BIOINFORMATICS

Vol. 00 no. 00 2016
Pages 1–6

Genotype Imputation using Bi-directional Recurrent Neural Network

Deepak Muralidharan [1,*] Manikandan Srinivasan [1,*]
[1]Department of Electrical Engineering, UCLA.

ABSTRACT
With the advancements in high-throughput single-nucleotide polymorphism (SNP) genotyping platforms, there has been an exponential increase in the number of measured genotypes. These advancements have enabled us to understand multi-gene interactions and also develop genetic models that more closely represent the polygenic nature of common disease risk. Studies have shown that the compounded effects of even small amounts of missing data on such analysis is considerable. In this work, we present a bidirectional recurrent neural network based method for imputing missing SNP genotype data. We perform our experiments on a subset of the 1000genomes dataset and our Bi-RNN method accurately predicts around 96% of the missing genotypes on average. We benchmark our results against the MENDEL imputation method and demonstrate that our bi-RNN based imputation method does as well other available methods such as MENDEL even with a vanilla architecture.

Keywords: genotype imputation, bidirectional recurrent neural network, deep learning, missing data imputation, genotype prediction.

1 INTRODUCTION

Genomic studies in recent times have started dealing with massive amounts of data. The availability of high-throughput single-nucleotide polymorphism (SNP) genotyping platforms has resulted in an exponential increase in the number of measured genotypes. However, in spite of the universal occurrence of missing data, downstream analysis such as GWAS usually require complete data for estimation of the model parameters. Many works (Efron *et al.*, 1994), (Little *et al.*, 1992), (Rubin *et al.*, 1996) have suggested that not imputing missing data might have serious consequences on statistical validity and that it might affect the estimability of parameters which need to be generalized to a larger population of inference. The effects of missing data might also result in a wide variability in association wide study results thus making them less replicable. More recently, (Dai *et al.*, 2006) showed that in genetic association studies of common human diseases, imputation results in a better accuracy compared to just ignoring the missing data.

Genotype imputation methods use genotype data in a reference panel to infer about ungenotyped variants in target samples. There are several software packages available for genotype imputation such as fastPHASE (Scheet *et al.*, 2006), MACH (Li *et al.*, 2006), IMPUTE2 (Marchini *et al.*, 2007), BEAGLE (Browing *et al.*, 2009) and Mendel (Ayers *et al.*, 2008). Inspired by the success of neural networks in other domains, in this work, we present a bi-directional recurrent neural network based architecture for imputing missing SNP genotype data. Although there have been some works in the past which use simple feed forward neural networks to predict missing genotype accurately (Sun *et al.*, 2008), a more complicated model neural network model such as RNN can be explored to potentially improve the performance under different situations.

Our paper is organized as follows. Section 2 introduces the reader to artificial neural networks. Section 3 dives deep in the recurrent neural network architecture which forms the heart of our model. In Sections 4 and 5, we explain the problem of genotype imputation and our experimental setup in a more detailed manner. In Section 6, we give an in-depth explanation of our bidirectional RNN modeled specifically to the genotype imputation problem and in Section 7 we discuss briefly about the Robust PCA based (Cands *et al.*, 2011) imputation method against which we compared our results. Finally, in Sections 8 and 9, we present our results and analysis and provide scope for future work in this area.

2 ARTIFICIAL NEURAL NETWORKS

2.1 Artificial Neuron

Artificial neural networks (ANNs) are a family of models inspired by biological neural networks. ANNs are generally presented as systems of interconnected ”neurons” which exchange messages between each other. The artificial neuron receives one or more inputs and generates a weighted sum of these inputs. This weighted sum is then passed through a non-linear activation function such as the sigmoid, tanh, ReLU (rectified linear unit) etc. Although more recently ReLU has been empirically shown to result in better convergence, there is no one best activation function but rather we choose the non-linearity specific to the problem we are trying to solve. Figure (1) describes the structure of an artificial neuron. Here,

- $x1$, $x2$ and $x3$ the inputs
- h is the state of the neuron
- $w1$, $w2$ and $w3$ are the ‘weights’ of the links
- f is called the neural-activation function and typically f is a sigmoid function i.e

$$f(x) = \frac{1}{1+e^{-x}}$$

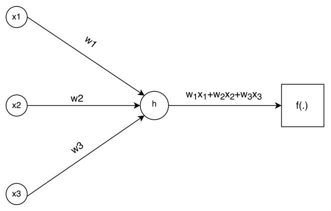

Fig. 1. An artificial neuron

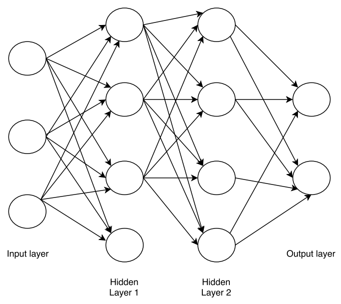

Fig. 2. A simple feed forward network

2.2 Feed Forward Artificial Neural Networks

In this section, we give a brief overview of the architecture of a feed forward neural network. Suppose we have a network as shown in Figure (2). The leftmost layer is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons. The middle layers are called the hidden layers, since the neurons in this layer are neither inputs nor outputs. While the design of the input and output layers of the network is mostly straightforward, the design of the hidden layers is a tricky art. In particular, it is not possible to sum up the design process for the hidden layers with a few rules of thumb. Instead, many design heuristics for the hidden layers have been developed over the years specific to the applications these nets are targeted at.

Although feed forward ANNs have a lot of applications like function approximation, classification, image recognition etc., they have some shortcomings as well. One of their main disadvantages is that the input to the Feed Forward ANN has a fixed dimension/ window length. We know that in applications such as genotype imputation, the Linkage Disequilibrium (LD) is high for adjacent SNPs but comparatively lower for SNPs which are far apart (Marchini *et al.*, 2007). However, this does not take away the fact that there might still be some correlation between these SNPs and we will fail to capture this information if we use a simple Feed Forward ANN of a fixed input dimension as we will not be looking at SNPs beyond a specific window. This motivates us to move towards a different neural network architecture which can take in its inputs dynamically, and hopefully learn to capture correlation statistics that persists across long distant SNPs. In the following sections, we will look at a neural network architecture called the Recurrent Neural Network which exactly does this.

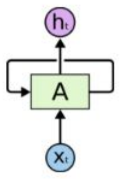

Fig. 3. A simple recurrent neural network

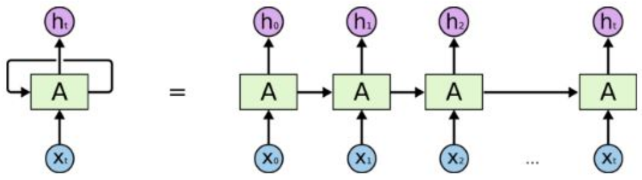

Fig. 4. An unrolled recurrent neural network

3 RECURRENT NEURAL NETWORKS

In simple terms, recurrent neural networks (RNNs) are neural-networks with loops in them, allowing information to persist. Figure (3) illustrates the architecture for a simple RNN, where a chunk of neural network A, looks at some input x_t and outputs h_t. The loop allows for the information to be passed from one 'time'-step of the network to the next.

If we think about carefully, RNNs arent all that different than a normal feed-forward RNNs. They can be thought of as multiple copies of the same network, each passing a message to a successor.These chain-like nature reveals that RNNs are intimately related to sequences and lists. Let us denote the input to a recurrent neural network by $\mathbf{X} = \{x_t\}_{t=1}^{t=N}$, output as $\mathbf{Y} = \{y_t\}_{t=1}^{t=N}$ and states as $\mathbf{H} = \{h_t\}$. It's easy to see the following mathematical relations hold.

$$\begin{aligned} h_t &= \Phi(W_h h_{t-1} + W_x x_t + b_h) \\ y_t &= f(W_y h_t + b_y) \end{aligned}$$

The weights W_h, W_x, W_y, b_h and b_y are computed by minimizing some cost function $J_W(X_{data}, Y_{data})$ through 'back-prorogation' over time.

4 GENOTYPE IMPUTATION PROBLEM

Genotype imputation is the process of filling in the missing genotype data. High linkage disequilibrium (LD) between alleles at different loci shows that there is a strong statistical correlation across SNPs among an individual in a population. This introduces some redundancy in data providing with the opportunities for imputation. Typically a completely sequenced genotype data is available for a bunch of reference individuals. We use this data for filling in SNPs for other individuals for whom the complete sequence isn't known. Figure (5) illustrates this problem, where we have the complete sequence for the reference panel, however entries are missing for inference panel.

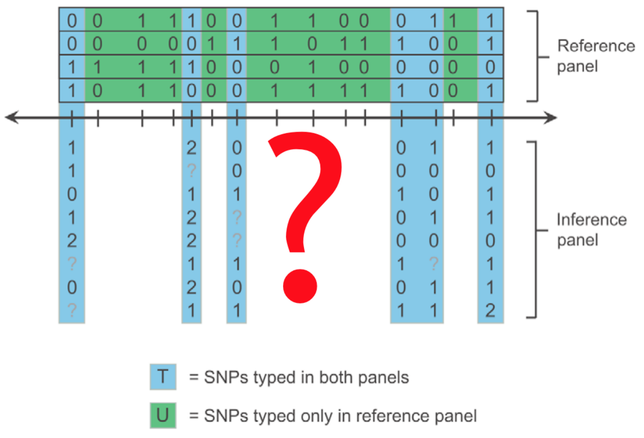
Reference panel
Inference panel
?
T = SNPs typed in both panels
U = SNPs typed only in reference panel

Fig. 5. Problem of Genotype Imputation

5 DATA AND EXPERIMENTAL SETUP

For our experiments, we used chromosome 22 of the 1000genomes dataset (www.1000genomes.org). This dataset consists of haploid data for 1092 individuals in approximately 300k SNP locations. We selected a window of 50 SNPs from SNP9950 - SNP10000 and performed imputation using both the haploid (0/1) and the diploid (0/1/2) versions of the reference data. The reference data was split into three parts. We used 80% of the rows (individuals) for training, 10% for cross validation and the remaining 10% for testing. Although we could have run the imputation engine for a specific population (eg. European population which consisted of 379 individuals), since we wanted to understand the effectiveness of RNNs for a more complex dataset, we decided to include individuals from all populations in the training set for our simulations. As explained in the previous sections, we performed imputation using three algorithms: Uni-directional RNN, Bi-directional RNN and Robust PCA. We bench-marked our results with the MENDEL-IMPUTE method. Tensorflow was used for implementation of uni-RNN and Bi-RNN and the Robust PCA algorithm was implemented in MATLAB and all the four algorithms were run on an i5 processor with 8 GB of RAM.

6 BI-DIRECTIONAL RECURRENT NEURAL NETWORKS BASED IMPUTATION

Conventional RNNs discussed above helps us to get y_t as a function of $x_1, x_2, \dots, x_t$. While this is great for predicting stock-market prices and other time series where the future input is unknown, for the specific problem of Genotype imputation, we have data available from both-sides of position where we have to impute. It turns out that there is a simple solution to using information from the 'future' data.

We have to train another RNN for the reversed sequence $X_R = \{x_t\}_{t=N}^{t=1}$. We combine outputs from both the 'forward RNN' and the 'backward RNN' to calculate the resultant function. This is the idea behind a bidirectional RNN. If we call the states of the 'forward RNN' and the 'backward RNN' as h_t^f and h_t^b at time instant t, then the mathematical model is now fairly easier to describe.

$$\begin{aligned} h_t^f &= \Phi(W_h^f h_{t-1}^f + W_x^f x_t + b_h^f) \\ h_t^b &= \Phi(W_h^b h_{t+1}^b + W_x^b x_t + b_h^b) \\ y_t &= f(W_y^f h_t^f + W_y^b h_t^b + b_y) \end{aligned}$$

Figure (6) shows the architecture of a bidirectional RNN. Here we have a sequence $S = \{s_t\}_{t=0}^{l}$ and reversed sequence. $S' = \{s_t'\}_{t=l}^{0}$.

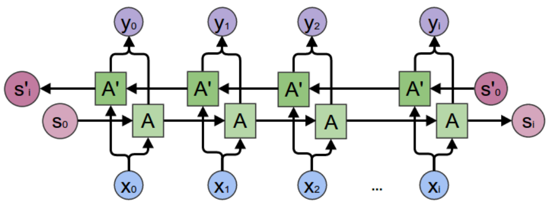

Fig. 6. A simple bidirectional recurrent neural network

In the problem of genotype imputation, we don't have access to x^t if we want to impute SNP position t. With a minor change in the architecture of vanilla bidirectional RNN, we can have

$$y_t = f(W_y^f h_{t-1}^f + W_y^b h_{t+1}^b + b_y)$$

Bidirectional RNNs like other ANNs are capable of learning functions of the input. For genotype imputation, we trained our neural network to learn

$$y_t = P(x_t | x_1, \dots, x_{t-1}, x_{t+1}, \dots, x_N)$$

where y_t, the output of bidirectional RNN at time t, is a column vector of dimension K (K=2 for haploid data and K=3 for diploid data). For training our neural network, we use cross-entropy loss for evaluating our model during training

$$J(X_{data}) = -\frac{1}{NK} \sum_{t=1}^{N} \sum_{k=1}^{K} y_{t,k}^g \log(y_{t,k})$$

where X_{data} is a single training sequence, y_t^g represents ground-truth distribution from data. The parameters of the model are obtained by minimizing $J(X_{data})$ through back-propagation through time. We can also have weighted cross entropy loss function if we want to control family-wise error rate (FWER).

$$J_\alpha(X_{data}) = -\frac{1}{NK}\sum_{t=1}^{N}\sum_{k=1}^{K}\alpha_k y^g_{t,k}\log(y_{t,k})$$

7 ROBUST PCA BASED IMPUTATION

Lets assume that we have a large data matrix D of dimensions $n \times m$ and we know that it can be decomposed into the following

$$D = A + E$$

where A is a low rank matrix and E is sparse error matrix. We have no prior information on how low is the rank of A, or the positions where there are errors. In addition to this, we don't observe the entire matrix D. Let $\Omega_{obs} \subset [n] \times [m]$ be the set of all indices of matrix D that get observed. Let us $\pi_\Omega(.)$ be a function on set of matrices, where $\Omega \subset [m] \times [n]$, defined as

$$\pi_\Omega(D) = \begin{cases} D_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{if } (i,j) \notin \Omega \end{cases}$$

Under some conditions, it is shown in (Cands *et al.*, 2011) that we can recover both A and E exactly from Ω_{obs} entries of D by solving the following optimization problem,

$$\begin{aligned} \text{minimize } ||A||_* &+ \lambda||E||_1 \\ \text{subject to } \pi_{\Omega_{obs}}(A+E) &= \pi_{\Omega_{obs}}(D) \end{aligned}$$

where $||.||_*$ denotes the nuclear norm of a matrix (i.e., the sum of its singular values), $||.||_1$ denotes the sum of the absolute values of matrix entries, and λ is a positive weighting parameter. Usually λ is taken as $1/\sqrt{n_{(1)}}$ where $n_{(1)} = \max(m,n)$. In literature, this is sometime called as solving robust PCA (RPCA). We use inexact augmented Lagrangian method to solve RPCA as in (Lin *et al.*, 2010).

Since there is high linkage disequilibrium (LD) across genotype data, there exists a strong correlation between different SNPs. LD breaks down in locations where there are sequencing errors or mutations. We can create think of a matrix D of dimensions $n \times m$ (n is the total number of individuals in a population and m is the number of SNPs) where $D_{ij} \in \{0,1,2\}$ represents genotype at j^{th} SNP for i^{th} individual. We expect this matrix to decompose into a low rank component (because of LD) A and sparse error component E (because of sequencing errors and mutations). For imputation when entries of this matrix are missing, we use Ω_{obs}, the set of observed entries to recover A and E.

8 RESULTS AND ANALYSIS

As described in the previous section, our simulations were carried out with both the haploid and the diploid versions of the reference panel. For both the haploid and the diploid version of the reference panel, out of the 1092 individuals, 900 were used for training data, 100 for cross-validation and 92 individuals for test.

8.1 Training on Haploid reference panel

For this work, we choose to ignore the case where the alleles from different SNPs are sporadically missing and only consider the scenario when an entire SNP needs to be imputed. The diagram given below gives a visual indication of our test case:

$$\begin{bmatrix} 1&1&0&?&1&1&0&0&1&0\\ 0&1&0&?&0&0&1&0&0&0\\ 1&0&0&?&0&1&1&0&0&1\\ 1&1&1&?&1&1&0&0&0&0\\ 1&0&0&?&1&0&1&1&0&1\\ 0&1&0&?&1&0&1&0&1&0 \end{bmatrix}$$

Since every individual has two homologous chromosomes, for the haploid version have 1800 binary sequences of length 50 for training.

$$X_{test} = \{X_i\}_1^{1800} \quad X_i = (x_1^i, \ldots, x_{50}^i) \quad X_i \in \{0,1\}^{50}$$

8.2 Training on Diploid reference panel

The procedure for training on the diploid reference panel is similar to the one that we have described above. However, since we use diploid data, each genotype is coded as 0,1 or 2 (refer figure below).

$$\begin{bmatrix} 1&2&0&?&1&1&0&0&1&0\\ 0&1&0&?&0&0&1&2&0&0\\ 1&0&0&?&0&2&1&0&0&2\\ 1&1&2&?&1&1&0&0&0&0\\ 2&0&0&?&1&0&2&1&0&1\\ 0&1&2&?&1&0&1&0&1&2 \end{bmatrix}$$

In a more formal way, we can write it as,

$$X_{test} = \{X_i\}_1^{900} \quad X_i = (x_1^i, \ldots, x_{50}^i) \quad X_i \in \{0,1,2\}^{50}$$

Figures (7) and (8) shows the variation in training loss with the number of epochs. For haploid reference, the lowest training loss that we could reach (with increase in cross validation loss) was around 0.1 and for diploid reference, the lowest loss was around 0.25.

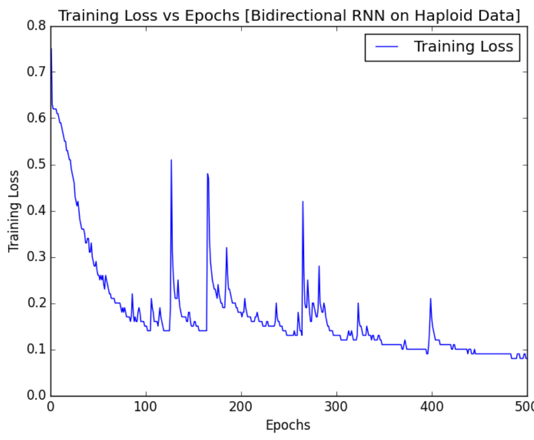

Fig. 7. Training loss vs iterations for bidirectional RNN on haploid data

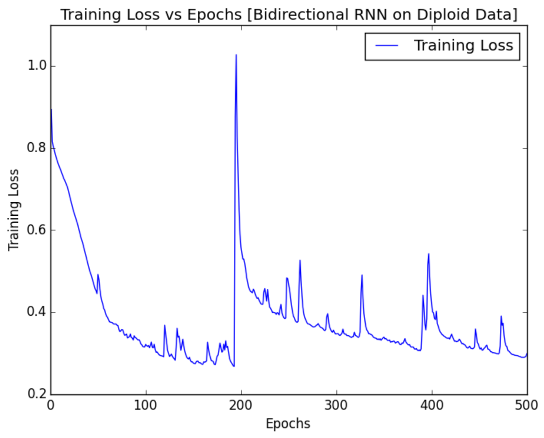

Fig. 8. Training loss vs iterations for bidirectional RNN on diploid data

8.3 Analysis of our results

Figures (9,10) plot the imputation accuracy (number of mismatches) along each SNP (from SNP9950 - SNP10000) for a bunch of 92 individuals by using different imputation methods. We chose the above SNP positions because we wanted a good distribution of 0s and 1s in the SNP positions. From the plots we can clearly see that our Bi-RNN method does as well as other available methods such as MENDEL impute (Ayers *et al.*, 2008) even with a vanilla architecture. The genotypes are predicted with near full accuracy in most of the SNP locations. However, we can observe that the error percentage is high in some locations e.g. SNP9960. Since this trend is present across all imputation methods, we hypothesize that this might be attributed to the fact that these SNP locations might be recombination hotspots. We can also clearly see that Bi-RNN performs considerably well compared to Uni-RNN indicating that information from both sides of the SNP is required for imputation to be accurate.

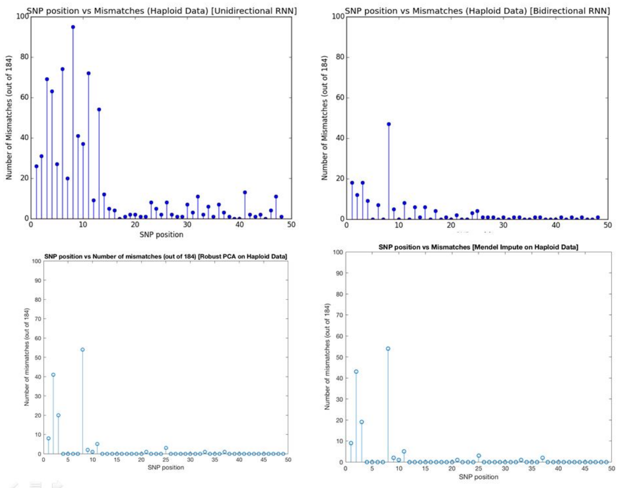

Fig. 9. Number of mismatches vs SNP postion for haploid data

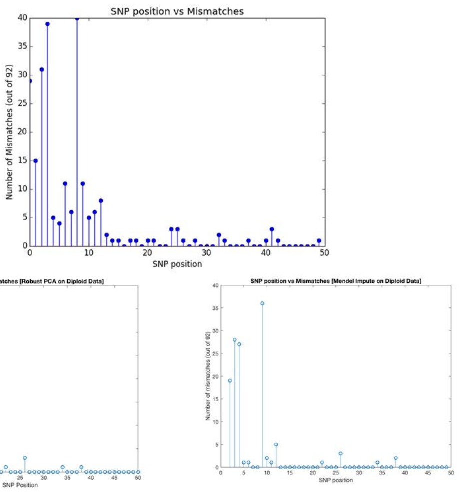

Fig. 10. Number of mismatches vs SNP postion for diploid data

9 EXTENSION - MORE THAN 1 SNP MISSING

So far we looked at exact inference, where we had the conditional probability distribution of the single missing SNP x^t given the other 49 SNPs for each individual. Our imputed value was simply the value of x_t that maximized this distribution. Now let us assume $x_{t_1}, x_{t_2}, \ldots, x_{t_D}$ to be the D SNPs missing for a individual. For imputation, we need the distribution $P(x_{t_1:t_D}|\{x_{1:N}\}\backslash\{x_{t_1:t_D}\})$. But we only have $P(x_{t_1}|\{x_{1:N}\}\backslash\{x_{t_1}\}), \ldots P(x_{t_D}|\{x_{1:N}\}\backslash\{x_{t_D}\})$. In order to approximate sample $x_{t_1:t_D}$ jointly from $P(x_{t_1:t_D}|\{x_{1:N}\}\backslash\{x_{t_1:t_D}\})$, we use Gibbs sampling. We initialize $x_{t_1:t_D}$ randomly at iteration 0. We set 10 iterations for burn in and run Gibbs sampler for 200 iterations. We plot the histogram for each of the sampled x_{t_i} $\forall \quad 1 \leq i \leq D$. We infer each missing x_{t_i} as the mode of the histogram.

10 CONCLUSION AND FUTURE WORK

To the best of our knowledge, this is the fist attempt in using deep learning techniques to solve the problem of genotype imputation. We have demonstrated that our method does as well other available methods even with a vanilla architecture. Deep learning is unreasonably effective and it won't be surprising to see it being used in genetics in near future. We saw this genotype imputation just as a sequence prediction problem where we don't use any other genetic information other than the fact that there is correlation across SNPs. This we believe is a major weakness of our model. In Bi-RNNs, weights W_y^f and W_y^b denote some kind of state-transitioning probability. A richer model could have Weights as functions of genetic data (genetic distance, recombination rate etc.) rather than just constants.This is one direction we can look into. We can also train our neural networks with more data and also look for sequences of length much greater than 50 SNPs.

ACKNOWLEDGEMENT

We would like to thank Professor Sriram Sankararaman for his guidance with our project and extremely helpful suggestions and ideas.

REFERENCES

Sun,Y.V., Kardia,S.L.R, Imputing missing genotypic data of single-nucleotide polymorphisms using neural networks, *European Journal of Human Genetics*, **16**, 487495.

Scheet,P., Stephens,M., A fast and flexible statistical model for large-scale population genotype data: Applications to inferring missing genotypes and haplotypic phase, *Am J Hum Genet*, **78**, 629-644.

Li.Y., Abecasis,G., Rapid haplotype reconstruction and missing genotype inference, *Am J Hum Genet*, **79**, 2290.

Marchini,J., Howie,B., Myers,S., McVean,G., Donnelly,P., A new multipoint method for genome-wide association studies by imputation of genotypes, *Nat Genet*, **39**, 906913.

Browning,B.L., Browning,S.R., A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals, *Am J Hum Genet*, **84**, 210223.

Ayers,K.L., Lange,K., Penalized estimation of haplotype frequencies, *Bioinformatics*, **24**, 1596-1602.

Browning,B.L., Browning,B.R., Genotype Imputation with Millions of Reference Samples, *Journal Name*, **199**, 133-154.

Efron,B, Missing data, imputation, and the bootstrap., *J Am Stat Assoc*, **89**, 463-478.

Little,R.J.A., Regression with missing X's: a review, *J Am Stat Assoc*, **87**, 1227-1237.

Rubin,D.B., Multiple imputation after 18 years, *J Am Stat Assoc*, **91**, 473-489.

Dai,J.Y., Ruczinski,I., LeBlanc.M., Kooperberg.C., Imputation methods to improve inference in SNP association studies, *Genet Epidemiol*, **30**, 690-702.

Marchini,J., Howie,B., Genotype imputation for genome-wide association studies,*Nat Rev Genet*, **11**, 499511.

Chi,E.C., Zhou,H., Chen,G.K., Del Vecchyo,D.O., Lange,K., Genotype Imputation via Matrix Completion, *Genome Res*, **23**, 509-518.

Wen,X., Stephens,M., Using linear predictors to impute allele frequencies from summary or pooled genotype data, *Ann Appl Stat*, **4**, 11581182.

Karpathy,A., Johnson,J., Fei Fei,L., Visualizing and Understanding recurrent neural networks, arXiv:1506.02078.

Hochreiter,S., Schmidhuber,J., Long Short Term Memory, *Neural Computation*, **9**, 1735-1780.

Berglund,M., Raiko,T., Honkala,M., Krkkinen,L., Vetek,A., Karhunen,J., Bidirectional Recurrent Neural Networks as Generative Models, arXiv:1504.01575.

Candes,E.J., Li,X., Ma,Y., Wright,J., Robust principal component analysis?, Journal of the ACM (JACM) 58.3 (2011): 11.

Lin,Z., Chen,M., Ma.Y., The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices, arXiv:1009.5055.