CM229 Final Project: Genotype Imputation using Bi-directional Recurrent Neural Network

Manikandan Srinivasan and Deepak Muralidharan

UCLA, Electrical Engineering Department

6/7/2016

Overview

Artificial Neural Networks

- 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
- Typically, these messages are functions of inputs to these neurons.

Artificial Neuron

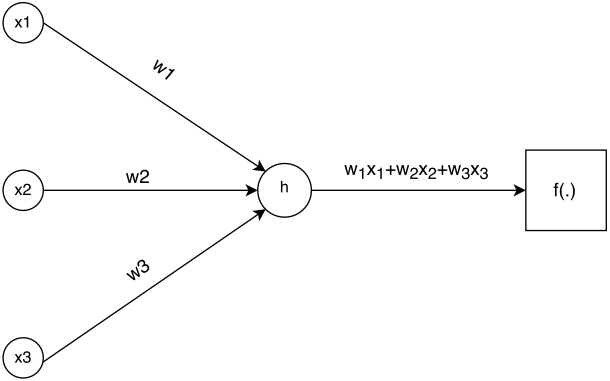


Figure 1: An artificial neuron

Artificial Neuron

- $x1$, $x2$ 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

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

Feed Forward Artificial Neural Networks

- Neurons are connected in-layers to create a feed forward artificial neural network.
- The connections have numeric weights that can be tuned based on experience i.e data, making neural nets adaptive to inputs and capable of learning.

Feed Forward Artificial Neural Networks

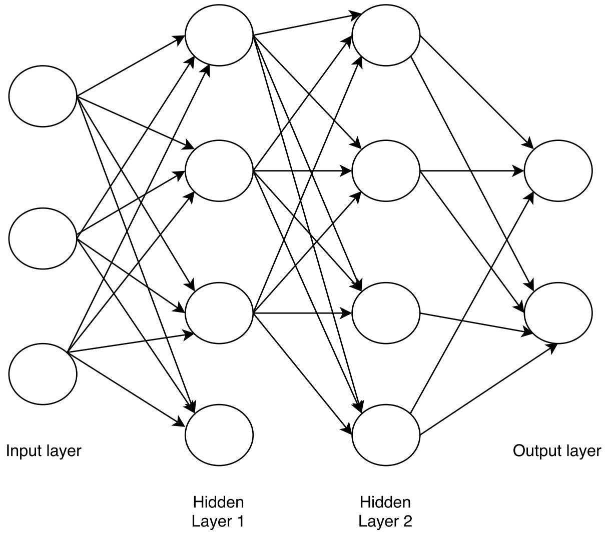


Figure 2: A fully connected feed forward neural network

Disadvantages of Feed-Forward ANNs

- Although FF-ANNs have a lot of applications like function approximation, classification, image recognition etc., they have some shortcomings.
- Imagine we want to classify what kind of event is happening at every point in a movie.
- It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.
- Recurrent neural networks address this issue.

Recurrent Neural Networks

- In simple terms, RNNs are neural-networks with loops in them, allowing information to persist.

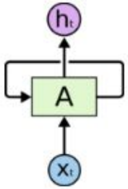


Figure 3: Architecture for RNNs

- A chunk of neural network A, looks at some input x_t and outputs h_t
- A loop allows information to be passed from one 'time'-step of the network to the next.

Recurrent Neural Networks

- RNNs aren't all that different than a normal FF-ANNs.
- They can be thought of as multiple copies of the same network, each passing a message to a successor.

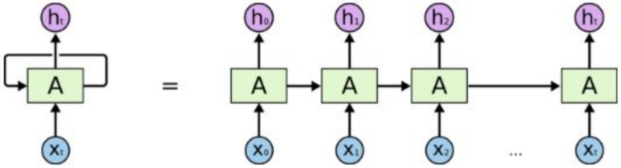


Figure 4: An unrolled recurrent neural network

- These chain-like nature reveals that RNNs are intimately related to sequences and lists.

Mathematical Model for RNNs

- 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.

Need for Bi-Directional RNNs

- The model above helps us to get y_t as a function of $x_1, x_2, \ldots, x_t$.
- 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.
- So how do we use this information from the 'future' data?

Need for Bi-Directional RNNs

- The model above helps us to get y_t as a function of $x_1, x_2, \dots, x_t$.
- 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.
- So how do we use this information from the 'future' data? - **Train another RNN for the reversed sequence** $X_R = \{x_t\}_{t=N}^{t=1}$

Bi-Directional RNNs

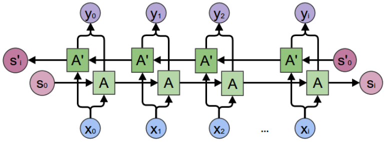


Figure 5: Bidirectional RNN

- Mathematical model is now fairly easier to describe.

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

Problems of Long-term dependencies in RNNs

- In cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

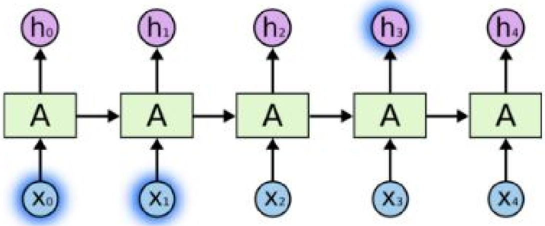


Figure 6: Short-term dependencies are captured by RNNs well

Problems of Long-term dependencies in RNNs

- Unfortunately in practice, as that gap grows, RNNs become unable to learn to connect the information.
- This is due to vanishing or exploding gradients when we to back-propagation over time during training phase.

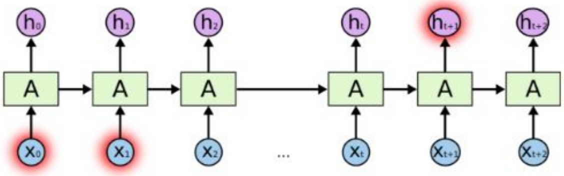


Figure 7: Long-term dependencies are not captured by RNNs in practice

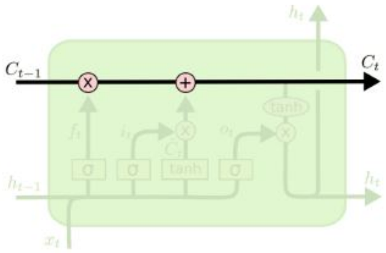
Problems of Long-term dependencies in RNNs

SOLUTION

Long Short Term Memory (LSTM) Networks

Long Short Term Memory Networks

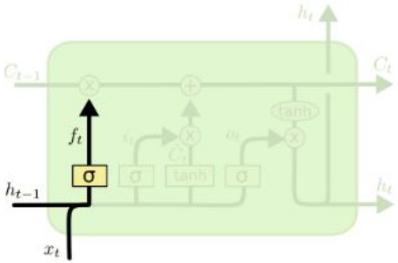
- LSTMs are special kinds of RNNs - designed to avoid the long-term dependency problem
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram
- It runs straight down the entire chain, with only some minor linear interactions



- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Understanding LSTMs

- The first step in our LSTM is to decide what information we're going to throw away from the cell state - *Forget Gate Layer*

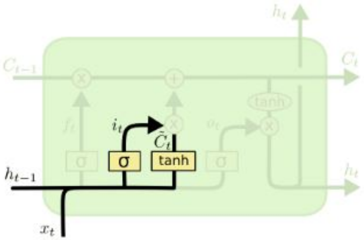


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

- f_t lies between 0 and 1.
- 0 represents completely forget and 1 means to remember the cell state.

Understanding LSTMs

- The next step is to decide what new information we're going to store in the cell state.
- Sigmoid layer decides which values we'll update - *Input Gate Layer*
- Tanh layer creates a vector of new candidate values that could be added to cell state.

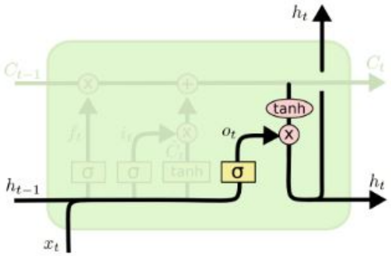


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- Cell state update $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$

Understanding LSTMs

- Finally, we need to decide what we're going to output
- First, we run a sigmoid layer which decides what parts of the cell state we're going to output
- Then, we put the cell state through tanh and multiply it by the output of the sigmoid gate.



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

Robust Principle Component Analysis

- Suppose $D = A + E$, A is low rank and E is sparse error
- In genotype data, E could represent genetic mutations that could occur randomly in a population
- High LD in genotype data $\implies$ A being low rank

Let $\Pi_{obs}(.)$ denote observed indices. A can be recovered exactly by solving

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

Genotype Imputation Problem

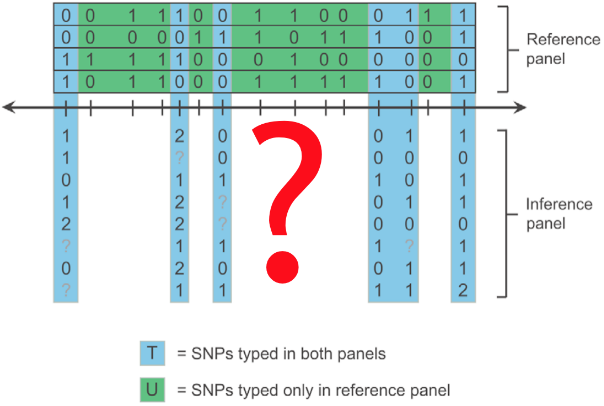

Reference panel
Inference panel
?
T = SNPs typed in both panels
U = SNPs typed only in reference panel

Figure 8: The genotype imputation problem

Genotype Imputation Problem

Genotype imputation has been used to:

- Aid fine-mapping studies,
- To increase the power of genome wide association studies,
- To extract maximum value from existing family samples,
- To facilitate meta-analysis of genomewide association data.

Several software packages available for genotype imputation:

- MaCH (Li and Abecasis 2006; Li et al. 2009, 2010),
- IMPUTE2 (Marchini et al. 2007; Howie et al. 2009),
- BEAGLE (Browning and Browning 2009)
- MENDEL IMPUTE (Ayers and Lange 2008).

Basic idea behind imputation

- High linkage disequilibrium between alleles at different loci
- In other words, there is a strong statistical correlation across SNPs among an individual in a population
- Because of this redundancy, we should possibly be able to recover allele at position t as

$$x_t = g(x_1, \ldots, x_{t-1}, x_{t+1}, \ldots, x_N)$$

Basic idea behind imputation using Bi-RNNs

- We saw these equations for bi-directional RNNs

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

- A minor tweak in the wiring of Bi-RNN and we can have

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

- Bi-RNNs, like other ANNs are capable of learning function of inputs.
- We train our Bi-RNN to learn

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

Training our Bi-RNN

- We use cross-entropy loss for evaluating our model during training

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

where X_{data} is a single training sequence, y^{g}_{t} 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).

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

Robust PCA based imputation

- We use inexact-augmented Lagrangian method to solve the convex optimization

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

- $\Pi_{obs}(D)$ is observed genotype matrix
- Inexact ALU - Proximal point method applied to dual of the above problem

Experimental Setup

- We used chr 22 of the 1000genomes dataset. The data set consists of haploid data for 1092 individuals in approximately 300k SNP locations.
- For our simulations, we selected a window of 50 SNPs from SNP9950 - SNP10000 and performed imputation using both the haploid (0/1) and diploid (0/1/2) versions of the reference data.
- 80% of the rows (individuals) were used as training data, 10% for cross-validation and around 10% was used for testing.

Experimental Setup

- NOTE: Although we could have tested only for a specific population, since we wanted to understand the effectiveness of RNNs for a more complex dataset, we included individuals from all populations in the training set for our simulations.
- 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. We reported run time results after running our algorithm on an i5 processor with 8 GB of RAM.

Training using Haploid data

- Out of the 1092 individuals, 900 were used for training, 100 were used for cross-validation and 92 for testing.
- Every individual has two homologous chromosomes $\implies$ 1800 binary sequences of length 50 for training.

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

- We will train the RNNs to learn the 'conditional distribution'.

How training loss changes with iterations?

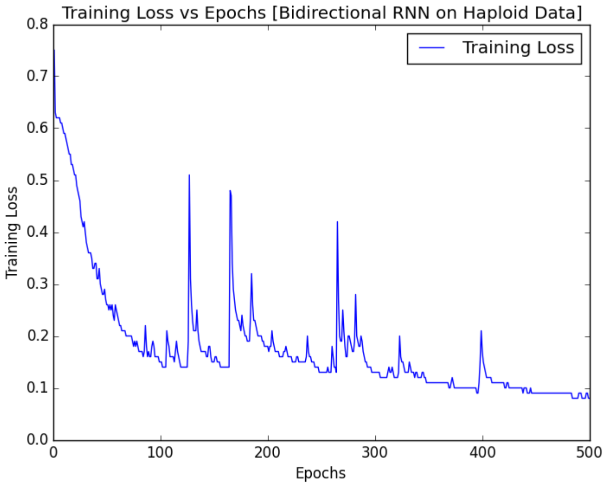


Figure 9: Training loss with iterations for haploid data

Systematic Imputation

Entire SNP is missing for all the 'test' individuals

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

Results

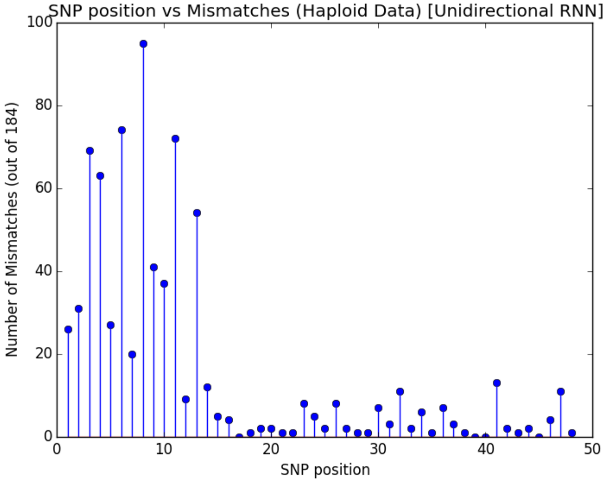


Figure 10: SNP position vs Mismatches Haploid Data- Uni-RNN

Results

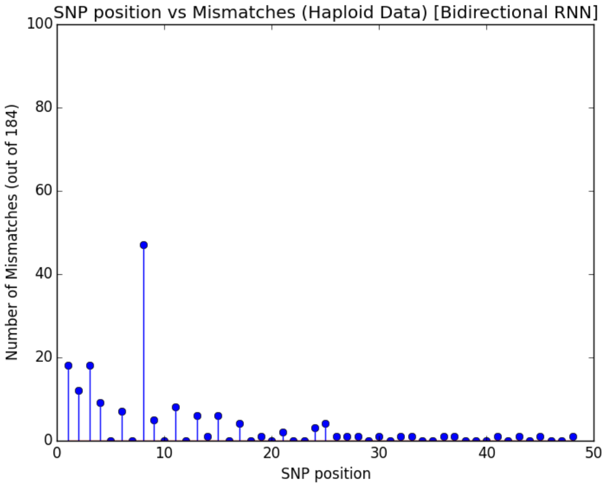


Figure 11: SNP position vs Mismatches Haploid Data- Bi-RNN

Results

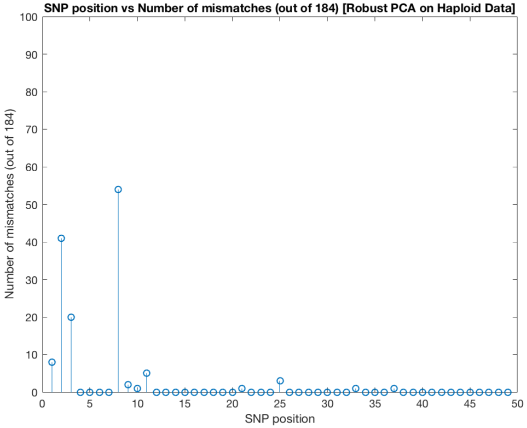


Figure 12: SNP position vs Mismatches Haploid Data- Robust PCA

Results

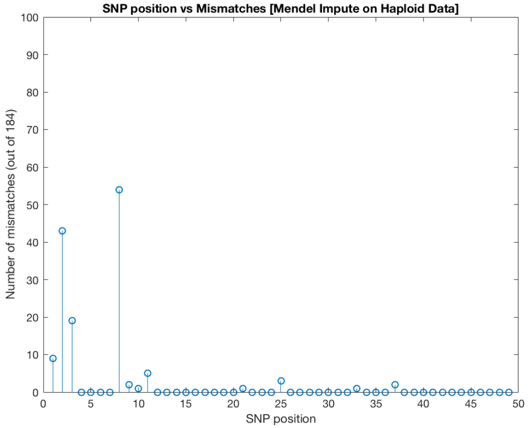


Figure 13: SNP position vs Mismatches Haploid Data- Mendel Impute

Training using Diploid data

- Out of the 1092 individuals, 900 were used for training, 100 were used for cross-validation and 92 for testing.
- We use diploid data $\implies$ each genotype is coded as 0,1 or 2.

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

- We will again train the RNNs to learn the 'conditional distribution'.

How training loss changes with iterations?

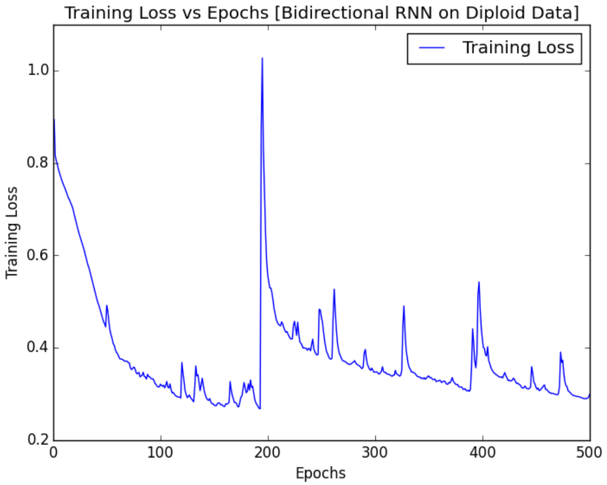


Figure 14: Training loss with iterations for diploid data

Systematic Imputation

Entire SNP is missing for all the ‘test’ individuals

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

Results

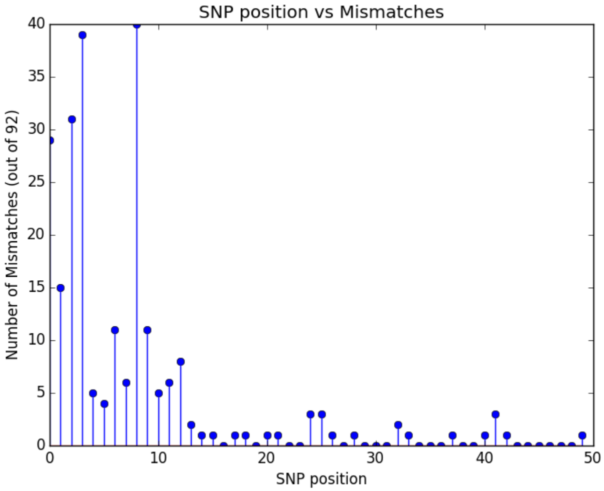


Figure 15: SNP position vs Mismatches Diploid Data- Bi-RNN

Results

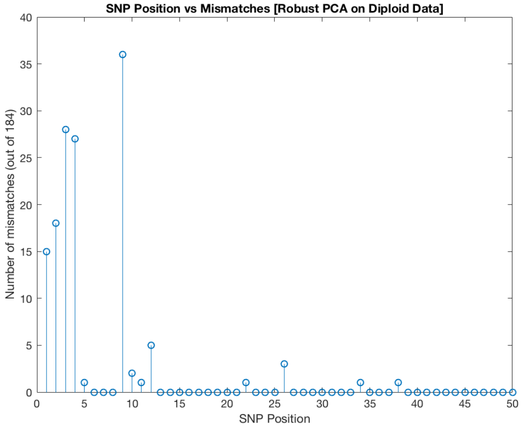


Figure 16: SNP position vs Mismatches Diploid Data- Robust PCA

Results

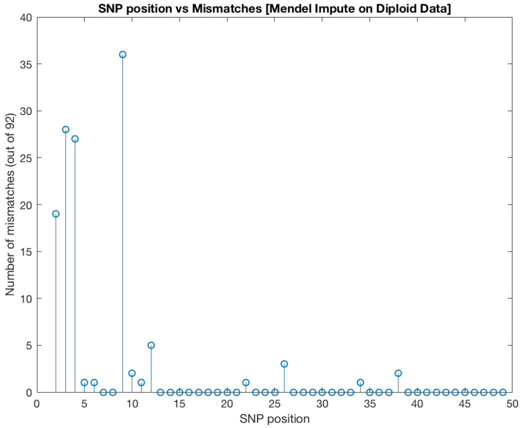


Figure 17: SNP position vs Mismatches Diploid Data- Mendel Impute

Results

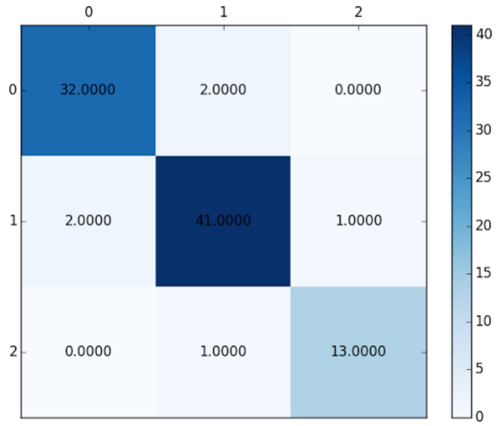


Figure 18: Heat Map for SNP Position 11 - 92 test individuals

Question

How do we do imputation when more than 1 SNPs are missing?

How do we do imputation when more than 1 SNP is missing?

GIBBS SAMPLING

Gibbs Sampling

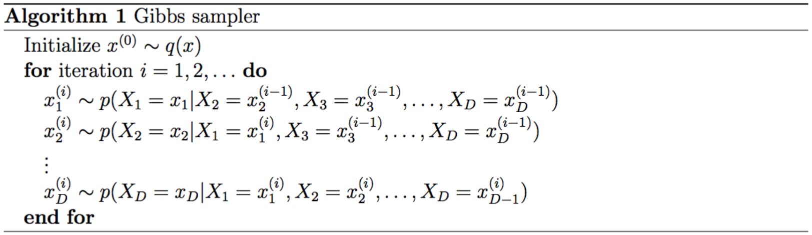
- Now let us assume $x_{t_1}, x_{t_2}, \ldots, x_{t_D}$ be D SNPs missing for a individual

  For imputation, we need to sample $x_{t_1:t_D}$ jointly from
  $$P(x_{t_1:t_D} | \{x_{1:N}\} \setminus \{x_{t_1:t_D}\})$$

- But we only have,

$$P(x_{t_1} | \{x_{1:N}\} \setminus \{x_{t_1}\})$$
$$\vdots$$
$$P(x_{t_D} | \{x_{1:N}\} \setminus \{x_{t_D}\})$$

Gibbs Sampler



```
Algorithm 1 Gibbs sampler
  Initialize $x^{(0)} \sim q(x)$
  for iteration $i = 1, 2, \ldots$ do
    $x_1^{(i)} \sim p(X_1 = x_1 | X_2 = x_2^{(i-1)}, X_3 = x_3^{(i-1)}, \ldots, X_D = x_D^{(i-1)})$
    $x_2^{(i)} \sim p(X_2 = x_2 | X_1 = x_1^{(i)}, X_3 = x_3^{(i-1)}, \ldots, X_D = x_D^{(i-1)})$
    $\vdots$
    $x_D^{(i)} \sim p(X_D = x_D | X_1 = x_1^{(i)}, X_2 = x_2^{(i)}, \ldots, X_D = x_{D-1}^{(i)})$
  end for
```

Figure 19: General purpose Gibbs Sampler

Conclusion and Future Work

- We have demonstrated that our method does as well other available methods even with a vanilla architecture.
- Deep Learning is unreasonably effective - won't be surprising to see it being used in genetics in near future.
- We saw the problem as sequence prediction given genotype data $\implies$ **Major shortcoming**
- In Bi-RNNs, weights W_y^f and W_y^b denote some kind of state-transitioning probability.
- A richer model - Weights as functions of genetic data (Genetic distance, recombination rate etc.) rather than constants.

References

- Browning et.al . *Genotype Imputation with Millions of Reference Samples.*. 2016.
- Marchini et.al. *Genotype imputation for genome-wide association studies*. Nature Review Genetics 2010.
- Chi et.al. *Genotype Imputation via Matrix Completion*. 2012.
- Wen et.al.*Using linear predictors to impute allele frequencies from summary or pooled genotype data*. 2010.
- Karpathy et.al. *Visualizing and Understanding recurrent neural networks*. arXiv 2015.
- Hochreiter et.al. *Long Short Term Memory*. 1997.
- Berglund et. al. *Bidirectional Recurrent Neural Networks as Generative Models*. 2015.