Neural Networks

Cristián Aguilera-Arellano
UMN

March 22, 2021

Introduction

- A neural network is a two-stage regression or classification model
- A single-layer neural network is a specific case of a **project pursuit regression**
- Cybenko (1989) proved a single-layer neural network is a **universal approximator**
- The central idea is to extract linear combinations of the inputs, and then model the target as a nonlinear function of these features

Projection Pursuit Regression

- Input vector $X \in \mathbb{R}^p$ and a target $Y \in \mathbb{R}^k$
- α_m $m = 1, ..., M$ is a unit p-vector of unknown parameters
- g_m are unspecified functions to be estimated along with α_m – use smoothers (smoothing splines or local regressions) to approximate g_m
- The project pursuit regression (PPR) model has the form:

$$f(X) = \sum_{m=1}^{M} g_m(\alpha_m^T X)$$

- The model can approximate any continuous function in $\mathbb{R}^p$ – **global approximator**
- Given training data (x_i, y_i) $i = 1, 2, ..., n$ the objective is to,

$$\min_{\{g_m, w_m\}_{m=1}^{M}} \sum_{i=1}^{n} \left[y_i - \sum_{m=1}^{M} g_m(\alpha_m^T x_i) \right]^2$$

- Neural Networks – Fix $g_m(\alpha_m^T X) = \beta_m \sigma(\alpha_{0m} + \alpha_m^T X)$ where β_m is a parameter and $\sigma(\cdot)$ is a sigmoid function

Solution Algorithm

Neural Networks

- Nonlinear statistical models
- For regression, usually there is only one output unit at the top (Y_1)
- For K-class classification (diagram), there are K target measurements $Y_k, k = 1, ..., K$ each being coded as a $0 - 1$
- Z_m $m = 1, ..., M$ hidden units (neurons) are created from linear combinations of the inputs and Y_k as a function of linear combinations of Z_m

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \ m = 1, ...M$$
$$T_k = \beta_{0k} + \beta_k^T Z, \ k = 1, ...K$$
$$f_k(X) = g_k(T), \ k = 1, ...K$$

- $\sigma(v)$ is called the activation function and is usually chosen to be the sigmoid $\sigma(v) = 1/(1 + e^{-v})$
- For a regression usually $g_k(T) = T_k$, and for the K-class classification $g_k(T) = e^{T_k} / \sum_{l=1}^{K} e^{T_l}$

Neural Networks diagram

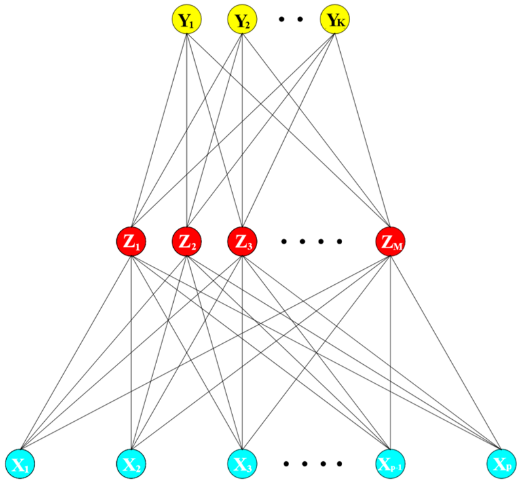


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

Fitting Neural Networks

- The unknown parameters of the model, also called weights consists of

$$\alpha \equiv \{\alpha_{0m}, \alpha_m; m = 1, \ldots, M\} \; M(p+1)\text{weights}$$
$$\beta \equiv \{\beta_{0k}, \beta_k; k = 1, \ldots, K\} \; K(M+1)\text{weights}$$
$$\theta \equiv (\alpha, \beta)$$

- For regression, we use sum-of-squared errors as a measure of fit,

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$$

- For classification,

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i)$$

- To avoid overfitting some regularization is needed (penalty term)
- The goal is to minimize $R(\theta)$ – gradient descent (***back-propagation***)

Neural Networks-Back Propagation (1/3)

- Let $A \equiv \alpha_{0m} + \alpha_m^T X$, $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$, and $z_i = (z_{1i}, ..., z_{mi})$
- The overall model,

$$X \xrightarrow{\alpha} A \xrightarrow{\sigma} Z \xrightarrow{\beta} T \xrightarrow{g} \hat{f}$$

- Our task is to compute $\nabla_\theta R(\theta)$,

$$\frac{\partial R(\theta)}{\partial \beta_{km}} = -2 \sum_{i=1}^{N} (y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}$$

$$\frac{\partial R(\theta)}{\partial \alpha_{ml}} = -\sum_{i=1}^{N} \sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}$$

- A gradient descent update at the $(r+1)st$ iteration has the form,

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \frac{\partial R(\theta)}{\partial \beta_{km}^{(r)}}$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \frac{\partial R(\theta)}{\partial \alpha_{ml}^{(r)}}$$

Neural Networks-Back Propagation (2/3)

- γ_r is the learning rate – it is usually a constant. It can also be optimized by a grid search that minimizes the error function at each update
- Rewriting the FOC,

$$\frac{\partial R(\theta)}{\partial \beta_{km}} = -2\sum_{i=1}^{N}(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi} \equiv \sum_{i=1}^{N}\delta_{ki}z_{mi}$$

$$\frac{\partial R(\theta)}{\partial \alpha_{ml}} = -\sum_{i=1}^{N}\sum_{k=1}^{K}2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il} \equiv \sum_{i=1}^{N}s_{mi}x_{il}$$

- s_{mi} can be expressed in terms of δ_{ki},

$$s_{mi} = \sigma'(\alpha_m^T x_i)\sum_{k=1}^{K}\beta_{km}\delta_{ki}$$

Neural Networks-Back Propagation -Summary (3/3)

1. In the **forward pass**, the current weights are fixed

$$X \xrightarrow{\alpha} A \xrightarrow{\sigma} Z \xrightarrow{\beta} T \xrightarrow{g} \hat{f}$$

2. In the **backward pass**, the errors δ_{ki} are computed and **back-propagated** by

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km}\delta_{ki}$$

3. δ_{ki} and s_{mi} are used to compute the gradients

$$\frac{\partial R(\theta)}{\partial \beta_{km}} = \sum_{i=1}^{N} \delta_{ki} z_{mi} \text{ and } \frac{\partial R(\theta)}{\partial \alpha_{ml}} = \sum_{i=1}^{N} s_{mi} x_{il}$$

4. **Update**

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \frac{\partial R(\theta)}{\partial \beta_{km}^{(r)}} \text{ and } \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \frac{\partial R(\theta)}{\partial \alpha_{ml}^{(r)}}$$

Some Issues in Training Neural Networks (1/2)

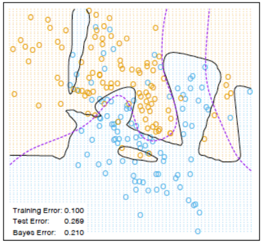
- **Starting values**: usually starting values for weights are chosen to be random values near zero
  - Use of exact zero weights leads to zero derivatives and perfect symmetry, and the algorithm never moves
  - Starting with large weights often leads to poor solutions
- **Overfitting**: An explicit method for regularization is weight decay. We add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

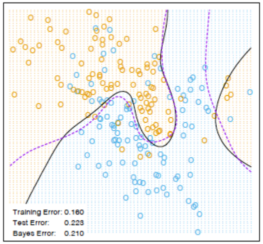
  $\lambda \geq 0$ is a tuning parameter. We can use cross-validation to estimate λ (weight decay)
- **Scaling inputs**: Since the scaling of the inputs determines the effective scaling of the weights in the bottom layer, it is recommended to standardize all inputs to have zero mean and standard deviation one.

Weight Decay - Example

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02



Some Issues in Training Neural Networks (2/2)

- **Multiple minima**: The error function $R(\theta)$ is nonconvex, possessing many local minima. As a result, the final solution depends on initial weights $\rightarrow$ try different random starting configurations

- **Number of hidden units and layers**: cross-validation can be used to estimate the optimal number of hidden units. Choice of the number of hidden layers is guided by background knowledge and experimentation

Example 1: Simulated Data

- Data is generated from two additive error models $Y = f(X) + \epsilon$:

$$\text{Sum of sigmoids} : Y = \sigma(a_1^T X) + \sigma(a_2^T X) + \epsilon_1$$

$$\text{Radial} : Y = \prod_{m=1}^{10} \phi(X_m) + \epsilon_2$$

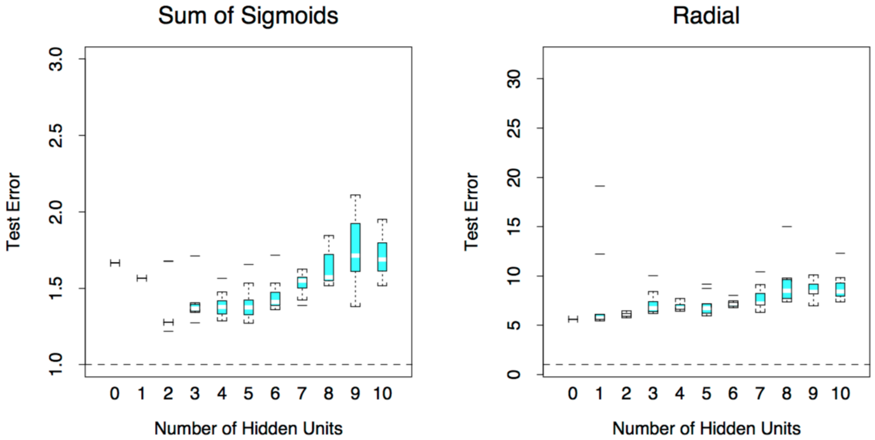
  where $X^T = (X_1, ..., X_p)$, each X_j being a standard Gaussian variate, with $p = 2$ in the first model, and $p = 10$ in the second.
- For the sigmoid model $a_1 = (3, 3)$, $a_2 = (3, -3)$; for the radial model, $\phi(t) = (1/2\pi)^{1/2} exp(-t^2/2)$
- ϵ_1 and ϵ_2 are Gaussian errors, with variance chosen so that the signal-to-noise ratio

$$\frac{Var(E(Y|X))}{Var(Y - E(Y|X))} = \frac{Var(f(X))}{Var(\epsilon)}$$

  is 4 in both models.
- Training sample of size 100 and test sample of size 10,000

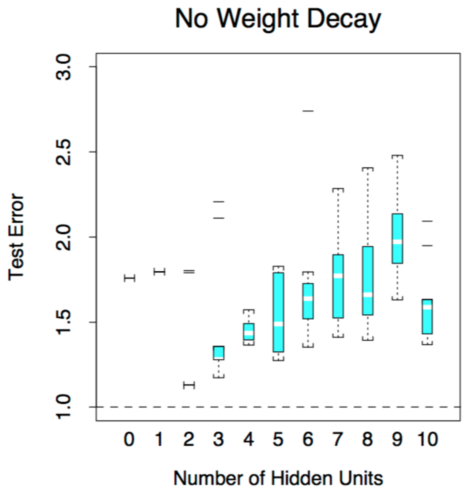
Example 1- Performance ($\lambda = 0.0005$)

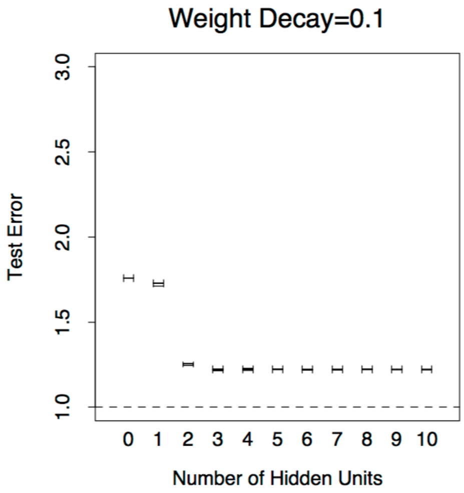
- Hidden units (0-10), weight decay $\lambda = 0.0005$, one hidden layer
- 10 random starting weights
- In the figures, it is reported the test error relative to Bayes error



Example 1- Performance of Sigmoid Model ($\lambda = 0$ vs $\lambda = 0.1$)

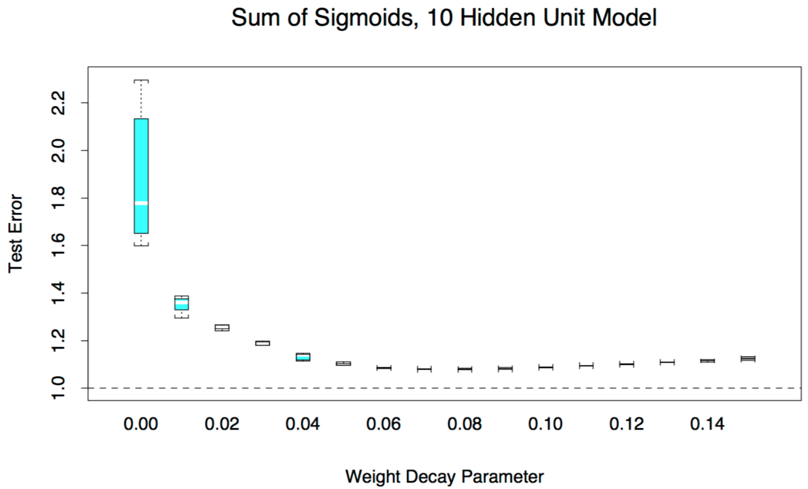
- True function is a sum of two sigmoids





Example 1- Performance of Sigmoid model (for different λ's)

- True function is a sum of two sigmoids



Example 2: ZIP Code Data

- The objective is to classify handwritten numerals
- Total of 256 inputs (16×16 pixels)

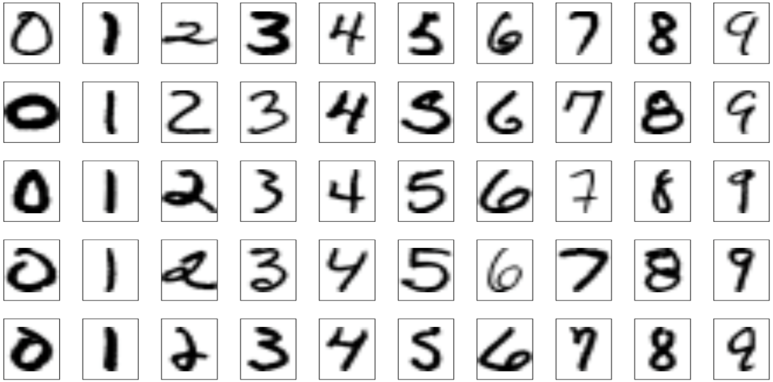


FIGURE 11.9. *Examples of training cases from ZIP code data. Each image is a 16 × 16 8-bit grayscale representation of a handwritten digit.*

Example 2-Structure

- **Net-1**: No hidden layer, equivalent to multinomial logistic regression

- **Net-2**: One hidden layer, 12 hidden units fully connected

Constrained networks:

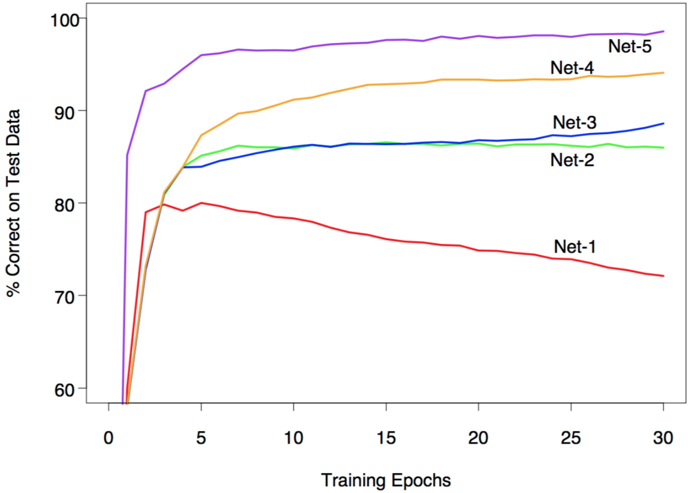
- **Net-3**: Two hidden layers locally connected
  - Each unit in the first hidden layer (8x8 array) takes inputs from a 3x3 patch of the input layer (each patch is two pixels apart from the next patch) NN diagram
  - The second layer (4x4 array) takes inputs from a 5x5 patch of the first layer
  - Weights of all other connections are set to zero

- **Net-4**: Two hidden layers, locally connected with weight sharing

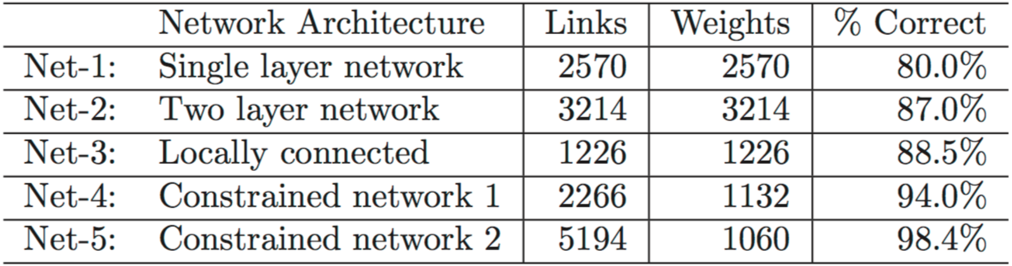
- **Net-5**: Two hidden layers, locally connected, two levels of weight sharing

Example 2- Test performance

- An epoch is a single pass over the entire data set



Example 2- Test performance



	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

Applications

- Kaji, Manresa, and Pouliot (2020) propose a new simulation-based estimation method that uses neural networks for structural models

- Neff (2021) adapts neural network methods for text analysis to analyze how the wording in statements of the Federal Open Market Committee (FOMC) impacts fed funds futures (FFF) prices

Conclusions

- Project Pursuit Regression
- Fit Neural Networks
- Issues training Neural Networks

References

- Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1, No. 10). New York: Springer series in statistics.
- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Neural Networks diagram

back

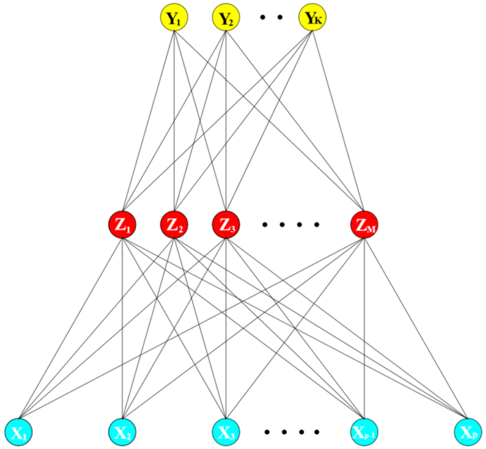
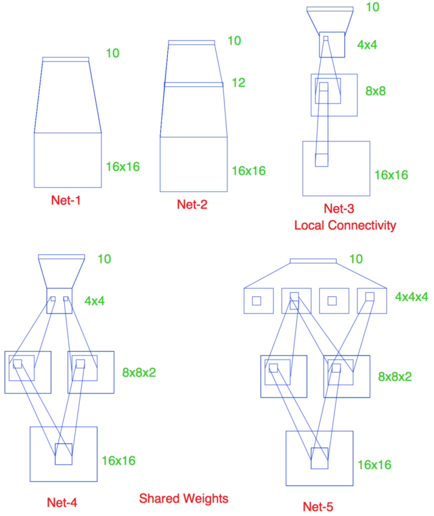


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

Example 2-diagram



Projection Pursuit Regression-Solution Algorithm back

- Impose constraints on the g_m, to avoid overfit solutions
- Suppose $M = 1$, given w, we have a one-dimensional smoothing problem
- Given g we want to minimize the objective function over w, we can use a Gaussian-Newton search,

$$g(w^T x_i) \approx g(w_{old}^T x_i) + g'(w_{old}^T x_i)(w - w_{old})^T x_i$$

$$\sum_{i=1}^{n} \left[y_i - g(w^T X) \right]^2 \approx \sum_{i=1}^{n} g'(w_{old}^T x_i)^2 \left[\left(w_{old}^T x_i + \frac{y_i - g(w_{old}^T x_i)}{g'(w_{old}^T x_i)} \right) - w^T x_i \right]^2$$

  where w_{old} is the current estimate for w
- Minimize the right-hand side and get a new vector w_{new}
- Estimation of g and w are iterated until convergence
- If $M > 1$ the model is built in a **forward stagewise** manner, adding a pair (w_m, g_m) at each stage
- After each step the g_m's can be readjusted using **backfitting**
- M is usually estimated as part of the **forward stagewise** strategy. The model building stops when the next term does not improve the fit of the model

Linear smoothers

- A linear smoother is a regression function of the training outputs

$$\hat{f}(x_*) = \sum_i w_i(x_*) y_i$$

  where $\hat{f}$ is the predicted value of the output and x_* are the inputs.
- Examples of Linear smoothers: kernel regression, locally weighted regression, Gaussian process regression, $\underbrace{\textbf{k-nearest-neighbors}}$, smoothing splines, etc.

$$\hat{f}(x_*) = \frac{1}{k} \sum_{i \in \underbrace{N_k(x_*)}_{\text{k-neighbor set of x}}} y_i$$

$$w_i(x_*) = \begin{cases} 0 & \text{if } i \notin N_k(x_*) \\ 1/k & \text{if } i \in N_k(x_*) \end{cases}$$