Variation Inference for Neural Network with Bayesian Regression Initialization

Liang Wang

May 3, 2020

1 Introduction

On class we introduced many interesting machine learning algorithms from statistical prospective. Among them, neural network has been a very popular algorithm that has been proven its effectiveness in speech recognition, natural language processing and biology science. Due to its flexibility in giving feature transformation, it can easily achieve nearly perfect in sample performance, which in the meanwhile has made the penalization to prevent an unnecessary complicated model inevitably essential. Most of the penalization tricks are similar to what we had on class. For example, L_1, L_2 regularizers being added to the objective function is similar to what we had in Lasso and Ridge in regression; Dropout [**6**] of some connections among network neurons can be interpreted as an approximation of model averaging. Imposing noise [**2**] to weights training can be viewed as a Bayesian regularization term as I am about to showing in the next section. However, all those penalization methods are preliminary ideas that have to be accompanied with numerical tractability. For example, L_1, L_2 regularization will complicate the backward propagation process, dropout are still expensive to conduct if we want to know exactly how many connections we want to drop, bayesian regularizer requires almost impossible derivation of weight posteriors. Under the assumption that the evidence of lower bound is a good approximation of marginal likelihood, the Variational Inference method [**1**], however, will introduce regularization through the bayesian prospective but instead of deriving an explicit form of the weights posterior, it approximate it by solving an explicit optimization function with penalization ideas incorporated automatically.

2 Variational Inference Setup

The inference problem [**4**] is to approximate the posterior of the latent variable, weights $p(w|x, y)$ in the case of neural network, by purposing an approximation probability $q(w|\theta)$. For example, $q(w|\theta) \sim N(\mu.\sigma^2)$ with $\theta = (\mu, \sigma^2)$. If we assume σ^2 to be deterministic, then this prospective is essentially the same to adding random normal random noises to weights training [**2**], which has been proved effective in improving neutral network's out of sample performance. Of course the approximation has to be accurate enough so that useful conclusion could be drawn from sampling

of the posterior, $\hat{y|x} \sim \int p(y|w,x)q(w|x,y)dz$ for example. To quantify the approximation accuracy the KL divergence is used to derive the Evidence Lower Bound:

$$q(w) = \underset{q_\theta(w)}{\operatorname{argmin}} KL[q(w|\theta) \;||\; p(w|,x,y)]$$

Notice that $\theta = (\mu, \sigma^2)$, then we have the above equation being reduced to

$$q(w) = \underset{\theta}{\operatorname{argmin}} KL[q_\theta(w) \;||\; p(w|,x,y)] \tag{1}$$

$$KL[q_\theta(w) \;||\; p(w|,x,y)] = E[log\ q_\theta(w)] - E[log\ p(w|,x,y)]$$

$$= E[log\ q_\theta(w)] - E[log\ p(w,x,y)] + log\ p(x,y)$$

Notice that the optimization process only involves θ, we thus can drop $log\ p(x,y)$ where equation 1 will be reduced to its lower bound since $log\ p(x)$ is dropped, which is exactly the ELBO function we use as approximation:

$$\begin{aligned} \underset{q}{\operatorname{argmax}}\, ELBO(q)\ with\ ELBO(q) &= E[log\ p(w,x,y)] - E[log\ q_\theta(w)] \\ &= E[log\ p(w)] + E[log\ p(y,x|w)] - E[log\ q_\theta(w)] \\ &= E[log\ p(y,x|w)] - KL[q(w)||p(w)] \end{aligned} \tag{2}$$

If x is given determistic, we have:

$$ELBO(q) = E[log\ p(y|x,w)] - KL[q(w)||p(w)] \tag{3}$$

where we can interpret the first term as a likelihood function $L^N(\theta,x,y)$, the second term as negative distance $-L^C(\phi,x,y)$ between purposed approximation density $q(w)$ and the prior density $p(w)$. We know that the prior $p(w)$ actually also affects our ELBO function value, if we parametrize the prior with parameter ϕ, then our optimization function turns out to depend on both ϕ and θ:

$$\underset{\theta,\phi}{\operatorname{argmax}}\, E[log\ p(y|x,w)] - KL[q_\theta(w)||p_\phi(w)] \tag{4}$$

Taking the negative sign, then we can change the argmax to argmin and this form resembles to the general objective function we defined on class $L(y,\hat{y}) = Fitting\ Loss + Regularization$. Usually, the expectation in Eq 4 is calculated by sampling $w \sim q(w|\phi,\theta)$ In my neural network training, I employed the sampling calculation of expectation and adopted the negative ELBO as my objective function to be minimized:

$$\begin{aligned} &\underset{\theta,\phi}{\operatorname{argmin}}\, L(\theta,\phi,x,y) = \underset{\theta,\phi}{\operatorname{argmin}} -L^N(\theta,x,y) + L^C(\phi,x,y) \\ &= -\frac{1}{S}\sum_{k=1}^{S}[log\ p(y|x,w_k)] + KL[q_\theta(w)||p_\phi(w)] \end{aligned} \tag{5}$$

In the case of my implementation, I used Gaussian distribution for both the purposed density q_θ the global prior p_ϕ with $\theta = \mu, \sigma^2$ and $\phi = a, b^2$, i.e $q(w_i) \sim N(\mu_i, \sigma_i^2)$, $p(w_i) \sim N(a, b^2)$ for weight w_i on the i-th connection between neurons, $i = 1..W$. The L^C term thus entitles an explicit form with $L^C(\phi, x, y) = \sum_{i=1}^{W} ln\frac{\sigma}{b} + \frac{1}{2\sigma^2}[(\mu_i - a)^2 + \sigma_i^2 - b^2]$

3 Optimization

With the ELBO defining the objective function, I now explain how the ELBO can be optimized.

3.1 Stochastic Gradient Descent

For Gaussian multivariate variable $X \sim N(\mu, \Sigma)$, Using Gaussian characteristic function and integration by part, it can be shown that, the derivative of a function $V(x)$ w.r.t μ_i, σ_i are:

$$\frac{\partial E[V(X)]}{\partial \mu}|_{X\sim N(\mu,\Sigma)} = E[\frac{\partial V(X)}{\partial X}]|_{X\sim N(\mu,\Sigma)}$$

$$\frac{\partial E[V(X)]}{\partial \Sigma}|_{X\sim N(\mu,\Sigma)} = \frac{1}{2}E[\frac{\partial^2 V(X)}{\partial X^2}]|_{X\sim N(\mu,\Sigma)}$$

Combined with $L^C(\phi, x, y) = \sum_{i=1}^{W} ln\frac{\sigma}{b} + \frac{1}{2\sigma^2}[(\mu_i - a)^2 + \sigma_i^2 - b^2]$, with batch size $|B|$ we have:

$$\begin{aligned}
\frac{\partial L(\theta, \phi, x, y)}{\partial \mu_i} &= \frac{\mu_i - a}{|B|b^2} + \sum_{(X,Y)\in B} \frac{1}{S}[\frac{\partial L^N(w^k, x, y)}{\partial w_i}] \\
\frac{\partial L(\theta, \phi, x, y)}{\partial \sigma_i^2} &= \frac{1}{2}[\frac{1}{b^2} - \frac{1}{\sigma_i^2}] + \sum_{(X,Y)\in B} \frac{1}{2S}\frac{\partial^2 L^N(w^k, x, y)}{\partial w_i^2}
\end{aligned} \tag{6}$$

Note that for σ_i^2 update, I didn't use the equation 17 given in the reference since it involves a further approximation of $E[\frac{\partial^2 V(X)}{\partial X^2}]|_{X\sim N(\mu,\Sigma)} \approx E[(\frac{\partial v(x)}{\partial x})^2]|_{X\sim N(\mu,\Sigma)}$ and its approximation made the gradient of σ_i update could only be positive and thus explode the sampling oftentimes. However, for initialization of $\frac{2}{i}$ and b^2, I imposed the constraint that $b^2 < \sigma^2$ to avoid negative σ^2.

While for $\phi = (a, b^2)$ optimization, we can use directly the first order condition to update

$$\begin{aligned}
a &= \frac{1}{W}\sum_{i=1}^{W} \mu_i \\
b^2 &= \frac{1}{W}\sum_{i=1}^{W} [\sigma_i^2 + (\mu_i - a)^2]
\end{aligned} \tag{7}$$

3.2 AdaGrad for adaptive learning rate

The choice of the learning rate in deep learning is essential because a large update rate could quickly explode the our μ and σ, in which case the learning is inefficient. On the contrast, a small learning rate would results in gradient vanishing where our neural network wouldn't learn anything.

Intuitively, when the variance of our gradient is too large, we want our learning rate to be small since it represents large uncertainty in our gradient estimation, thus the AdaGrad **[5]** algorithm was proposed to adaptively change the learning rate vector at step t:

$$\rho_t = \frac{\eta}{\sqrt{cI + diag(G_t)}} \tag{8}$$

where η is the initial learning rate. I choose $\eta_\sigma = 0.02, \eta_\mu = 1$. c is positive number to maintain numerical stability. G_t is the sum of the outer product of gradient up to step t:

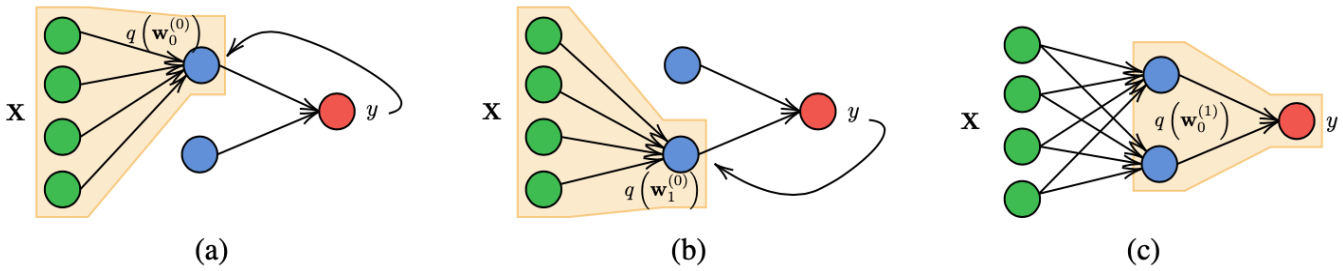
$$G_t = \sum_{i=1}^{t} g_i g_i^T$$

where g_i is the gradient vector in i-th step for all the parameters.

3.3 Bayesian Linear Regression Initialization

Since the objective function is non-convex w.r.t the, as the neural network is enriched with more layers and neurons, the algorithm can suffer from the local minimum problem. In fact, in my training process, I found the Variational inference network turns out to be more dependent on the initial μ_0 vector and initial variance matrix Σ_0. A great starting point can decrease the loss toward 0 quickly, however, some starting points, even with AdaGrad implemented, could make the ELBO oscillate around some large error and stop the algorithm. Thus the iterative Bayesian Linear Regression (I-BLM)**[3]** method was proposed to address this particular concern.

Figure 1: I-BLM Workflow



Essentially the algorithm proceed by iteratively running bayesian regression between the output of the previous layer x_{layer} to the output of outcome y and use the regression coeffecicients as the weight initalization.

Algorithm 1: Sketch of the I-BLM Initializer

Inputs : Model M, Dataset D
foreach *layer in* M **do**
  foreach *outfeature in layer* **do**
    $X, Y \leftarrow$ next batch in D;
    propagate X;
    $X_{\text{BLM}} \leftarrow$ output of previous layer;
    if *layer is convolutional* **then** ▷
      $X_{\text{BLM}} \leftarrow$ patch extraction(X_{BLM});
    if *likelihood is classification* **then** ▷
      $\text{var}(Y_{\text{BLM}}) \leftarrow \log\left[(Y+\alpha)^{-1}+1\right]$;
      $\text{mean}(Y_{\text{BLM}}) \leftarrow \log(Y+\alpha) - \text{var}(Y_{\text{BLM}})/2$;
    else
      $Y_{\text{BLM}} \leftarrow Y$;
    $p(\mathbf{w}|X,Y) \leftarrow \text{BLM}(X_{\text{BLM}}, Y_{\text{BLM}})$; ▷
    $q(\mathbf{w}) \leftarrow$ best approx. of $p(\mathbf{w}|X,Y)$; ▷

However, such an initialization algorithm requires high computation efficiency in Bayesian regression sampling as the dimension of our input increases. For my implementation, I used *PyMC3*.

3.4 Algorithm

Here I summarize my algorithms as follow

Algorithm 2 Variational Inference BNN

Data: (X_{train}, Y_{train})
Initialization define batches $|B|$, learning rate η_σ, η_μ, starting prior value $\phi_0 = (a_0, b_0)$
  use I-BLM to initialize $\theta_0 = (\vec{\mu_0}, \vec{\sigma_0}^2)$ according to Algorithm 1
for t=0 : T **do**
  load batch data X_{B_t}
  draw S sample by $q_\theta = N(\vec{\mu_t}, \vec{\sigma_t})$// in parallel
  evaluate g_t according to Eq 6
  calculate ρ_t according to Eq 8
  update $\theta_{t+1} = \theta_t + \rho_t g_t$
end
Result: Weights posterior parameter $\theta = (\vec{\mu_T}, \vec{\sigma_T}^2)$

4 Experiments

To exam the power of variational BNN with I-BLM intialization, I conducted classification experiments on both real world dataset and simulated dataset. Combined with the neural network structure, I have the conditional likelihood of binary classification in Eq 5 to be

$$-log\ p(y|x, w_k) = -\sum_{i=1}^{n} y_i log[h(x_i, w_k) + \alpha] - (1 - y_i) log[1 - h(x_i, w_k) + \alpha]$$

This is similar to logistic regression deviance except, we have $h(x_i, w_k)$ to be the transformation of neural network that map feature x into $[0, 1]$. I choose $\alpha = 0.01$ to maintain the numerical stability of the stochastic gradient descent. The computation was conducted on Boston University Shared Computing Cluster [BUSCC] after parallelization.

4.1 Cancer Dataset

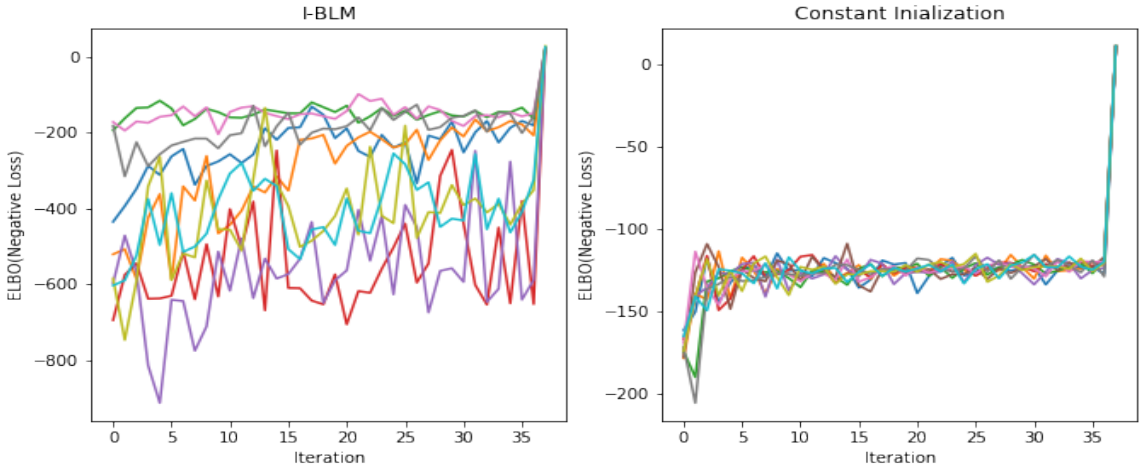
Breast Cancer Wisconsin (Diagnostic) Database provides data for benign and malignant diagnosis of cancer with 3 group (mean, standard error, largest value), 10 of each real valued feature input. Specifically, those values are the mean, standard error and the largest value of:

1. radius
2. texture
3. perimeter
4. compactness
5. area
6. smoothness
7. concavity
8. concave points
9. symmetry
10. fractal dimension

In the appendix Figure 8, I visualized their relationships to the diagnosis result. For simplicity, I am demonstrating only the mean value of each features in my training process. I used scatter plot of every pair of the features to demonstrate their relationship difference(red is benign and blue is malignant)

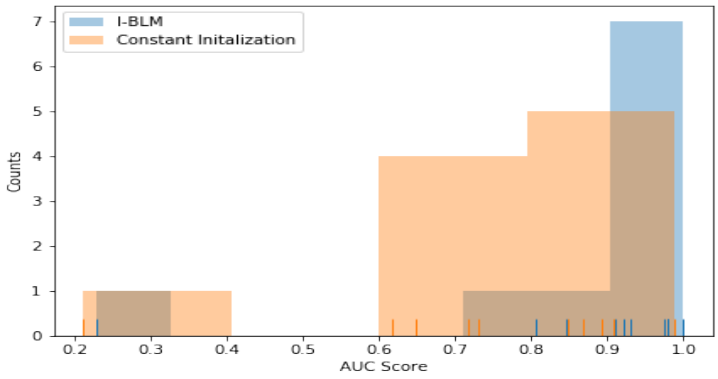
As we can see, almost all pairs of those 10 mean features demonstrates different cluster center according to the group of benign and the group of malignant. Since this is a small dataset, I used batch size 20 per training and 200 samples to approximate the gradients, 3 layers neural network of 7 neurons in the hidden layer. Those choices are also generally consistent Yann Lecun's empirical instructions [**7**]. To better evaluate the out of sample performance, I bootstrapped the dataset 10 times and keep the last 50 observations as my testing dataset and here is the AUC distribution plot and optimized ELBO plot for the result.

Figure 2: Iteration Convergence Comparison



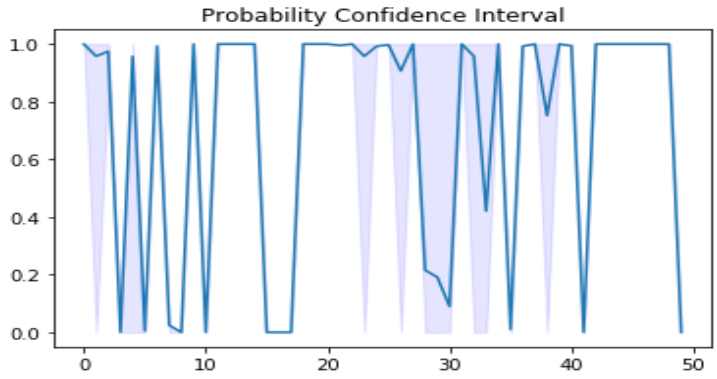
As we can see, the I-BLM method adjust itself according to different batch data but the constant initialization initialize $\theta_0 = (0, 1)$ for all the data point. In this small neural network example, we know that it both of them converges to 0. However in terms of the out of sample AUC plot, the I-BLM approach has way better and stable performance compared to the constant initialization method.

Figure 3: AUC 10 Fold Comparison



Note that we not only could see what is the predict probability, we are also able to know how confident we are in that predication with the confidence intervals plotted below.

Figure 4: Confidence Interval of Predicted Probability



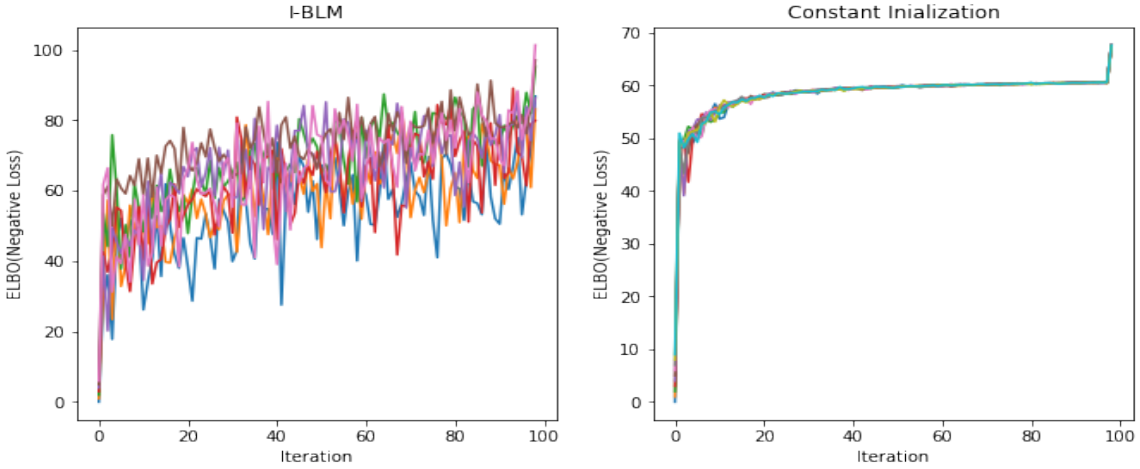
Not surprisingly, since the dataset is small and our neural network structure is also small, the out of sample predicted probability has small variance, which means the neural network are almost certain about most of the predicted values.

4.2 Simulated Dataset

To study the initialization improvement for a larger neutral network, I simulated 2000 data of 30 dimensions where X is generated according to standard Gaussian distribution with multiple centers of clusters, then I randomly combined within each cluster in order to add covariance. The scatter plot is presented in the appendix Figure 9. Similarly, I bootstrapped the dataset 10 times

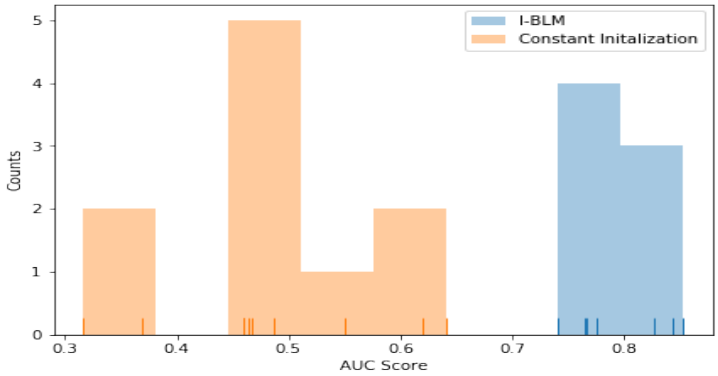
and adopted empirical rule [**7**] of 15 neurons to draw some useful conclusion.

Figure 5: Iteration Convergence Comparison



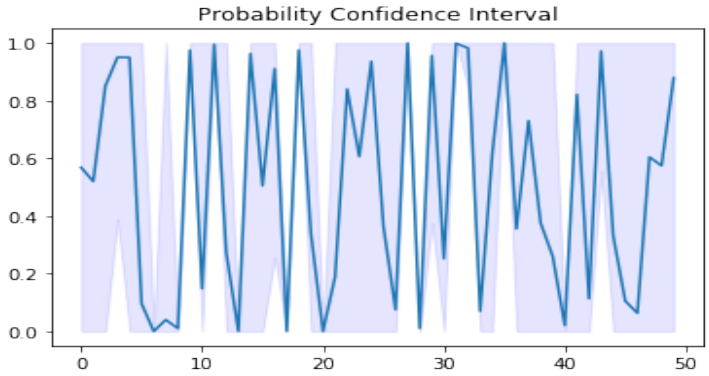
Again, the I-BLM method adjust itself according to different batch data but the constant initialization initialize $\theta_0 = (0, 1)$ for all the data points. obivously, the I-BLM gives us a higher ELBO compared to the constant initialization network since the constant initialization network tends to get stuck by its local minimum, which can be further confirm by the following auc comparison plot:

Figure 6: AUC K- fold Comparison



The blue I-BLM AUC is at least 0.75 compared to the AUC of the constant initialized neural network. And to judge for whether or not the learning is complete, we could again take a look at the predicted probability confidence interval plot.

Figure 7: Confidence Interval of Predicted Probability



Since I just simulated 2000 samples, I have the predicting probability with larger variance, which indicates the requirements for more training samples.

5 Conclusion

In conclusion, I studied and reimplemented vartional BNN with I-BLM intialization and AdaGrad algorithm. Compared to original neural network machine learning, the variational BNN model not only provides us with the uncertainty quantification but also requires less computation in hyper parameters tuning to generalize out of sample performance. The AdaGrad method was implemented and has showed tremendous improvements on preventing weights exploding. The I-BIM initialization accelerate the convergence and also improves on the local minimum problem.

References

[1] A Graves(2011), Practical variational inference for neural networks, Advances in neural information processing systems

[2] Ian Goodfellow, and Yoshua Bengio (2015), Deep Learning Book by Aaron Courville,

[3] Simone Rossi, Pietro Michiardi, Maurizio Filippone (2018), Good Initializations of Variational Bayes for Deep Models, 36th International Conference on Machine Learning.

[4] DM Blei, A Kucukelbir, JD McAuliffe (2017), Variational inference: A review for statisticians, Journal of the American Statistical Association,

[5] Duchi, John; Hazan, Elad; Singer, Yoram (2011). "Adaptive subgradient methods for online learning and stochastic optimization" (PDF). JMLR. 12: 2121–2159.

[6] Pierre Baldi, Peter J. Sadowski (2013), Understanding Dropout, Advances in Neural Information Processing Systems 26 (NIPS 2013)

[7] YA LeCun, L Bottou, GB Orr, KR Müller (1998), Efficient backprop Neural networks: Tricks of the trade, 9-48

Appedix

1. Python Code

I uploaded the python3.6 code on github.

2. Breat Cancer Scatter Plot

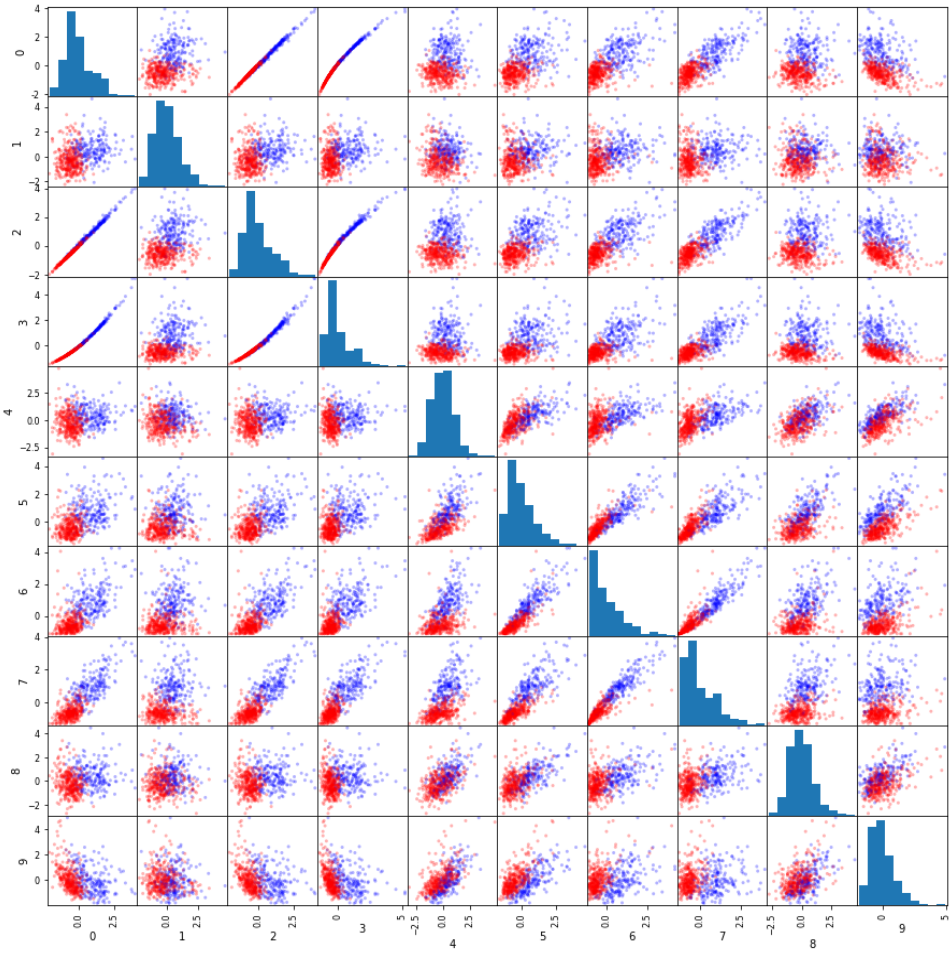


Figure 8: Scatter Plot of Breast Cancer Data

3. Simulated Data Scatter Plot

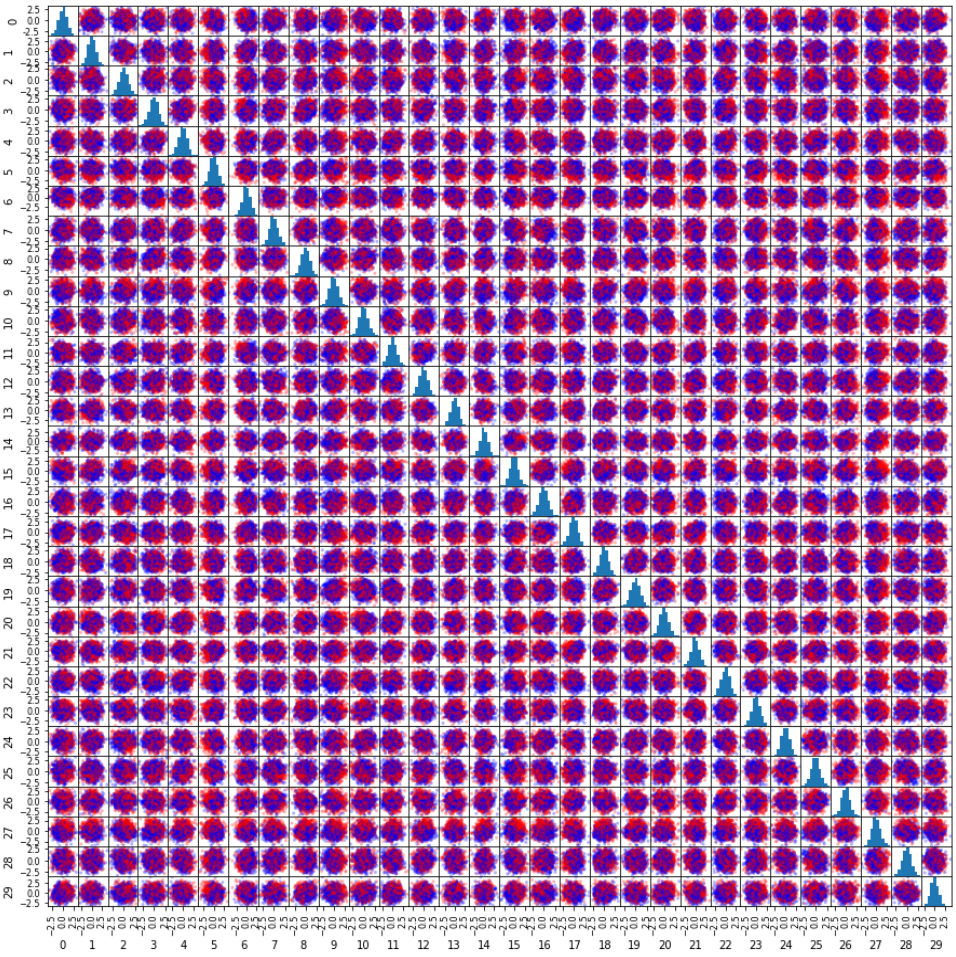


Figure 9: Scatter Plot of Simulated Data