Extreme Learning Machine

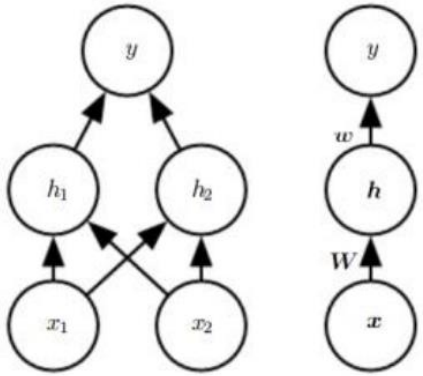
Qi Wang

ELM Application

- Introduction to NN
- ELM Theory
- One Dimensional Case
- Two Dimensional Case
- Result

Introduction to NN

Neural Networking (Single Layer)



1. With each input x_i, we expand it into many "Hidden Units" h_{ij}, meaning the j^{th} hidden unit of the x_i. $I = (W^T x + c)$

2. Activate the function. $f^{(1)}(x; W, c) = g(W^T x + c)$

- ReLU: $f(x) = \max\{0, g(x)\}$, Sigmoid: $f(x) = logit^{-1}(x) = \frac{\exp(x)}{1+\exp(x)}$

3. Getting the output layer parameter by using the LSE. $f^{(2)}(h; \omega, b) = \omega^T h + b$

Unknown Parameters: W, c, ω, b

So complicated!

What if we can generate W, c ?

- It becomes LSE!

ELM Theory

ELM Theory(Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006))

Theorem 1.2.1

Given a standard SLFN with N hidden nodes and activation function $g{:}R \to R$ which is infinitely differentiable in any interval, for N arbitrary distinct samples (x_i,t_i), where $x_i \in R^n$ and $t_i \in R^m$, for any w_i and b_i randomly chosen from any intervals of R^n and R, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix H of the SLFN is invertible and $||H\beta - T|| = 0$

Theorem 1.2.2

Given any positive value $\varepsilon > 0$ activation function $g{:}R \to R$ which is infinitely differentiable in any interval, there exists $\tilde{N} \leq N$ such that for N arbitrary distinct samples (x_i,t_i), where $x_i \in R^n$ and $t_i \in R^m$, for any w_i and b_i randomly chosen from any intervals of R^n and R, respectively, according to any continuous probability distribution, then with probability one, $||H_{\tilde{N}\times N}\beta_{\tilde{n}\times m} - T_{N\times m}|| = 0$

Therefore, we can randomly generate the slope and intercept in the first step. But which distribution to choose?

Main Purpose:

- Centered around 0.
- Easy to generate.

N(0,1) & Unif[-1, 1]

One Dimensional Case

One Dimensional Case

Data Simulation

$$Y_i = f(X_i) + \epsilon_i \ , \ \epsilon_i \sim_{iid} N(0, \sigma_1^2)$$

$$f(x) = D_1(x) + 7 * D_2(x) + 2 * D_3(x)$$

D_i is the PDF of $N(\nu_i, \tau_i^2)$, i = 1, 2, 3

$$\nu_1 = 1, \nu_2 = 5, \nu_3 = 9, \tau_1 = \tau_2 = \tau_3 = 1$$

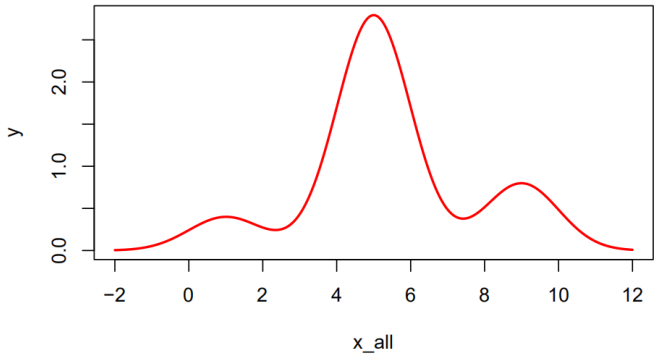


Figure 1: Data Generating Process for Model 1

Factors affecting fitting result:

1. Distribution of Generating Parameters
- Normal? Uniform?

2. Number of Hidden Nodes
- 10, 50, 100, 200, 500?

3. Activation Function
- ReLU? Sigmoid?

One Dimensional Case

Fitting Results

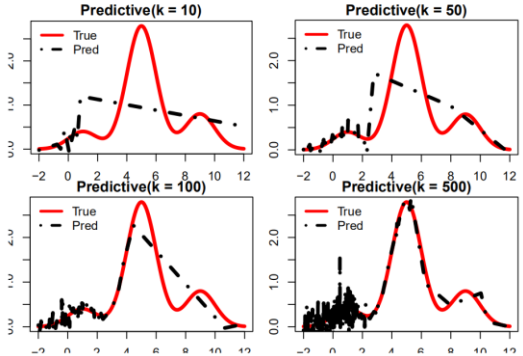


Figure 4: Uniformly Generating(ReLU Activation)

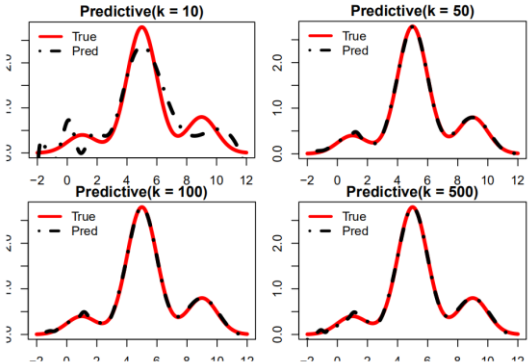


Figure 5: Normally Generating(Sigmoid Activation)

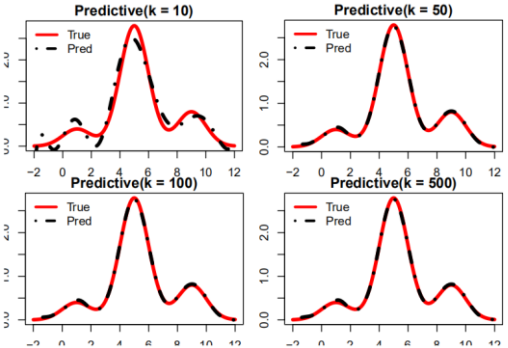


Figure 3: Uniformly Generating(Sigmoid Activation)

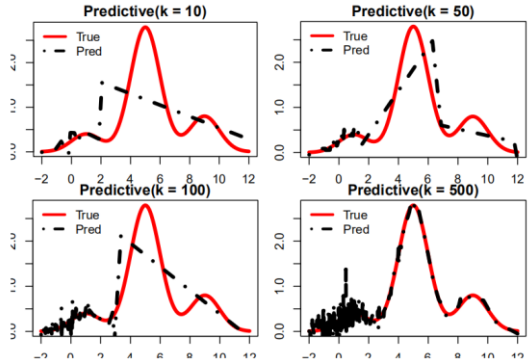


Figure 6: Normally Generating(ReLU Activation)

Sum of Squared Error on each 0.01 step in [-2, 12]

Table 1: Prediction Error(Uniform)

	Sigmoid	ReLU
k=10	41.2319964	673.26028
k=50	0.6247817	411.76893
k=100	0.6388939	153.94517
k=500	0.6248940	29.00473

Table 2: Prediction Error(Normal)

	Sigmoid	ReLU
k=10	92.7380924	548.10285
k=50	0.8518948	199.56869
k=100	0.9884457	266.45380
k=500	1.0671437	14.47762

Two Dimensional Case

Two Dimensional Case

Data Simulation

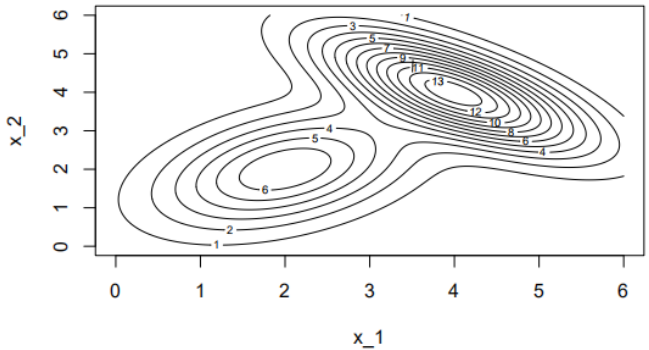
$$Y_i = f(X_{1i}\ ,\ X_{2i}) + \epsilon_i\ ,\ \epsilon_i \sim_{iid} N(0, \sigma_2^2)$$

$$f(x_1, x_2) = 4 * D_1(x_1, x_2) + 6 * D_2(x_1, x_2)$$

D_i is the PDF of $\mathrm{MV}N(\mu_i, \Sigma_i)$, i = 1, 2

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$$

$$\mu_2 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & -0.7 \\ -0.7 & 1 \end{bmatrix}$$



Factors affecting fitting result:

1. Distribution of Generating Parameters
- Normal? Uniform?

2. Number of Hidden Nodes
- 10, 50, 100, 200, **500**?

3. Activation Function
- ReLU? Sigmoid?

Two Dimensional Case

Fitting Results

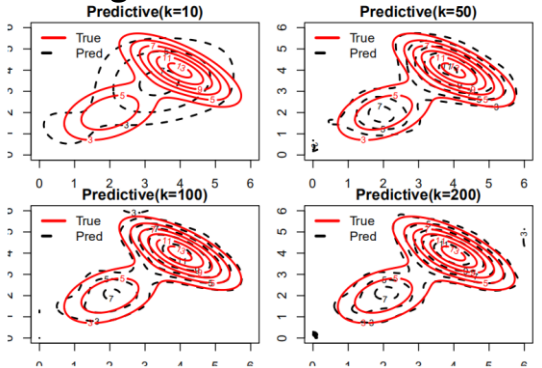


Figure 7: Uniformly Generating(Sigmoid Activation)

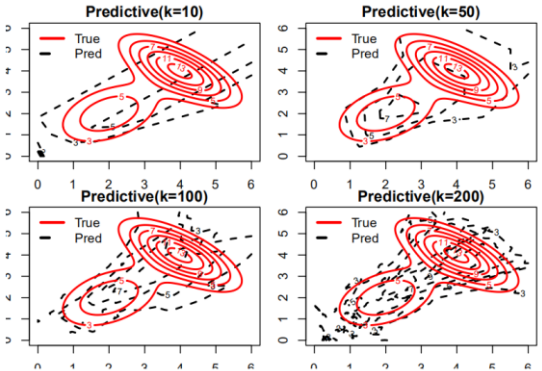


Figure 8: Uniformly Generating(ReLU Activation)

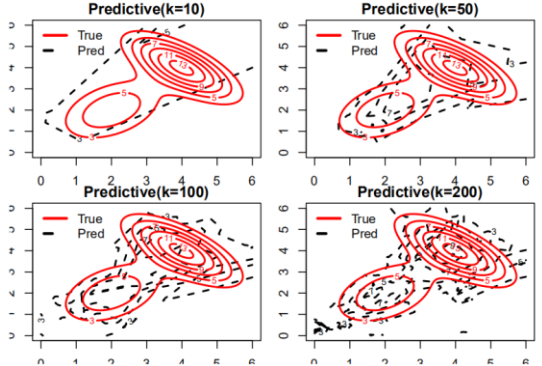


Figure 10: Normally Generating(ReLU Activation)

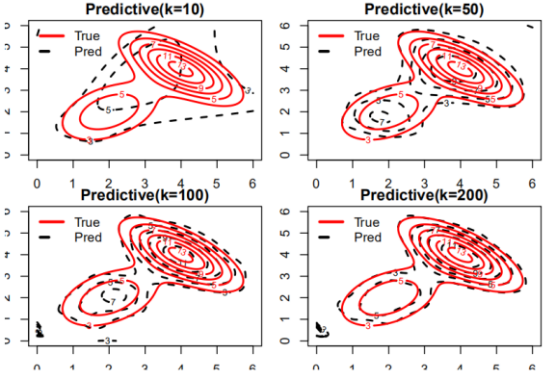


Figure 9: Normally Generating(Sigmoid Activation)

Sum of Squared Error on each 0.01 grid in [0,6]*[0.6]

Table 3: 2D Prediction Error(Uniform)

	Sigmoid	ReLU
k=10	1441691.3	2142516
k=50	244945.0	1602831
k=100	170364.5	1290168
k=200	260025.8	1096784

Table 4: 2D Prediction Error(Normal)

	Sigmoid	ReLU
k=10	1681628	2391985
k=50	400342	1442464
k=100	1339670	1221827
k=200	1675013	1281670

Result

Conclusion

For one-dimensional case, the sigmoid activation function performance better than ReLU function. And uniformly generating function performs better for sigmoid activation, but normal performs better in ReLU activation case. It could be explained by the extreme case generated by normal is rare, which leads more “useful” hidden units whose components are not all 0.

Table 1: Prediction Error(Uniform)

	Sigmoid	ReLU
k=10	41.2319964	673.26028
k=50	0.6247817	411.76893
k=100	0.6388939	153.94517
k=500	0.6248940	29.00473

Table 2: Prediction Error(Normal)

	Sigmoid	ReLU
k=10	92.7380924	548.10285
k=50	0.8518948	199.56869
k=100	0.9884457	266.45380
k=500	1.0671437	14.47762

Conclusion

For two dimensional case, the results become more complicated. But sigmoid function still performs better than ReLU function when the parameters are uniformly generated.
However, when the parameter is normally generated, and the number of hidden nodes is large, ReLU function case has a better prediction. That could be the case that the sigmoid function will be easier to overfit the model because when k=50, it approaches the smallest prediction error.

Table 3: 2D Prediction Error(Uniform)

	Sigmoid	ReLU
k=10	1441691.3	2142516
k=50	244945.0	1602831
k=100	170364.5	1290168
k=200	260025.8	1096784

Table 4: 2D Prediction Error(Normal)

	Sigmoid	ReLU
k=10	1681628	2391985
k=50	400342	1442464
k=100	1339670	1221827
k=200	1675013	1281670

Conclusion

Overall, in this case, since the function are both smooth, sigmoid performs better than ReLU. Also, Uniformly generating the parameters will let the model explore the properties of the data better. Normally generating the parameters will have better performance in the case when overfitting exists.