

# Machine learning and deep learning for demographic and selection inference

Flora Jay, CNRS, LISN  
Matteo Fumagalli, QMUL  
Jean Cury, Erik Madison Bray, LISN

+ credits for some slides: T Sanchez



# About us

Flora Jay (CNRS) + Matteo Fumagalli (QMUL)

**EvoGenomics.AI**

`www.evogenomics.ai`

sign up to the mailing list for seminars and training opportunities  
(e.g., 20th June: Jonas Meisner)

---

# Outline

Machine Learning: basic concepts and terminology

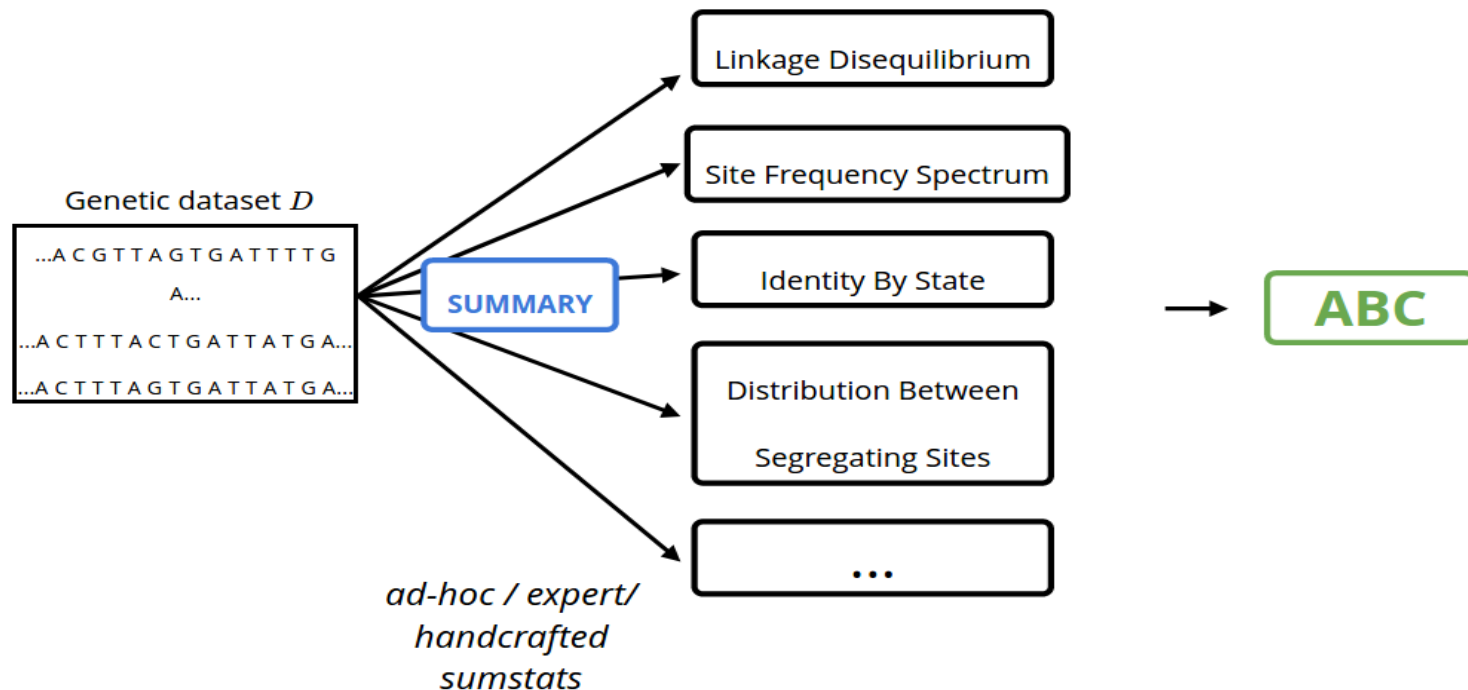
ML, application to popgen ; neural networks

- I. From ABC to deep learning for population genetics
- II. Learning directly from SNP data with neural networks
- III. Dissecting two published networks for effective population size inference
- IV. Opening on applications of unsupervised deep learning to popgen
- V. Tonight's hands-on: building/training/re-using DNNs for population genetics (demography/selection) inference with dnadna

# Approximate Bayesian Computation:

## likelihood free inference based on simulations

- Data summarized by **handcrafted summary statistics**
- **Real and simulated** summary statistics are **compared**
- The comparison informed on the likely demographic scenario
- Application to demography: Boitard et al 2016 , Jay et al 2019 and many other works



Here and in all DL methods presented afterwards training is based on large datasets of simulated data with labels (i.e. for which we know the evolutionary parameters)

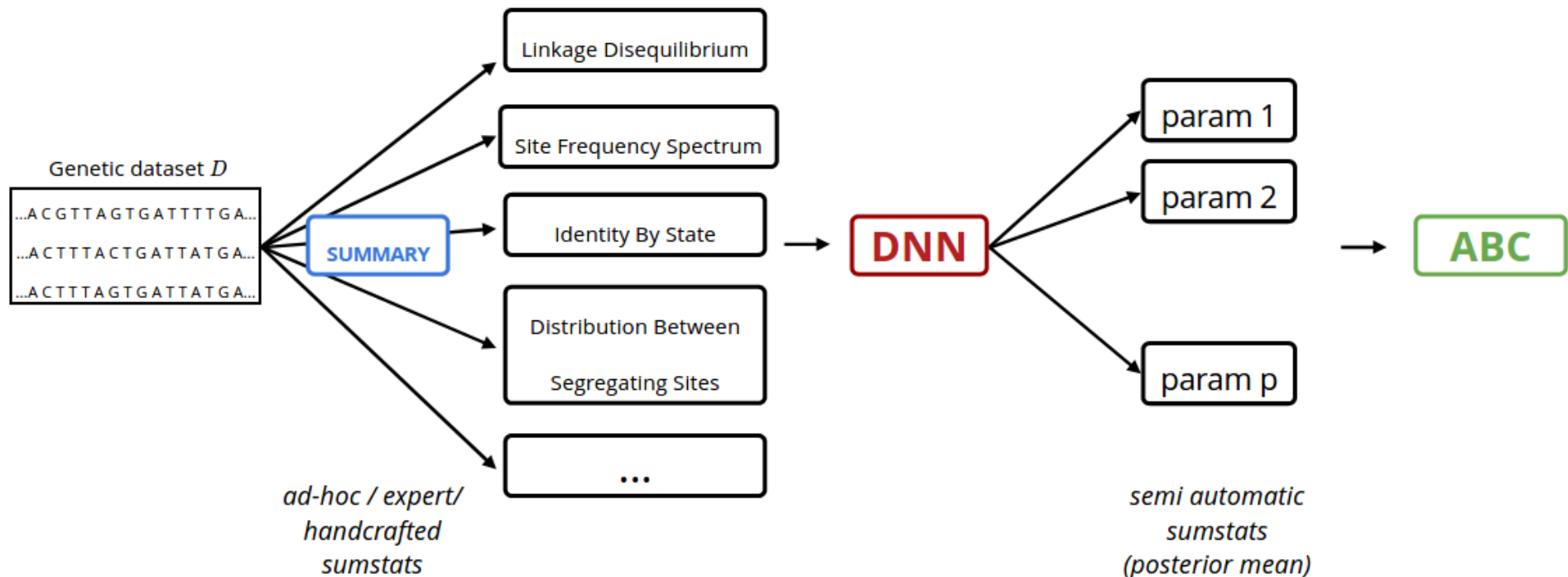
# Deep learning on summary statistics (+ABC)

- Generally fully connected net / multilayer perceptron (MLP), e.g.:

- Selection and demo inference, [Sheehan and Song 2016 \(no ABC\)](#)
- Model selection+inference (archaic admixture models), [Mondal et al. 2019 \(with ABC\)](#)

- Those were inspired by Jiang et al 2017 (MLP)

but see as well Creel 2017 (MLP), Raynal et al. 2017 (random forest), Fernhead and Prangle 2012 (posterior mean as  $s(.)$ )



## Deep learning on summary statistics

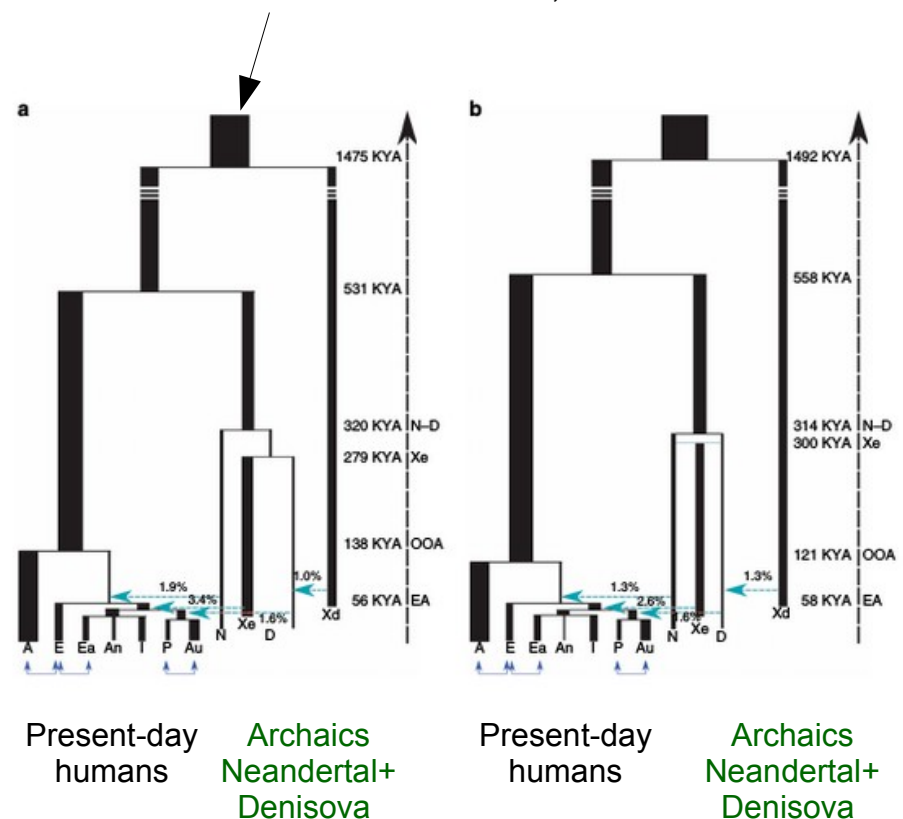
- Generally fully connected net / multilayer perceptron (MLP), e.g.:
  - Selection and demo inference, Sheehan and Song 2016 (no ABC)
  - Model selection+inference (archaic admixture models), Mondal et al. 2019 (with ABC)

**Mondal et al. 2019 :  
ABC [ MLP (joint SFS) ]**

2 models selected among 8 models + parameter estimation

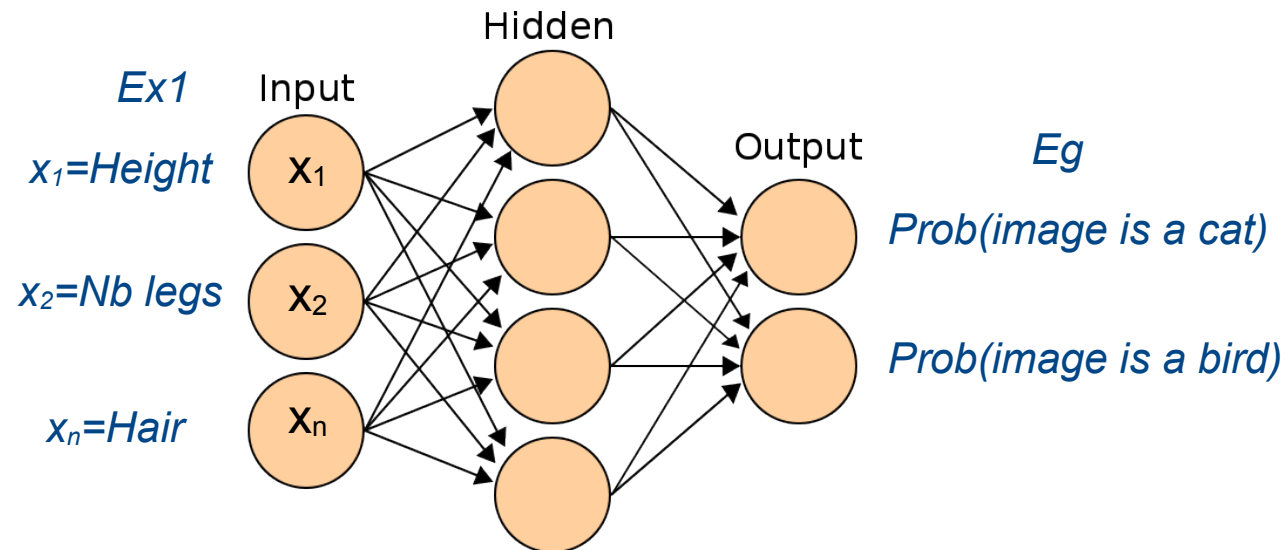
-> third archaic introgression in Asia and Oceania from Neandertal-Denisova clade or from Denisova related lineage (early divergence)

## Ancestors of modern humans, Neandertal and Denisova

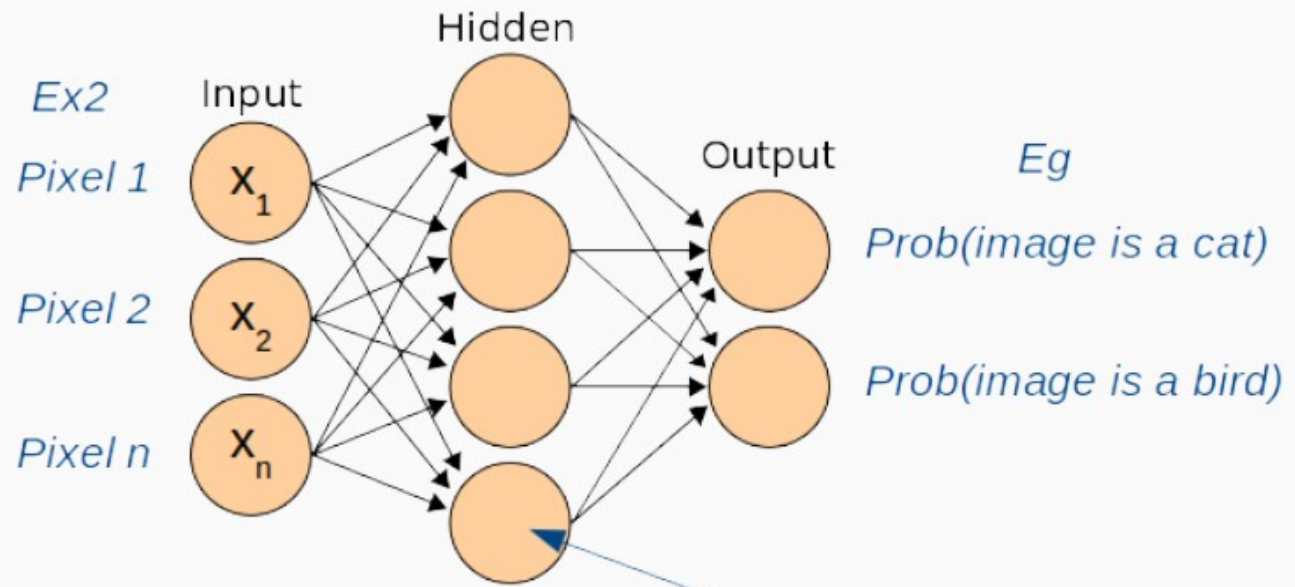


# Reminder: you could bypass summary statistics

From summary statistics (handcrafted features):



# Reminder: you could bypass summary statistics



*Internal layer(s) : learn an hidden representation from the data (ie learn how to encode the data)*

NOT handcrafted features!



---

# Outline

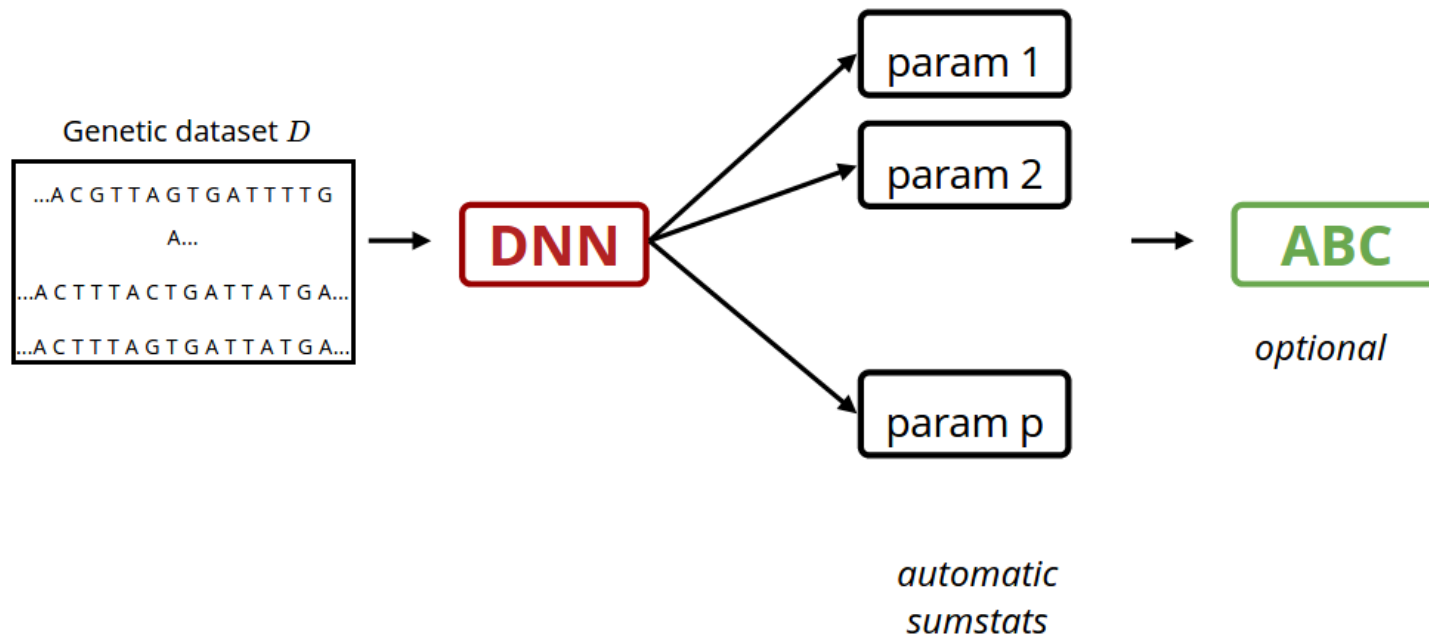
Machine Learning: basic concepts and terminology

ML, application to popgen ; neural networks

- I. From ABC to deep learning for population genetics
- II. Learning directly from SNP data with neural networks**
- III. Dissecting two published networks for effective population size inference
- IV. Opening on applications of unsupervised deep learning to popgen
- V. Tonight's hands-on: building/training/re-using DNNs for population genetics (demography/selection) inference with dnadna

# Deep learning on "raw" genetic data

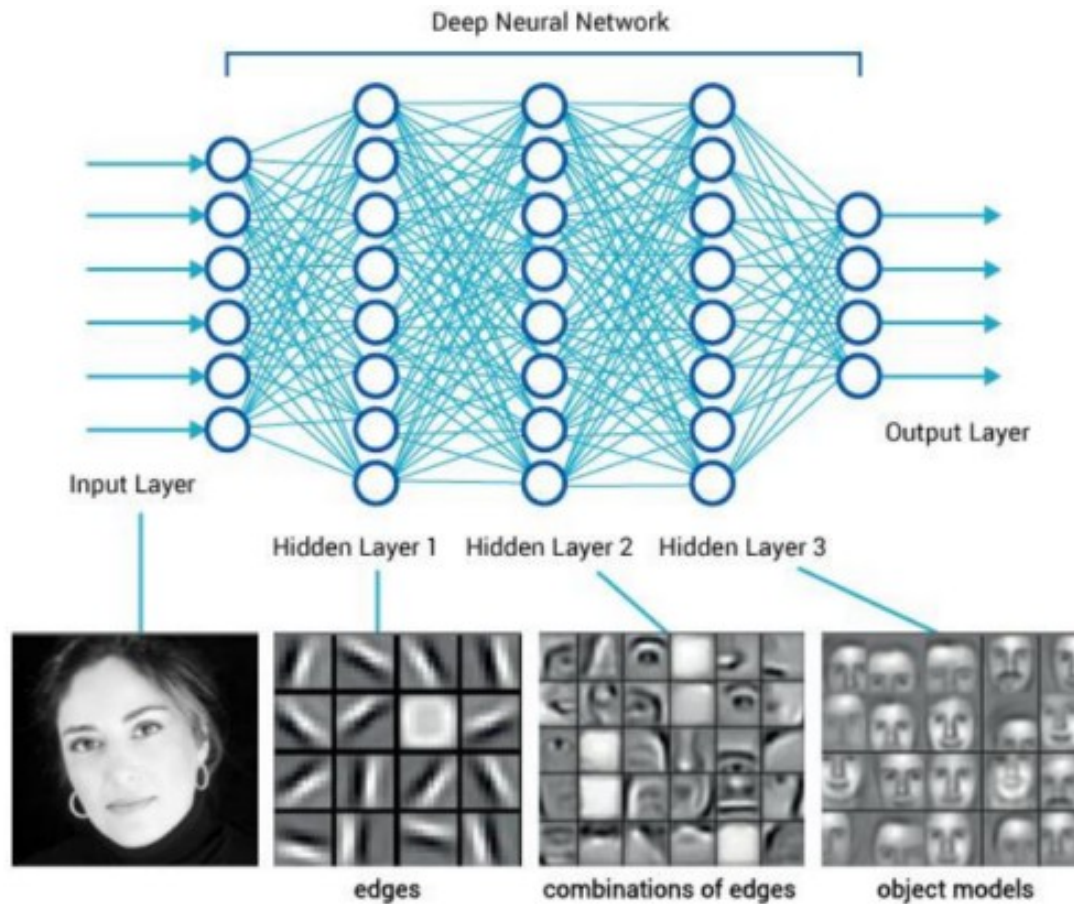
- Process directly the genetic data to bypass handcrafted features
- Often convolution neural networks (CNN)
- Inspired by Jiang et al 2017 but previous works in popgen skip the ABC step



(e.g. DNN tries to predict  $N_1$ , ...  $N_p$  etc) and these are used later as automatic summary statistics (or automatic features) processed by ABC

# DL - learning hierarchical representations

Deep Learning (DL) = deep neural networks = nnet with multiple layers



---

# DL - learning hierarchical representations

- Able to learn a hierarchy of representations with increasing level of abstraction

Eg. for image :

pixel → edge → motif → part → full object → combination (eg landscape, scene)

Eg. for text :

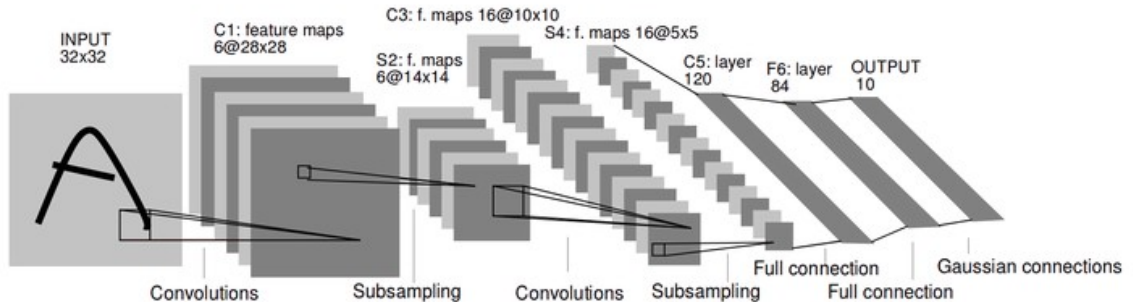
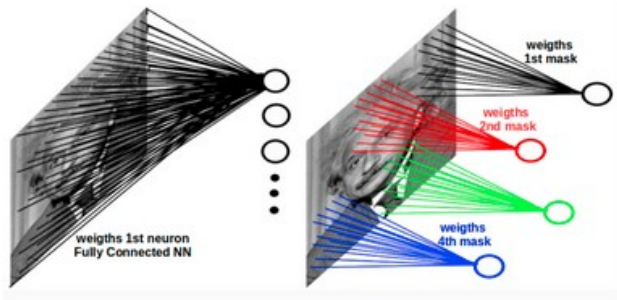
letter → word → word group → sentence → story

...

- A layer = trainable function that transforms input into features at a certain hierarchy level

# Deep learning on "raw" genetic data

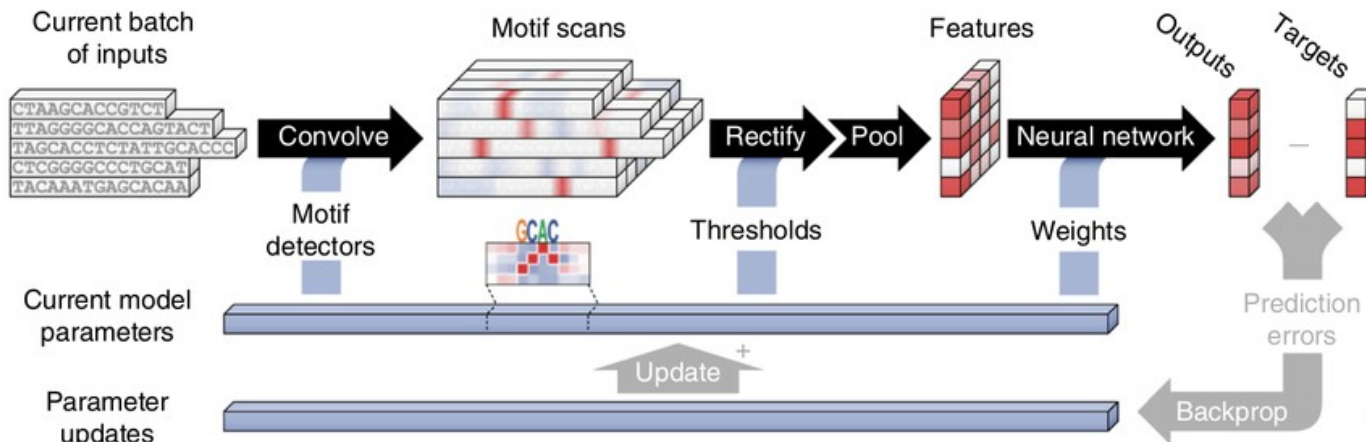
- Convolution networks work well for **computer vision**:



*Lecun et al.*

1998

- Already used on **DNA sequences**:



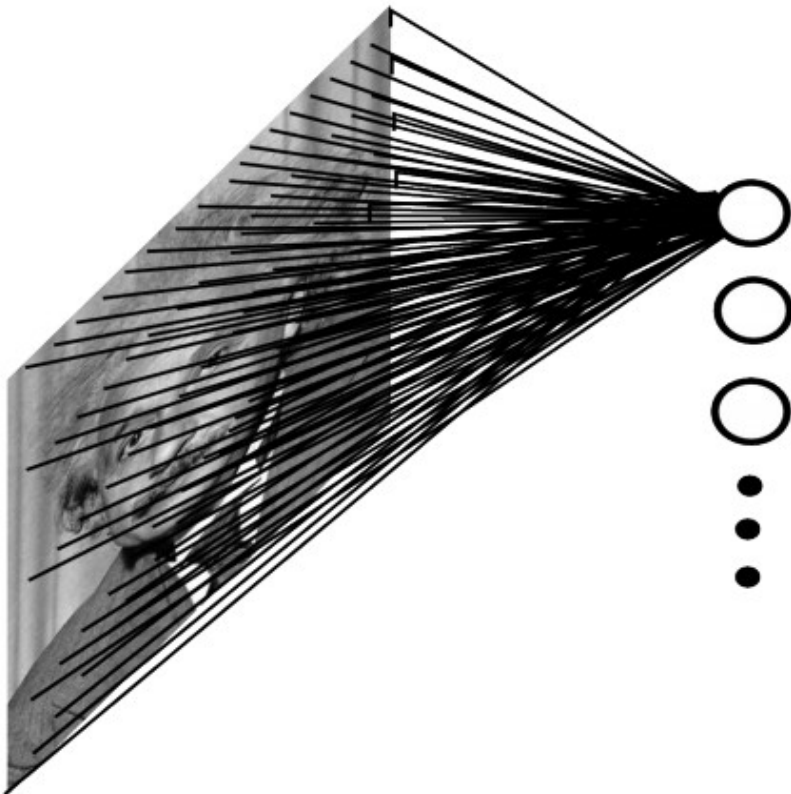
*Alipanahi et al.*

2015

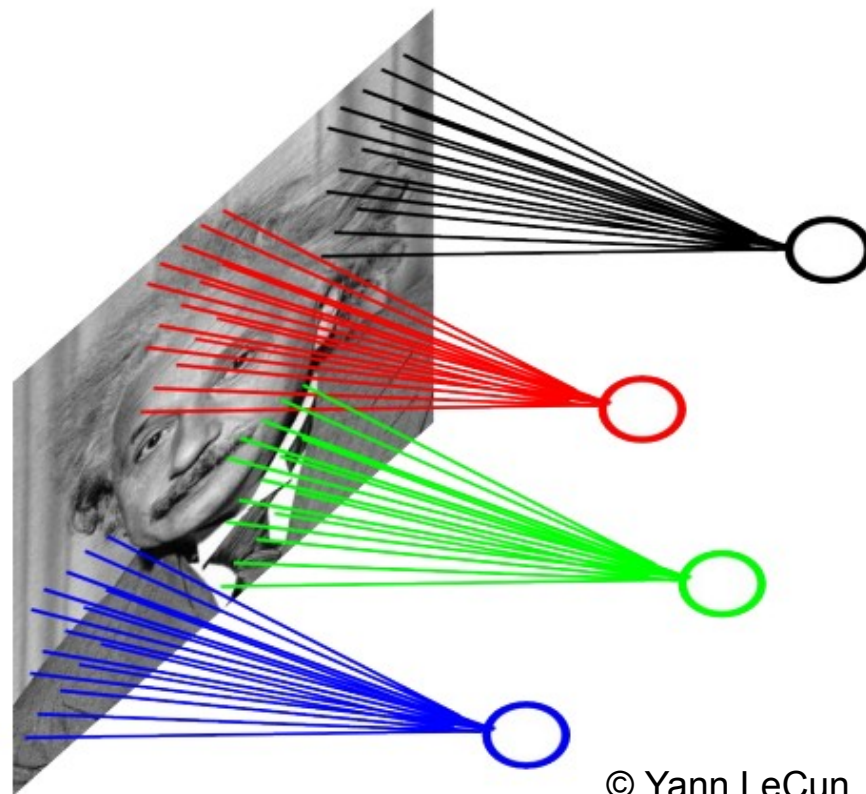
# DL - convolution

- Why using convolution networks? (convnet)

Fully connected :  
→ huge number of parameters  
to learn (each edge  
correspond to a weight)



Convnet  
locally connected  
→ smaller number of  
parameters to learn

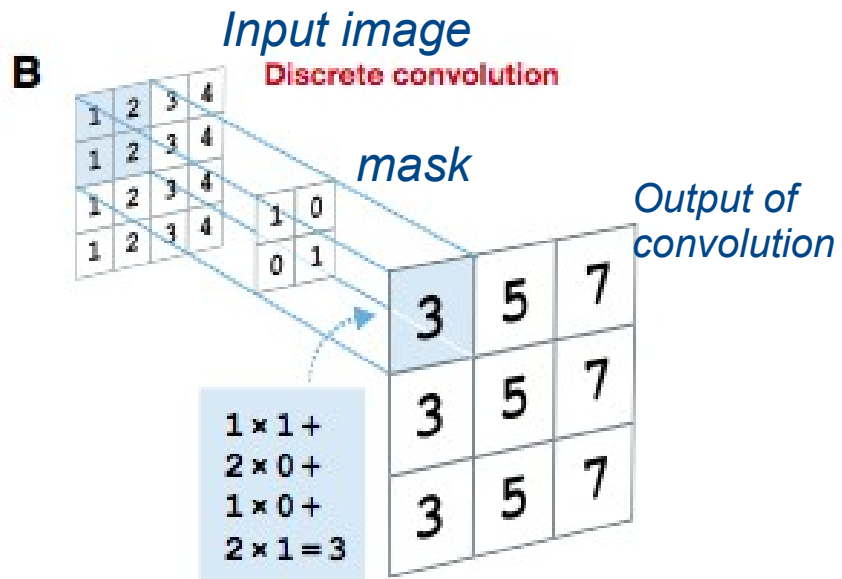


© Yann LeCun

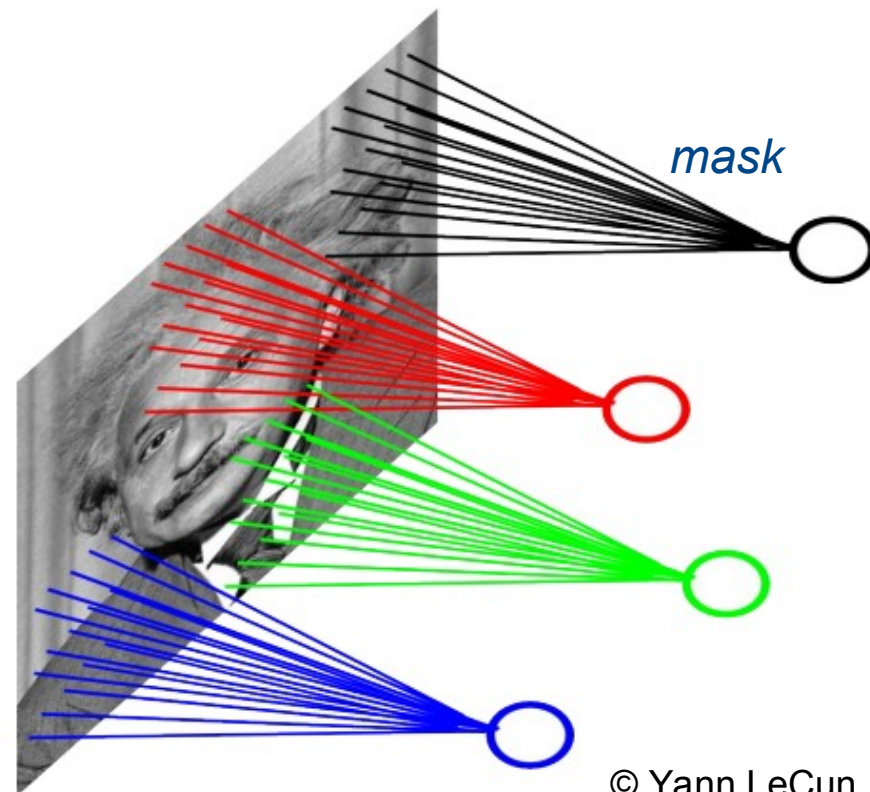


# DL - convolution

- Why using convolution networks? (convnet)



Convnet  
locally connected  
→ smaller number of  
parameters to learn



© Yann LeCun

# Convolution operation

- Convolution with padding and stride=1 → no dimension reduction

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346			

- No padding and/or stride > 1 → dimension reduction

Kernel

0	-1	0
-1	5	-1
0	-1	0

60	113	56	139	85
73	121	54	84	128
131	99	70	129	127
80	57	115	69	134
104	126	123	95	130

266	-61	-30
116	-47	295
-135	256	-128



# Exercise

- Compute by hand a convolution operation (see below)
- Compute the number of parameters (how does it scale with input size)
- Design a 3x3 filter (with fixed weights) that could detect horizontal edges (detect a pattern ~ maximal activation for this pattern)

Input data

```
0 1 1 0 0
1 1 0 1 0
1 0 0 0 1
```

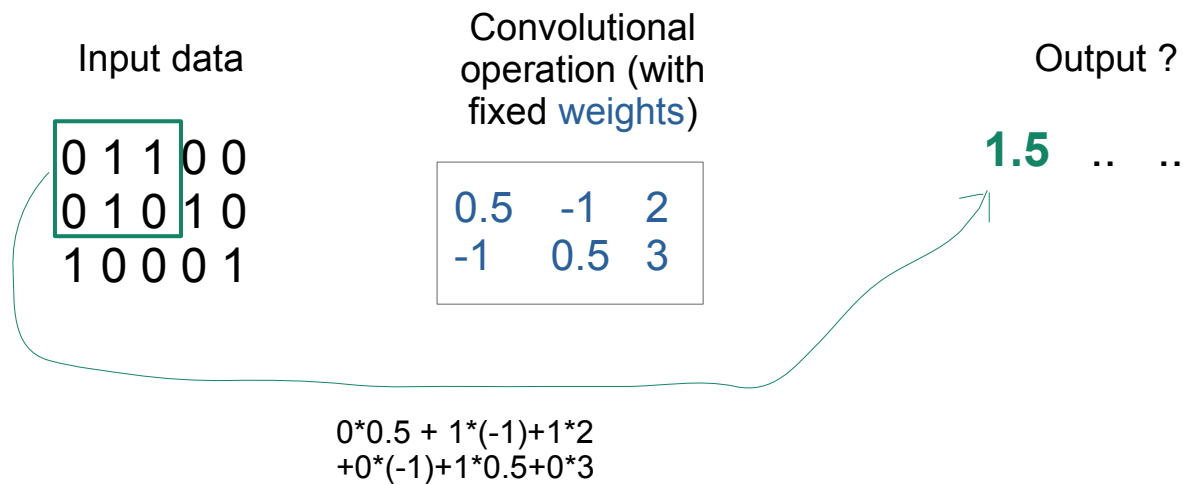
Convolutional  
operation (with  
fixed weights)

0.5	-1	2
-1	0.5	3

Output ?

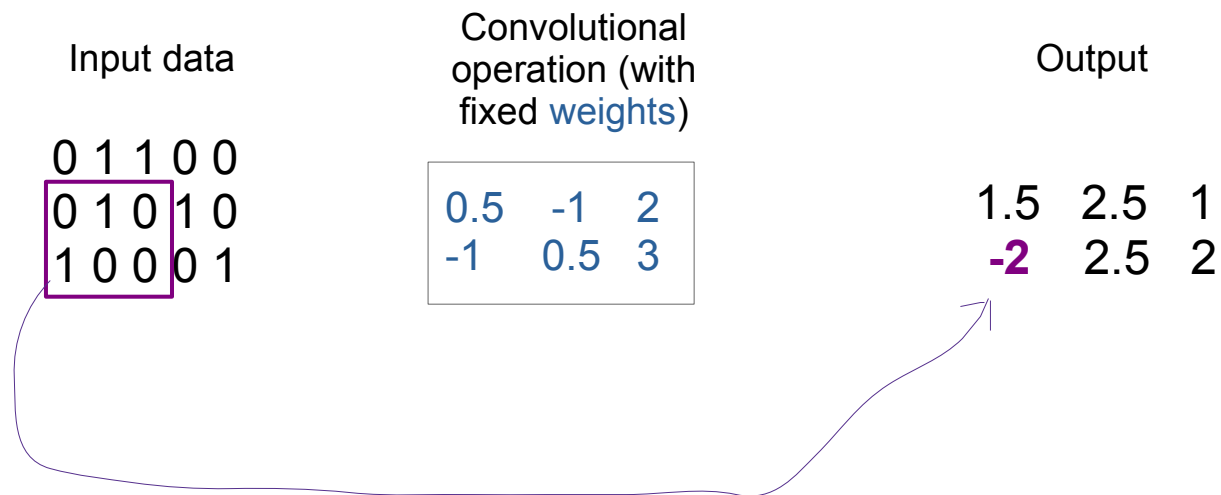
# Exercise

- Computer by hand a convolution operation (see below)
- Compute the number of parameters
- Design a 3x3 filter (with fixed weights) that could detect horizontal edges (detect a pattern ~ maximal activation for this pattern)



# Exercise

- Computer by hand a convolution operation (see below)
- Compute the number of parameters
- Design a 3x3 filter (with fixed weights) that could detect horizontal edges (detect a pattern ~ maximal activation for this pattern)



# Exercise

- Computer by hand a convolution operation (see below)
- Compute the number of parameters
- Design a 3x3 filter (with fixed weights) that could detect horizontal edges (detect a pattern ~ maximal activation for this pattern)

Input data

0	1	1	0	0
0	1	0	1	0
1	0	0	0	1

Convolutional  
operation (with  
fixed **weights**)

0.5	-1	2
-1	0.5	3

Output

1.5	2.5	1
<b>-2</b>	2.5	2

Output of max  
pool operation

2.5

# Exercise

- Computer by hand a convolution operation (see below)
- Compute the number of parameters
- Design a 3x3 filter (with fixed weights) that could detect horizontal edges (detect a pattern ~ maximal activation for this pattern)

Input data

```
0 1 1 0 0
0 1 0 1 0
1 0 0 0 1
```

Convolutional  
operation (with  
fixed **weights**)

0.5	-1	2
-1	0.5	3

Output

1.5	2.5	1
<b>-2</b>	2.5	2

OR applying a  
**RELU** activation  
and another conv  
filter (e.g. 2x2)

1.5	2.5	1
<b>0</b>	2.5	2

Output

...

another conv filter  
with fixed weights

4	-0.1
-1.2	0.6

# Exercise

- Compute by hand a convolution operation (see below)
- Compute the number of parameters
- Design a 3x3 filter (with fixed weights) that could detect horizontal edges (detect a pattern ~ maximal activation for this pattern)

Convolutional operation (with  
fixed **weights**) detecting  
horizontal edges

-1	-1	-1
1	1	1
-1	-1	-1

# Reminder of steps for a simulation-based supervised ML approaches

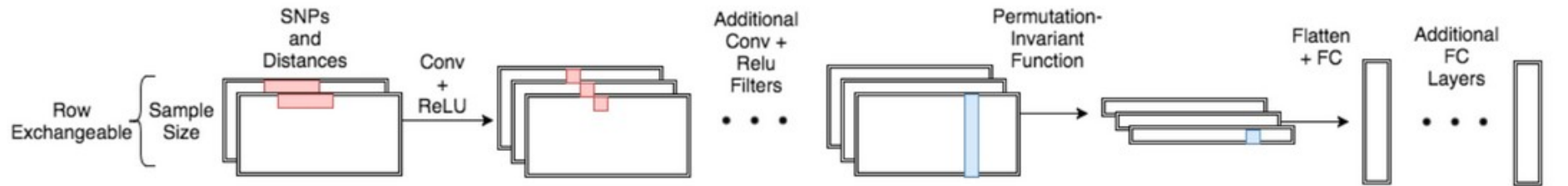
- Define clearly your task
  - Regression/classification of  $xXx$  ; score function, loss
  - Define one or several models e.g. Constant size, Fluctuating size, Fluctuating+selection
  - Pick priors for the parameters of these models  
e.g.  $N_e \sim U[0, 100]$ , selection coeff  $\sim N(0, 10)$ , ...
- Randomly draw parameters
- Simulate thousands/millions of such SNP matrices thanks to genetic simulators (msprime, msms, slim, bactSLiMulator, ...) using the random parameters
- **Design**, train and evaluate a **model** directly on these matrices

*Typical input for population genetics methods*

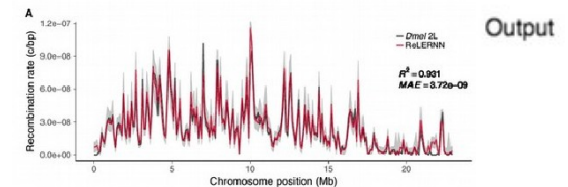


# DNN on "raw" data in popgen: SNPs (and distance) matrix

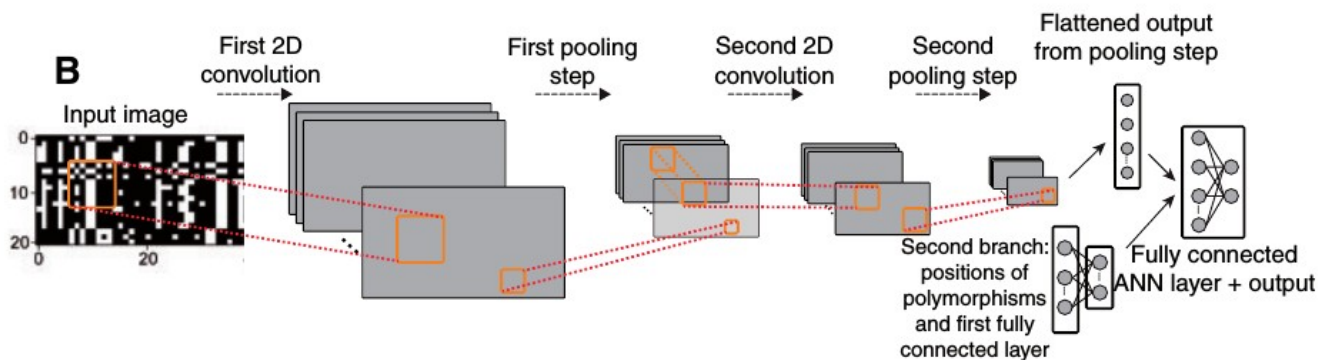
- Detection of recombination hotspot, *exchangeable CNN net*



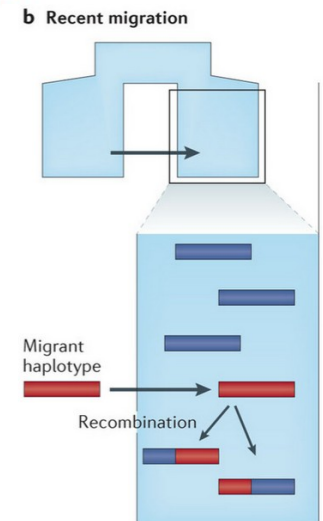
Chan et al. 2018



- Inference of introgression, selection, recombination rate and population size histories with 5 parameters (3-step history), CNNs



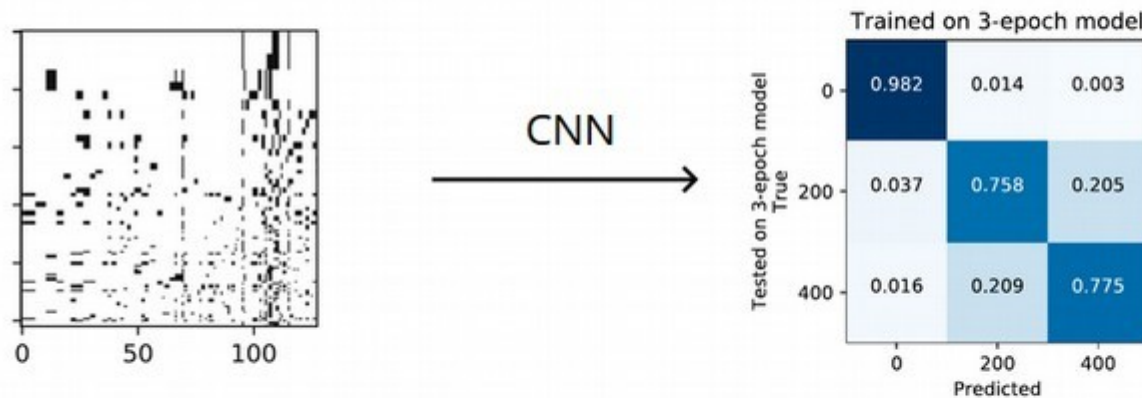
Flagel et al. 2019





# DNN on "raw" data in popgen: SNPs (and distance) matrix

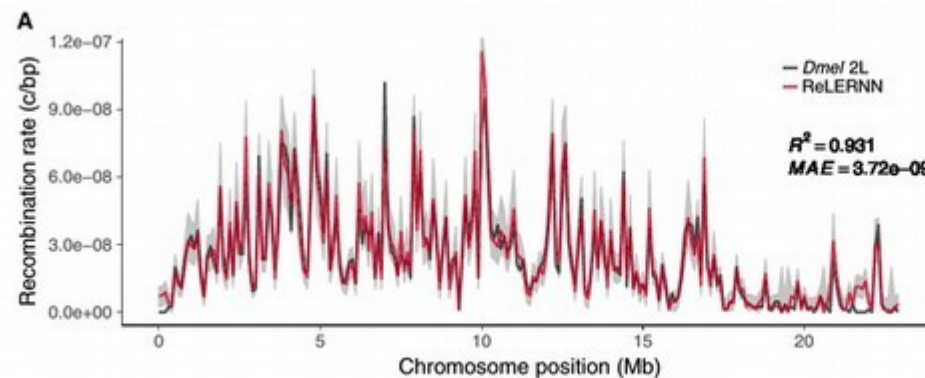
- Predicting selection (under fixed demography), *CNN*



*Torada et al.*  
2019

related: Isildak et al bioRxiv (balancing selection vs incomplete sweep),

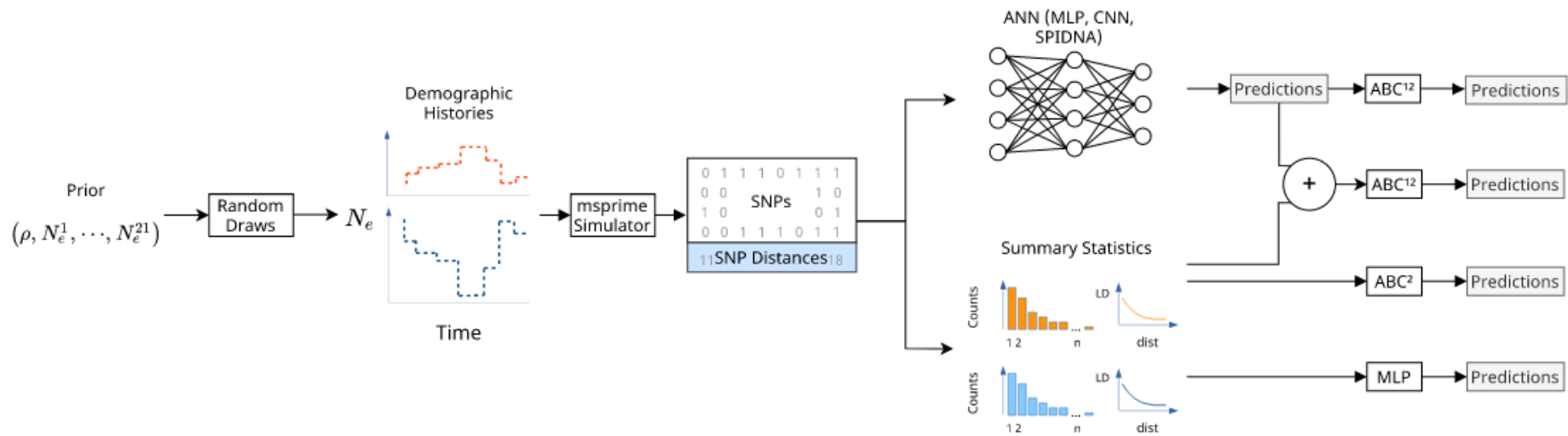
- Inference of recombination with *recurrent networks (RNN)*



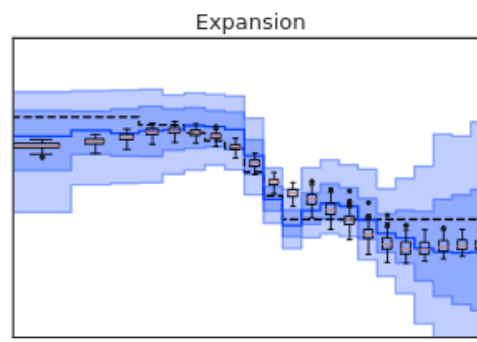
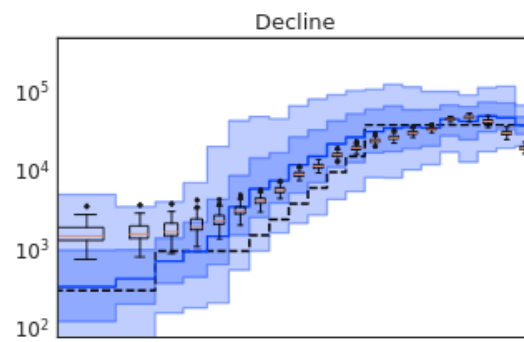
*Adrion et al.*  
2020

# DNN on "raw" data in popgen: SNPs (and distance) matrix

- Predicting fluctuating population size (21 steps), exchangeable CNN
- Comparison and combination with ABC



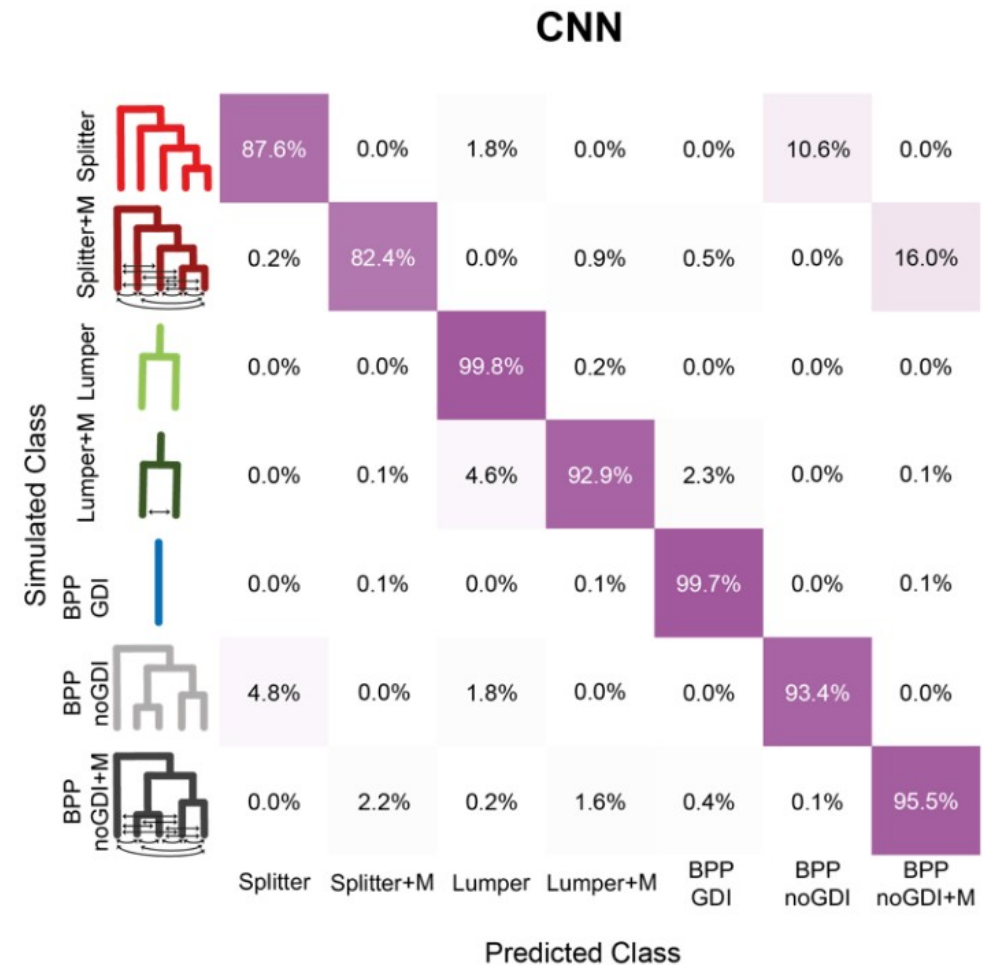
Effective population size (log scale)



*Sanchez et al.*  
2020

# DNN on "raw" data in popgen: SNPs (and distance) matrix

“Coalescent-based species delimitation meets deep learning: Insights from a highly fragmented cactus system.” Perez et al 2021



To keep going: the introduction of Sanchez\*, Bray\*, et al (preprint) lists many more papers on DNN for popgen

Sanchez\*, Bray\*, et al (preprint) <https://hal.archives-ouvertes.fr/hal-03352910v2>

“Dnadna: Deep Neural Architecture for DNA - A deep learning framework for population genetic inference”

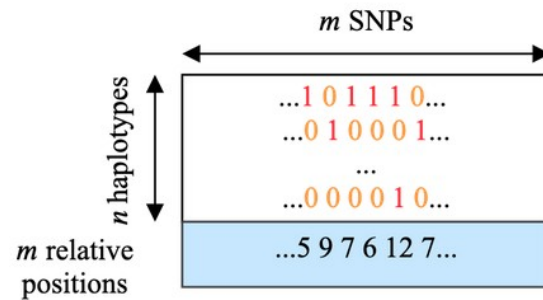
---

# Outline

Machine Learning: basic concepts and terminology

ML, application to popgen ; neural networks

- I. From ABC to deep learning for population genetics
- II. Learning directly from SNP data with neural networks
- III. Dissecting two published networks for effective population size inference**
- IV. Opening on applications of unsupervised deep learning to popgen
- V. Tonight's hands-on: building/training/re-using DNNs for population genetics (demography/selection) inference with dnadna

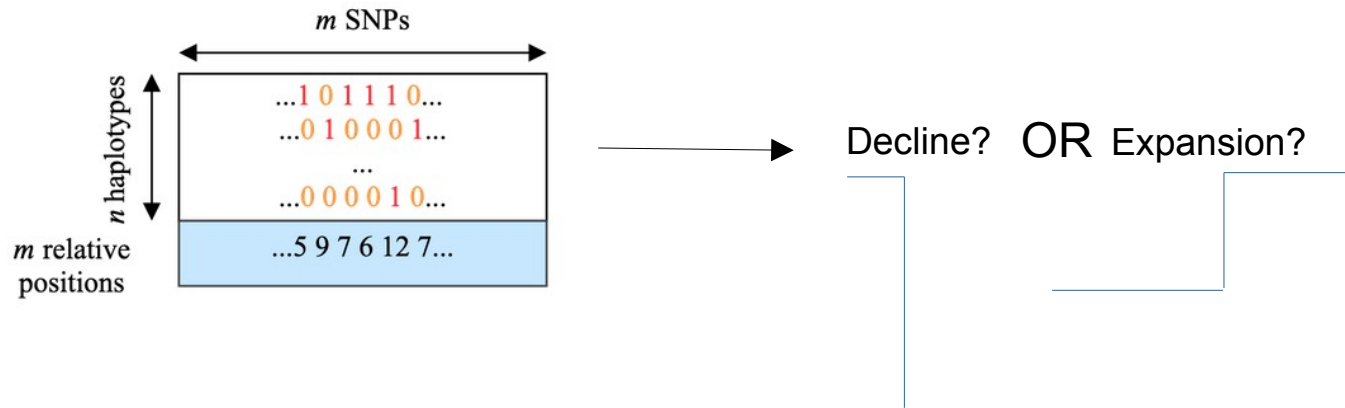
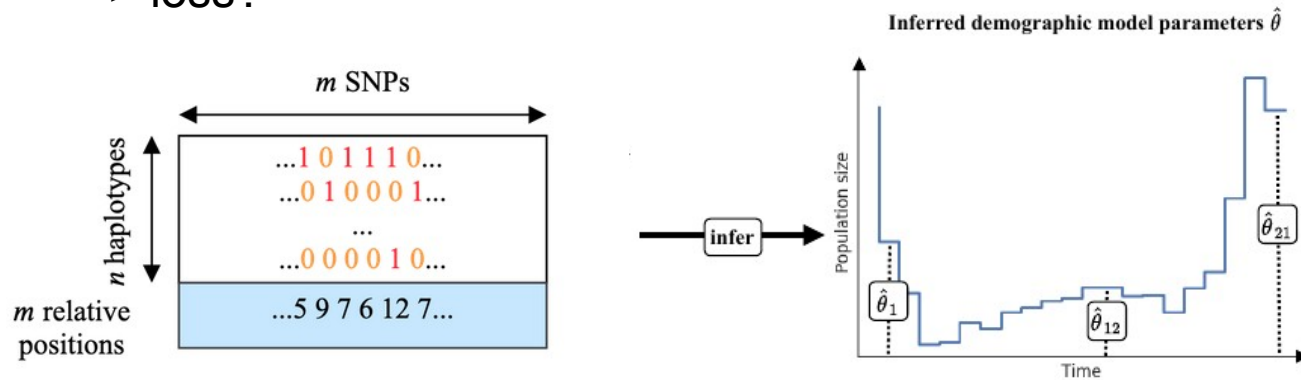


Exercise: define the task and loss for a net that could inform you whether there was a strong decline of effective population size

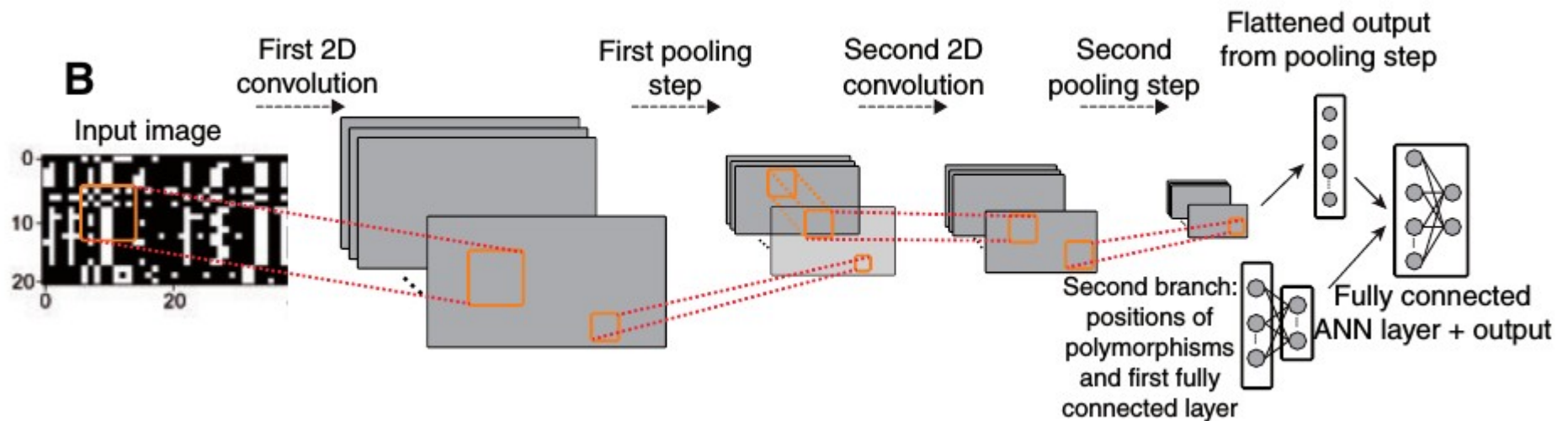
What's my model(s)? What are the parameters? Regression or classification task? --> loss?

Exercise: define the task and loss for a net that could inform you whether there was a strong decline of effective population size

What's my model(s)? What are the parameters? Regression or classification task? --> loss?

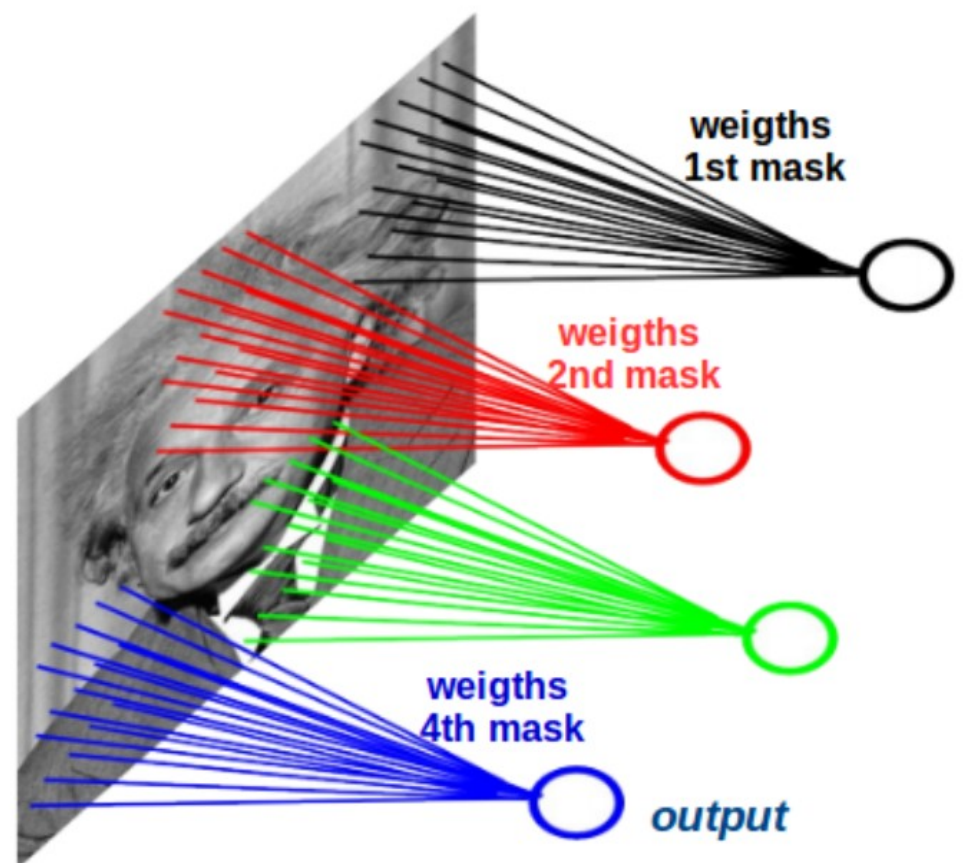
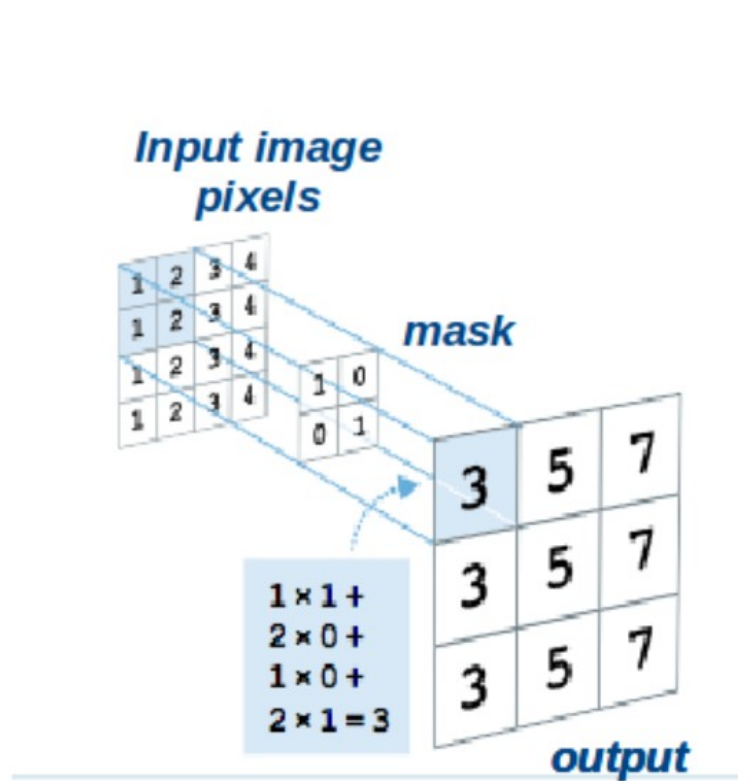


- *Flagel et al. 2019* network





How?





# Convolution operation

- Convolution with padding and stride=1 → no dimension reduction

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346			

- No padding and/or stride > 1 → dimension reduction

60	113	56	139	85
73	121	54	84	128
131	99	70	129	127
80	57	115	69	134
104	126	123	95	130

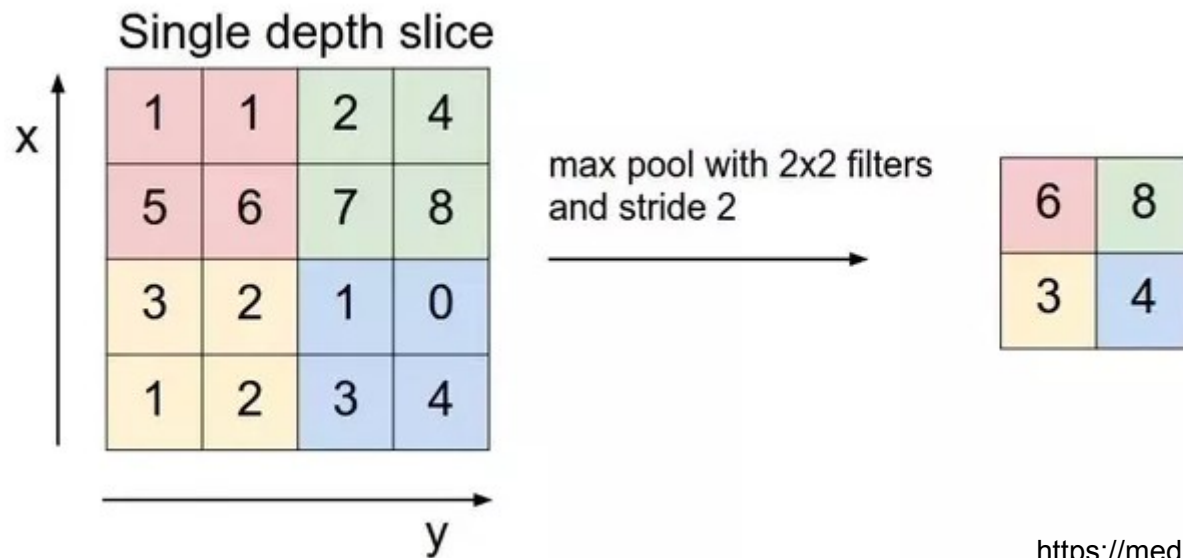
Kernel

0	-1	0
-1	5	-1
0	-1	0

266	-61	-30
116	-47	295
-135	256	-128

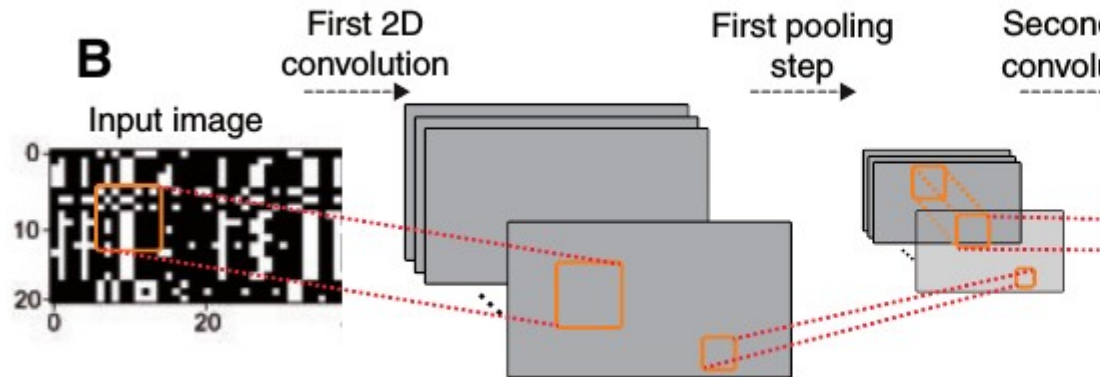
# Pooling operation

- Pooling (max, average, sum pooling)  
-> reducing dimension without additional parameter to learn



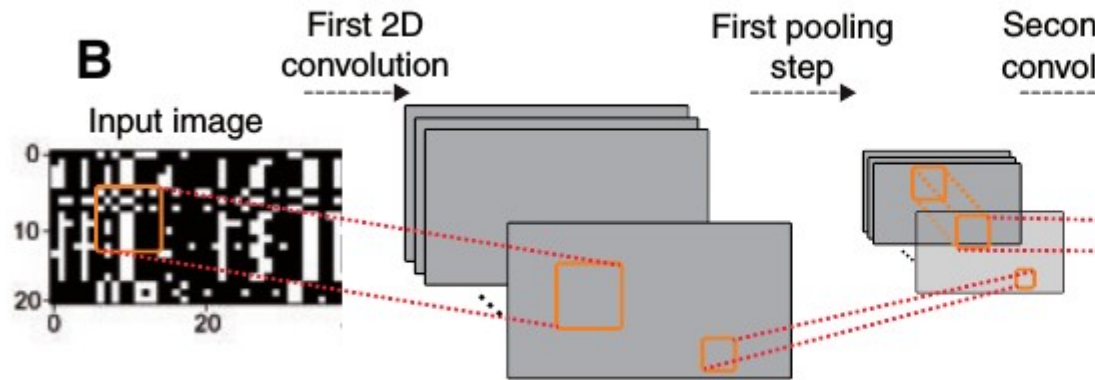
<https://medium.com/@RaghavPrabhu>

# Why a 2D convolution ?

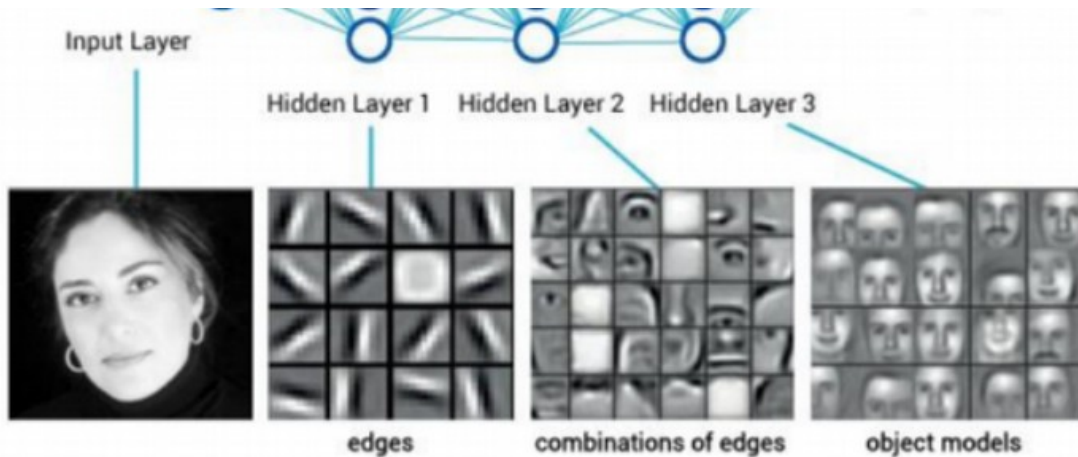


*Flagel et al. 2019*

# Why a 2D convolution ?



*Flagel et al. 2019*

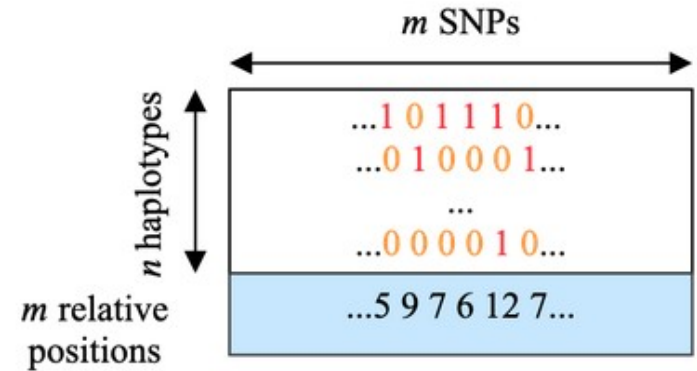


Y LeCun

But popgen data is different from a classical image!

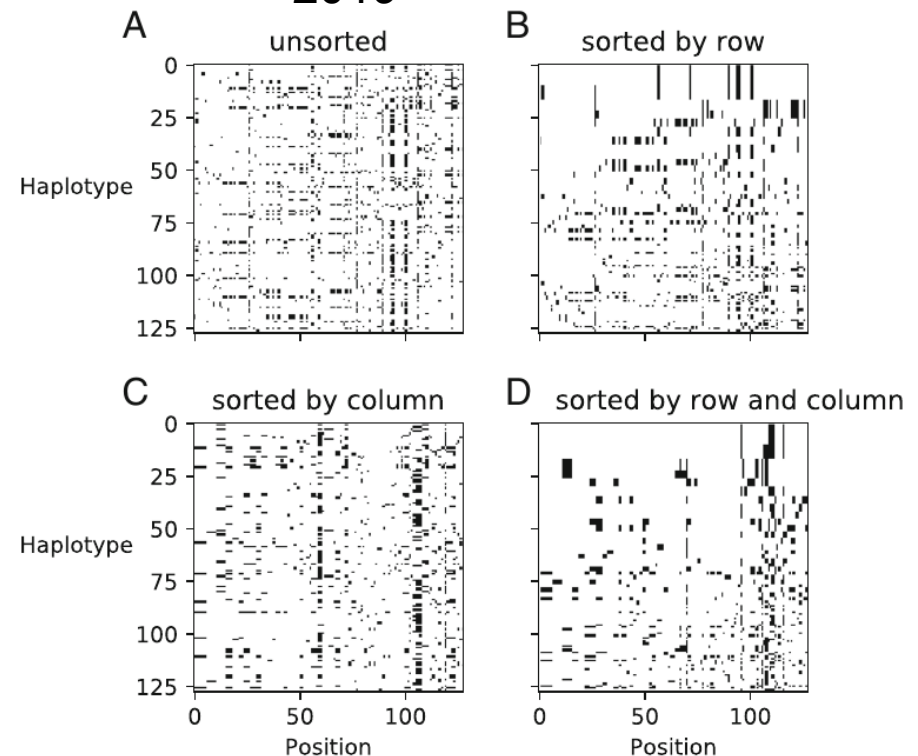
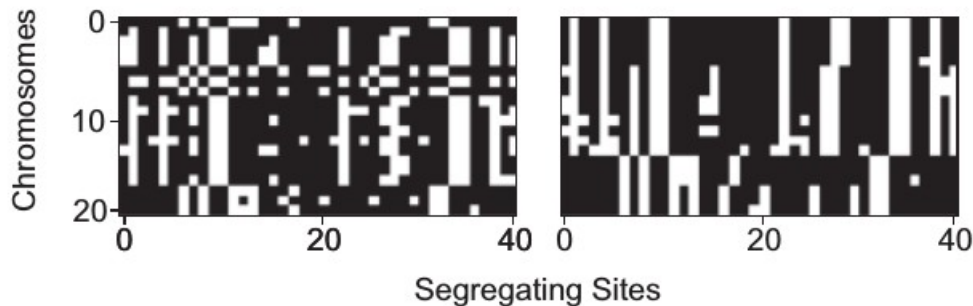
# How to be invariant ?

- Solution 1: specify an **order**  
ex: individual similarity (rows)  
SNP similarity (columns)

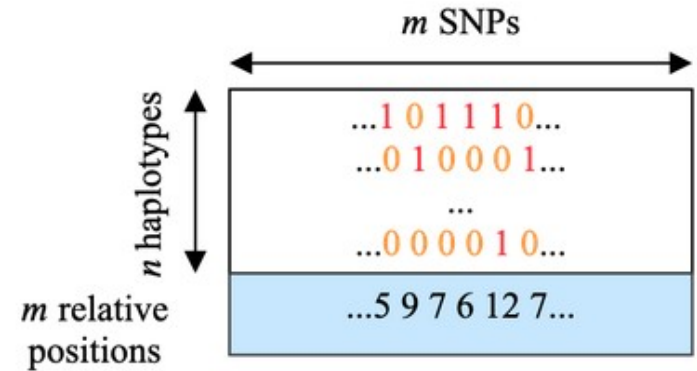


Torada et al.  
2019

Flagel et al. 2018



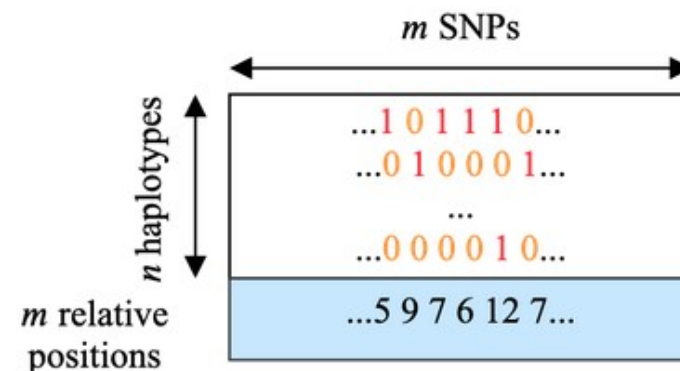
# How to be invariant ?



- Solution 1: specify an **order** (ex: individual similarity) (e.g. Fligel et al. 2018, Torada et al. 2019, ... )
- Solution 2: data **augmentation**: apply transformation(s) to the input data that should not affect its label.  
Ex. in some tasks of computer vision: rotating images

*Exercise: give examples of data augmentation relevant for population genomics*

# How to be invariant ?

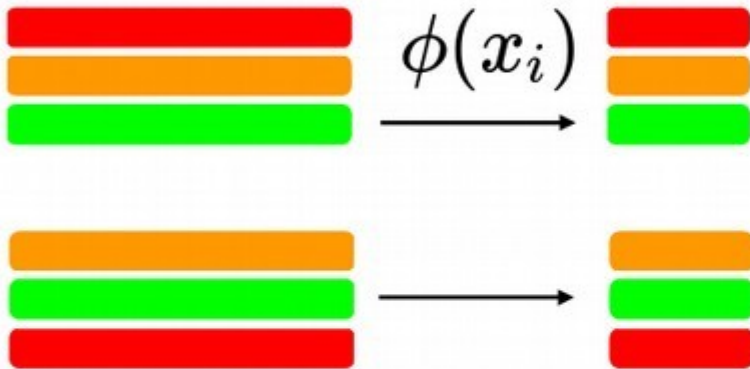


- Solution 1: specify an **order** (ex: individual similarity) (e.g. Fligel et al. 2018, Torada et al. 2019, ... )
- Solution 2: data **augmentation** (e.g. shuffling the lines)
- Solution 3: **encode invariance** in the network (permutation-invariant network, exchangeable network) (e.g. Chan et al. 2018, Wiqvist et al. 2019, Sanchez et al 2020)

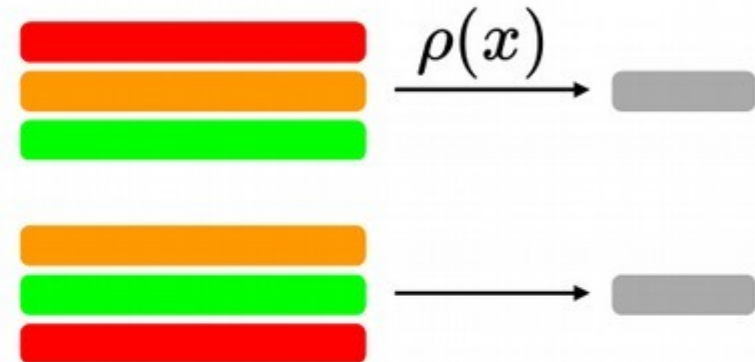
# Adapt the network to the data features

- **Invariant** to the permutation of rows

## Equivariant function



## Invariant function



*Exercise: give an example of invariant operation*



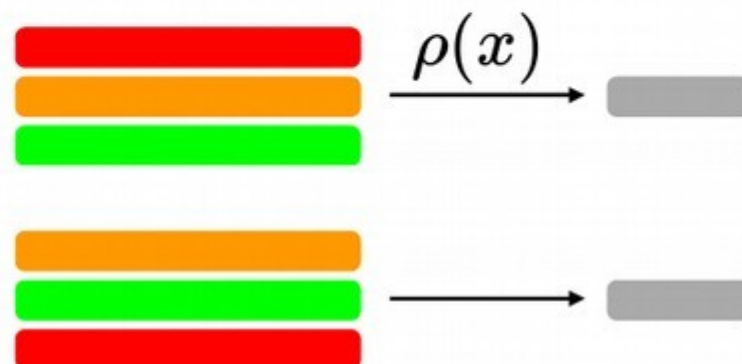
# Adapt the network to the data features

- **Invariant** to the permutation of rows

## Equivariant function

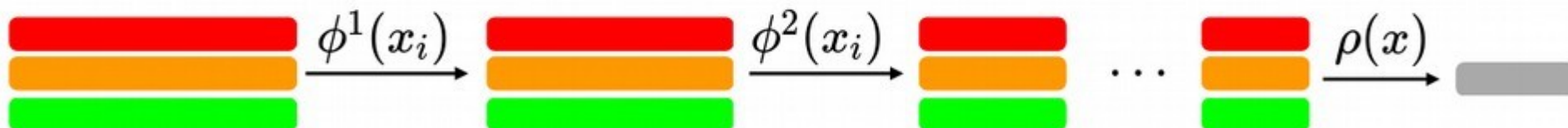


## Invariant function



---

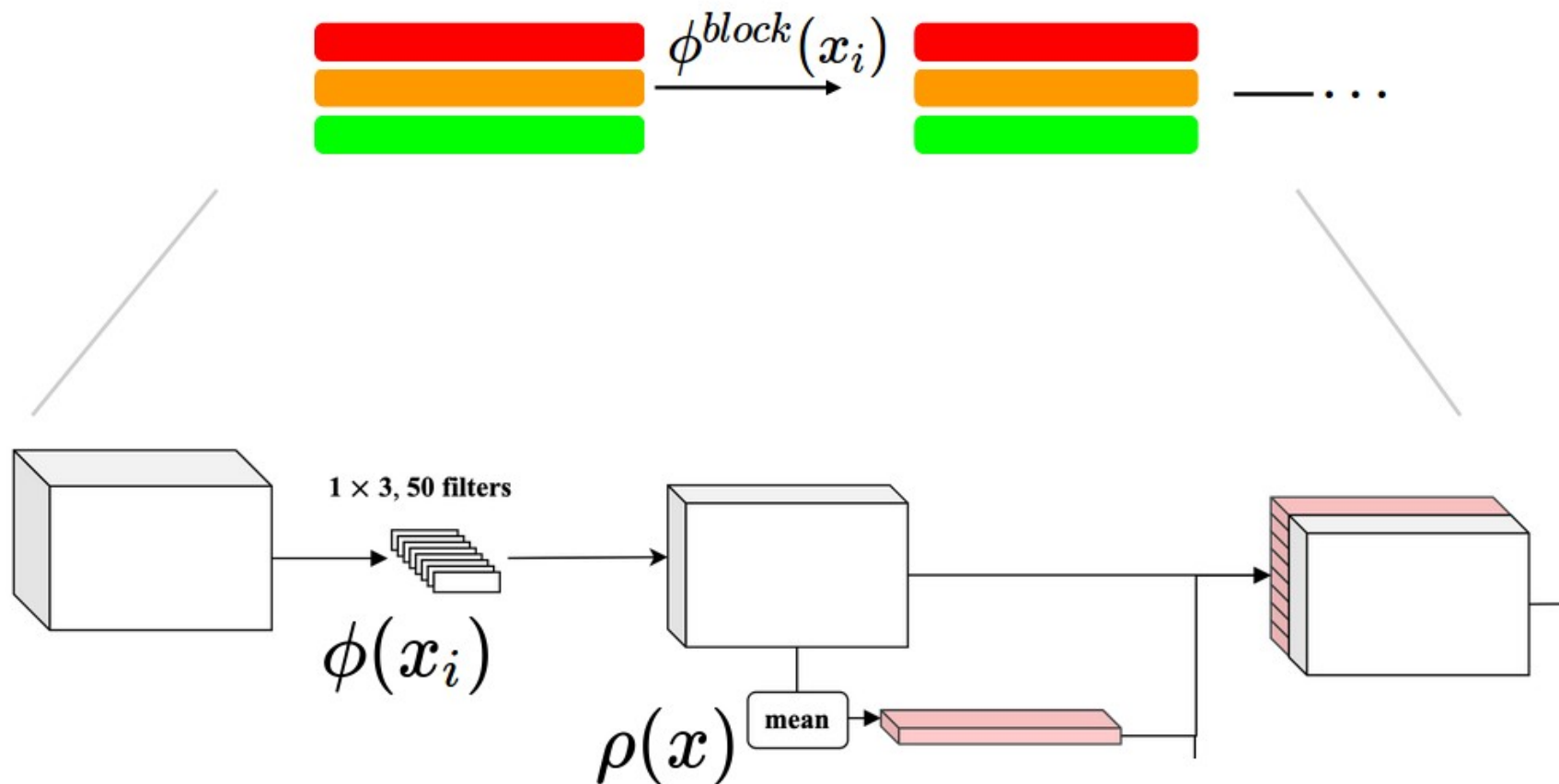
**Stacking** invariant and equivariant layers **covers the full space of permutation invariant** function (Zaheer et al. 2017, Lucas et al. 2018)



# Adapt the network to the data features

- Each block of SPIDNA defines an equivariant function

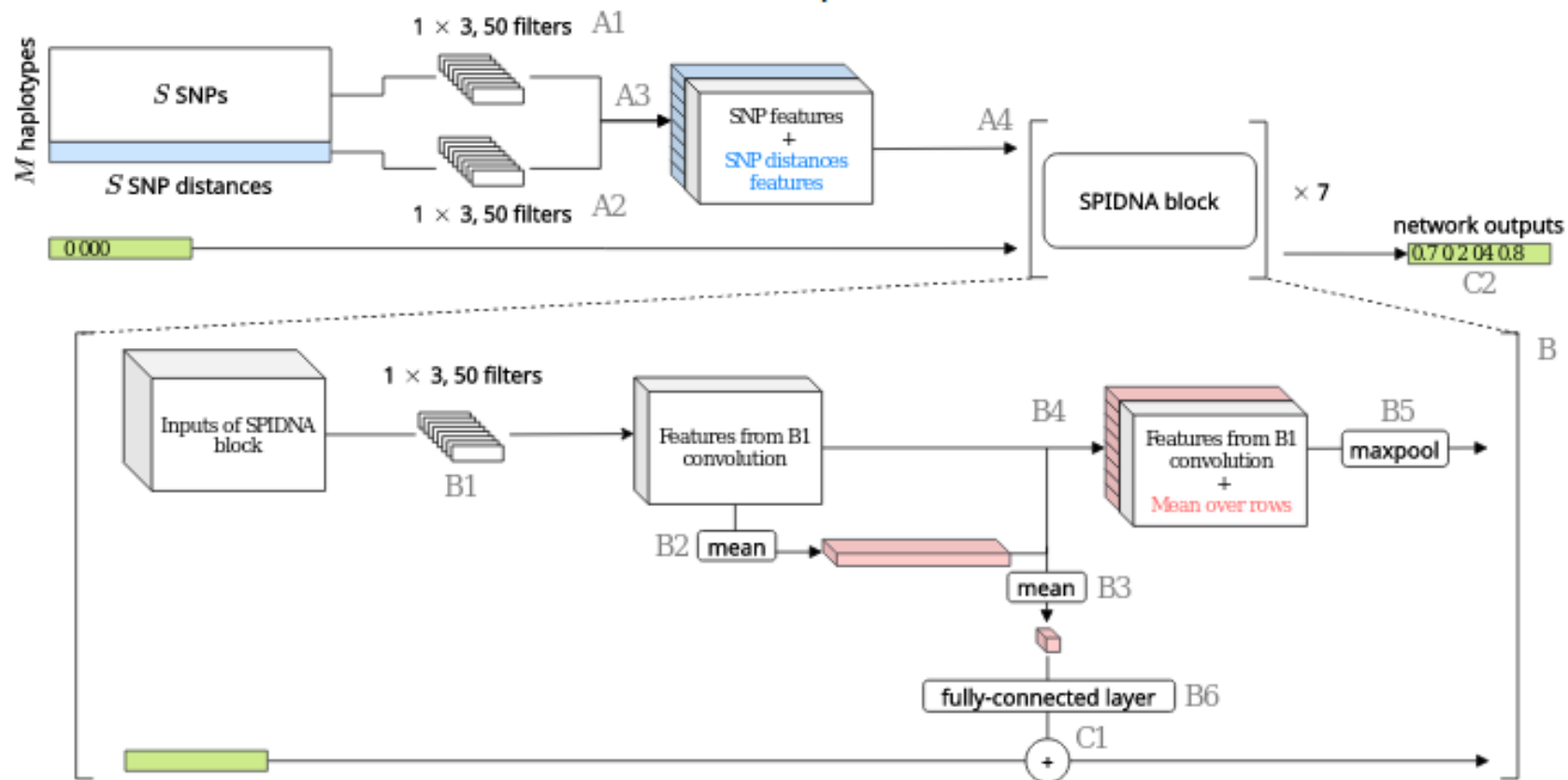
Sanchez et al 2020



# Our Neural Network Architecture: SPIDNA

## Sequence Position Informed Deep Neural Architecture

- Combines relative positions and SNPs
- Invariant to haplotype permutation
- Adaptive to the number of SNPs



---

# Final note on DL applied to population genetics

- Still a **recent** “combo” (2016 for DL on summary statistics, 2018 for DL on raw population genetic data) -> space for creativity and new proposals!
- Remember some important rules in stat/ML/DL:
  - **Think** properly about your **task**, statistical **model**, **evaluation** scheme/metrics
  - Watch out for **overfitting**, use train/validation sets
  - **Compare** to previously published methods
    - Choose or explore **hyperparameter space** for each approach (grid search, Bayesian hyperoptimisation, ...) based on validation set
    - Final comparison on an **independent test set**
- Evaluate method **robustness** to data **corruption**, model **misspecification**, ...  
Particularly relevant for simulation-based inference approaches where simulators and simulation scenarios have underlying **assumptions** that can be violated in the real life.
- You might gain in accuracy but lose in **explainability**/interpretability (ex: CNN versus a previous approach based on SFS). Improving interpretability is actively studied in DL field.

---

# Robustness?

- Eg to selection while predicting demography or vice versa  
Sanchez et al. 2020, Torada et al 2019.
- To data damage, ...

Same care should be taken for all model-based inference models (even without simulations, such as PSMC, dadi etc.)

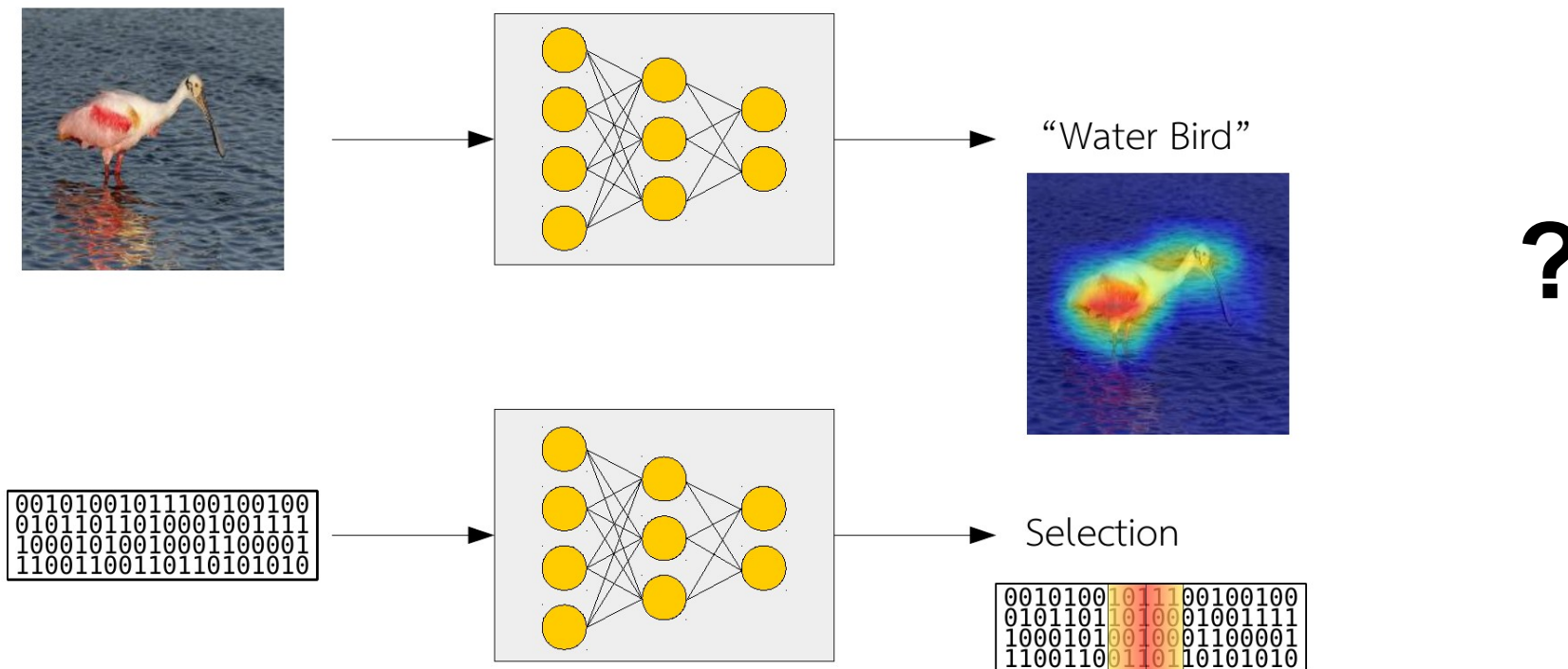
+ Additional unknown regarding what the NN is using

# Interpretation?

## Examples

- Sheehan and Song 2016. Pinpoint summary statistics used by the NN for a prediction
- Gower et al. 2021 Pinpoint parts of an image used for predicting adaptive introgression

Dream goal? And what about demography?



Can we have such a clear signal?

Active area of research in the machine/deep learning community

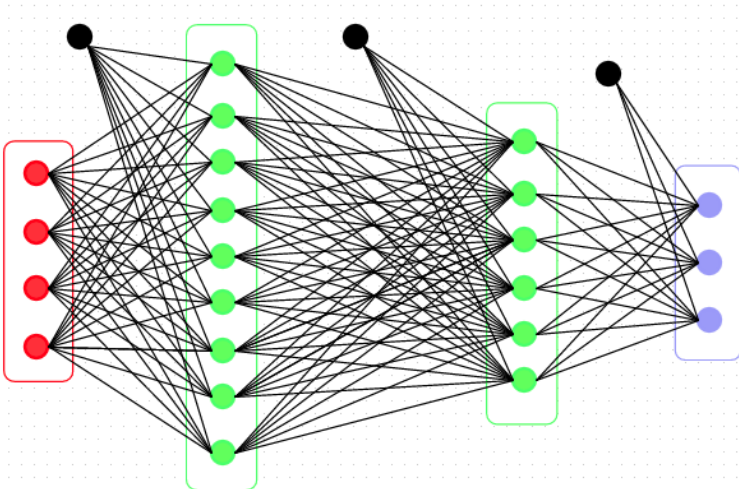
# Deep learning hyper-parameters (HP)

- You still have to make decisions about (1) your architecture (#layers, #nodes per layer, layer type,...) ; (2) the algorithm/optimization hyper-parameters
- Usually done by training numerous networks with numerous HP and keeping the one performing the best. Can be done in a smart way with e.g. bayesian HP optimization. Automatic Deep Learning : active area of research

## Just a taste of some NN algo hyper-parameters

Table 2. Central parameters of a neural network and recommended settings.

Name	Range	Default value
Learning rate	0.1, 0.01, 0.001, 0.0001	0.01
Batch size	64, 128, 256	128
Momentum rate	0.8, 0.9, 0.95	0.9
Weight initialization	Normal, Uniform, Glorot uniform	Glorot uniform
Per-parameter adaptive learning rate methods	RMSprop, Adagrad, Adadelata, Adam	Adam
Batch normalization	Yes, no	Yes
Learning rate decay	None, linear, exponential	Linear (rate 0.5)
Activation function	Sigmoid, Tanh, ReLU, Softmax	ReLU
Dropout rate	0.1, 0.25, 0.5, 0.75	0.5
L1, L2 regularization	0, 0.01, 0.001	



---

# Outline

Machine Learning: basic concepts and terminology

ML, application to popgen ; neural networks

- I. From ABC to deep learning for population genetics
- II. Learning directly from SNP data with neural networks
- III. Dissecting two published networks for effective population size inference
- IV. Opening on applications of unsupervised deep learning to popgen**
- V. Tonight's hands-on: building/training/re-using DNNs for population genetics (demography/selection) inference with dnadna



---

# Unsupervised / Supervised Tasks

- Learning something from **data**
- Either **unsupervised** (no labels) or **supervised** (discrete or continuous labels)

---

# Unsupervised learning

- **Learning something from data without labels**
- **Unsupervised = discovering patterns in data without prior knowledge**  
You do NOT have labels, or you do NOT use them

Exercise: Give examples of unsupervised tasks in population genetics

- 
- 
-

---

# Unsupervised learning

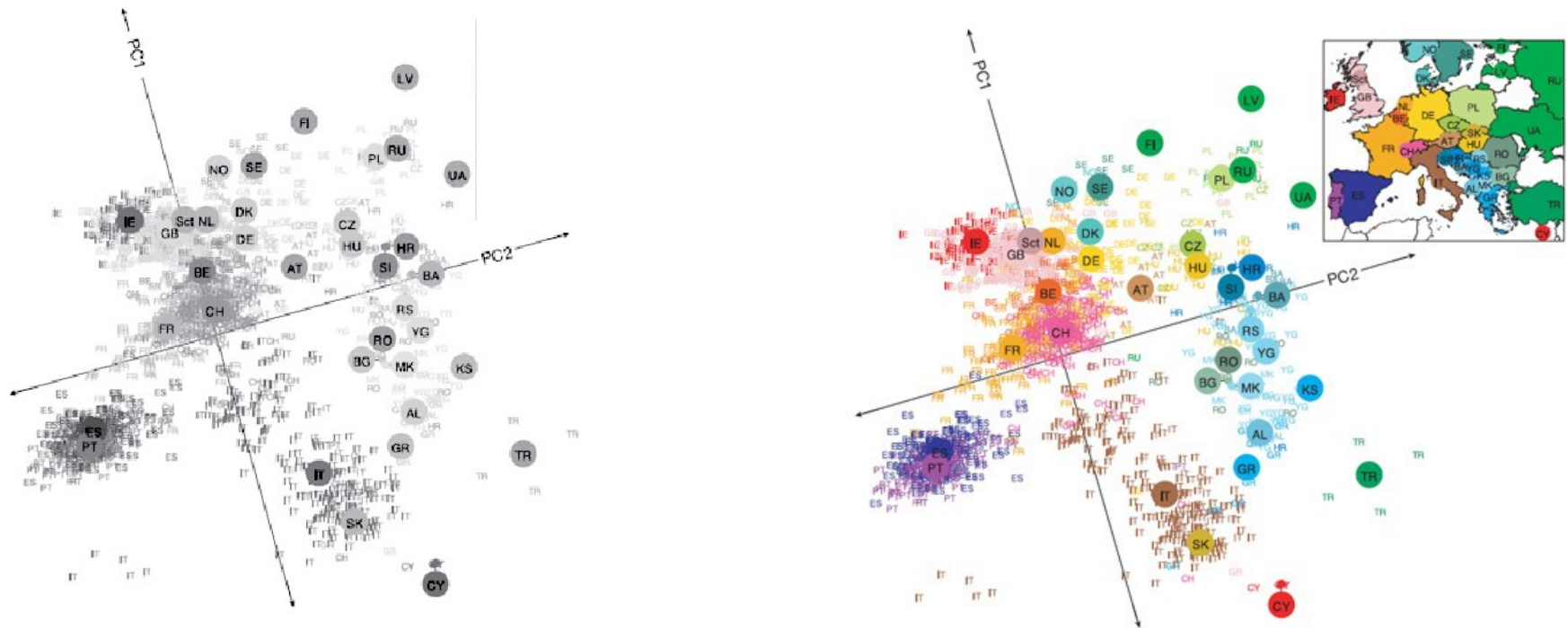
- **Unsupervised = discovering patterns in data without prior knowledge**

You do NOT have labels, or you do NOT use them

- Dimension reduction methods, e.g. PCA, Matrix factorization
- Clustering algorithms, e.g. K-means, hierarchical clustering, ...
- Outlier detection (can be then used for filtering, ..)
- ...

# Unsupervised learning examples (not only deep neural nets here)

- Dimension reduction methods, e.g. PCA, Matrix factorization
- Clustering algorithms, e.g. K-means, hierarchical clustering, SNMF ...
- Outlier detection (can be then used for filtering, ..)



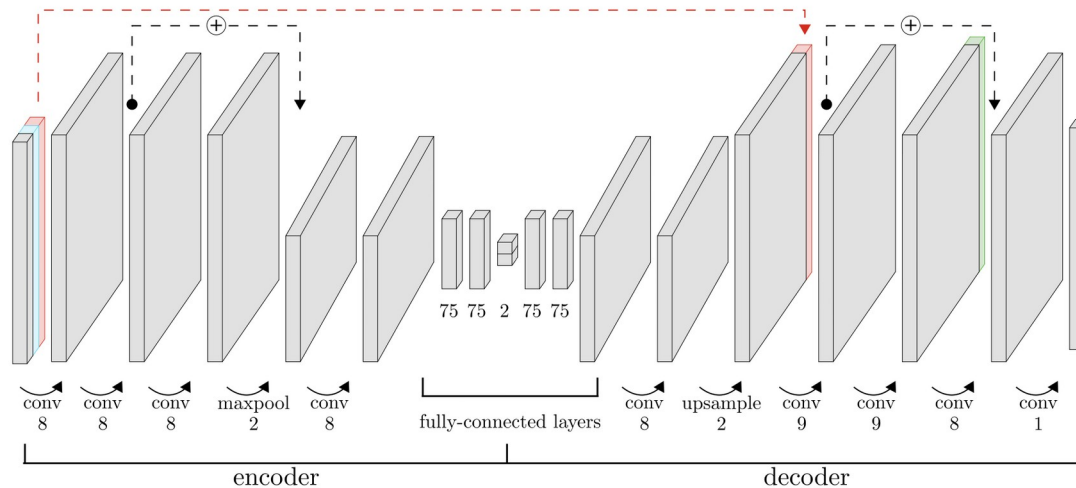
PCA to reduce high dimensional genotype data for human populations

*The 1st axis (ie the linear combination of markers) explains the largest part of the variance among samples. The 2nd axis explains the largest part of the remaining variance, and so on..*

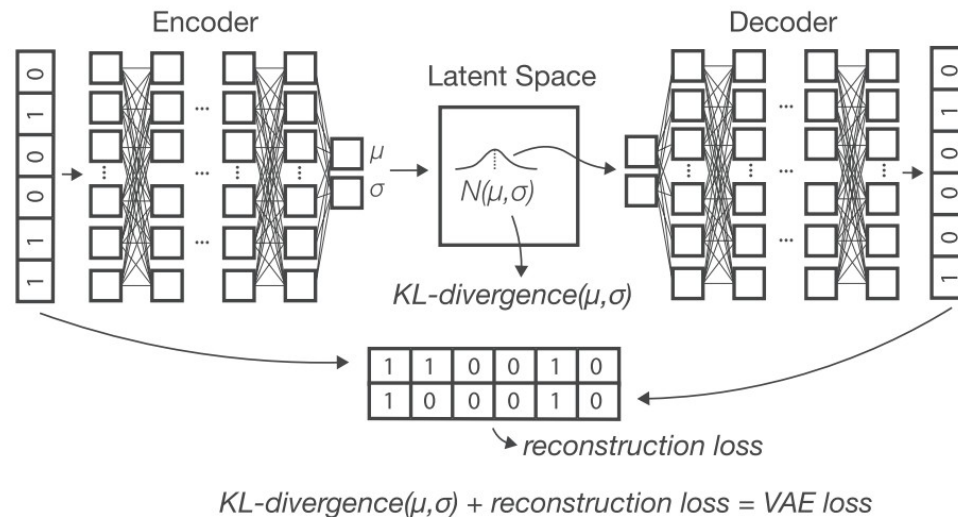
Novembre et al 2008

# Neural networks for unsupervised tasks

(i) reconstructing oneself after a strong reduction in dimension



**Convolutional Autoencoder (AE)**  
Ausmees et al 2021

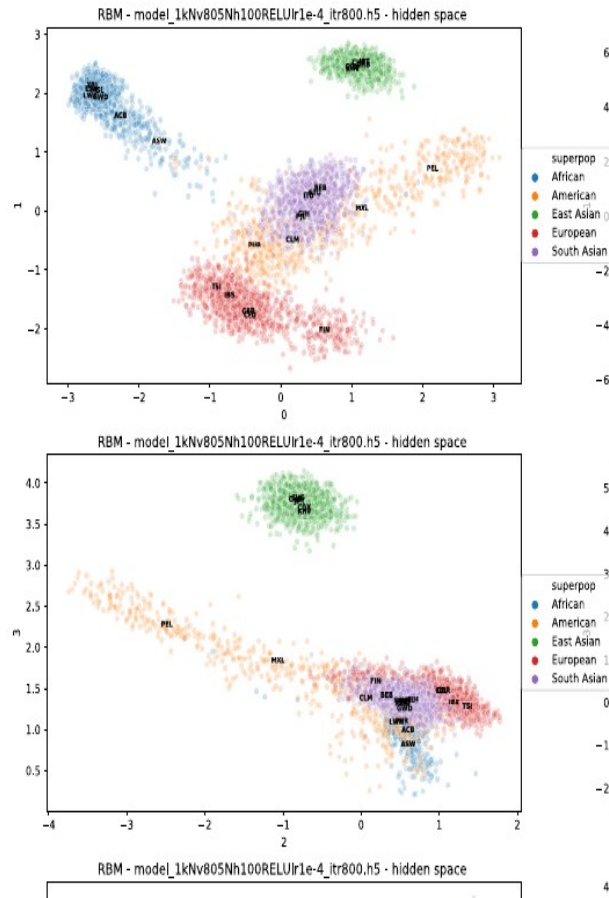


**Variational Autoencoder (VAE)**  
Battey et al 2021

# Non-linear dimension reduction based on neural networks for visualizing genetic data

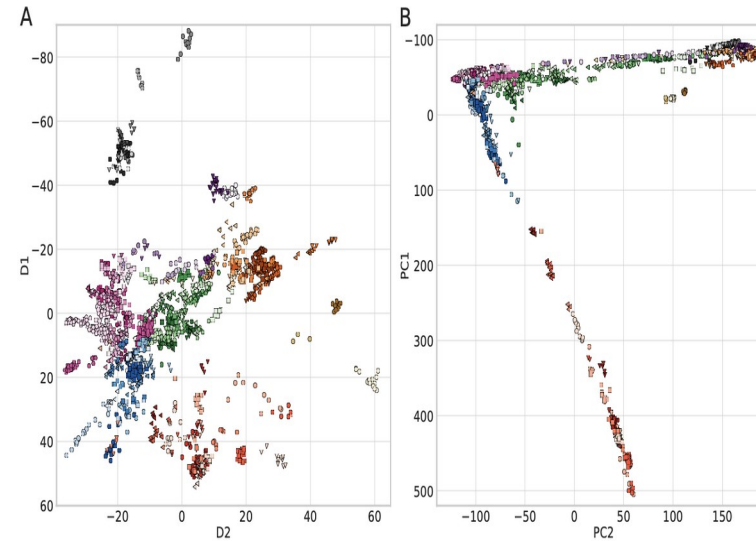
Ausmees & Nettelblad 2022 convolutional autoencoders

Yelmen et al. 2021  
restricted Boltzman machine  
RBM



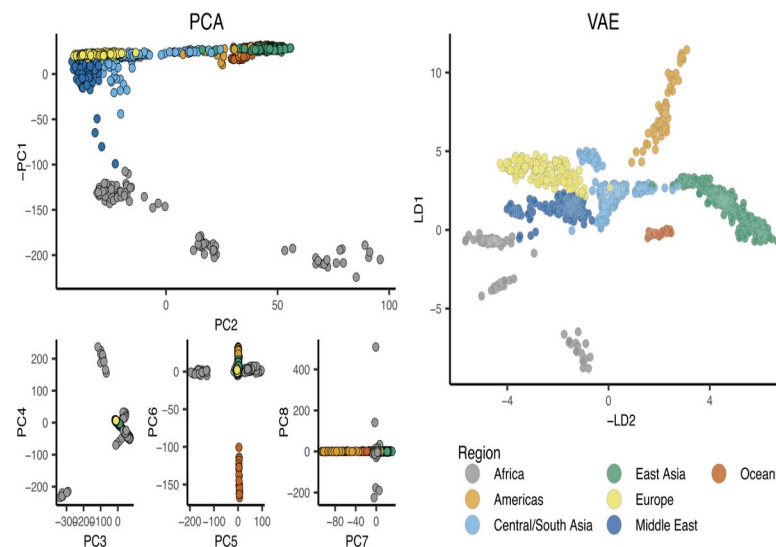
GCAE

PCA

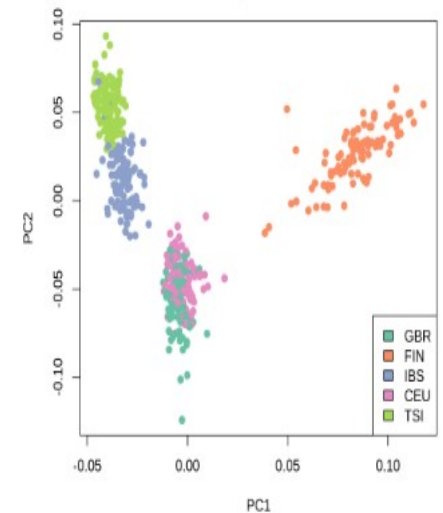


Batthey et al. 2021 - variational autoencoders (VAE)

Meisner & Albrechtsen (preprint) VAE



HaploNet



# Generative models (unsupervised learning)

**Data** with no label

**Goal** Generate samples having the same distribution as the data

Training data  $\sim p_{\text{data}}(x)$   
(distribution unknown)

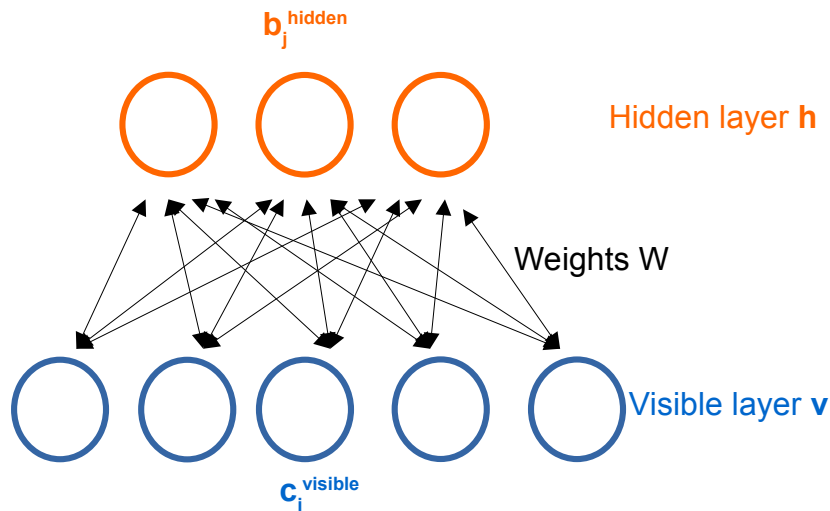


Generated samples  $\sim p_{\text{model}}(x)$



## Neural networks for unsupervised tasks

(ii) generating realistic genomes that do not belong to a real individual



Probabilistic model of the joint distribution of  $\mathbf{v}$  and  $\mathbf{h}$

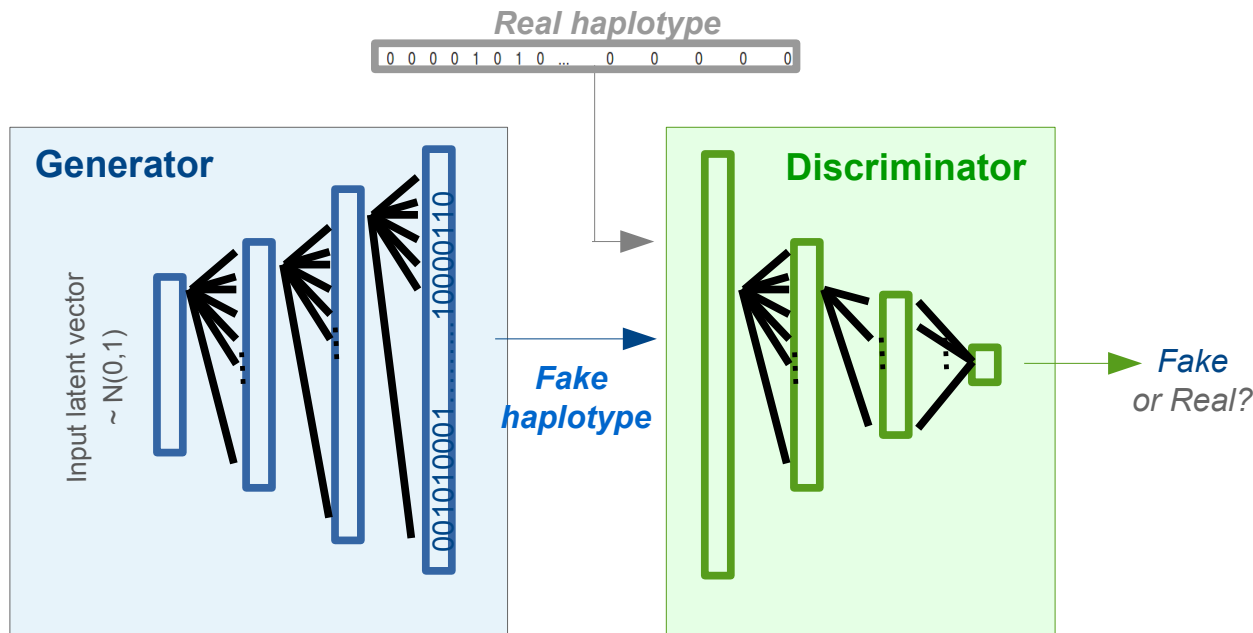
$$P(v, h) = e^{-E(v, h)} / Z$$

$Z$ : partition function

$$E(v, h) = \sum_{ij} W_{ij} v_i h_j + \text{bias terms}$$

**Restricted Boltzman Machine (RBM)**

Yelmen et al 2021



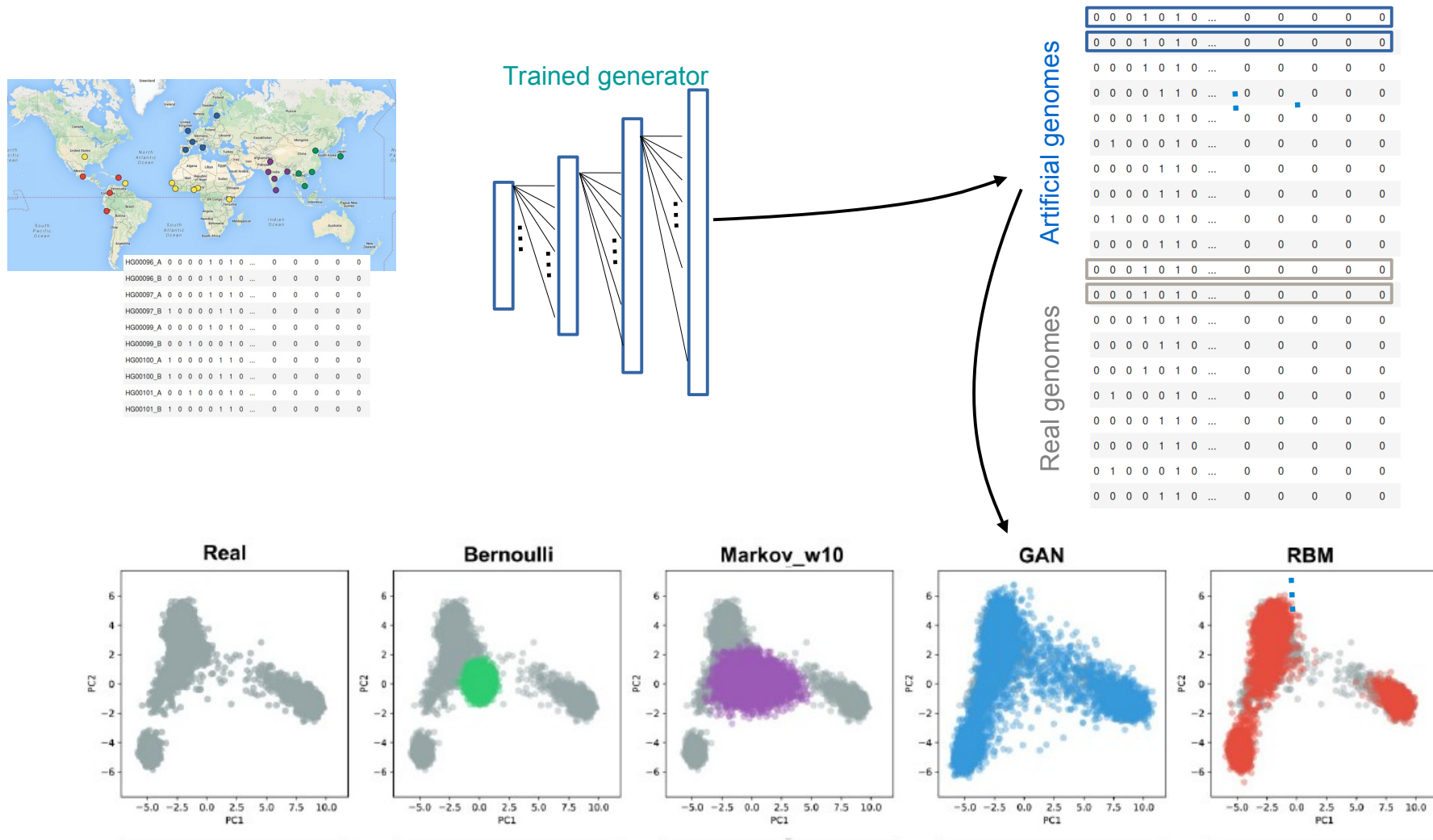
**Generative Adversarial Networks (GAN)**

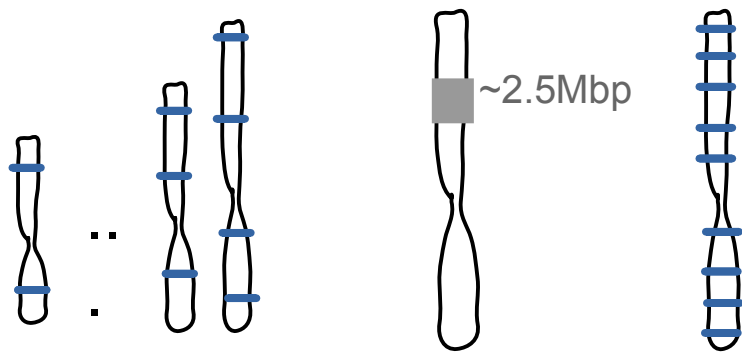
Yelmen et al 2021



# Unsupervised learning for generating realistic genomes

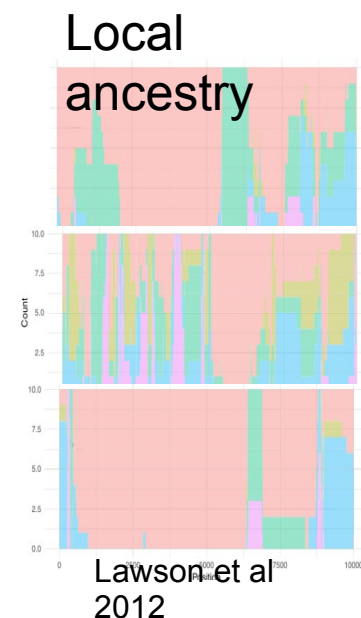
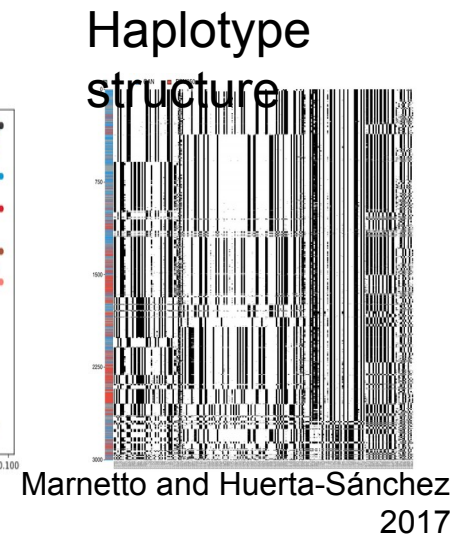
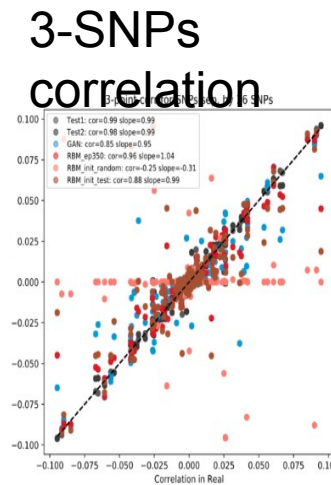
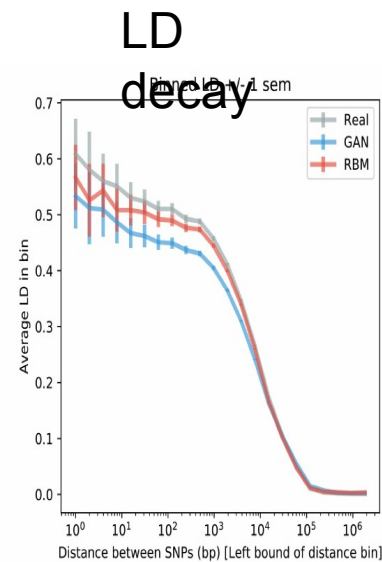
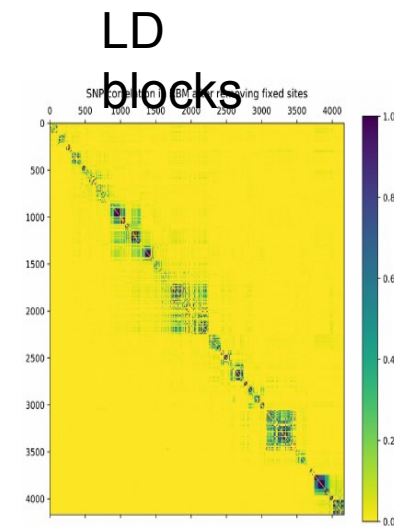
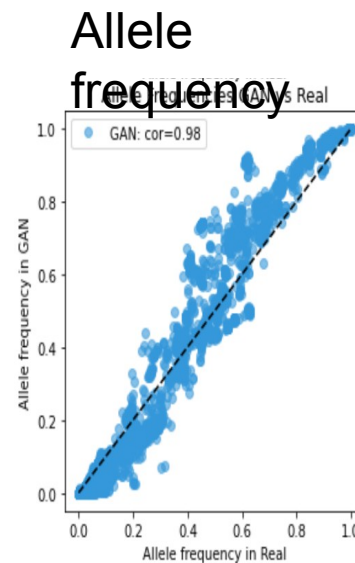
- Generative models (can also be used for dimension reduction and exploring latent space) Yelmen et al 2021 (GAN and RBM neural networks) ; Battey et al 2021 (VAE) ; Ausmees et al 2021





## Quality checks

- PCA, tSNE, UMAP
- Allele frequencies (1-point correlation)
- Linkage disequilibrium patterns (2-point correlation)
- Haplotype structure (and 3-point correlation)
- Local ancestry block patterns
- Pairwise distance distributions, ...



→ Are characteristics preserved in generated genomic sequences ?

---

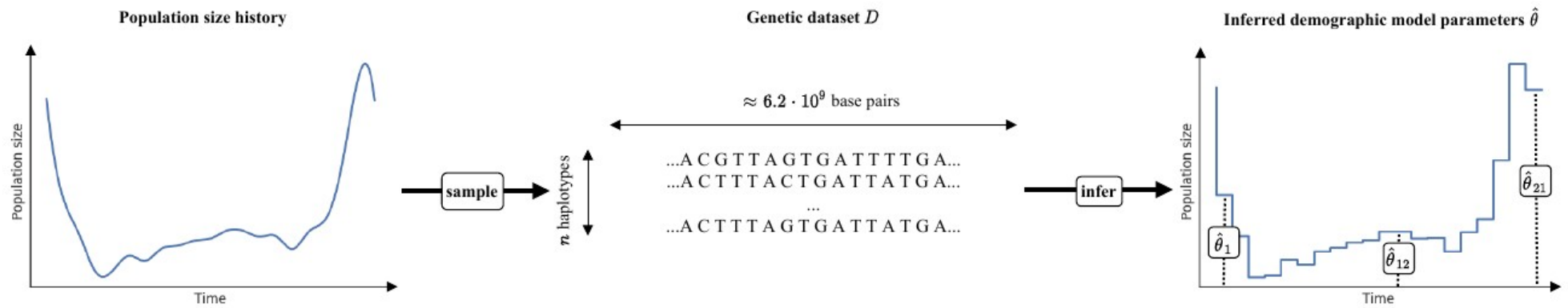
# Outline

Machine Learning: basic concepts and terminology

ML, application to popgen ; neural networks

- I. From ABC to deep learning for population genetics
- II. Learning directly from SNP data with neural networks
- III. Dissecting two published networks for effective population size inference
- IV. Opening on applications of unsupervised deep learning to popgen
- V. Tonight's hands-on: building/training/re-using DNNs for population genetics (demography/selection) inference with dnadna**

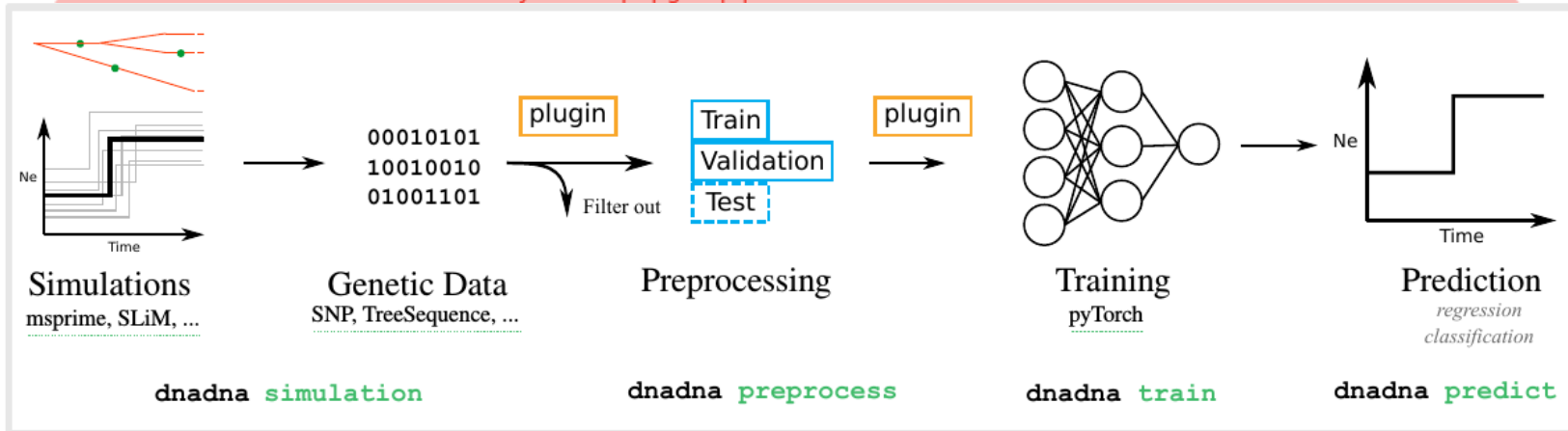
# Task of the tutorial




# DNADNA: *Deep neural architectures for DNA,* *a toolbox for population genetics inference*

Théophile Sanchez\*, EM Bray\*, Pierre Jobic, Jérémy Guez, Guillaume Charpiat, Jean Cury\*, Flora Jay\*

**dnadna**: a command-line tool for your DL-popgen pipeline <https://gitlab.com/mlgenetics/dnadna>



 Python-based package  
+ High level interface  
+ **extendable** YAML-based config format

## Aim:

- **Reproducibility + sharing** more easily networks within/outside your lab
- **Designing** networks or **training** an already designed network on your training set/task
- **Predicting** evolutionary history for your data using a **pretrained** network
- Being flexible with proper test, continuous integration, documentation

**Beta version → feedback welcome!**

**dnadna**: a command-line tool for your DL-popgen pipeline <https://gitlab.com/mlgenetics/dnadna>

**B**

Standard workflow: train a newly implemented network on existing simulations

```
1/ dnadna preprocess Demo_preprocessing_config.yml
   --dataset-config=Demo_dataset_config.yml
```

describes a previously simulated training set

```
2/ dnadna train Demo_training_config.yml --plugin local_net.py
```

→ output data and results in different self-contained folder named run\_xxx

- Try new architecture
- Update hyperparameters

```
3/ dnadna predict run_xxx/Demo_run_xxx_best_net.pth Testset/*/*.npz
```

plugins are embedded in the .pth file to facilitate sharing and reusing

**C**

Standard workflow: reuse a trained network on one's dataset

```
1/ dnadna predict trained_net.pth myData/*.npz --preprocessing
```

Contains optimized weight, and all config parameters used for training.

Contains means and std to unstandardize prediction

Apply same preprocessing e.g. filter out sequences with less than X SNPs and N individuals

**D**

Example of a training config file

```
...
# the simulation configuration
simulation:
  inherit: model_simulation_config.yml
learned_params:
  event_time:
    type: regression
    log_transform: true
    loss_func: MSE
  event_size:
    type: regression
    log_transform: true
    loss_func: MSE
network:
  name: myNet
  params:
    param1: 3
  n_epochs: 5
  optimizer:
    name: Adam
    params:
      learning_rate: 0.001
      weight_decay: 0.1
...
```

**E** Example of a network plugin

```
from dnadna import nets
from torch.nn.functional import relu

class myNet(nets.Network):
    def __init__(param1):
        super().__init__()
        self.param1 = param1
        ...
    def forward(x):
        ...
```

Only change compared to using only pytorch