [18] Deep Learning library. The codebase of the full pipelines serves as the template for this current project from the preprocessing down to network performance evaluation.

The results were remarkable and provide a proof that different architecures (RNNs) to those published recently (CNNs) can work with similar results for the germline variant calling task. While somatic variant calling is more difficult, the performance of NeuSomatic is encouraging for this space and by the end of this thesis the reader will find out how RNN based approaches compare.

The justification for choosing Recurrent Neural Networks (RNNs) instead of Convolutional Neural Networks (CNNs) is the following.

CNNs are designed for hierachical classification tasks, for instance object detection in images, where features need to be extracted in a position invariant way. In works like DeepVariant [13] the sequential genomic data is transformed into an ”imagelike” representation to take advantage of the strengths of CNNs.

Since the data is originally sequential, architectures that were specifically designed for sequence data could and perform similarly if not better. RNNs, introduced in Chapter 3, are such models and the results of the previous project indicated that the performance is indeed satisfatory with them.

Thus this line of thought is carried on for this project and similar architectures are used this time as well.

Chapter 2

Data and Pre-processing

Intro TODO

2.1 Data Source: Genome in a Bottle Dataset

All of the data used in this project comes from the Genome in a Bottle (GIAB) project [4] [5]. This project is a consortium hosted by the US National Institute of Standards and Technology (NIST) and it was piloted in 2014 with additional samples released in 2016.

Motivated by the issue of the discrepancy of variant calls being made by different algorithms built for different sequencing methods, the aim of the project was to establish a standard reference dataset with highly accurate genotypes across reference genomes. This data was made publicly available to be used by anyone in the field to benchmark their methods and enable algorithmic improvements as well as leading to potential new discoveries about the human genome.

Altogether there are 7 whole sequenced human genomes in the dataset. The first sample released was the widely used reference genome of a Utah woman, referred to as NA12878 in the literature. The other 6 genomes belong to two father-mother-son trios, one of Ashkenazi Jewis and one of Han Chinese descent.

Several resources are available for all of these genomes including but not limited to raw reads, alignment files and high confidence consensus variant calls. These resources were produced by using various datasets, sequencing technologies and variant calling methods to ensure the high overall quality for all of them.

Of particular interest for the present project are the alignment files and the files with the variant calls. These are given in two standard formats: the Binary Alignment Map (BAM, a binary version of the Sequence Alignment Map, SAM format [19]) and the Variant Calling Format (VCF [20]) for each sample.

A BAM file contains a header and an alignments section. The former contains information about the file such as the name of the sample, the length of it and the alignment method used to create the alignments. The later describes the aligned reads them-

selves with their names, quality, etc in addition to other pieces of information with the optional custom tags.

A VCF file also consists of a header and a main data part. It describes variants found in a reference genome. The header lists the annotations used in the file, information about the file version, variant caller version, a reference to the BAM file(s) used and the column headings of the variants. The main part contains various pieces of information about individual variants, such as the chromosome, position, quality, zygosity, etc.

2.2 Pre-processing

TODO

Chapter 3

Deep Neural Network Architectures

This chapter introduces the neural network architectures that have been experimented with. These were the following: single layer perceptron, vanilla RNN, LSTM, GRU and a very basic Transformer.

Fully connected feedforward neural networks are powerful and flexible models. In theory they are universal approximators that can fit any function given enough nodes in a single hidden layer with some nonlinearity applied to the outputs.

Even a simple, single layer perceptron can be used in many cases when the task is binary classification. Linear classifiers can work well diverse types of data even without having to apply nonlinerities.

Hence, although my tasks fall into the multiclass classification category, the first model tried to form a sort of baseline was such a single layer perceptron, a single layer of fully connected nodes of the sequence of a data point mapped to the nodes representing the different classes.

A softmax layer is applied to the output, a nonlinear transformation that ensures that the final result is a probability distribution over the classes, easing comparison and interpretation of the model's best choice. It is defined as follows:

$\sigma(z_i) = exp(z_i) / \sum_{j=1}^{K} exp(z_j)$

Since the result is a probability distribution over the values, they will all fall in the [0,1] range and sum to 1. The output of all other models were fed through the same softmax activation function as well.

3.1 Recurrent Neural Network Architectures

A big drawback of such simple fully connected models is that they have no memory. Previously encountered inputs that could give useful context to correctly process present and future ones is not taken advantage of.

This limitation is especially apparent in domains like natural language processing or really in any sequential or time series data analysis.

To remedy this issue, a special class of neural networks was created called recurrent neural networks (RNNs for short). RNNs have many varieties but the core idea around them is the same. A recurrent connection is introduced to create a loop with which the network can modify an internal state in light of the current data processed. This internal state encodes information about the previously seen data.

This setup can also be thought of as several copies of the same network, each having an extra input and output. These networks pass on their internal state to the next network in line and accept the previous network's interal state as an input of their own.

As the length of the sequence to be processed increases a probem arises with vanilla RNN setups. Theoretically all information about earlier parts of the sequence should be well represented and easily taken into account when making inferences, learning connections between elements of the sequence. In practice this does not happen, as the gradient signal that drives the learning mechanism of backpropagation simply vanishes as a result of many multiplications with smaller and smaller values. The contribution of early elements to the loss of the later elements diminishes and long range dependencies are not learned.

To remedy this issue, several improved architectures have been developed. All of these have their own merits and may be useful on specific data. The most widespread variants are the Long Short Term Memory and the Gated Recurrent Unit architectures.

3.2 Long Short Term Memory

Long Short Term Memory networks or LSTMs are not a new addition to the zoo of neural networks. The architecture was first introduced in [15] in 1997 and soon proved to be very useful on a wide range of problems.

In the standard RNN architecture there is only one internal state and the hyperbolic tangent activation function applied to the output of it. In contrast, an LSTM cell has four internal subcompontents with 3 "gates". Figure 3.1 provides an overview of this structure.

There are 3 inputs to the LSTM cell, x_t the input data at timestep t, c_{t-1} the previous cell state and h_{t-1} the previous hidden state. The outputs are the current cell state and hidden state (c_t, h_t) with the hidden state serving as the model output as well.

The first operation that gets applied to the inputs is the forget gate, denoted by f_t. Mathematically it is defined as:

$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$

Where σ denotes a standard sigmoid function and W_f, b_f are the weights and biases associated with the forget gate. The part with square brackets means the weights are applied to both the previous hidden state and the inputs, and then combined using a Hadamard (elementwise) product.

The name points to the functionality, the forget gate consideres the previous hidden state and the incoming input two decide how much of the preivious cell state should

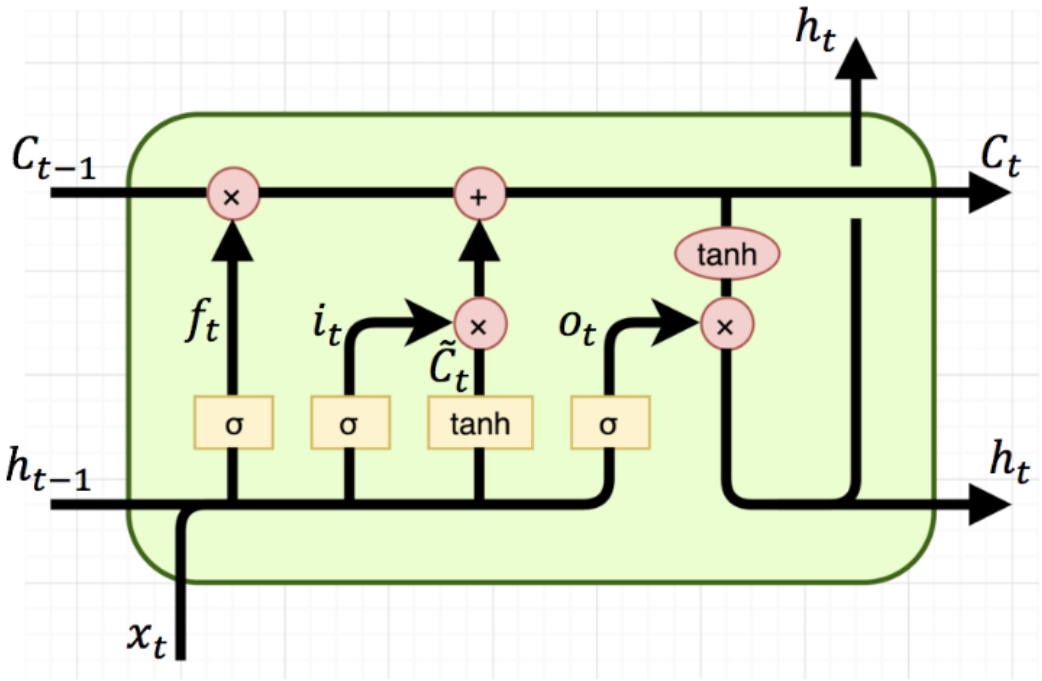

Figure 3.1: Illustration of the internal configurations of an LSTM cell. Image source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

be forgotten. A value of 0 means complete wiping of the cell state, while 1 means no forgetting. This is applied to the cell state via a multiplication.

The next step is deciding how much of the input information should be saved in the new cell state. This is done via combining two operations, an input gate and a tanh layer that creates the candidate values. These are defined as:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

Where all variables are defined similarly as before. The input gate decides how much of the new candidate values to memorize, while the tanh layer is applied to the previous hidden state and inputs to create these values. The two are combined via multiplication and then added to the cell state.

Then the update of the cell state can be written down as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The output, that is the hidden state is based on this cell state. A tanh operation is applied to it to push the values between -1 and 1, and then multiplied with an output gate to produce the output of h_t. This can be written as:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

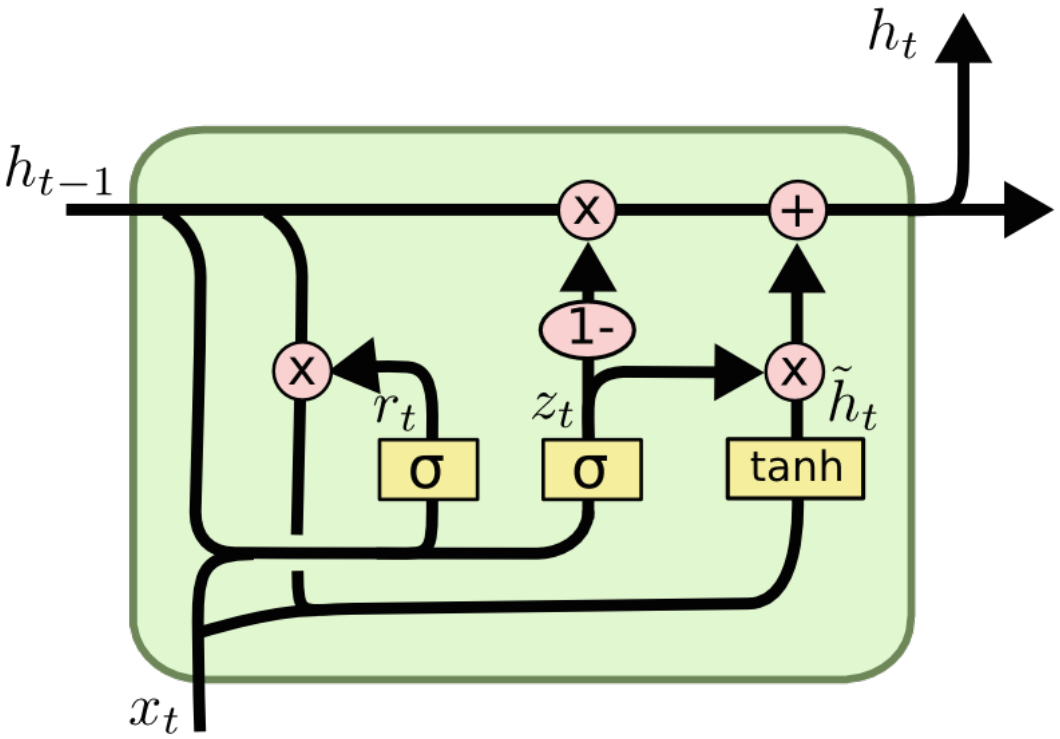

Figure 3.2: Illustration of the internal configurations of a GRU cell. Image source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

$h_t = o_t * tanh(C_t)$

The parameters of these gates are also optimized during the learning process, and in practice this makes the LSTM setup overcome the weaknesses of the vanilla RNN.

3.3 Gated Recurrent Unit

Gated Recurrent Unit or GRU, is a more recent architecture from 2014, presented in [16]. It is essentially a simplification of the LSTM cell albeit it may not seem that way upon first inspection. An illustration of the GRU cell is presented in Figure 3.2.

The main changes compared to LSTM are the combination of input and forget gates into an update gate and the merging of the cell and hidden states.

The update gate decides how much of the previously stored hidden state is passed along. The equation is simlar to the ones before, but there is no bias involved:

$z_t = \sigma(W_z * [h_{t-1}, x_t])$

r_t stands for the reset gate of the current timestep, which is defined very similarly to the update gate:

$r_t = \sigma(W_r * [h_{t-1}, x_t])$

This gate decides what information should be removed from the total that has been accumulated in the previous hidden state.

Next the reset gate and the inputs are used to determine what parts of the previous hidden state should be propagated in the following way:

$\tilde{h_t} = tanh(W * [r_t * h_{t-1}, x_t])$

Finally this is added to the hidden state after the 1 minus the results of the update gate is multiplied by the previous value:

$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$

This architecture has also successfully combatted the vanishing gradient problem and is in widespread use due to its great performance.

3.4 Transformer

The Transformer is a novel architecture type from 2017, introduced in [21]. Since the original paper being somewhat impenetrable, a simpler and clearer introduction is given a famous blogpost [22], on which the explanation here is based on.

The fundamental building block of this architecture is the self-attention operation. Intitially people started to use this operation to augment RNNs, especailly in sequence-to-sequence tasks where encoder-decoder architectures are used. In these architecures there are two models (usually some RNNs, for instance LSTMs), an encoder which transforms the input sequence into a more compact intermediate representation and a decoder which takes this representation and sequentially builds an output sequence from it.

Self-attention in this context is most often used in the decoder part. The operation is simply a weighted average of an input:

$y_i = \sum_j w_{ij} * x_j$

Where the weights are obtained from the data, more specifically usually produced via a dot product:

$w_{ij} = x_i^T x_j$

These weights are scaled via the a softmax before used for calculating y_i , which was shown before. When used in a decoder model, the attention weights are used to put emphasis on the more important parts of the sequence rather instead of assuming that all parts are equally important.

Using self-attention improved performance and in some cases was adapted in the encoder as well to assign weights to the inputs as well. Naturally, the question arose if this mechanism could work in itself without requiring the original recurrent models.

Suprisingly the answer was yes, which lead to the Transformer architeture and it's great success. However there is more to this model than the simple self-attention mechanism.

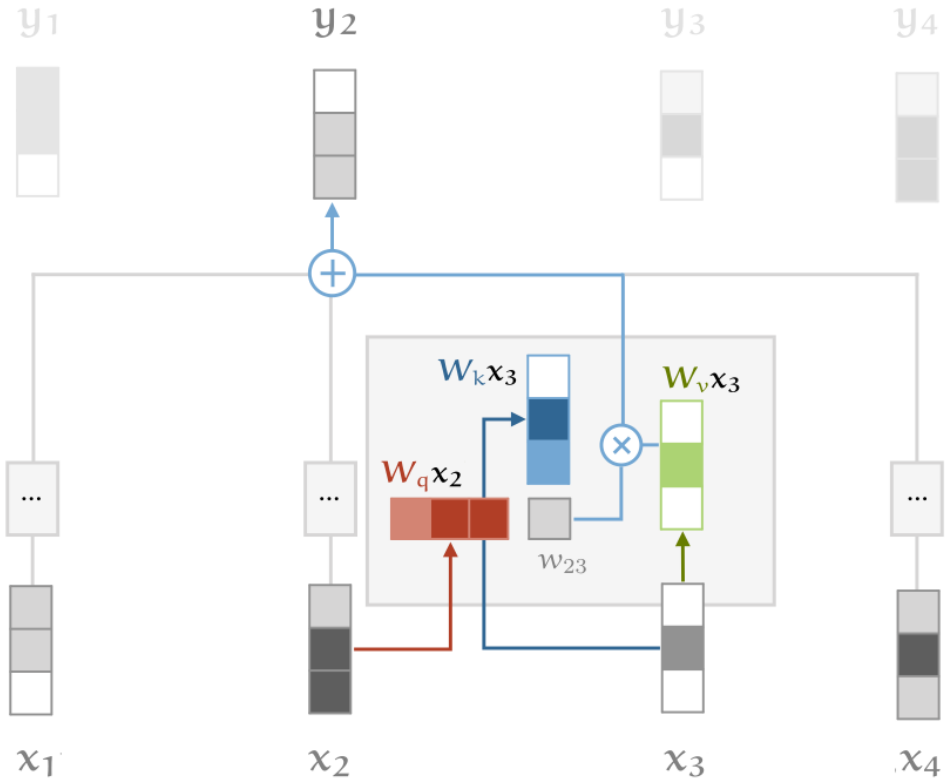

Figure 3.3: Self-attention mechanism with key, query and value linear transformations (softmax not indicated). Image source: http://peterbloem.nl/blog/transformers

First of all, three different roles are filled by each input. These are the query, key and value. An input is a query when it is compared to all other inputs to produce weight for its own output, it is a key when it is compared to other inputs to produce their output weights, and finally it is a value when it is used in the finaly weighted sum to produce the output. Described with equations:

$q_i = W_q x_i \qquad k_i = W_k x_i \qquad v_i = W_v x_i$

$w'_{ij} = q_i^T k_j$

$w_{ij} = softmax(w'_{ij})$

$y_i = \sum_j w_{ij} v_j$

Notice that three weight matrices were added to differentiate the inputs for their different usages. These are just linear transformations, but they add parameters that can be tuned during model training. An illustration of this is shown in Figure 3.3.

Usually the unnormalized wiehgts w'_{ij} are scaled by a factor of the square root of the input vector dimension to avoid saturation and with it impediment to the learning in the softmax function, which is sensitive to very large values.

The last addition to the attention mechanism that needs to be considered is a splitting into multiple heads. This just means that instead of a single focus, there can be multiple ones. Multiple W_q, W_k, W_v matrices are defined which combined with the same input can lead to different values. These are then concatenated and put through a linear

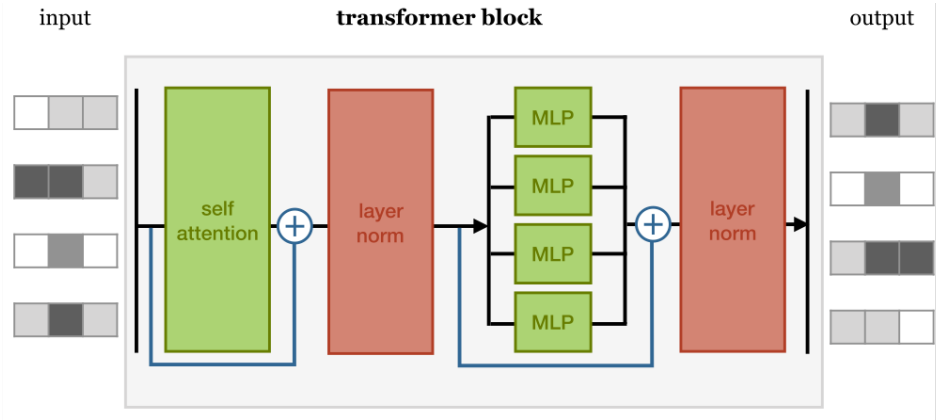

Figure 3.4: A full Transformer block. MLP (multi-layer perceptron) refers to a single fully connected feedforward layer. The plus signs indicate residual connections. Image source: http://peterbloem.nl/blog/transformers

transformation to get back to the original dimensionality.

This setup is called wide self-attention. A computationally cheaper and more memory efficient alternative is spltting up the input into number of attention heads parts and letting each head attend only to its designated chunk of the input.

A Transformer block, in addition to self-attention, also contains a layer normalization after the attention, a fully connected feedforward layer applied to each input individually, followed by another layer normalization. Residual connections are used as well to connect the inputs to the the self-attention and feedforward layers to their outputs before normalization. Layer normalization usually just means scaling and shifting the values to have a mean of zero and unit variance. A complete Transformer block is illustrated in Figure 3.4.

Additionally since the sequential processing aspect of RNNs is lost in this setup, in many cases positional embeddings are added to the inputs to introduce a notion of positions in the sequence for Transformers as well.

Transformers work remarkably well and in the past few years have been gradually replacing RNNs in many contexts, and recently even CNNs in some Computer Vision tasks (e.g. Vision Transformer [**?**]).

Due to the basic Transformer block having relatively few parameters and high efficiency compared to for instance LSTM layers, there has been a tendency to scale up Transformer models to previously unseen sizes, resulting in enormous number of parameteres and breakthrough performance. Some famous examples are Google's BERT [**?**] and OpenAI's GPT-3 [**?**].

The model used in this work is much more modest. It's setup along with the other previously mentioned models is introduced in the next chapter.

Chapter 4

Experiments and Results

Intro TODO

4.1 Classification Task

TODO

4.1.1 Initial Experiments

TODO

4.1.2 Increased Training Time

TODO

4.1.3 Discussion

TODO

4.2 Genotyping Task

TODO

4.2.1 Results

TODO

4.2.2 Discussion

TODO

Bibliography

[1] Campbell P.J., Getz G., Korbel J.O., and et al. Pan-cancer analysis of whole genomes. *Nature*, 578:82–93, 2020.

[2] Bert Vogelstein, Nickolas Papadopoulos, Victor E. Velculescu, Shibin Zhou, Luis A. Diaz, and Kenneth W. Kinzler. Cancer genome landscapes. *Science*, 339(6127):1546–1558, 2013.

[3] Surui Pei, Tao Liu, Xue Ren, Weizhong Li, Chongjian Chen, and Zhi Xie. Benchmarking variant callers in next-generation and third-generation sequencing analysis. *Briefings in Bioinformatics*, 07 2020. bbaa148.

[4] J. Zook, B. Chapman, J. Wang, et al. Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls. *Nat Biotechnol*, 32:246–251, 2014.

[5] J. Zook, D. Catoe, J. McDaniel, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Sci Data*, 3, 2016.

[6] Donald Freed, Renke Pan, and Rafael Aldana. Tnscope: Accurate detection of somatic mutations with haplotype-based variant candidate detection and machine learning filtering. *bioRxiv*, page 250647, 2018.

[7] Donald Freed, Rafael Aldana, Jessica A Weber, and Jeremy S Edwards. The sentieon genomics tools-a fast and accurate solution to variant calling from next-generation sequence data. *BioRxiv*, page 115717, 2017.

[8] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, 31(3):213–219, 2013.

[9] A. McKenna, M. Hanna, E. Banks, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome Research*, 20:1297–1303, 2010.

[10] Sayed Mohammad Ebrahim Sahraeian, Ruolin Liu, Bayo Lau, Karl Podesta, Marghoob Mohiyuddin, and Hugo YK Lam. Deep convolutional neural networks for accurate somatic mutation detection. *Nature communications*, 10(1):1–10, 2019.

[11] Daniel C Koboldt, Qunyuan Zhang, David E Larson, Dong Shen, Michael D McLellan, Ling Lin, Christopher A Miller, Elaine R Mardis, Li Ding, and Richard K Wilson. Varscan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome research*, 22(3):568–576, 2012.

[12] Sangtae Kim, Konrad Scheffler, Aaron L Halpern, Mitchell A Bekritsky, Eunho Noh, Morten Källberg, Xiaoyu Chen, Yeonbin Kim, Doruk Beyter, Peter Krusche, et al. Strelka2: fast and accurate calling of germline and somatic variants. *Nature methods*, 15(8):591–594, 2018.

[13] R. Poplin, P. Chang, D. Alexander, et al. A universal snp and small-indel variant caller using deep neural networks. *Nat Biotechnol*, 36:983–987, 2018.

[14] R. Luo, F.J. Sedlazeck, T. Lam, et al. A multi-task convolutional deep neural network for variant calling in single molecule sequencing. *Nat Column*, 10:998, 2019.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[18] A. Paszke, S. Gross, F. Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] L. Heng, B. Handsaker, A. Wysoker, et al. The sequence alignment/map format and samtools. *Bioinformatics*, 25:2078–2079, 2009.

[20] P. Danecek, A. Auton, Abecasis G., et al. The variant call format and vcftools. *Bioinformatics*, 27:2156–2158, 2011.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[22] P. Bloem. Transformers from scratch, 2019.