GROUPNAME – Title of project for the 2024-2025 assignment

James Brown, Barry White, and Simply Red

(Dated: March 31, 2025)

Weights initialization – The main idea behind weights initialization is to help the model to break the symmetry, in fact all weights are initialized to the same value, the hidden units will learn the same feature.

In the RBM we used, the procedure of initialization for the bias a_i of a visible unit v_i is given by the formula $a_i = \log(\frac{p_i}{1-p_i})$ [3], where p_i is the fraction of training samples in which the unit i is on. When updating the visible states, p_i corresponds to the probability of having a visible state equal to 1, which is used to stochastically pick the value of each visible unit. The probability of activation for a visible neuron is

$$P(x_i = 1|h) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{1}$$

where $\sigma(x)$ is a sigmoid function. If we set $w_{ij} \sim 0$, we have $P(x_i = 1) \sim \sigma(a_i)$, hence $\sigma(a_i) = p_i$.

The biases are set so that the neurons have on average an activation that matches the distribution of the training data [2]. This helps stabilize learning and prevents the model from learning too slowly. In a qualitative way, if a pixel is often active in the dataset ($p \sim 1$), the bias will be highly positive, encouraging the neuron to be active. On the contrary, if a pixel is rarely active ($p_i \sim 0$), the bias will be highly negative, discouraging activation. If $p_i = 0.5$, the bias will be zero, meaning the neuron has a neutral activation probability.

In our code, an average on the data is made for each column obtaining the vector of means. The vector is then *refined* setting values that are too close to the lower (-1 or 0) and upper (1) state to a higher (first case) or lower value (second case). This is to avoid divergences when computing the logarithms. Finally, the function returns the vector of biases based on the formula above.

Log-likelihood – To solve a problem using Bayesian method, as we do, we have to define

- the *likelihood function*, $p(X|\theta)$, that describes the probability of observing a dataset X given the value of parameters θ
- the *prior distribution*, $p(\theta)$, that describes the *a-priori* knowledge we have about the parameters.

These two are used to compute the *posterior distribution*

$$p(\theta|X) = \frac{p(X|\theta)\,p(\theta)}{\int d\theta' p(X|\theta')\,p(\theta')} \tag{2}$$

that describes the knowledge we have about parameters θ after observing the data X. As we will see the denominator of the posterior distribution in may cases is not possible to compute analytically. Markov Chain Monte Carlo methods are required to draw random samples of $p(\theta|X)$. The likelihood function is determined by the model and the measurement noise. Many generative models follows a *Maximum Likelihood Estimation* (MLE). In MLE parameters $\hat{\theta}$ that maximize the likelihood of generating observed data are chosen. Equivalently, the log-likelihood since log is monotonic.

$$\hat{\theta} = \arg_\theta \max \log p(X|\theta) \tag{3}$$

The most common approach used for training a generative model is to maximize the log-likelihood of the training dataset. By choosing the negative log-likelihood as the cost function, the learning procedure tries to find parameters that maximize the probability of the data. The log-likelihood $\ell_\theta(x)$ per data point x, averaged over M data points, gives the log-likelihood of data

$$\mathcal{L} = \frac{1}{M} \sum_{m \le M} \ell_\theta\left(x^{(m)}\right) \tag{4}$$

[2] In training RBMs, our goal is to maximize the log-likelihood of the observed data x given the model parameters represented by a,b,w, respectively visible biases, hidden biases, weights.

$$\ell_\theta(x) = \ln \sum_z e^{-E(x,z)} - \ln \sum_{x'} \sum_z e^{-E\left(x',z\right)} \tag{5}$$

where the second therm is the partition function Z. The computation of the latter is intractable, the hard part resides in summing up the Boltmann weights of all possible configurations in Z, with D visible units and L hidden units, there are 2^{D+L} possible configurations. We followed instead the procedure suggested by Baiesi [1], that takes advantage of the energy function.

$$H_i(z) = a_i + \sum_\mu w_{i\mu} z_\mu \tag{6}$$

$$E(x,z) = -\sum_i H_i(z)\,x_i - \sum_\mu b_\mu z_\mu \tag{7}$$

$$e^{-E(x,z)} = \prod_\mu e^{b_\mu z_\mu} \prod_i e^{H_i(z) x_i} \tag{8}$$

in eq. 8 the first factor is the hidden units contribution to the energy, defined as $G(z)$. With this we can reach a reduced partition function $Z(z)$ defined as

$$Z(z) = G(z) \prod_i \left(1 + e^{H_i(z)}\right) \tag{9}$$

Parameter	Searched Interval	Description
`L`	$[3, 10]$	Hidden units
`SPINS`	*boolean*	Ising units
`POTTS`	*boolean*	One-hot encoding
`Gamma`	$\left[10^{-4}, 0\right]$	Regularization
`Grad`	SGD, RMSprop, Adam	Optimization function
`Nt`	$[1, 15]$	CD steps
`Nepoch`	$[50, 350]$	Epochs
`Nmini`	$[10, 50]$	Minibatches per epoch
`Nini`	$[5, 50]$	Initial minibatch's size
`Nfin`	$[\texttt{Nini}, 350]$	Final minibatch's size

TABLE I. Selected parameters in the random search.

This is easy to compute and becomes numerically stable limiting the argument to avoid overflow. Since we used low value of L in our RBM we can compute the partition function at the start of the training

$$\ln Z = \ln\left[\sum_z G(z)\prod_{i=1}^{D}\left(1+e^{H_i(z)}\right)\right] \tag{10}$$

for Bernoulli units, while using spin units, -1,1

$$\ln Z = \ln\left[\sum_z G(z)\prod_{i=1}^{D}\left(1+\cosh\left(H_i(z)\right)\right)\right]. \tag{11}$$

We then averaged it over $x^{(m)}$ points of the dataset to get $\mathcal{L}$. This computation of the log-likelihood is used to identify the best models after a random search. The $\mathcal{L}$ behaves well with Bernoulli variables $\{0, 1\}$ leading to values that reach ~ -140 for our best models.

Validation − We validated the log-likelihood score obtained for each model by setting N different random seeds and repeating the training N-times, then computing the mean of the log-likelihood score and standard deviation. The goal of this procedure is to account for the stochastic nature of the training process, quantifying its effect on the performance of each model.

Best model − We implemented a random search to tune the following hyperparameters as showed in table , we then looked for the best model and started fine tuning a handful of them.

[1] Marco Baiesi **Log-likelihood computation for restricted Boltzmann machines**, 1–2 (2025).

[2] P. Mehta, M. Bukov, et al. **A high-bias, low-variance introduction to Machine Learning for physicists**, (2018).

[3] G. Hinton, **A Practical Guide to Training Restricted Boltzmann Machines**, (2012)