CS 559 – Machine Learning: Fundamental and Application

Lecture 6 – Neural Network

Outline

6.1. Lecture 5 Review
6.2. Introduction to Neural Network
6.3. Feedforward Functions
6.4. NN – Regression
6.5. NN – Classification
6.6. Optimization
6.7. Backpropagation
6.8. Numerical Example
6.9. Conclusion

6.1. Lecture 5 Review

In Lecture 5, we discussed linear classifiers.

1. The assumptions in linear regression will be carried over.
2. Discriminant Model – Directly assigns the target class label.
    1. LDA: Transforms into 1-D space and constructs the hyperplane model.
    2. Perceptron: Updates weights from the previous learn and constructs the hyperplane model.
3. Probabilistic Model – Assigns the target class label based on the Bayes learning
    1. Generative Model – MLE
        1. Finds the posterior probability of weights
    2. Discriminative Model – Logistic Regression
        1. Uses the logistic sigmoid function and stochastic descent optimization

6.1. Review - Optimization

Gradient Descent (GD)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L(\boldsymbol{w})$$

$\nabla_w TrainLoss$ denotes the gradient of the (average) total training loss with respect to ***w.***

Stochastic Gradient Descent (SGD)

For each $(x, y) \in D_{train}$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L(x, y, \boldsymbol{w})$$

$\nabla_w L$ denotes the gradient of one example loss with respect to ***w.***

6. Neural Networks

6.2. Introduction to Neural Networks

- Often associated with biological devices (brains), electronic devices, or network diagrams
  - Proposed by Alexander Bain in 1873 and William James 1890
  - Interactions among neurons within the brain
- But the best conceptualization for this presentation is none of these: Think of neural network as a mathematical function.

6.2. Introduction to Neural Networks

Predicting Car Collision (Yes or No)

Suppose the 1-D coordinate of two vehicles (x_1 and x_2) are known. Predict if the collision will occur or not based on the simple classification model

$$y = sign(|x_1 - x_2| - 1).$$

We will say no if $y = +1$ and yes if $y = -1$.

Input: position of two oncoming cars $\boldsymbol{x} = [x_1, x_2]^T$

Output:

- $\boldsymbol{x} = [1,3]^T, y = +1$; No
- $\boldsymbol{x} = [3,1]^T, y = +1$; No
- $\boldsymbol{x} = [1,0.5]^T, y = -1$; Yes

x_1	x_2	y
1	3	1
3	1	1
1	0.5	-1

Predicting Car Collision (Yes or No)

Suppose we know when y will be +1 based on the position of one vehicle with respect to another vehicle. Let the new learning model be

$$y = sign(h_1 + h_2)$$

where h_i determines if the vehicle i is far right of j vehicle $h_i = (x_i - x_j \geq 1 : 0)$.
The collision will not occur if either h is true.

$\boldsymbol{x}$	h_1	h_2	y
$[1,3]^T$	0	1	+1
$[3,1]^T$	1	0	+1
$[1,0.5]^T$	0	0	-1

Input Data

Intermediate Learning (hidden layers)

Prediction

6. Neural Networks

6.3. Feedforward Network Functions

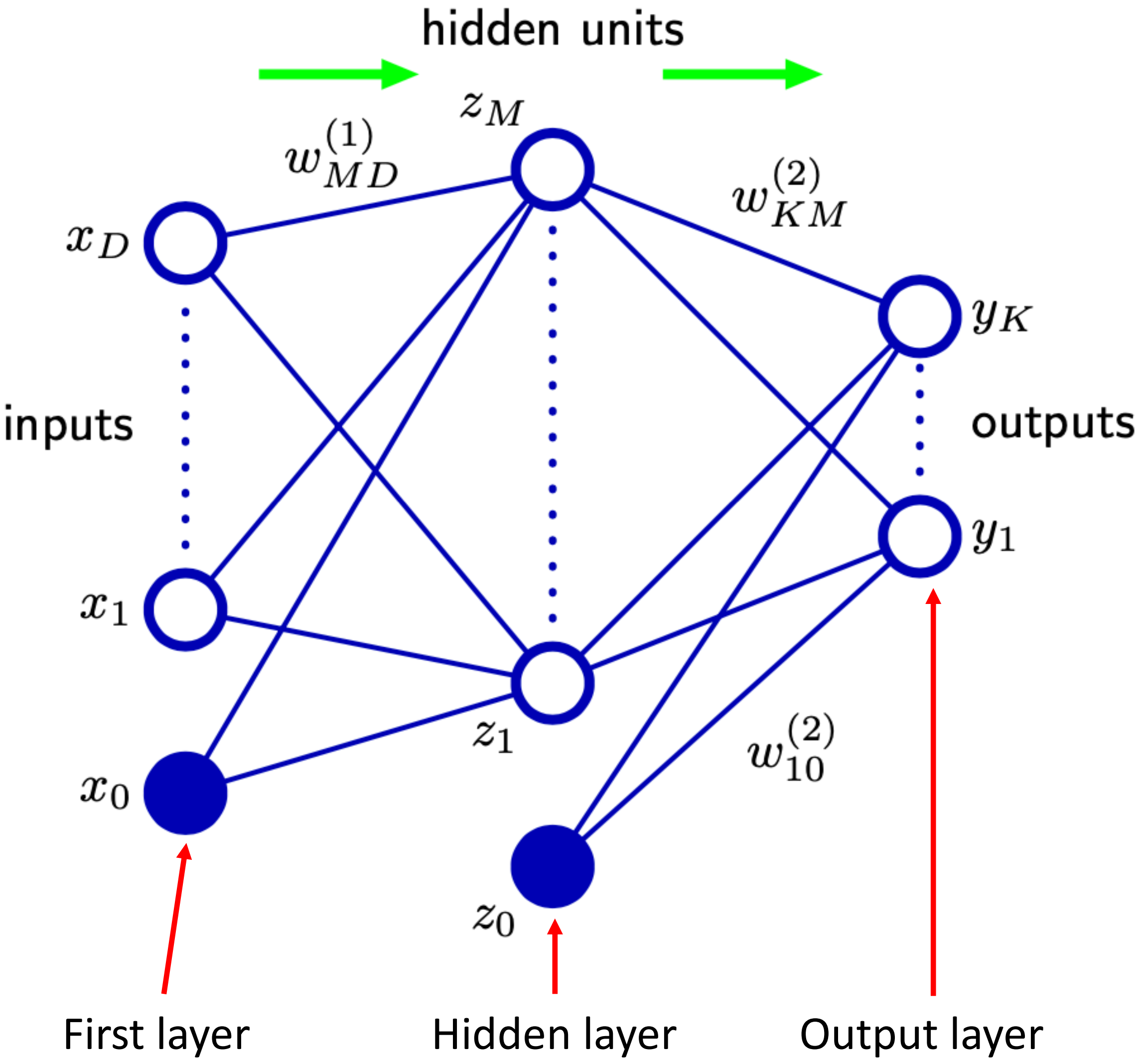
❑ Consider a linear model based on linear combinations or fixed nonlinear basis function $\phi_j(\boldsymbol{x})$:

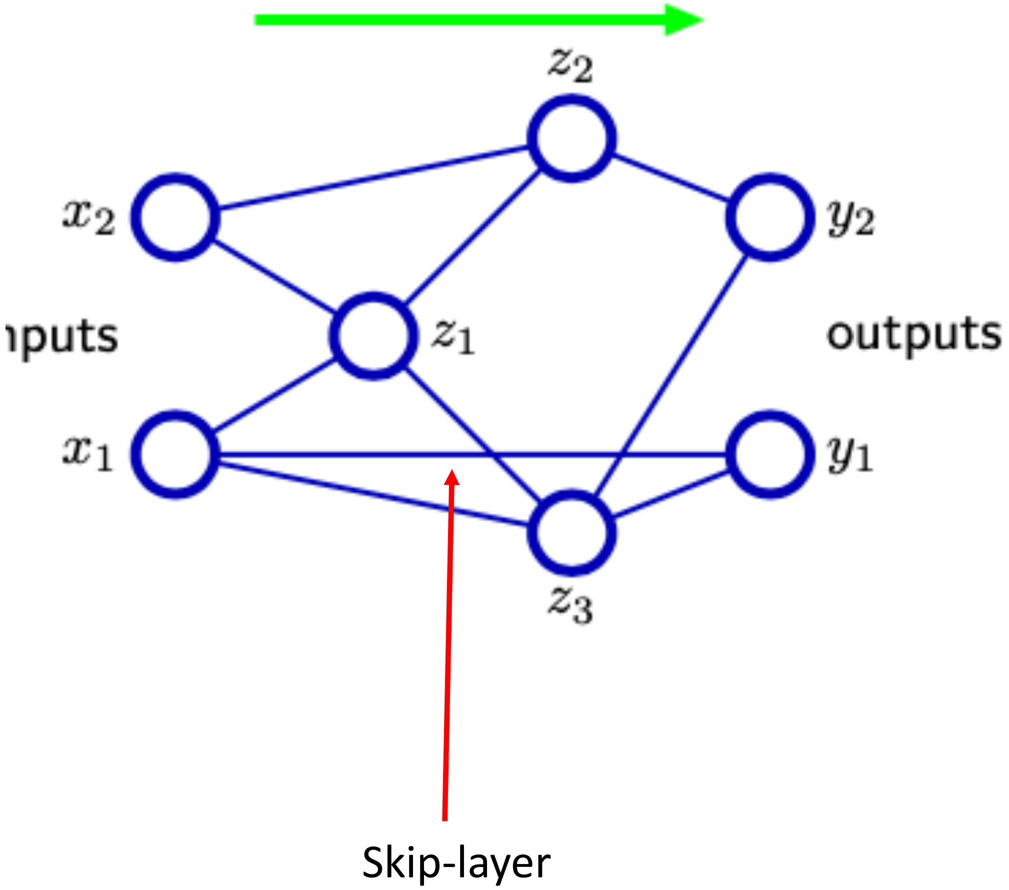
$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right) \tag{6-1}$$

where $f(\cdot)$ is a nonlinear activation function for classification and identity for regression.

- Recall that perceptron uses the activation function.

6.3. Feedforward Network Functions





6.3. Feedforward Network Functions

- Neural networks (NN) use basis functions that follow the same form as Eq (1), so that each basis function is itself a nonlinear function of a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

- The basic NN model can be described as a series of functional transformation:
  1. Construct M linear combinations of input variables $x_1, \ldots, x_D$ in the form of

$w_{ji}^{(1)}$: weights
$w_{j0}^{(1)}$: baises

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{6-2}$$

where $j = 1, \ldots, M$, and the superscript (1) indicates that the corresponding parameters are in the first '*layer*' of the network.

6.3. Feedforward Network Functions

2. Transform each a_j using a differentiable, nonlinear activation function $h(\cdot)$,

$$z_j = h(a_j): \tag{6-3}$$

   - These are called hidden units in NN that corresponds to the outs put $\phi_j(\boldsymbol{x})$.
   - The nonlinear $h(\cdot)$ are either the logistic sigmoid or $tanh$ function.

3. Linearly recombine to give output unit activations:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{6-4}$$

   where $k = 1, \ldots, K$ and K is the total number of outputs.

4. Transform the output unit activations (6-4) to predict y_k:
   - For standard linear regression: $y_k = a_k$
   - For binary classification: $y_k = \sigma(a_k)$
   - For multiclassification: softmax

6.3. Feedforward Network Functions

- The NN ***forward propagation*** can be summarized as

$$y_k(x,w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{6-5}$$

- The NN model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector $\boldsymbol{w}$ of adjustable parameters.
- We can absorb the bias into the set of weights as $a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$ then Eq (6-5) becomes

$$y_k(x,w) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{6-6}$$

- NN is also known as the *multilayer perceptron* using continuous sigmoidal nonlinearities.
- If the activation function of the hidden units is linear, we always find an equivalent network.

6. Neural Networks

6.4 NN – Regression

- Consider a set of input vectors $\{\boldsymbol{x}_n\}$ and target vectors $\{\boldsymbol{t_n}\}$ where $n = 1, \dots, N$.
- For the simplicity, assume that t has a Gaussian distribution with an $\boldsymbol{x}$-dependent mean

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$$

  where β is an inverse variance of the Gaussian noise.
  - To take the output unit activation function be the identity.
- The corresponding likelihood function is

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} p(t_n|\boldsymbol{x_n}, \boldsymbol{w}, \beta) \tag{6-7}$$

- We can learn the parameters $\boldsymbol{w}$ and β by taking the negative log:

$$-\log p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \frac{\beta}{2} \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi) \,. \tag{6-8}$$

6.4. NN – Regression

- In NN, it is usual to consider the minimization of an error function

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\boldsymbol{x_n}, \boldsymbol{w}) - t_n\}^2 . \tag{6-8}$$

- The maximum likelihood (ML) solution $\boldsymbol{w_{ML}}$ is found by minimizing $E(\boldsymbol{w})$.

6.4. NN – Regression

- Using $\boldsymbol{w_{ML}}$, the value of $\boldsymbol{\beta}_{ML}$ can be found by minimizing Equation (6-8):

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\boldsymbol{x_n}, \boldsymbol{w_{ML}}) - t_n\}^2 \tag{6-9}$$

- If we have multiple target variables,

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}|\boldsymbol{y}(\boldsymbol{x_n}, \boldsymbol{w_{ML}}) - \boldsymbol{t_n}|^2 \tag{6-10}$$

  Where K is the number of target variables.

- An output activation is the identity, so $y_k = a_k$ and the corresponding error function has the property

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{6-11}$$

6. Neural Networks

6.5. NN – Classification

- Binary classification having a single target variable $t, t \in \{0,1\}$

- An activation function is a logistic sigmoid:

$$y = \sigma(a) = \frac{1}{1 + \exp(-a)}$$

so $0 \leq y(\boldsymbol{x}, \boldsymbol{w}) \leq 1$.

- We can interpret the conditional probability for each class

$$C_1: p(C_1|\boldsymbol{x}) = y(\boldsymbol{x}, \boldsymbol{w})$$

$$C_2: p(C_2|\boldsymbol{x}) = 1 - p(C_1|\boldsymbol{x})$$

And the conditional distribution $p(t|\boldsymbol{x}, \boldsymbol{w})$ is then a Bernoulli distribution:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = y(\boldsymbol{x}, \boldsymbol{w})^t \{1 - y(\boldsymbol{x}, \boldsymbol{w})\}^{1-t}$$

6.5. NN – Classification

- The cross-entropy error function after taking the negative log likelihood is then

$$E(w) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t)\ln(1-y_n)\}. \quad (6\text{-}12)$$

  Note $\beta = 0$ if the target values are assumed to be correctly labelled.

- In NN, training is faster and easier to generalize using the cross-entropy function.

6.5. NN – Classification

- K classes and $t_k \in \{0,1\}$ have a 1-of-K.
- The activation function is the *softmax* function:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \frac{\exp\left(a_k(\boldsymbol{x}, \boldsymbol{w})\right)}{\sum_j \exp\left(a_j(\boldsymbol{x}, \boldsymbol{w})\right)} \quad (6\text{-}13)$$

where $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.

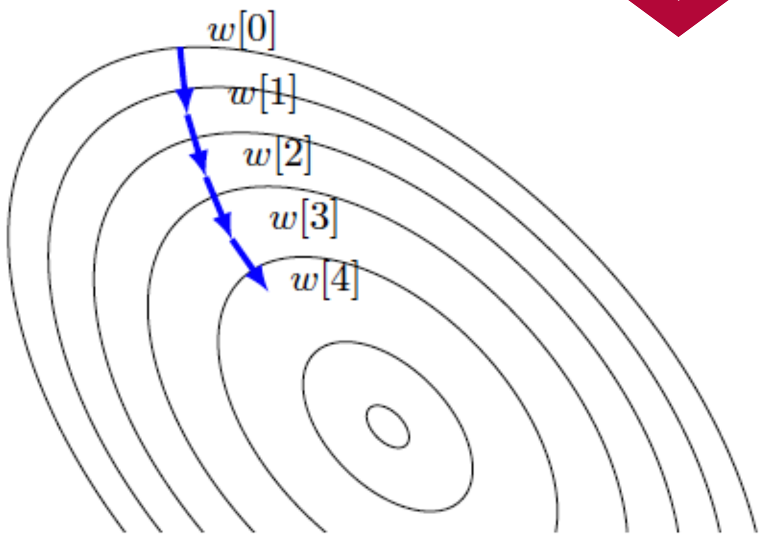
- NN output $y_k(\boldsymbol{x}, \boldsymbol{w}) = p(t_k = 1|\boldsymbol{x})$ leads to the error function is the multiple cross-entropy function

$$E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\boldsymbol{x_n}, \boldsymbol{w}) \quad (6\text{-}14)$$

6. Neural Networks

6.6. Optimization

- Use Gradient descent optimization by updating weight
$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E\left(\boldsymbol{w}^{(\tau)}\right)$$
where η is the learning rate being > 0 and τ is the labelled iteration number.



- To evaluate the gradient of error efficiently, we use **error backpropagation** through the network.

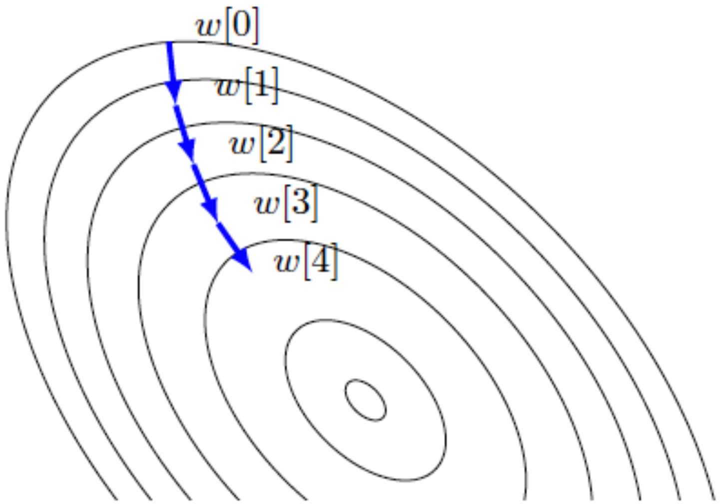
- In iteration $[t+1]$, choose a weight update $\Delta w[t]$ and set
$$w[t+1] = w[t] + \Delta \mathrm{w}[\mathrm{t}].$$

6.6. Optimization

- Example: GD minimizes the error/loss $L_n(w)$ by taking steps in the direction of the negative gradient:

$$\Delta w[t] = -\eta \frac{\partial L_n}{\partial w[t]} \tag{6-15}$$

- Problem: How to evaluate $\frac{\partial L_n}{\partial w[t]}$ in iteration $[t+1]$?
  - "Error Backpropagation" allows to evaluate $\frac{\partial L_n}{\partial w[t]}$ in $\mathcal{O}(W)$!

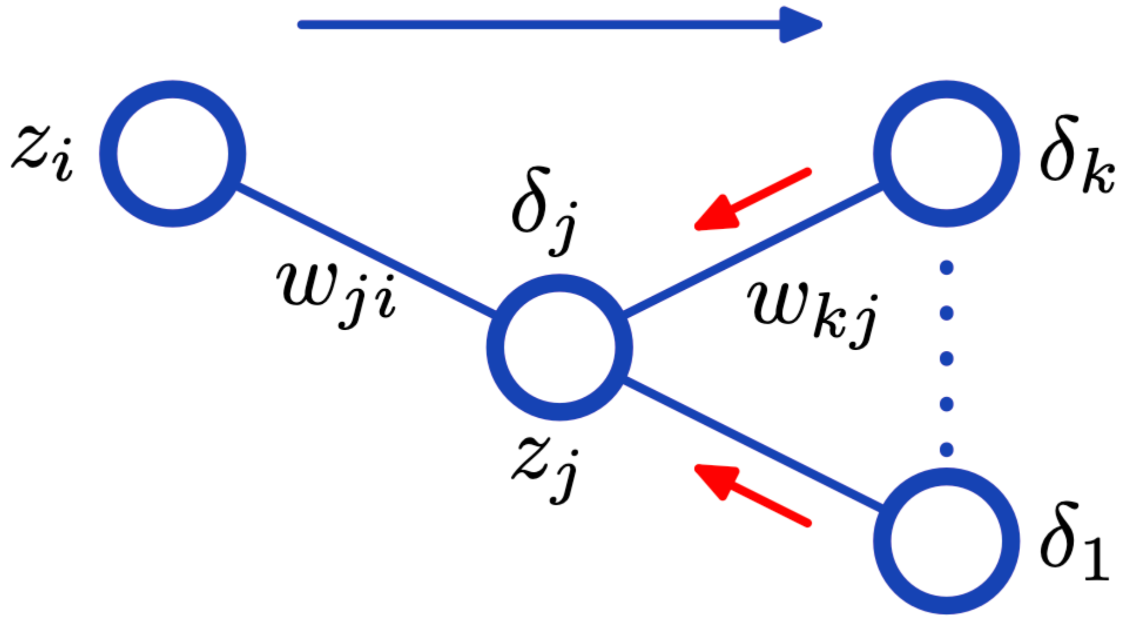


6. Neural Networks

6.7. Error Backpropagation

- Backpropagation is used to describe the training using gradient descent applied to an SSE function.

- Two steps:
  - Evaluation of derivatives.
  - Adjustment of weights.



)

6.7. Error Backpropagation

- Consider a *forward propagation:*
  - Let the error function be $E(\boldsymbol{w})$ and y_k be a linear combination of x_i:

$$y_k = \sum_i w_{ki} x_i$$

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

  - The gradient of error r.t. w_{ji} - the "local" computation with product of an error associated with the output end of the link w_{ji} and x_{ni}

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} = \frac{\partial E_n}{\partial a_j} \left(\frac{\partial a_j}{\partial w_{ji}} \right) = \delta_j z_i \tag{6-16}$$

$$\boxed{\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \left(\frac{\partial a_k}{\partial a_j} \right)}$$

$\frac{\partial a_j}{\partial w_{ji}} = z_i$ and $z_j = h(a_j)$

$$a_j = \sum_i w_{ji} z_i$$

6.7. Error Backpropagation

❑ We can obtain the backpropagation formula by

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{6-17}$$

Derivation:

$$\delta_j = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j} \left(\sum_i w_{ki} z_i \right) = \sum_k \delta_k \frac{\partial}{\partial a_j} \left(\sum_i w_{ki} h(a_i) \right)$$

$$\delta_k = \frac{\partial E_n}{\partial a_k}$$

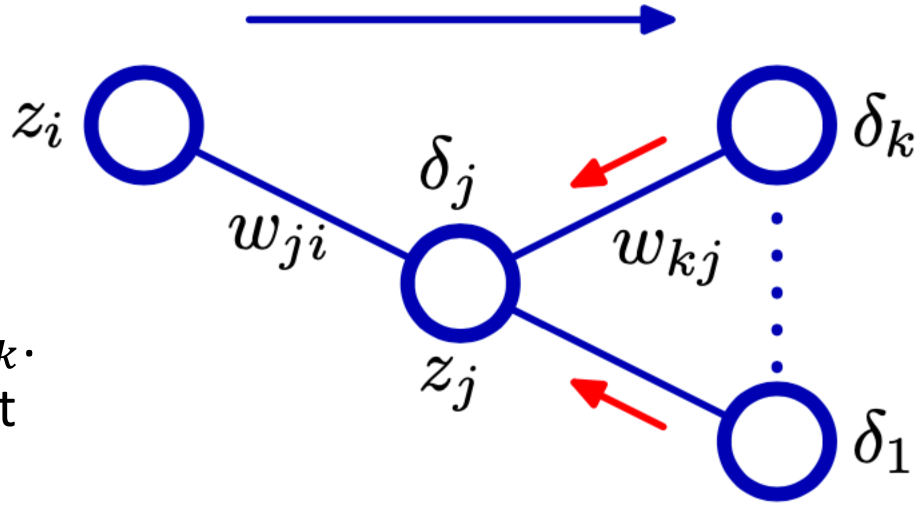
$$a_k = \sum_i w_{ki} z_i$$

$$z_i = h(a_i)$$

All partial derivatives are 0 except when $j = i$

6.7. Error Backpropagation

- Procedure Summary:
  1. Apply an input $\boldsymbol{x}_n$ to the network and forward propagate through network using definitions in Eq. (6-16) to find the activations of all the hidden and output units.
  2. Evaluate δ_k for all the output units : $\delta_k = y_k - t_k$.
  3. Backpropagate and obtain δ_j for each hidden unit in the network using Equation (6-17).
  4. Use Equation (6-16) to evaluate the required derivatives.



6.7. Error Backpropagation

- Example: two-layer network regression ($y_k = a_k$) with hidden units $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$.
  - $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$
  - $h'(a) = \frac{dh}{da} = 1 - h(a)^2$
  - A forward propagation using $a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$, $z_j = \tanh(a_j)$, $y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$
  - Compute δ's for each output unit using $\delta_k = y_k - t_k$
  - Then backpropagate to obtain δs for the hidden units using

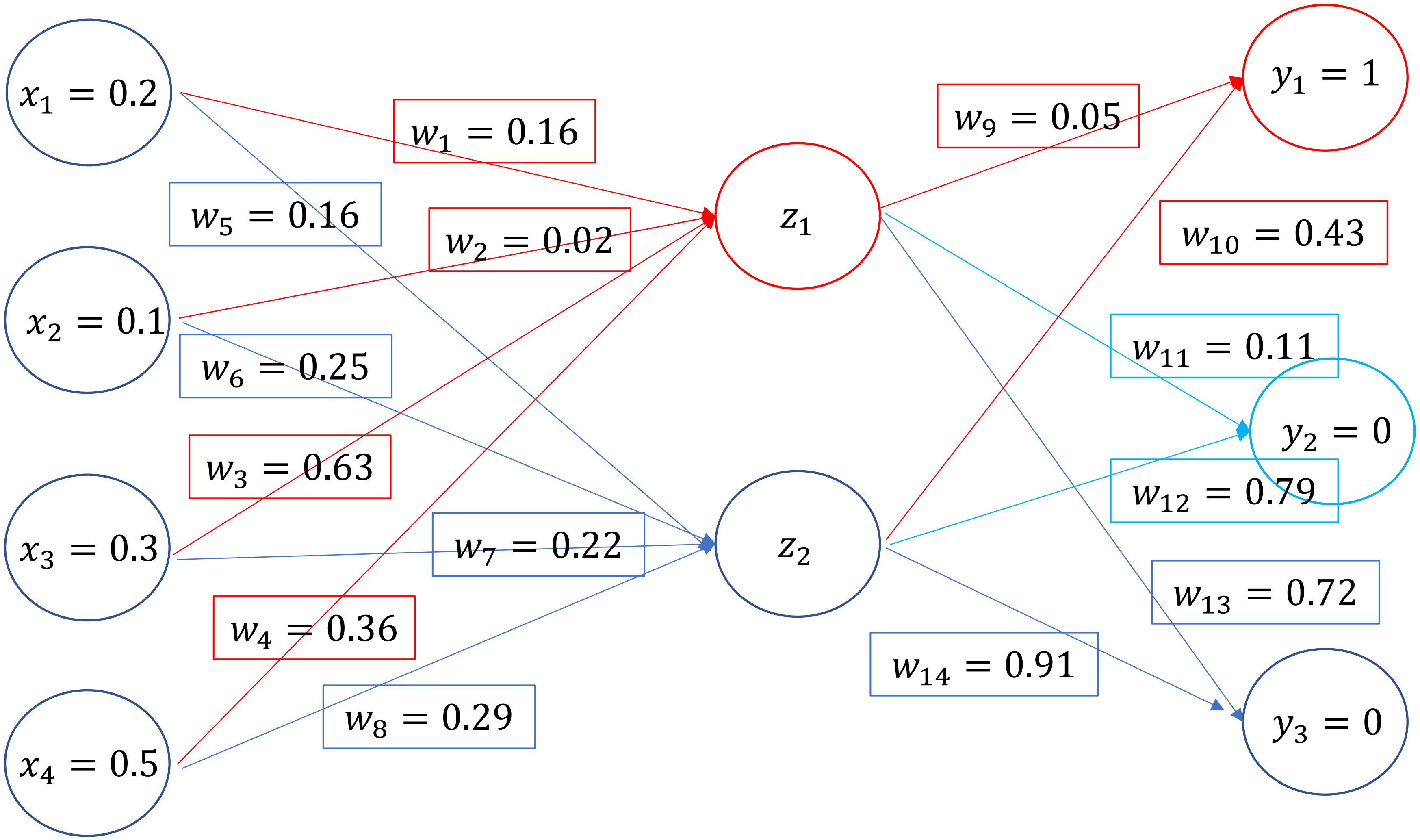
$$\delta_j = \left(1 - z_j^2\right) \sum_{k=1}^{K} w_{kj}\delta_k$$

  - Derivatives w.r.t. the first-layer and second-layer weights:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

6. Neural Networks

6.8. Numerical Example



6.8. Numerical Example

Forward Propagation using Eq. 6-3:

$$\boldsymbol{z} = \sigma(\boldsymbol{w}_1^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}_1^T \boldsymbol{x})} = \frac{1}{1 + \exp\left(-\left([0.2\ 0.1\ 0.5]\begin{bmatrix} 0.16 & 0.16 \\ 0.02 & 0.25 \\ 0.63 & 0.22 \\ 0.36 & 0.29 \end{bmatrix}\right)\right)} = [0.5994\ 0.5666]$$

Prediction using Eq. 6-5: $\hat{\boldsymbol{y}} = \sigma\left(\boldsymbol{w}_2^T \boldsymbol{z}\right)$

$$\hat{\boldsymbol{y}} = \frac{1}{1 + \exp(-\boldsymbol{w}_2^T \boldsymbol{z})} = \frac{1}{1 + \exp\left(-\left([0.5994\ 0.5666]\begin{bmatrix} 0.05 & 0.33 & 0.72 \\ 0.43 & 0.79 & 0.91 \end{bmatrix}\right)\right)} = [0.5680\ 0.6560\ 0.7205]$$

Error Function using Eq. 6-6: $L = \sum_{i=1} \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^2 = 0.5681$

6.8. Numerical Example

Back Propagation from output to last hidden layer: $\boldsymbol{G}_p = \boldsymbol{E}'\widehat{\boldsymbol{y}}'$

Derivative of error function (Eq. 6-11):

$$\boldsymbol{E}' = \widehat{\boldsymbol{y}} - \boldsymbol{y} = [0.5680\ 0.6560\ 0.7205] - [1\ 0\ 0] = [-0.4320\ 0.6560\ 0.7205\ ]$$

Derivative of $\widehat{\boldsymbol{y}}$: $\widehat{\boldsymbol{y}}' = \widehat{\boldsymbol{y}}(1 - \widehat{\boldsymbol{y}}) = [0.2454\ 0.2257\ 0.2014]$

The gradient of layer: $\boldsymbol{G}_p = \boldsymbol{E}'\widehat{\boldsymbol{y}}' = [-0.1060\ 0.1480\ 0.1451]$

$$\delta \boldsymbol{W}_2 = \left(\boldsymbol{G}_p^T \boldsymbol{z}\right)^T = \left(\begin{bmatrix} -0.1060 \\ 0.1480 \\ 0.1451 \end{bmatrix} [0.5994\ 0.5666]\right)^T = \begin{bmatrix} -0.0635\ 0.0887\ 0.0870 \\ -0.0601\ 0.0839\ 0.0822 \end{bmatrix}$$

$$\widehat{\boldsymbol{W}}_2 = \boldsymbol{w}_2 - \delta \boldsymbol{W}_2 = \begin{bmatrix} 0.05\ 0.33\ 0.72 \\ 0.43\ 0.79\ 0.91 \end{bmatrix} - \begin{bmatrix} -0.0635\ 0.0887\ 0.0870 \\ -0.0601\ 0.0839\ 0.0822 \end{bmatrix} = \begin{bmatrix} 0.1135\ 0.2413\ 0.6330 \\ 0.4901\ 0.7061\ 0.8278 \end{bmatrix}$$

6.8. Numerical Example

Back Propagation from last hidden layer

Derivative of z: $\boldsymbol{z}' = \boldsymbol{z}(1-\boldsymbol{z}) = [0.5994(1-0.5994)\ 0.5666(1-0.5666)] = [0.2401\ 0.2456]$

The gradient of 1st layer: $\boldsymbol{G} = \boldsymbol{G}_p \boldsymbol{w}_2^T \boldsymbol{z}'$

$$\boldsymbol{G} = [-0.1060\ 0.1480\ 0.1451] \begin{bmatrix} 0.05 & 0.43 \\ 0.33 & 0.79 \\ 0.72 & 0.91 \end{bmatrix} [0.2401\ 0.2456] = [0.0355\ 0.0499]$$

$$\delta \boldsymbol{W}_1 = \boldsymbol{x}^T \boldsymbol{G} = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.3 \\ 0.5 \end{bmatrix} [0.0355\ 0.0499] = \begin{bmatrix} 0.0071 & 0.0100 \\ 0.0036 & 0.0050 \\ 0.0107 & 0.0150 \\ 0.0178 & 0.0250 \end{bmatrix}$$

$$\widehat{\boldsymbol{W}}_1 = \boldsymbol{w}_1 - \delta \boldsymbol{W}_1 = \begin{bmatrix} 0.16 & 0.16 \\ 0.02 & 0.25 \\ 0.63 & 0.22 \\ 0.36 & 0.29 \end{bmatrix} - \begin{bmatrix} 0.0071 & 0.0100 \\ 0.0036 & 0.0050 \\ 0.0107 & 0.0150 \\ 0.0178 & 0.0250 \end{bmatrix} = \begin{bmatrix} 0.1529 & 0.1500 \\ 0.0164 & 0.2450 \\ 0.6193 & 0.2050 \\ 0.3422 & 0.2650 \end{bmatrix}$$

6.8. Numerical Example

Calculate new z:

$$z' = \frac{1}{1 + \exp\left(-\widehat{W}_1^T x\right)} = [0.5961\ 0.5618]$$

Predict y:

$$\hat{y} = \frac{1}{1 + \exp\left(-\widehat{W}_2^T z\right)} = [0.5849\ 0.6319\ 0.6990]$$

Estimate SSE:

$$E = \frac{1}{2}\sum(\hat{y} - y)^2 = 0.5301$$

6. Neural Networks

6.9. Conclusion

Pros:

- Successfully used on a variety of domains: computer vision, speech recognition, gaming, etc.
- Can provide solutions to very complex and nonlinear problems.
- If provided with sufficient amount of data, can solve classification and forecasting problems accurately and easily.
- Once trained, prediction is fast.

Cons:

- The outcome contains some uncertainty that is not always desirable.
- It has to undergo a “learning” phase.
- The quality of the outcome depends on the quality of the data used in the learning phase.