# Lecture 22 Neural Network II

## ECE 625: Data Analysis and Knowledge Discovery

Di Niu

Department of Electrical and Computer Engineering
University of Alberta

April 6, 2021

# Outline

# Fitting Neural Networks

- The unknown parameters in neural network model are called weights, denoted by $\theta$, which includes

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \cdots, m\}\ M(p+1)\text{weights},$$
$$\{\beta_{0k}, \beta_k;\ k = 1, 2, \cdots, K\}\ K(M+1)\text{weights}.$$

- In regression, we minimize RSS

R_i

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

- In classification, we minimize cross-entropy (deviance)

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

and the corresponding classifier is $G(x) = \arg\max_k f_k(x)$.

## Back-Propagation

regression

- The generic approach to minimizing $R(\theta)$ is by gradient descent, called back-propagation in this setting.
- Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and $z_i = (z_{1i}, \cdots, z_{Mi})$. The derivatives of $R(\theta)$ are

df_k(x_i)/d beta_km

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi},$$

dT_k/d alpha_ml

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

=dT_k/dZ_m * dZ_m/d alpha_ml

- which can be rewritten as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}, \quad \frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}, \tag{1.1}$$

where the quantities $\delta_{ki}$ and $s_{mi}$ are "errors" from the current model at the output and hidden layer units, respectively.

# Back-Propagation

- It can easily be shown that

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km}\delta_{ki}, \tag{1.2}$$

known as the back-propagation equations.

- Given the derivatives, a gradient descent update at $(r+1)$ iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \tag{1.3}$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},$$

where $\gamma_m$ is the learning rate.

# Back-Propagation

- In the forward pass, the current weights are fixed and the predicted values $\hat{f}_k(x_i)$ are computed from formula in the last lecture.
- In the backward pass, the errors $\delta_{ki}$ are computed, and then back-propagated via (1.2) to give the errors $s_{mi}$.
- Both sets of errors are then used to compute the gradients for the updates in (1.3) via (1.1).
- This two-pass procedure is what is known as back-propagation.
- Back-propagation can be slow. Other methods include second-order techniques, conjugate gradients and variable metric methods.

# Training Neural Networks

- Starting Values. Usually starting values for weights are chosen to be random values near zero. Hence the model starts out nearly linear, and becomes nonlinear as the weights increase.
- Overfitting. Often neural networks have too many weights and will overfit the data at the global minimum of $R$.
- A more explicit method for regularization is weight decay, which is analogous to ridge regression, that is

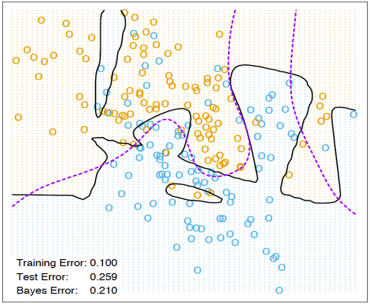
$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2.$$

- Other penalties are proposed as well, for example, the weight elimination penalty

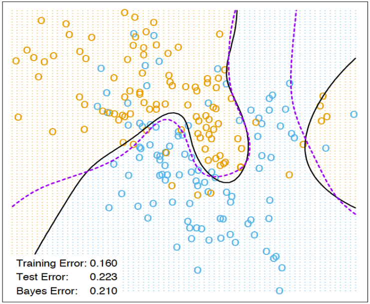
$$J(\theta) = \sum_{km} \beta_{km}^2/(1 + \beta_{km}^2) + \sum_{ml} \alpha_{ml}^2/(1 + \alpha_{ml}^2).$$

# Training Neural Network

Neural Network - 10 Units, No Weight Decay




Neural Network - 10 Units, Weight Decay=0.02




The broken purple boundary is the Bayes error rate. Both use the softmax activation function and cross-entropy error.

# Training Neural Network

- Number of hidden units and layers. Generally speaking it is better to have too many hidden units than too few.
- Multiple Minima. The loss function $R(\theta)$ is nonconvex and hence possesses many local minima.
- One must at least try a number of random starting configurations, and choose the solution giving lowest (penalized) error.
- Another approach is via bagging, which averages the predictions of networks training from randomly perturbed versions of the training data.

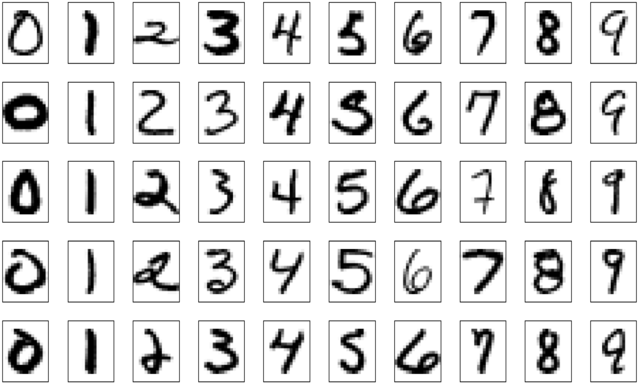
# Training Neural Network

- In summary, there are two free parameters to select: the weight decay $\lambda$ and number of hidden units $M$ as in

$$R(\theta) + \lambda J(\theta).$$

- As a learning strategy, one could fix either parameter at the value corresponding to the least constrained model, to ensure that the model is rich enough, and use cross-validation to choose the other parameter.

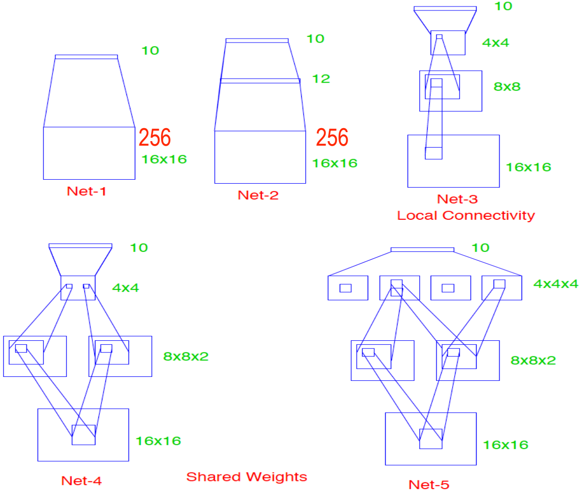
# Zip code data



Examples of training cases from ZIP code data. Each image is a $16 \times 16$ 8-bit grayscale representation of a handwritten digit.

# Zip code data

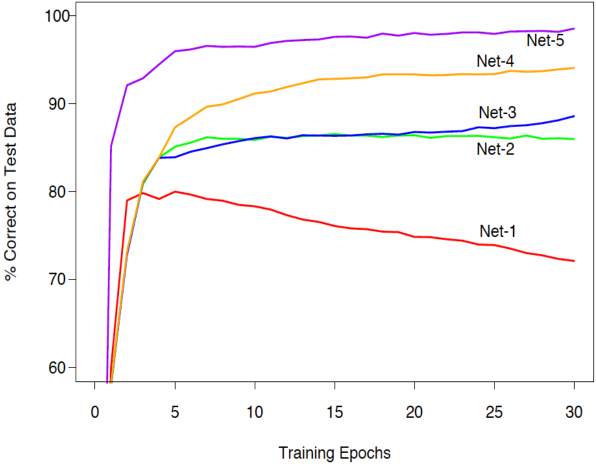
convolutional neural networks (CNN)



# Zip code data

- Net-1: No hidden layer, equivalent to multinomial logistic regression.
- Net-2: One hidden layer, 12 hidden units fully connected.
- Net-3: Two hidden layers locally connected.
- Net-4: Two hidden layers, locally connected with weight sharing.
- Net-5: Two hidden layers, locally connected, two levels of weight sharing.

# Zip code data



# Summary and Remark

- Back propagatioon
- Training neural network
- Zip code data
- Read textbook Chapter 11 and `R` code
- Do `R` lab