Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Genome analysis

Genomic Imputation using Deep Learning

Parth Patwa (605729087) [1], Parth Shettiwar(105710622) [1] and Satvik Mashkaria (405729026) [1],

[1] UCLA, Los Angeles, USA

Abstract

Motivation: Genome imputation refers to the statistical inference of unobserved genotypes. It is an import and and challenging problem in bioinformatics. Most of the previous solutions use statistical methods like SVD to tackle this problem. In this work we frame the genome imputation problem as a language generation task and leverage and train the existing deep learning approaches in an auto-regressive fashion, to achieve superior results compared to previous approaches. We evaluate our models performance on R^2 and accuracy metrics on 1000 Genomes dataset, followed by ablation studies.
Results: Our best method achieves 96.845 ± 1.8 % accuracy and 0.76 ± 0.02 R^2 when MAF $\approx$ 0.1. When MAF $\approx$ 0.6, our method achieves 64.84 ± 1.1 % accuracy and 0.47 ± 0.02 R^2.
Availability: Complete code and data is at the `https://github.com/parth-shettiwar/Genomic_Imputation_using_Deep_Learning`
Contact: parthpatwa,parthshettiwar,satvikm@g.ucla.edu

1 Introduction

To understand the genes which affect the human features, has been long studied in the field of genetics. However, it is a well known fact that human genome consists of atleast 100,000 genes and resequencing the whole genome and then finding associations of each of the genes with external attributes is not feasible. In past, geneticists have recognized the importance of few causal genes which can help to predict the various other genes of human genome. Genome imputations plays a key role here where it performs the task of imputing the unobserved genomic sequence in humans, thereby reducing the overhead of resequncing the whole genomic sequence for each new individual. In addition to this, its well known fact that current approaches for sequencing human genome are error prone. Genomic imputation plays a vital role by imputing these "corrupted" genes, subsequently decreasing the error per basses sequenced.

Traditionally, researchers have tried to use statistical methods like KNN, iterative SVD, matrix completion methods etc. However these methods perform deterministic operations and lack the complexity required to model the genome imputation problem. In recent past, deep learning approaches, like Kojima *et al.* (2019), have shown the efficacy of learning based approaches in this setup.

The past few years have seen a multitude of developments in the language modelling domain using deep learning approaches. LSTM, Transformer, BERT and GPT have been shown to be successful in solving numerous language processing tasks. With the advent of these models, we frame the problem of genomic imputation as a auto-regressive generation task i.e. the model learns to generate the genome, given as input to it, in a sequential manner. We train our own LSTM and transformer based models on the 1000 Genomes dataset Auton *et al.* (2015), and compare their performance with traditional approaches on standard coefficient of determination R^2 and Accuracy metrics.

The rest of the paper is organized as follows: Section 2 mentions the related work to our research, Dataset details are described in Section 3. In section 4, we discuss the approach used by us in this work, and discuss the baselines implementations. This is followed by evaluation metrics in section 5. Finally we report our experimental results in section 6 with implementation details in 7. Section 8 discusses the conclusion and potential future work extending this work.

2 Related work

In this section, we describe the advances both in genome imputation and deep generative models.

2.1 Genomic Imputation

There are several methods established for the task of genomic imputation, notably fastPHASE (Scheet and Stephens (2006)), MACH (Li *et al.* (2010)), IMPUTE2 (Marchini *et al.* (2007)), BEAGLE (Browning and Browning (2009)) and Mendel (Ayers and Lange (2008)). They mainly use HMMs With the recent advancements in Deep Learning and release of larger datasets like Auton *et al.* (2015), researchers have started to explore

it as a tool. Kojima *et al.* (2019). used RNNs for genome imputation. However, the method they used involves saving a weight matrix for every SNP position to be imputed and doesn't look very efficient. We propose a different formulation, which uses LSTMs in a different way and in a different dataset setting.

2.2 Transformer based models

Transformer was introduced by Vaswani *et al.* (2017) in 2017. Transformer achieved state of the art on many generativ NLP tasks. Transformer models utilize the attention mechanism to focus on particular parts of sequence and hence are able to model complex dependencies in sequences.

GPT(generative pre-training) Radford *et al.* (2018), which later became popular in industry also, is a transformer based-model autoregressive generative model. It uses causal self-attention mask to generate novel sequences. The next versions of GPT i.e. GPT-2 and GPT-3 gained a lot of media attentions because of the wonderful results they obtained in text generation tasks. In this work, we test the effectiveness of GPT for the genomic imputation task.

3 Dataset

We use a subset of 1000 genomes dataset Auton *et al.* (2015) for our experiments. It consists of genome sequences from more than 2500 individuals across 26 populations from five continents. We perform our analysis for phased data of chromosome 22 for 500 randomly chosen individuals. We uniformly randomly select $10,000$ SNP columns from the dataset according to the required MAF values described next. To start with, we experiment with two type of datasets: 1) MAF ≈ 0.1 and 2) MAF ≈ 0.6. This is mainly done to observe the behaviour of model in extreme cases. We have also created a robust dataset which contains SNP columns of all kinds of MAF values, and we discuss more about it in the future work section. We further divide our dataset into three parts: 1) reference panel containing 200 individuals 2) train data containing 200 separate individuals 3) test data containing 100 individuals. Note that the reference panel is kept disjoint from the train set because if we train our models on reference panel itself, and the features themselves come from reference panels, large models like LSTM and GPT might just memorize the data i.e. learn something like an identity matrix as weight matrix. As shown in figure 1, the data is divided into chunks of $M = 500$ and we assume that the first $F = 100$ values are observed and the next $M - F = 400$ values are unobserved. This assumption is reasonable as we know genome data doesn't have long temporal dependencies. We call this kind of continuous observed region a "flaking region", consistent with other literature. We also vary the length F of this flanking region in our experiments.

Once we obtained promising results with these primitive datasets, we carefully constructed a larger dataset from 1kg dataset for our next set of experiments. We mainly kept two factors in mind - 1) In the previous method, we are not incorporating the temporal dependence properly. 2) We don't have diverse MAF SNPs in the previous method. We want to have all ranges of SNPs in our new dataset. Keeping these two driving forces in mind, we constructed a new dataset according to the following steps:

1. We have around 1M SNP columns for chromosome 22. We divide them into continuous chunks of length 500. This will give us around 2000 chunks. Out of these, we will consider some chunks in our dataset according to next steps. This will ensure that the temporal dependence in the original dataset is maintained.
2. Now, we use the AF values provided by the 1kg dataset to find average MAF for each chunk. Then we create 5 bins corresponding to the MAf values $[0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1, 0]$ and then randomly choose 20 chunks of length 500 from each bin. This will ensure that we have sufficient SNPs of each MAF type.
3. We extend our reference panel size to 500, training individuals to 500 and testing individuals to 200. Thus, with 1200 individuals and $50,000$ SNPs, this will be a significantly larger dataset than the previous one (about 10 times larger). This extension is useful in removal of possible confounding due to population memberships through randomized control experiment. (Taking all individuals from same population can also be a method to remove confounding but then our training data will be reduced).

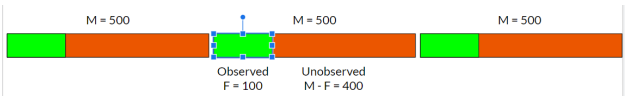

Fig. 1. Division of data into observed and unobserved chunks. Green part indicates the observed "flanking region" and red part indicates unobserved values to be imputed

We believe that this dataset will be a good representative of the entire genome data will be a good judge of which method is better. We are still working on tuning parameters for this dataset and have not fully obtained all the results.

4 Methods

First we describe the baseline methods, then we describe our method.

4.1 Baselines

We implement 3 baselines and compare our results with them:

- Naive randomization: The value of the SNP is randomly predicted from (0,1,2) with equal probability.
- KNNImpute: This method selects the genes of profiles which are similar to to the gene of interest to perform imputation. The imputation is performed by using the data from the K nearest neighbors of the unobserved SNP. Euclidean distance is used to find the nearest neighbors. The predicted value of the SNP is given by majority voting of the values of the nearest neighbors. The number of neighbours to be selected, K, is the hyperparameter in this method. The method has been adopted from. Troyanskaya *et al.* (2001)
- SVDImpute: We use an iterative approach where we find out the SVD of input matrix and consider the first k columns in SVD. We then multiply these low rank matrices to get an approximation for our original matrix. For this method, we assume that the gene matrix was generated by product of low rank matrices.
  Following algorithm from Troyanskaya *et al.* (2001) was adopted:

  1. Given input gene matrix *A*
  2. Compute SVD of $A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$
  3. Take the first k columns of U and V, and corresponding eigenvalues from Σ matrix and compute the imputed A as: $\tilde{A}_{m\times n} = U_{m\times k}\Sigma_{k\times k}V^T_{k\times n}$
  4. Set $A = \tilde{A}$ and go to step 2 for T iterations

  In this method, the number of iterations T and number of columns to be selected k, are the hyperparameters

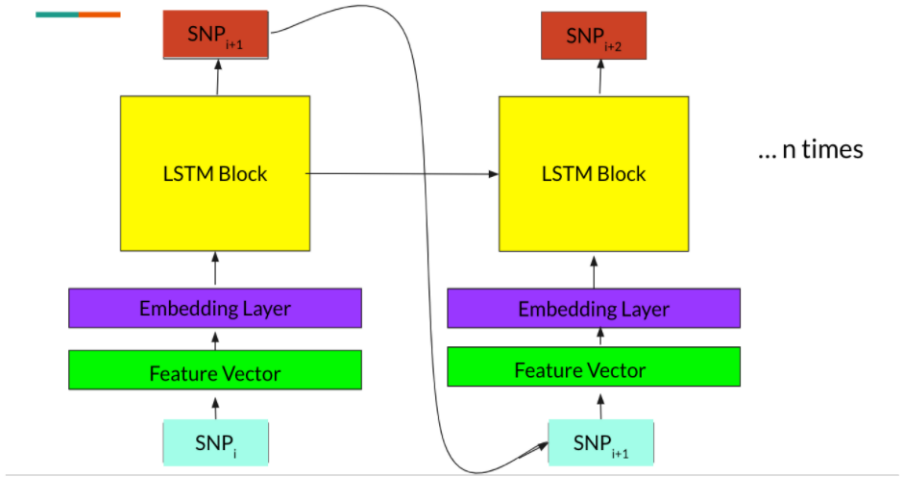

Fig. 2. Architecture diagram of our model. The model is autoregressive, where a cell at a step tries to predict the SNP at the next step.

Note that the baseline methods used here are model-agnostic and typically do not use a reference panel, and we are use a model-based method. To meaningfully compare between both methods, we augment the test matrix with the reference panel before applying these methods, and thus implicitly providing them advantage.

4.2 Deep Learning Algorithms

Statistical baselines are simple and not very good at capturing the complex dynamics of genomes. We train autoregressive generative models for genome imputation. An autoregressive model tries to predict the future values (unobvserved SPN) from past values (observed SNPs). We experiment with 2 Deep Learning neworks - LSTM and GPT. In the section 4.1 we describe the method we used for determining the features and in subsequent sections 4.2 and 4.3 we describe the models we use.

4.2.1 Feature construction

We use the reference panel to construct the feature vector for a SNP position i. Let's denote the allel with no mutations as aa, with one mutation as Aa and with two mutations as AA. Usually we use the encoding $aa = 0$, $Aa = 1$, $AA = 2$. Let's focus on a particular SNP position i. Let's say the genome vector at SNP i in the reference panel is

$$\mathbf{g} = [aa, Aa, AA, Aa, ..., AA] \tag{1}$$

Let's say the genome at position i for the data point, whose feature vector we want to find, is aa. Then in that case, we keep our encoding as it is and our feature vector will simply be $[0, 1, 2, 1, ..., 2]$. If the genome is Aa, we will encode $Aa = 0$ and correspondingly shift all other encoding by one, so our new encoding will be $aa = 2$, $Aa = 0$, $AA = 1$. Similarly, when $AA = 2$, the encoding will be $aa = 1$, $Aa = 2$, $AA = 0$. Mathematically, this can be achieved simply by the formula $(\mathbf{g_e} - G) \mod 3$, where G is the usual encoding of the gene of the data point at SNP i and $\mathbf{g_e}$ is the encoding of reference panel vector $\mathbf{g}$ according to the usual encoding. the subtraction and modulo operator both are element-wise. This feature vector construction is just a simple extension of a method found in literature for constructing feature vectors for haploid data, for example in the work Kojima *et al.* (2019). Note that the length of the feature vector will be equal to the number of individuals in the reference panel.

4.2.2 LSTM

LSTM are a modified form of RNN. Unlike RNNs, they are better at handling longer sequences. LSTMS have 3 gates - input gate, forget gate, output gate. These gates control the flow of information within the LSTM. Figure 2 shows the architecture diagram. We refrain from going into detailed gating mechanism in this report but can be found in the paper Hochreiter and Schmidhuber (1997). The input to the model is a genome. At each cell of LSTM, one gene (in form of feature vector) is inserted and the cell tries to predict the next gene in a generative manner. This will be passed through a linear layer to obtain embedding. The embedding are passed to LSTM cells. Each LSTM takes the embedding as well as the hidden output of previous LSTM as inputs. Each cell of the LSTM is connected to a fully connected layer, which predicts the probabilities of next SNP value (between 0,1,2) using softmax function. We output the value which has the maximum softmax probability. In this way, the model learns to generate genomes. The generated genome is compared with the actual genome to compute loss. In our experiments, we use Cross Entropy Loss with three classes. During training, the correct next gene is force fed to the model at every step. During testing, the observed values are given as input and the unobserved values are generated by the LSTM.

4.2.3 GPT

Generative Pre-trained Transformer (GPT) is an advanced generative model. It is based on transformer (Vaswani *et al.* (2017)) and uses self-attention to learn dense representations. GPT is much better than transformer at handling long sequences and can be trained efficiently in a paralelized manner, unlike LSTMs. It has capability to focus on the important parts of the genome through attention mechanism. Further, it has skip (residual) connections so it is less prone to gradient vanishing problem even in deeper networks. A GPT has the decoder part of the transformer. Each decoder layer has two sub-layers - masked self attention sub-layer and feed forward sub-layer. each sub-layer is followed by layer normalization and skip connection. GPT stacks multiple decoder layers. Like LSTM, GPT is also trained in an autoregressive model. For more details about the model architecture, please refer Radford *et al.* (2018). We hypothesize that GPT should be better than LSTM because of its superior properties.

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

5 Evaluation Metrics

An obvious evaluation metric is accuracy. Accuracy is simply the fraction of genomes correctly imputed. However, this will not be a good evaluation metric for a biased dataset. For example, the dataset with MAF 0.12, 88% of the values will be 0. In this case, a model which always output 0 will also give a very high accuracy of 88%. Thus, we need a better evaluation metric. A commonly used evaluation metric is R^2. There are multiple ways of using R^2 as an evaluation metric. We use the method that we found to be commonly used in literature for this kind of problems. We call this method "Allelic Dosage based Coefficient of Determination".

Let's say the softmax probability output of our model is (p_0, p_1, p_2) which is basically the probabilities of values $(0, 1, 2)$. Dosage of an allel is calculates using the formula $\text{dosage} = 0 * p_0 + 1 * p_1 + 2 * p_2$. For a particular individual, we calculate dosages of output probabilities for all SNPs to be imputed and construct a dosage vector. We then simply regress this dosage vector with the gene values in ground truth and find the coefficient of determination R^2. Thus, we found R^2 value of an individual. We then take average over all the individuals in the test data to find overall R^2 value.

Note that these metrics are more robust than the accuracy because they also incorporate the confidence probability that our model is outputting. For example, no matter our prediction probabilities are $(0.4, 0.3, 0.3)$ or $(0.8, 0.1, 0.1)$, our model will will predict 0 and accuracy will be same, however, R^2 metric discussed above will be better for the latter case, which is ideal. Also, as simple sanity check - consider a model which output all 0's. Accuracy will be 88% on MAF ≈ 0.12 data, however the R^2 value will be 0.0. Note that our baseline methods doesn't output a probability distribution, so in that case we simply consider our probability vector as a one-hot vector with value 1 at the prediction and 0 for other two values. Note that this R^2 value can be negative as the coefficient of determination between two vectors can be negative Pedregosa *et al.* (2011)(though it will be upper bounded by 1.0). For example, for randomized algorithm, the value can be negative.

6 Experimental Results

Tables 1 and 2 show the results of our models for both accuracy and R^2 metric. GPT performs the best, followed by LSTM. KNN is the best performing baseline and is lower than LSTM. As expected, Naive randomization performs the worst. MAF ≈ 0.6 shows about 10% improvement using deep learning methods. For MAF ≈ 0.1 dataset, deep learning methods achieve around 95% percent accuracy. Accuracy is not a good metric here, so we compute R^2 value, and surprisingly the R^2 value for deep learning models are also quite high ($\approx 0.75\%$), which shows that they are giving meaningful predictions. This came as a surprise to us because the model was able to predict spurious 1 between a comparatively long sequence of 0's. Figure 3 show the change in accuracy and R^2 with the change in the length of flanking region for the two datasets. We can see that GPT significantly outperforms all other models consistently.

Note that for the large dataset we created for our next set of experiments, we plan to calculate R^2 values for each MAF bin separately. We expect the R^2 values to be decreasing with increase in MAF. In the current set of experiments, the R^2 values are higher for low MAF, which is counter-intuitive because it is a general perception that lower MAF SNPs are harder to impute. However, note that the entire dataset on which the model is trained is low MAF, and hence, the model is only learning the low MAF behaviour and hence able to perform well. When our dataset contains all kinds of MAF values (for example in our new dataset), then higher MAF SNPs will be easier to impute and we will have a decreasing behaviour of R^2. We plan to conduct this experiment to check this hypothesis.

Method	**Accuracy (MAF $\approx$ 0.1)**	**Accuracy (MAF $\approx$ 0.6)**
Naive Randomization	33.28 ± 0.02	33.39 ± 0.02
SVD	88.26 ± 2.31	39.84 ± 4.68
KNN	91.46 ± 1.23	53.39 ± 2.49
LSTM	96.24 ± 1.6	63.91 ± 1.2
GPT	**96.845 ± 1.8**	**64.84 ± 1.1**

Table 1. Results of our experiments.

Method	R^2 **(MAF $\approx$ 0.11)**	R^2 **(MAF $\approx$ 0.6)**
Naive Randomization	-5.38 ± 1.2	-1.01 ± 1.02
SVD	0.397 ± 0.003	0.12 ± 0.07
KNN	0.552 ± 0.004	0.25 ± 0.08
LSTM	**0.78 ± 0.03**	0.45 ± 0.02
GPT	0.76 ± 0.02	**0.47 ± 0.02**

Table 2. R^2 value. Negative values are possible, refer section 5

7 Implementation details

We use PyTorch for our implementation. We modify the implementation of GPT by HuggingFace (Wolf *et al.* (2020)). We mainly make the following changes - we remove positional encoding from the model and input our own positional encoding (simply $1, 2, ..., T$). We directly feed input embedding to the model in oppose to word tokens in NLP. We use GPU available on Google Colab Pro to train all our models. Our code is available at `https://github.com/parth-shettiwar/Genomic_Imputation_using_Deep_Learnin` On an average, LSTM takes 30 minutes to train on the 10,000 SNPs data. Imputing on 100 individuals takes around 1 hour. Imputation time is high because we have to impute autoregressively, so we have to make forward pass of the model multiple times (one for each SNP column imputed). Training time for GPT is around 1 hour and imputation time is around 1.5 hours. We use 1 LSTM layer, embedding size of 64, and hidden dimension of 12. For more details about architecture and hyper-parameters, please refer the code.

8 Conclusion and Future Work

In this paper, we attempt genome imputation. We use statistical baselines and compare then with two autoregressive generative models, namely LSTM and GPT. The result show that our methods outperform baselines in the dataset we studied. We hope our work opens up possibilities for future research.

In future, we first plan to finalize our results on the new larger dataset we created, described in the section 3. Apart from that, for the current datasets, we speculate that margin of improvement of transformer models can be improved further by proper training with larger data and hyperparameter tuning. Further, we want to Train using different loss functions which would specifically account for the underlying Mean allele frequency in dataset. For example the loss function we are trying currently is the following

$$\text{loss} = \sum_{i=1}^{m} (\text{MAF}_i + \delta) y_i \log(P(y_i))$$

where δ is a small positive constant, y_i is the true value, $P(y_i)$ is the softmax probability of the true value output by the model. This is basically

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

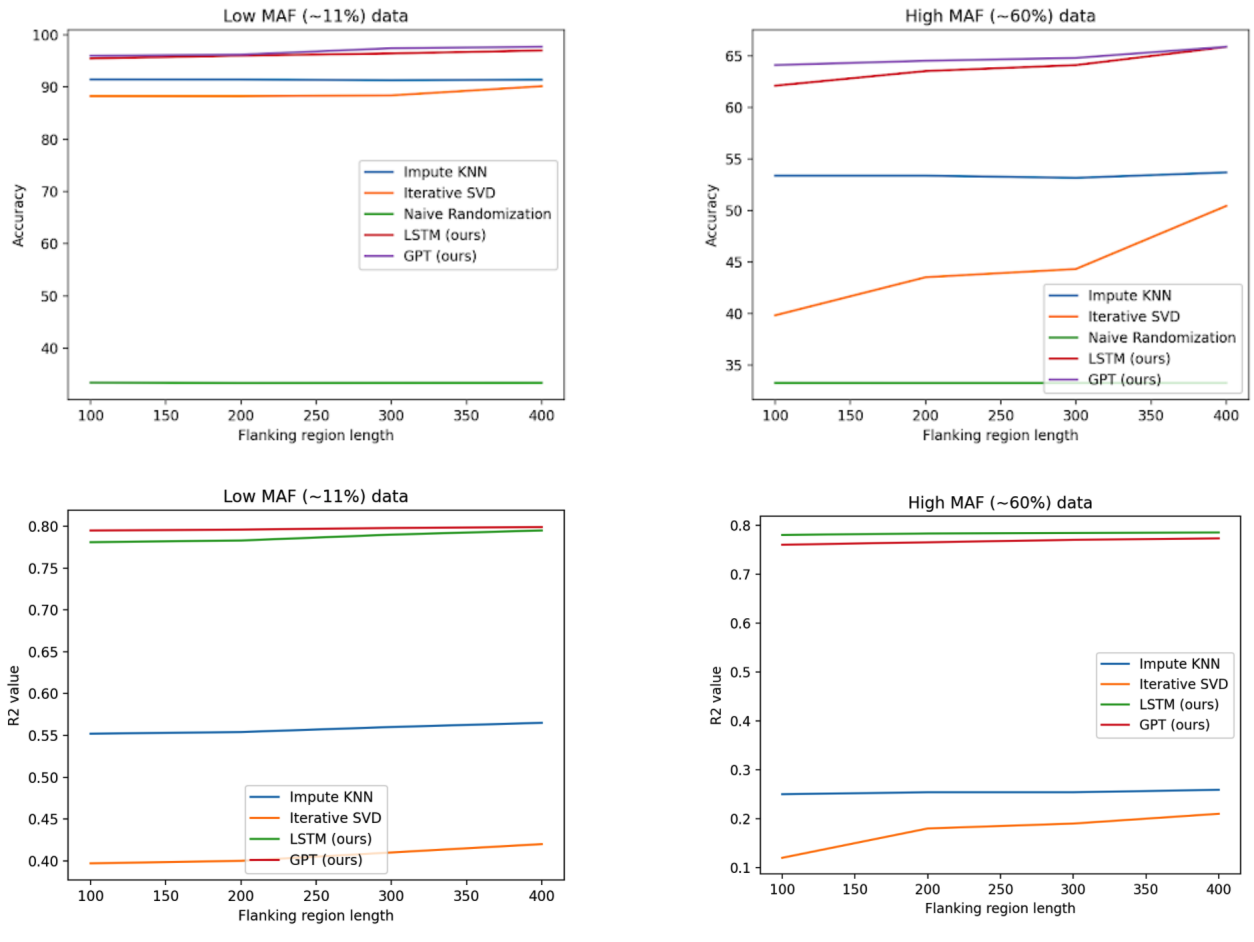

Fig. 3. Experimental results (for R^2 values, we have omitted randomization algorithm for visual purposes

cross entropy loss weighted by MAF. We also plan to conduct ablation studies to observe the effect of various components of our models, like number of layers for LSTM and GPT, feature vector size etc. We also note here that we are only utilizing one directional dependency in gene data. However, genes often have bidirectional dependencies, so we plan to try out models having bidirectional nature, like BERT (Devlin *et al.* (2019)).

Acknowledgements

We acknowledge the guidance provided by our professor Sriram Sankararaman through out this project. We also acknowledge the support given by the TA Boyang Fu.

References

Auton, A. *et al.* (2015). A global reference for human genetic variation. In *Genome Research*. Cold Spring Harbor Laboratory Press.

Ayers, K. L. and Lange, K. (2008). Penalized estimation of haplotype frequencies. *Bioinformatics*, **24**(14), 1596–1602.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Browning, B. L. and Browning, S. R. (2009). A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, **84**(2), 210–223.

Chi, E. C., Zhou, H., Chen, G. K., Vecchyo, D. O. D., and Lange, K. (2013). Genotype imputation via matrix completion. In *Genome Research*. Cold Spring Harbor Laboratory Press.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Kojima, K., Tadaka, S., Katsuoka, F., Tamiya, G., Yamamoto, M., and Kinoshita, K. (2019). A recurrent neural network based method for genotype imputation on phased genotype data. *bioRxiv*.

Li, Y., Willer, C. J., Ding, J., Scheet, P., and Abecasis, G. R. (2010). MaCH: using sequence and genotype data to estimate haplotypes and unobserved genotypes. *Genetic Epidemiology*, **34**(8), 816–834.

Marchini, J., Howie, B., Myers, S., McVean, G., and Donnelly, P. (2007). A new multipoint method for genome-wide association studies by imputation of genotypes. *Nature Genetics*, **39**(7), 906–913.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2018). Language models are unsupervised multitask learners.

Scheet, P. and Stephens, M. (2006). A fast and flexible statistical model for large-scale population genotype data: Applications to inferring missing genotypes and haplotypic phase. *The American Journal of Human Genetics*, **78**(4), 629–644.

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. (2001). Missing value estimation methods for dna microarrays.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture