Neural ADMIXTURE: rapid population clustering with autoencoders

Albert Dominguez Mantes[1,2,*], Daniel Mas Montserrat[1], Carlos D. Bustamante[1], Xavier Giró-i-Nieto[2], Alexander G. Ioannidis[1,3,*]

1 Department of Biomedical Data Science, Stanford University, Stanford, CA, United States
2 Signal Theory and Communications Department, Universitat Politècnica de Catalunya, Barcelona, Catalonia, Spain
3 Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA, United States
***** {adomi and ioannidis} @stanford.edu

Abstract

Characterizing the genetic substructure of large cohorts has become increasingly important as genetic association and prediction studies are extended to massive, increasingly diverse, biobanks. ADMIXTURE and STRUCTURE are widely used unsupervised clustering algorithms for characterizing such ancestral genetic structure. These methods decompose individual genomes into fractional cluster assignments with each cluster representing a vector of DNA marker frequencies. The assignments, and clusters, provide an interpretable representation for geneticists to describe population substructure at the sample level. However, with the rapidly increasing size of population biobanks and the growing numbers of variants genotyped (or sequenced) per sample, such traditional methods become computationally intractable. Furthermore, multiple runs with different hyperparameters are required to properly depict the population clustering using these traditional methods, increasing the computational burden. This can lead to days of compute. In this work we present Neural ADMIXTURE, a neural network autoencoder that follows the same modeling assumptions as ADMIXTURE, providing similar (or better) clustering, while reducing the compute time by orders of magnitude. In addition, this network can include multiple outputs, providing the equivalent results as running the original ADMIXTURE algorithm many times with different numbers of clusters. These models can also be stored, allowing later cluster assignment to be performed with a linear computational time.

Introduction

The rapid growth in numbers of sequenced human genomes and the proliferation of population-scale biobanks have enabled the creation of increasingly accurate models to predict traits and disease risk based on an individual's genome. However, different predictive models can be required depending on an individual's genetic ancestry, and this necessitates accurately characterizing an individual's genetic ancestry composition at the individual level [1]. Such characterization is also an essential part of most modern population genetic studies and national biobanking projects. However, many existing algorithms for population genetic analyses struggle to keep up with next generation

sequencing data sets, where both the number of samples and the number of sequenced positions along the genome, are much greater. This has created an intense need for more computationally efficient and accessible methods for detailed large-scale structure analyses. To date the vast majority of association studies, such as genome-wide association studies (GWAS), which look for correlations between genomics sequences and phenotypes, and predictive models like polygenic risk scores (PRS), which indicate genetic predisposition to phenotypes, rely on samples from individuals of European descent, thus excluding most of the world's population and creating a new divide in healthcare [2]. The inclusion of fast, interpretable algorithms that characterize the ancestry makeup of genetic sequences is an important part of facilitating the creation of diverse association studies and expanding the reach of personalized genomic medicine.

A common approach for resolving the population structure within a genetic dataset is to describe each sample by a set of proportional cluster assignments obtained through an unsupervised clustering algorithm. Such methods take as input each individual's sequence of single nucleotide polymorphisms (SNPs), that is, those positions along the genome known to vary between individuals. There are over 10 million known SNPs in the human genome with most of the remainder of the human DNA sequence shared in common between all humans. Such positions are commonly encoded with a binary value, where 0 is used to encode the most common (or reference) variant at that SNP position on the genome, and 1 is used to encode the minority (or sometimes called "alternative") variant. This binary encoding works, because the vast majority of such variable positions have only one alternative to the common variant (are biallelic). The frequency distribution of these variants, and the correlations (linkage disequilibrium) between neighboring SNPs, will vary between populations due to different founder events, migration histories, and genetic drift experienced by those different populations. These differences can lead to predictive models trained on one population failing when faced with sequences from an unseen population. In addition, characterizing differing variant frequencies between populations can provide valuable historical and demographic information such as divergence times, migration events, and historical census size [3].

In this paper we present an autoencoder that implements one of the most widely used clustering methods for population genetics applications: ADMIXTURE [4, 5]. ADMIXTURE was developed as a more computationally efficient solution than Structure, and we now take this pursuit of efficiency to the next generation. Our proposed method, *Neural ADMIXTURE*, follows the same modeling assumptions as ADMIXTURE but re-frames the task as a neural network-based autoencoder, providing much faster computational times, both on GPU and CPU, and higher quality cluster assignments. Additionally, we introduce *Multi-head Neural ADMIXTURE*, which combines multiple decoders to obtain clustering results equivalent to running the original ADMIXTURE repeatedly with different priors for numbers of clusters. Both methods also include a supervised version that performs regular classification given ground truth training labels. The proposed method is fully compatible with the original ADMIXTURE framework, allowing use of ADMIXTURE results as initialization for Neural ADMIXTURE parameters, and vice-versa.

Related work

Model-based clustering methods such as FRAPPE [6], STRUCTURE [7] and ADMIXTURE [4, 5] are the most commonly used unsupervised clustering techniques for analyzing the population structure of genomic sequences. These methods, which resemble probabilistic versions of the Non-negative Matrix Factorization (NMF), decompose each input sequence into a set of cluster assignments and a set of centroids (average SNP variant sequences) for each cluster. Specifically, the cluster assignments

specify what proportion of each ancestry cluster an individual has, while the centroids indicate the SNP variant frequencies at each genetic position for each cluster. These methods allow the user to visualize ancestry composition within genetic datasets, compute how genetically distant different population groups are, and compute statistics that allow the dating of migration history. STRUCTURE [7] makes use of Bayesian models, using a Dirichlet prior for cluster assignments and the cluster centers, trained with Markov chain Monte Carlo (MCMC), making it highly computationally intensive. FRAPPE [6] and ADMIXTURE [4, 5] make use of maximum-likelihood point estimates, obtaining predictions with a quality compared to STRUCTURE, but with much faster computational times. FRAPPE makes use of the Expectation-Maximization algorithm (EM) while ADMIXTURE, as explained in depth in the following section, makes use of a faster block relaxation quasi-Newton optimization technique. However, each method still requires many hours of compute time and is not well suited for modern biobank datasets with tens or hundreds of thousands of samples and millions of SNP feature dimensions.

Several autoencoder architectures similar to our work have been presented. Some examples include the Dirichlet Variational Autoencoder [8], Deep Archetypal Analysis (DeepAA) [9], and Genotype Convolutional Autoencoder (GCAE) [10]. Such networks encode each sample as a point within a convex hull, or as a set of proportions and probabilities. The Dirichlet VAE replaces the commonly used Gaussian prior in the bottleneck by a Dirichlet prior. DeepAA adds constraints to enforce that the bottleneck representation is non-negative and sums to one. GCAE is a convolutional neural network with a Softmax activation in the bottleneck that provides similar clustering results as ADMIXTURE, while being more computationally intensive. Such methods are composed by non-linear encoders and decoders, which deny them the interpretability that our method provides. In fact, our proposed method can be seen as a non-variational version of the Dirichlet VAE with a linear decoder (without bias) and additional constraints in the dynamic range of its weights.

Neural Network-based supervised methods for ancestry classification have also been introduced. Some examples include LAI-Net [11], which provides a high-resolution ancestry estimate along a chromosome sequence, Diet Networks [12], which proposes a genome classifier with different regularization techniques to deal with the high dimensionality of genomic data, and Locator [13], which treats ancestry inference as a geographical prediction problem. While these methods can accurately classify genomes once trained, the ground truth labels used to train these supervised methods are typically hand-crafted reflections of concepts such as ethnicity, or self-reported race of the individual samples. These human-informed classes do not always reflect the full spectrum, or significant clusters, of genetically relevant substructure within and between populations. Therefore, in many genetic applications, it is preferred to use unsupervised methods that do not rely on the complexity of socially-constructed labeling schemes.

ADMIXTURE

In this work, we follow the notation presented in Alexander et al [5]. Note that each individual human has two copies of each chromosome (one paternal and one maternal). Therefore, for a given individual at each genomic position we have the possibility of four different combinations of biallelic SNPs (0/0, 0/1, 1/0, 1/1). It is common practice to sum both maternal and paternal sequences, obtaining a count sequence n_{ij}. In this scenario, an individual i has $n_{ij} \in \{0, 1, 2\}$ copies of the minority SNP j. ADMIXTURE models each individual's sequence, given a fixed number of clusters (population groups) K, as $n_{ij} \sim Bin(2, p_{ij})$, where $p_{ij} = \sum_k q_{ik} f_{kj}$, with q_{ik} denoting the fraction of population k assigned to i, and f_{kj} denoting the frequency of SNPs with value "1" j in

population k. ADMIXTURE applies block relaxation to try to find the parameters Q and F that minimize the following negative log-likelihood function:

$$\begin{aligned} \min_{Q,F} \quad & \mathcal{L}_C(Q,F) = -\sum_{i,j} n_{ij} \log(\sum_k q_{ik} f_{kj}) + (2 - n_{ij}) \log(1 - \sum_k q_{ik} f_{kj}) \\ \text{subject to} \quad & 0 \leq f_{kj} \leq 1 \\ & \sum_k q_{ik} = 1 \\ & q_{ik} \geq 0 \end{aligned} \tag{1}$$

where $Q = (q_{ik})$ and $F = (f_{kj})$. ADMIXTURE also allows an expectation-maximization (EM) based optimization, identical to FRAPPE [6], but this approach is slower than the block relaxation approach [4]. The value of K is typically chosen by using a cross-validation procedure, [5] necessitating runs across a range of values.

ADMIXTURE allows, among others, two valuable optimization alternatives, which are the *projective* analysis and the *supervised* training. In the projective analysis, F is initialized to a previously estimated matrix, and only Q is optimized. The initialization of F may come from previously fit ADMIXTURE models for which the learnt population structure is considered robust. This is especially useful in scenarios where ADMIXTURE is fit in a large dataset and new unseen samples need to be processed. The projective analysis allows estimation of the cluster assignments without the need of fitting the complete model with all the dataset samples. On the other hand, the supervised version requires that some population ancestries are known, so some rows of Q are initialized and fixed to these ancestries, while the rest of the rows of Q and F are optimized normally.

The block relaxation optimization in ADMIXTURE runs much faster than its main competitors, namely FRAPPE [6] and STRUCTURE [7]. Moreover, it can be run in multi-threading mode, greatly boosting the execution time. However, this boost is still insufficient when dealing with either a large number of samples or a large number of SNPs. A neural network version of the algorithm, however, benefits from massive speedups during training (*e.g.* minibatch training, GPU usage), as well as during inference time with a well-chosen architecture.

Neural ADMIXTURE

Network architecture

ADMIXTURE can be formulated as a non-negative matrix factorization problem. Let X denote the training samples, where the features are the alternate allele counts per SNP. Then, $X \approx QF$, where Q are the assignments, F are the alternate allele frequencies per SNP and population, and the negative log-likelihood in Equation (1) is the distance metric between X and QF. This can be naturally translated into the neural network world as a vanilla autoencoder, with $Q = f_\theta(X)$ being the bottleneck estimated by the encoder f_θ and F being the decoder weights themselves. The encoder-decoder architecture is depicted in Figure 1. The fact that Q is estimated at every forward pass and not learnt as a whole for the training data means that, at inference time, we will not have to run the optimization process again, as in ADMIXTURE's projective analysis, but instead perform a simple forward pass.

Note that the restrictions in the optimization problem (Equation (1)) impose restrictions in the architecture. Those relating to Q ($\sum_k q_{ik} = 1$ and $q_{ik} \geq 0$) can be

enforced by applying a softmax activation at the encoder output, making the bottleneck equivalent to the population estimates. Furthermore, while the decoder restriction ($0 \leq f_{kj} \leq 1$) could also be enforced in the architecture itself (*e.g.* applying the sigmoid function to the decoder weights), we have found that it suffices to simply project the weights of the decoder to the interval $[0, 1]$ after every optimization step, which is one of the most common forms of projected gradient descent [14].

We note that, critically, the decoder must be linear and cannot be followed by a non-linearity, as it would break the interpretability of the F matrix and the equivalence between the decoder weights and the cluster centroids (frequencies per SNP and per cluster) would be lost. On the other hand, the encoder architecture is free from constraints, and it may be composed of several neural layers with its corresponding non-linearities, if deemed appropriate. In fact, the proposed Neural ADMIXTURE includes a 512-dimensional non-linear intermediate layer with a ReLU activation before the bottleneck, as well as a batch normalization layer that acts directly on the input.

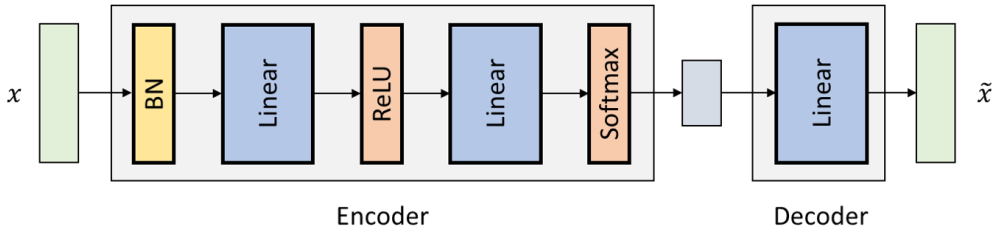


Fig 1. Neural ADMIXTURE architecture. The batch-normalized input sequence is projected into 512 dimensions and processed by a ReLU non-linearity. The cluster assignment estimates Q are computed by feeding the 512-dimensional sequence to a K-neuron layer activated with a Softmax. Finally, the decoder reconstructs the input using a linear layer with weights F. Note that the decoder is restricted to have this architecture to ensure interpretability.

The classical ADMIXTURE model does not exactly reconstruct the input data as a regular autoencoder would do, as the input SNP genotype sequences, $n_{ij} \in \{0, 1, 2\}$, and the reconstructions $p_{ij} \in [0, 1]$, do not have matching ranges. This can easily be remedied by dividing the genotype counts by two, so that now the input data are $n'_{ij} = \frac{n_{ij}}{2} \in \{0, 0.5, 1\}$. Moreover, instead of minimizing $\mathcal{L}_C$ (Equation (1)), we propose minimizing the binary cross-entropy instead, using a penalty term on the Frobenius norm of the first non-linear layer weights, W_1:

$$\mathcal{L}_N(Q, F) = -\sum_{i,j} n'_{ij} \log(\sum_k q_{ik} f_{kj}) + (1 - n'_{ij}) \log(1 - \sum_k q_{ik} f_{kj}) + \lambda ||W_1||_F^2. \quad (2)$$

This regularization term avoids hard assignments in the bottleneck, which helps during the training process and reduces overfitting. In Equation (3) we show that the proposed optimization problem and the classical ADMIXTURE one are equivalent (excluding the regularization term) by using the definitions of the parameters as well as Equations (1) and (2):

$$\begin{aligned}\mathcal{L}_N^{\lambda=0}(Q,F) &= -\sum_{i,j} n'_{ij}\log(p_{ij}) + (1-n'_{ij})\log(1-p_{ij}) \\ &= -\sum_{i,j} \frac{n_{ij}}{2}\log(p_{ij}) + (1-\frac{n_{ij}}{2})\log(1-p_{ij}) = \\ &= -\frac{1}{2}\sum_{i,j} n_{ij}\log(p_{ij}) + (2-n_{ij})\log(1-p_{ij}) = \\ &= \frac{1}{2}\mathcal{L}_C(Q,F).\end{aligned} \tag{3}$$

Note that a perfect reconstruction can be obtained by setting the number of clusters equal to the number of training samples or to the number of input dimensions. However, we want the bottleneck to capture elementary information about the population structure of the given sequences, therefore we make use of low-dimensional bottlenecks.

Decoder initialization

Due to the restriction that non-linearities cannot be used in the decoder, as well as the fairly large number of parameters for a single layer, the decoder weights (and thus, the overall performance of the model) are quite sensitive to the initialization. Common initializations, such as Xavier [15], do not work successfully in this architecture. However, the fact that the decoder is interpretable can be exploited in our favour, as we can try to insert information about the population structure into the initialization in an unsupervised manner. As the entries of (f_{kj}) are the frequencies of the alternate variant of SNP j in population k, f_k almost coincides with the centroid of the samples in population k. This suggests that classical clustering methods can be performed with the results used to initialize the decoder weights.

In the high dimensional space that we work, even fast clustering algorithms such as K-Means would yield high execution times. Instead of clustering in the original feature space, we propose to project the data using Principal Components Analysis (PCA) into a lower dimensional subspace of only a few (2 to 6) principal components and then perform K-Means. PCA is widely used in genetic analyses, as a small number of principal components often explain much of the population substructure of the sequences [16, 17, 18], which is what we are interested in. Hence, to initialize the decoder weights, we propose Algorithm 1:

Algorithm 1: PCK-Means Initialization

Result: Initialization weights W_0
Input : Training data $\mathcal{X}$, Number of populations K
Compute first two principal components on $\mathcal{X}$
$\mathcal{X}_P$ = projection of $\mathcal{X}$ onto first two principal components
C = partition of $\mathcal{X}_P$ given by K-Means
for *every cluster* C_i **do**
  $W_i = \frac{1}{|C_i|}\sum_{x\in C_i} x$
end
Return : Inverse PCA transform of W

Multi-head architecture

In ADMIXTURE, cross-validation must be performed in order to choose the number of population clusters (K), unless specific prior information about the number of population ancestries is known. Furthermore, in many applications, practitioners desire to observe how cluster assignments change as the number of clusters increase. With the

number of both sequenced individuals and variants increasing, the feasible number of different trials of cross-validation rapidly decreases due to its computational cost. As a solution, we propose a variation to Neural ADMIXTURE: the *Multi-head Neural ADMIXTURE* (MNA), which takes advantage of the 512-dimensional latent representation (from now on, shared bottleneck) computed by the encoder. In MNA, the shared bottleneck is jointly learnt for different values of K, $\{K_1 \dots K_H\}$.

Figure 2 shows how the shared bottleneck of the multi-head structure is split into H different heads. The i-th head consists of a non-linear projection to a K_i-dimensional vector, which corresponds to an assignment assuming there are K_i different populations in the data. While every head could be combined and fed through a decoder, this would cause the decoder weights F not to be interpretable. Therefore, every head needs to have its own decoder and, thus, H different reconstructions of the input are performed in every forward pass.

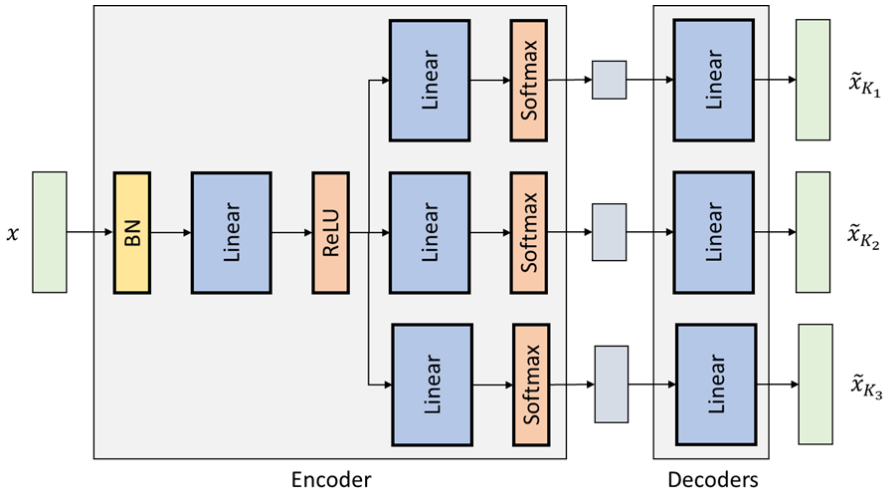


Fig 2. Multi-head Neural ADMIXTURE architecture ($H = 3$).

As we have H reconstructions, we will now have H different loss values. We can train this architecture by minimizing,

$$\mathcal{L}_{MNA}(Q', F') = \sum_{h=1}^{H} \mathcal{L}_N(Q'_h, F'_h), \tag{4}$$

where Q'_h and F'_h are, respectively, the cluster assignments and the SNP frequencies per population for the h-th head. The restrictions of the ADMIXTURE optimization problem (Equation (1)) must be satisfied by Q'_h and F'_h $\forall h \in \{1, \dots, H\}$.

The multi-head architecture allows computation of H different cluster assignments, for different values of K, efficiently in a single forward pass. Results can then be both quantitatively and qualitatively analysed in order to decide which value of K is the most suitable for the data.

Supervised training

ADMIXTURE allows for supervised training by fixing some (or all) entries in the Q matrix. The same approach cannot be applied to the neural network architecture because Q are not learnt parameters (like F) but are instead the activation of the encoder estimated at every forward pass. As a solution, we propose to add a

classification loss to the bottleneck entries. Let Y denote the ground truth ancestries and $\mathcal{L}_{CE}(Q, Y)$ denote the cross-entropy loss. In the supervised version, the optimization problem (assuming a single-head architecture) is formulated as

$$\min_{Q,F} \quad \mathcal{L}_{NS}(Q, F, Y) = \mu\mathcal{L}_N(Q, F) + \eta\mathcal{L}_{CE}(Q, Y), \tag{5}$$

along with the restrictions from Equation (1). Note that unsupervised Neural ADMIXTURE can be seen as a particular case by setting $\eta = 0$. Furthermore, having both losses allows for semi-supervised training, where only part of the training samples have ground truth labels.

In the supervised learning setting, instead of initializing the decoder weights using PCK-Means (Algorithm 1), we can exploit the fact that we know to which population each sample belongs; the decoder weights can simply be initialized as the centroid of each ground truth population, a straightforward computation. Moreover, this initialization will avoid permutation issues (*i.e.* cluster "i" found by PCK-Means may not correspond to population encoded as "i") which would make convergence slower, or even have a negative impact on performance.

Pretrained Neural ADMIXTURE

As a final contribution, we propose a training scheme which allows reusing the results of a previously optimized ADMIXTURE to speedup inference on a novel dataset. This is especially useful when many ADMIXTURE runs have already been computed, so that full retraining, which is computationally expensive with large datasets, is not desirable.

Let F_A denote the F matrix estimated by ADMIXTURE. This matrix can be used in Neural ADMIXTURE so the Q estimates will be similar to those given by ADMIXTURE by (a) initializing the decoder weights W_D to F_A, and (b) freezing W_D and learning Q in a few epochs. While the resulting Q estimates will not be exactly equal to the estimates coming from the classic ADMIXTURE, the computation of cluster assignments will be sped up noticeably. We prove this in the next section.

Experiments

Datasets

We use a comprehensive set of publicly available human whole genome sequences from diverse populations across the world, combining the 1000 Genomes Project [19], the Simons Genome Diversity Project [20], and the Human Genome Diversity Project [21]. We include 550 Africans (AFR), 75 Native Americans (AMR), 651 East Asians (EAS), 496 Europeans (EUR), 27 Oceanians (OCE), 590 South Asians (SAS), and 127 West Asians (WAS). Each category is defined geographically with the American populations additionally filtered to exclude post-colonial groups with recent origins from other continents (eg. Europe and Africa) by considering only samples with over 95% indigenous local ancestry segments. The genome sequences are from anonymous individuals sequenced with their full consent. Samples are randomly split into train and validation using a 80/20 split. We make use of three different datasets: Chm-22, Chm-22-SIM and Chm-1. Chm-22 and Chm-1 include the same set of individuals, but with only the subset of their genome sequence encoded on chromosome 22 and chromosome 1, respectively, considered. Chm-22-SIM is an augmented version of the Chm-22 data: it contains simulated descendants of the real individuals, created using a recombination simulation program, PyAdmix [22] with the simulations performed independently on the train and validation partitions of Chm-22. A total of 400

individuals per ancestry are generated in the training set and 50 in the validation set. Both Chm-22 and Chm-22-SIM have 317,408 SNPs, while Chm-1 has 362,605 variants. Originally, chromosome 1 sequences contained ≈ 5 times more SNPs, but inspired by linkage disequilibrium pruning recommendations in classical ADMIXTURE, we uniformly down-sampled the sequences. While down-sampling is required for long sequences using GPU compute (to avoid memory problems), the complete sequence can be used when using CPU. Furthermore, even when down-sampling the sequence, centroid estimates for all the SNPs can be recovered after training by simply computing the weighted average of the training samples, using as weights the cluster assigments.

Dataset	# Training samples	# Validation samples	# variants (SNPs)
Chm-1	2,072	444	362,605
Chm-22	2,072	444	317,408
Chm-22-SIM	4,872	794	317,408

Table 1. Description of datasets used in experiments

Benchmark setup

ADMIXTURE models are optimized (both in training and projection mode) using 16 threads on an Intel Xeon 2.6GHz (x86_64), with 32 cores and 260GB of RAM. The same architecture is used to train and run inference using Neural ADMIXTURE for the CPU experiments. For the GPU experiments, all networks are trained on a NVIDIA GeForce RTX 2080 Ti. The same type of graphics card is used to run inference on the trained models.

All models are trained using $K = 7$ clusters. Classic ADMIXTURE models are stopped after 20 iterations in training mode. In the projective analysis mode, the software stops upon convergence (which typically happens at less than 15 iterations). Unsupervised networks (including the pretrained Neural ADMIXTURE) are trained using Adam [23] for 10 epochs, except for Chm-22-SIM dataset, where the networks are trained for 20 epochs. The supervised network is optimized over 6 epochs. To run inference on the validation data using the networks, a batch size of 200 sequences is used. All Neural ADMIXTURE networks are trained using a learning rate of 10^{-4} and a batch size of 200. The regularization parameter, λ, is set to 10^{-2}. During our experiments, we observed that the value of λ is correlated with the distribution of the assignments (with $\lambda = 0$ assignments are hard, while large values of λ results in completely uniform assignations). The networks are implemented using the PyTorch framework [24].

To quantitatively evaluate performance, we compute $\mathcal{L}_N^{\lambda=0}$ on the validation data. Inference is understood as running a projective analysis on the validation set in ADMIXTURE, and performing a forward pass to obtain the Q estimates in Neural ADMIXTURE. Furthermore, we use the Adjusted Mutual Information (AMI) [25] between the ground-truth ancestries and the Q estimates of the validation data. The AMI is defined as:

$$AMI(q, y) = \frac{H(q:y) - \mathbb{E}\{H(q:y)\}}{mean(H(q), H(y)) - \mathbb{E}\{H(q:y)\}} \tag{6}$$

where $H(q)$ denotes the entropy of q and $H(q:q)$ denotes the mutual information between q and y. $\mathbb{E}\{H(q:y)\}$ can be calculated using the equation introduced in [26]. A score of one indicates perfect agreement between the assignments, and is the maximum value this metric can accept. The AMI can be computed using the SciKit-Learn [27] package. Note that the AMI is not defined for soft clustering

evaluation, so in order to compute it we assume that the population assigned to a sample corresponds to the cluster to which the individuals most belong.

On the other hand, we also evaluate the soft clustering itself with a new metric, Δ, as

$$\Delta(Q, Y) = \frac{1}{N^2}||QQ^T - YY^T||^2, \tag{7}$$

where Q are the cluster assignments matrix, Y the one-hot encoded ground truth and N the number of samples. This is equivalent to computing the mean squared difference between the covariance matrices of both the estimated and the target population. In case the estimates Q completely agree with Y (up to permutation), then Δ will be 0. The more the disagreement, the higher the value of Δ.

We are interested in these metrics (AMI and Δ) as they are more easily interpretable than the loss function value itself. We are aware that these pseudo-supervised metrics will not give us the true quality of the predictions of the models, as some labels may not be accurate. However, assuming that most of the labels are correct, it will allow us to quantitatively analyze the agreement between the handcrafted labels and the model estimates, and therefore give us an estimation of the quality of the predictions, which we will use to compare between the classical and the neural approaches and draw the appropriate conclusions in case the difference among these metrics is high.

Moreover, let $T_{\mathcal{C}}$ be the execution time for classical ADMIXTURE (in either training or projective analysis) and let $T_{\mathcal{N}}$ be the execution time for the corresponding Neural ADMIXTURE. We define the speedup as the ratio between execution times, $S = \frac{T_{\mathcal{C}}}{T_{\mathcal{N}}}$. When $S > 1$, Neural ADMIXTURE runs faster than ADMIXTURE, and otherwise if $S < 1$.

Single-head results

Dataset	Architecture	$\mathcal{L}_{N_V}^{\lambda=0}$	Δ_T	Δ_V	AMI_T	AMI_V	S_T (GPU)	S_I (GPU)
Chm-1	Classic	**7.506e8**	11	2.5	.85	.81	1.00	1.00
	Neural	7.674e8	**2.1**	**.52**	**.89**	**.88**	5.78 (83.3)	114 (563)
Chm-22-SIM	Classic	**1.195e9**	15	3.0	.90	.84	1.00	1.00
	Neural	1.203e9	**.84**	**.30**	**.94**	**.92**	2.87 (40.9)	137 (642)
Chm-22	Classic	6.644e8	6.7	1.5	.83	.79	1.00	1.00
	Neural	6.802e8	**2.4**	**.67**	**.88**	**.87**	5.08 (81.3)	147 (635)
	Neural (P)	**6.621e8**	6.0	1.5	.79	.78	-	140 (630)
	Classic (S)	6.697e8	**3.4e-7**	.47	1.0	.88	1.00	1.00
	Neural (S)	**6.695e8**	1.1e-5	**.26**	1.0	**.90**	.890 (11.2)	169 (735)

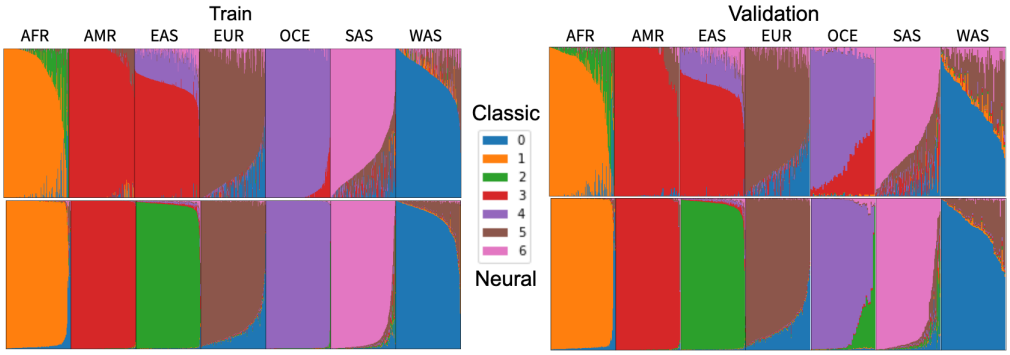
Table 2. Performance comparison of ADMIXTURE and Neural ADMIXTURE. T and V sub-indices denote train and validation, respectively. (P) and (S) distinguish pretrained and supervised modes, respectively, from unsupervised mode. S_T denotes the training speedup and S_I is defined as inference speedup. These two columns denote relative CPU speedups by default. Note that the pretrained version of Neural ADMIXTURE does not include a training speedup as it makes no sense to perform a direct training time comparison.

The results in Table 2 show that, when training on GPU, Neural ADMIXTURE is at least an order of magnitude faster than ADMIXTURE, while achieving very competitive

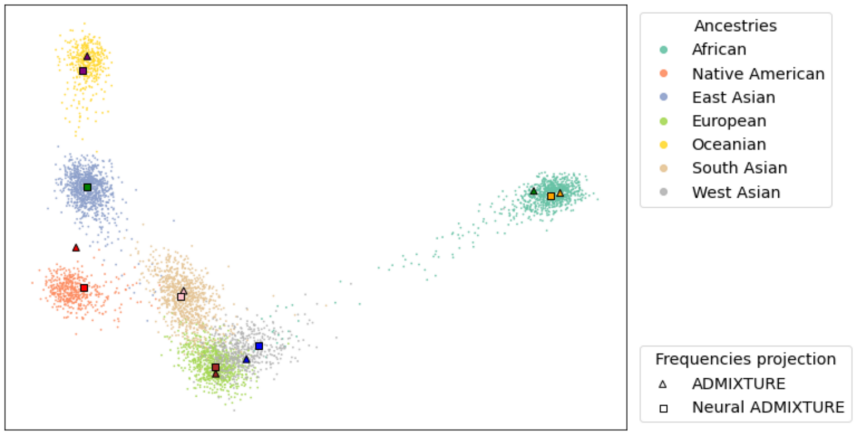
results. Moreover, it is approximately two orders of magnitude faster in inference, both on CPU and on GPU. In the supervised case the CPU version of Neural ADMIXTURE is slower than ADMIXTURE. We believe this is due to overhead introduced by the extra gradient computation on the added supervised loss term.

We also compare ADMIXTURE and Neural ADMIXTURE by visualizing their Q estimates and their respective SNP frequencies F. Figure 3a contains the prediction over the training and validation data of the dataset Chm-22-SIM. The SNP frequencies (that is, the entries in the F matrix) from both models can be observed (projected onto the first two principal components of the training data) in Figure 3b. Qualitatively, Neural ADMIXTURE estimates tend to be more polarized, with many samples being assigned only to a single population, while ADMIXTURE appears to be more conservative. On this dataset ADMIXTURE does not differentiate Native Americans (AMR) and East Asians (EAS), and instead partitions Africans (AFR) into two different different ancestry clusters. Neural ADMIXTURE, however, is able to split EAS and AMR populations. Qualitative examples of the Multi-head Neural ADMIXTURE are shown in the Appendix.

Such qualitative results are on par with the AMI and Δ values in both train and validation (Table 2), and supervised and unsupervised, indicating that the Neural ADMIXTURE provides estimates which are closer to the ground truth labels as compared to classic ADMIXTURE.



(a) Visualization of Q estimates choosing K=7 clusters. Each vertical bar represents an individual sample and bar color lengths represent the proportion of the sample's ancestry assigned to that colored cluster. Classical ADMIXTURE results shown in top row and Neural ADMIXTURE results shown in bottom.



(b) Training data projected onto its first two principal components along with the projected F matrix (cluster centers) for both classical ADMIXTURE and Neural ADMIXTURE.

Fig 3. Results from running both ADMIXTURE and Neural ADMIXTURE trained on Chm-22-SIM.

Multi-head results

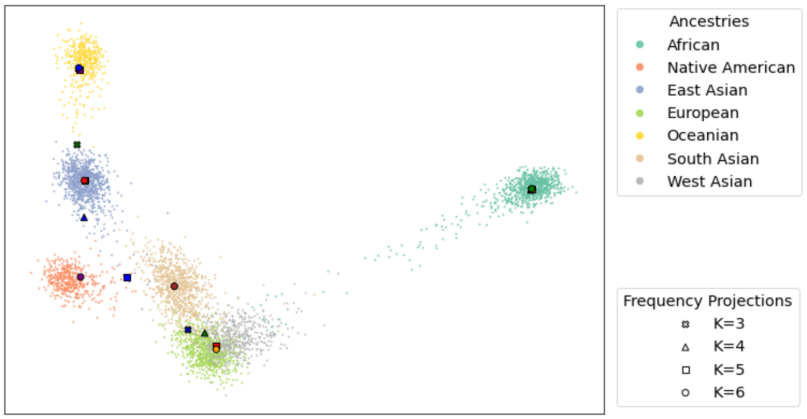
The main advantage of the Multi-head Neural ADMIXTURE (MNA) architecture is that it can perform simultaneous clustering and inference for multiple values of K (number of ancestry clusters). Running many different values for K allows geneticists to obtain a more complete picture of the variation within populations, and is recommended practice. Figure 5 and 6 show examples of the output of MNA for different clustering results under values of K ranging from 3 to 10.

In Figure 5a (K=3) we can observe that EUR, WAS and SAS are combined within the same cluster, while OCE and EAS are clustered together, and AFR has its own cluster. These results reflect the genetic similarity between the respective groups due to their Out-of-Africa migration patterns and subsequent gene flow.

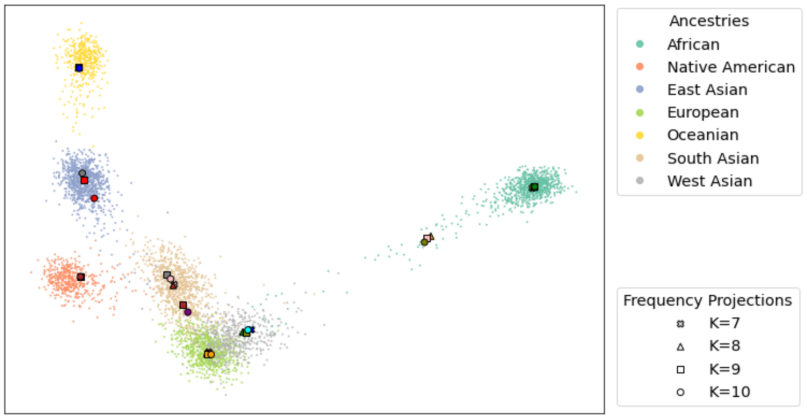
After adding a new cluster (Figure 5b) OCE obtains its own cluster, reflecting the ancient divergence of that population from the others. As more clusters are incorporated, AMR and EAS obtain their own clusters and OCE is divided between a component found predominantly in OCE and a component characteristic of EAS. The latter likely reflects the later migration of Austronesian speakers from East Asia out into the Pacific Islands, where they contributed their ancestry to the Oceanian

inhabitants. A shared component between EUR, SAS and WAS is maintained, independent of the cluster number K, which could be linked to early farmer expansions out of West Asia and into both Europe and South Asia, following the birth of agriculture, as well as to the much later expansion of the Indo-European language family across all of these regions. Other genetic exchanges between these neighboring regions doubtlessly played a role.

With a sufficiently high number of clusters (Figure 6d) a shared component between WAS and some AFR populations appear, which might reflect North African gene flow.

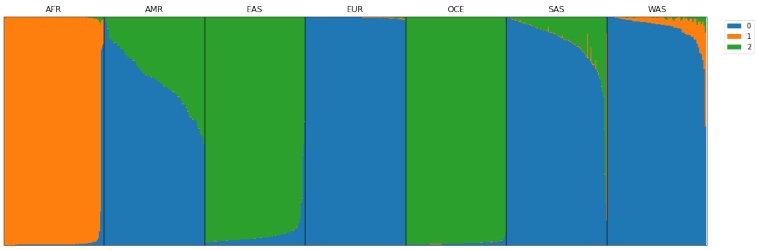


(a) $K \in \{3, 4, 5, 6\}$.

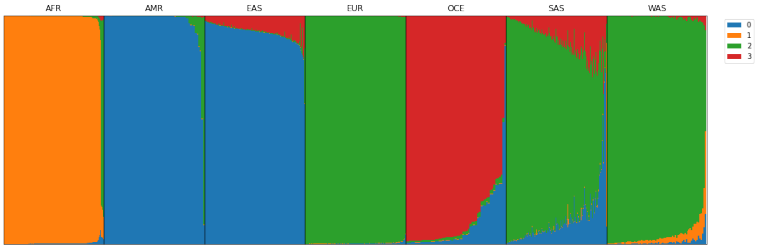


(b) $K \in \{7, 8, 9, 10\}$.

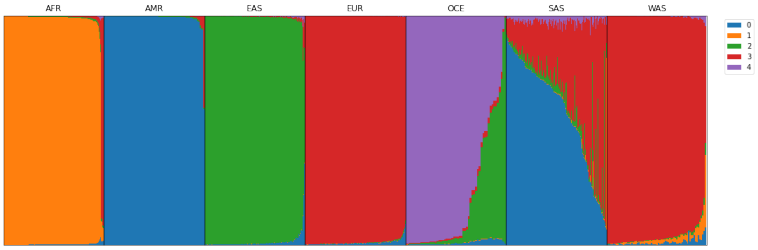
Fig 4. Projected training data along with the decoders of Multi-head Neural ADMIXTURE. Trained on Chm-22-SIM (K from 3 to 10).



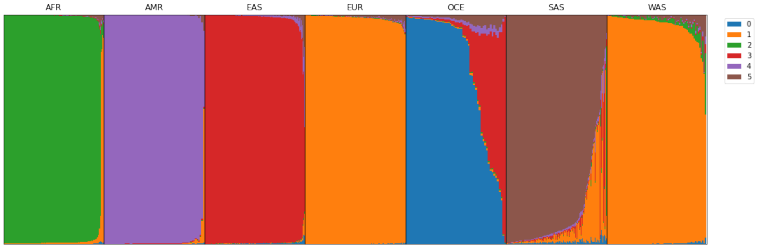
(a) $K = 3$.



(b) $K = 4$.

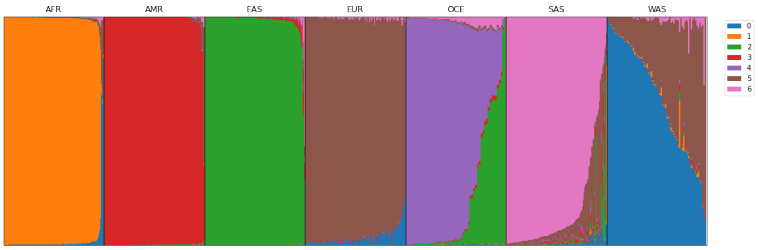


(c) $K = 5$.

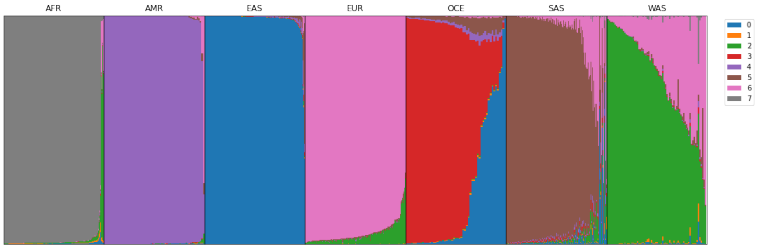


(d) $K = 6$.

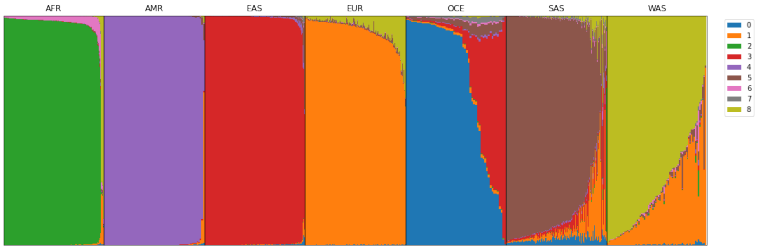
Fig 5. Validation results from Multi-head Neural ADMIXTURE trained on Chm-22-SIM. K from 3 to 6.



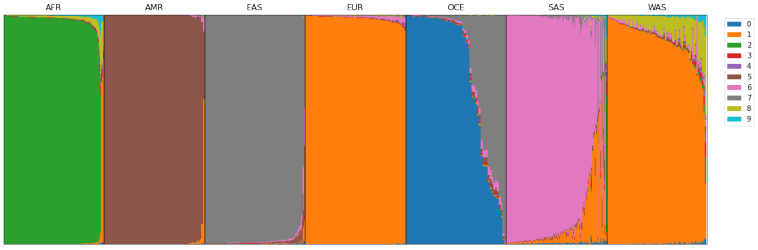
(a) $K = 7$.



(b) $K = 8$.



(c) $K = 9$.



(d) $K = 10$.

Fig 6. Validation results from Multi-head Neural ADMIXTURE trained on Chm-22-SIM. K from 7 to 10.

Discussion

In this paper, we demonstrate that the unsupervised clustering algorithm ADMIXTURE can be re-framed as an autoencoder. This novel framing, which we name Neural ADMIXTURE, allows for the use of common optimization techniques such as SGD or

Adam and provides rapid inference through the encoder, two orders of magnitude faster than original ADMIXTURE algorithm. Furthermore, by adding more heads, multiple estimates with different priors on the cluster number (K) can be performed simultaneously, reducing overall compute time still further. This approach, combined with the use of GPU compute, can enable rapid results on even large modern biobanks.

Acknowledgments

This work was supported in part by NIH grant 7U01HG009080.

References

1. Martin AR, Kanai M, Kamatani Y, Okada Y, Neale BM, Daly MJ. Clinical use of current polygenic risk scores may exacerbate health disparities. Nature genetics. 2019;51(4):584–591.
2. Morales J, Welter D, Bowler EH, Cerezo M, Harris LW, McMahon AC, et al. A standardized framework for representation of ancestry data in genomics studies, with application to the NHGRI-EBI GWAS Catalog. Genome biology. 2018;19(1):1–10.
3. Nielsen R, Akey JM, Jakobsson M, Pritchard JK, Tishkoff S, Willerslev E. Tracing the peopling of the world through genomics. Nature. 2017;541(7637):302–310.
4. Alexander DH, Novembre J, Lange K. Fast model-based estimation of ancestry in unrelated individuals. Genome research. 2009;19(9):1655–64. doi:10.1101/gr.094052.109.
5. Alexander DH, Lange K. Enhancements to the ADMIXTURE algorithm for individual ancestry estimation. BMC bioinformatics. 2011;12:246. doi:10.1186/1471-2105-12-246.
6. Tang H, Peng J, Wang P, Risch NJ. Estimation of individual admixture: analytical and study design considerations. Genetic epidemiology. 2005;28(4):289–301.
7. Pritchard JK, Stephens M, Donnelly P. Inference of population structure using multilocus genotype data. Genetics. 2000;155(2):945–59.
8. Joo W, Lee W, Park S, Moon IC. Dirichlet variational autoencoder. Pattern Recognition. 2020;107:107514.
9. Keller SM, Samarin M, Torres FA, Wieser M, Roth V. Learning extremal representations with deep archetypal analysis. International Journal of Computer Vision. 2021;129(4):805–820.
10. Ausmees K, Nettelblad C. A deep learning framework for characterization of genotype data. bioRxiv. 2020;.
11. Montserrat DM, Bustamante C, Ioannidis A. LAI-Net: Local-Ancestry Inference With Neural Networks. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2020. p. 1314–1318.
12. Romero A, Carrier PL, Erraqabi A, Sylvain T, Auvolat A, Dejoie E, et al. Diet networks: thin parameters for fat genomics. arXiv preprint arXiv:161109340. 2016;.
13. Battey CJ, Ralph PL, Kern AD. Predicting geographic location from genetic variation with deep neural networks. ELife. 2020;9:e54507.
14. Lin CJ. Projected Gradient Methods for Nonnegative Matrix Factorization. Neural Computation. 2007;19:2756–2779. doi:10.1162/neco.2007.19.10.2756.
15. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics; 2010.
16. Novembre J, Johnson T, Bryc K, Kutalik Z, Boyko AR, Auton A, et al. Genes mirror geography within Europe. Nature. 2008;456(7218):98–101. doi:10.1038/nature07331.

17. Patterson N, Price AL, Reich D. Population structure and eigenanalysis. PLoS genetics. 2006;2(12):e190.

18. Price AL, Patterson NJ, Plenge RM, Weinblatt ME, Shadick NA, Reich D. Principal components analysis corrects for stratification in genome-wide association studies. Nature genetics. 2006;38(8):904–9.

19. 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature. 2015;526(7571):68.

20. Mallick S, Li H, Lipson M, Mathieson I, Gymrek M, Racimo F, et al. The Simons genome diversity project: 300 genomes from 142 diverse populations. Nature. 2016;538(7624):201–206.

21. Bergström A, McCarthy SA, Hui R, Almarri MA, Ayub Q, Danecek P, et al. Insights into human genetic variation and population history from 929 diverse genomes. Science. 2020;367(6484).

22. Kumar A, Montserrat DM, Bustamante C, Ioannidis A. XGMix: Local-Ancestry Inference With Stacked XGBoost. bioRxiv. 2020;.

23. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. In: Bengio Y, LeCun Y, editors. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings; 2015.Available from: http://arxiv.org/abs/1412.6980.

24. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R, editors. Advances in Neural Information Processing Systems 32. Curran Associates, Inc.; 2019. p. 8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

25. Vinh NX, Epps J, Bailey J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. The Journal of Machine Learning Research. 2010;11:2837–2854.

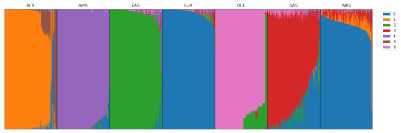
26. Vinh NX, Epps J, Bailey J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: Proceedings of the 26th Annual International Conference on Machine Learning; 2009. p. 1073–1080.

27. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825–2830.

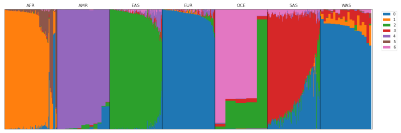
Appendix A: Results

The order of the labels in all figures are the same as in Figure 3a.

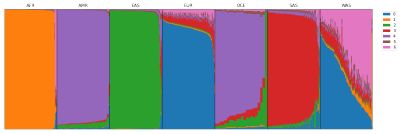
Chm-1 (unsupervised)



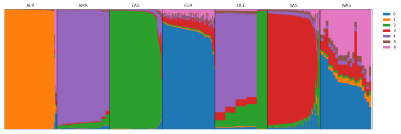
(a) Classical - Training.



(b) Classical - Validation.



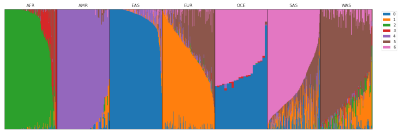
(c) Neural - Training.



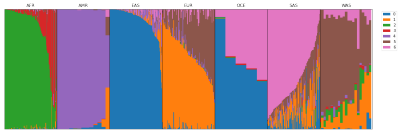
(d) Neural - Validation.

Fig 7. Q estimates on Chm-1 (unsupervised mode).

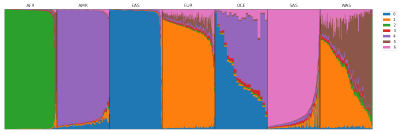
Chm-22 (unsupervised)



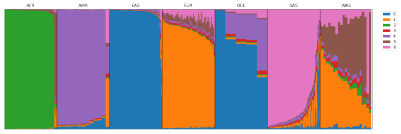
(a) Classical - Training.



(b) Classical - Validation.



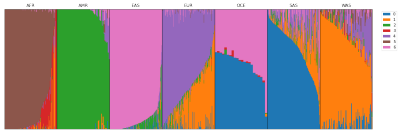
(c) Neural - Training.



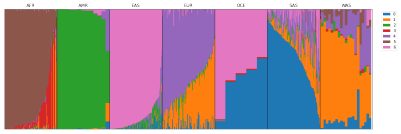
(d) Neural - Validation.

Fig 8. Q estimates on Chm-22 (unsupervised mode).

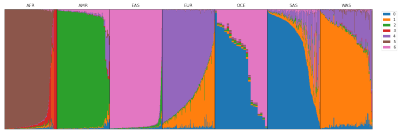
Chm-22 (pretrained)



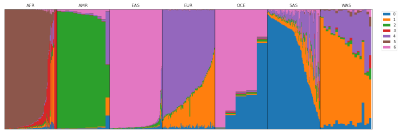
(a) Classical - Training.



(b) Classical - Validation.



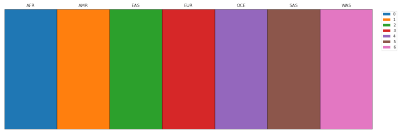
(c) Neural - Training.



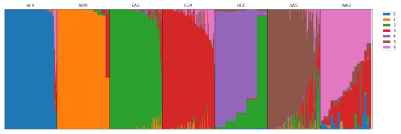
(d) Neural - Validation.

Fig 9. Q estimates on Chm-22 (pretrained mode).

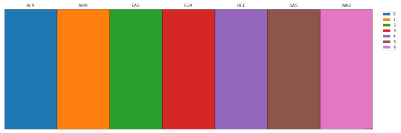
Chm-22 (supervised)



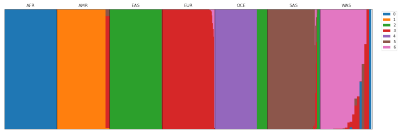
(a) Classical - Training.



(b) Classical - Validation.



(c) Neural - Training.



(d) Neural - Validation.

Fig 10. Q estimates on Chm-22 (supervised mode).