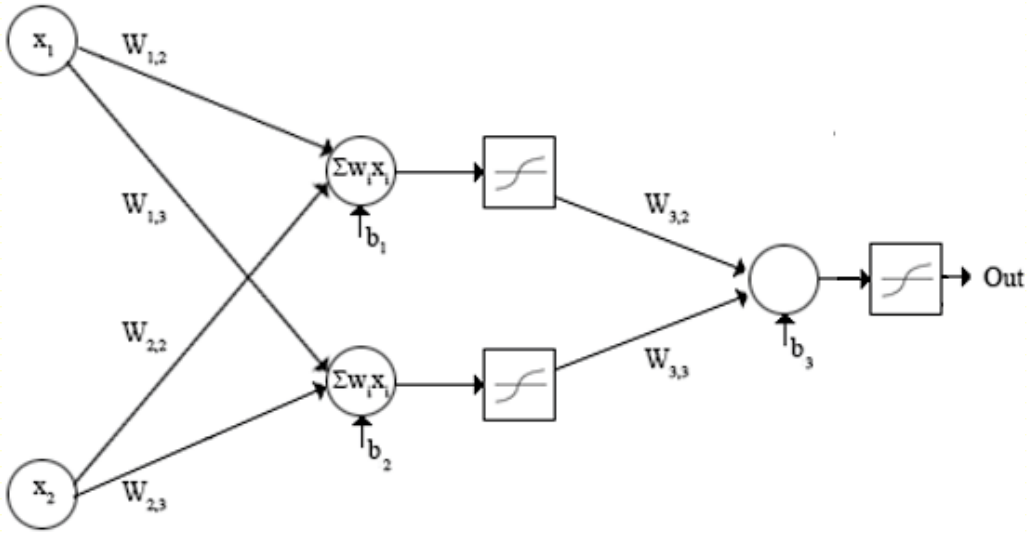
Course Outline

- TOPICS
  1. What is Machine Learning and Image Processing
  2. Traditional Features, K-NN classifier
  3. Linear Classification
  4. Perceptron Algorithm, Sigmoid Activation Function, Gradient Descent
  5. Stochastic Gradient Descent, Back-Propagation
  6. Multi-Layer Neural Network
  7. Convolution and Pooling
  8. Mid-Term Examination
  9. Mid-Term Examination
  10. Convolutional Neural Networks.
  11. Training Convolutional Neural Networks: Hyper-Parameters, Activation functions, initialization, dropout, batch normalization
  12. Recurrent Neural Networks
  13. Applications of Convolutional Neural Networks for Image Segmentation and Object Classification
  14. Project Presentations

Examples of Some MLNN

- Today, we will see some examples of MLNN by analyzing, designing structures and dealing with implementation.
- At the end of this course, you would be able to cover the general procedures of a NN structure.
  - What is layers
  - What is nodes and their size
  - What is output node
  - Role of weights and bias terms
  - Backpropagation to update weights



- You can see that a node is represented with circle.
- Each node has 2 weights and 1 bias as inputs and 1 outputs from activation function.
- Sum is calculated inside of the node.
- Note that 2 weights ($w_{3,2}$,$w_{3,3}$) are multiplied with 2 outputs of first hidden layer.

BackPropagation

For each input vector:

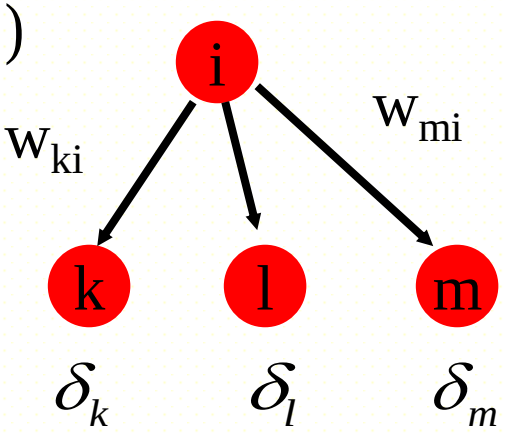
1. Propagate the inputs forward through the network.
2. Propagate the errors backward through the network.

Error term for output units:

$$\delta_i = -\frac{\partial E_d}{\partial sum_{id}} = -\frac{\partial E_{id}}{\partial sum_{id}} = (t_{id} - o_{id}) o_{id} (1 - o_{id})$$

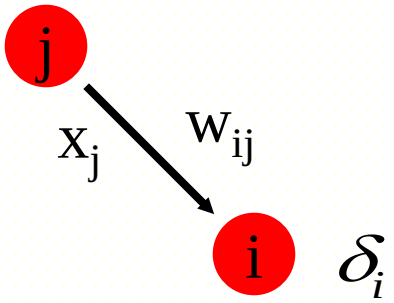
Error term for hidden units:

$$\delta_i = -\frac{\partial E_d}{\partial sum_{id}} = o_{id}(1 - o_{id}) \sum_{k \in Outputs} w_{ki} \delta_k$$



3. Compute all weights changes

$$\Delta w_{ij} = \eta \delta_i x_j$$



Backpropagation Algorithm

- Learning rule

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m),$$

- Hidden-to-output

$$\Delta w_{kj} = \eta\delta_k y_j = \eta(t_k - z_k) f'(net_k) y_j.$$

- Input-to-hidden

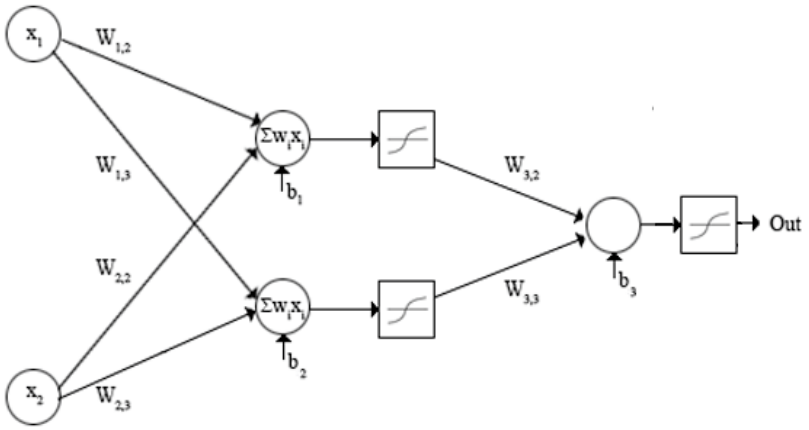
$$\Delta w_{ji} = \eta x_i \delta_j = \eta \underbrace{\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(net_j)}_{\delta_j} x_i.$$

- Note, that w_{ij} are initialized with random values

Backpropagation Algorithm

Error term for output units:

$$\delta_i = -\frac{\partial E_d}{\partial sum_{id}} = -\frac{\partial E_{id}}{\partial sum_{id}} = (t_{id} - o_{id}) o_{id} (1 - o_{id})$$



Error term for hidden units:

$$\delta_i = -\frac{\partial E_d}{\partial sum_{id}} = o_{id}(1 - o_{id}) \sum_{k \in Outputs} w_{ki}\delta_k$$

3. Compute all weights changes

$$\Delta w_{ij} = \eta \delta_i x_j$$

Learning Rule

$$w_{m+1} = w_m + \Delta w_m$$

Hidden to output

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

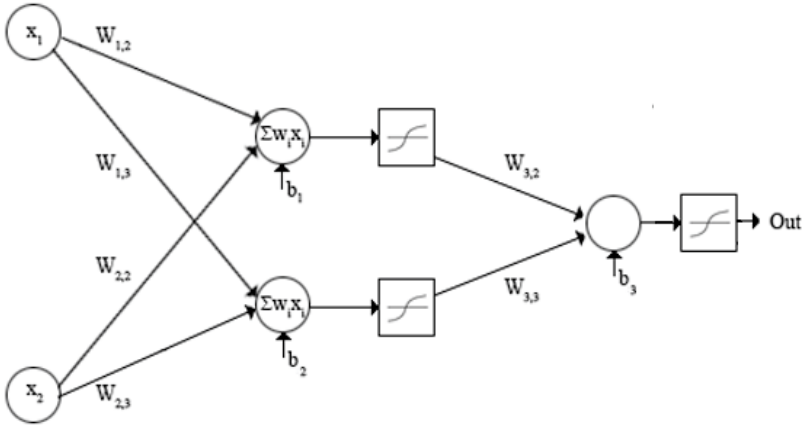
Input to Hidden

$$\Delta w_{ji} = \eta \delta_j x_i = \eta \underbrace{\left[\sum_{k=1}^{c} w_{kj} \delta_k \right] f'(net_j)}_{\delta_j} x_i$$

$c =$ *number of connected weights*

$c = 1$ *for this case*

Gradient Descent vs Stochastic Gradient

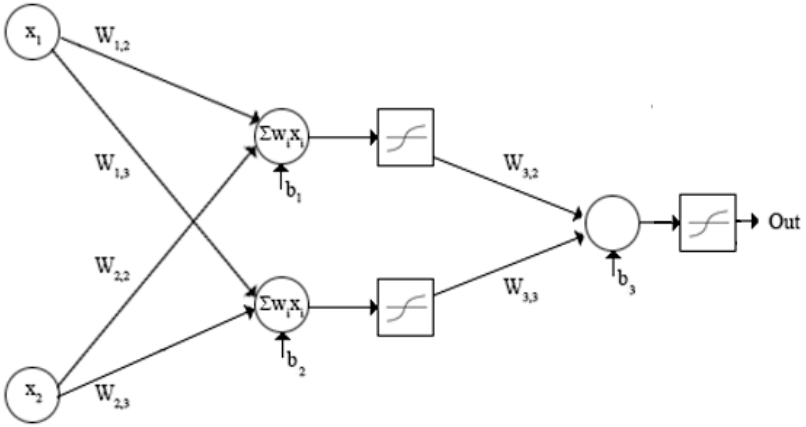


- **Gradient Descent** and **Stochastic Gradient Descent** are similar in terms how updating the weights.
- The **only difference** is relied on when updating the weights.
- **Gradient Descent:**
  - Updating weights after all inputs passed-forward through network
- **Stochastic Gradient Descent:**
  - Updating weights once any input passed-forward through network
- We will use the Stochastic Gradient Descent since it is robust.

Coding Example1: Single Hidden Layer

inputs		*outputs*
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Lets help NN to learn the XOR problem



$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$weights = \begin{bmatrix} b_{11} & w_{11} & w_{12} \\ b_{21} & w_{22} & w_{23} \\ b_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 \end{bmatrix}$$

We have

1 *input layer*

1 *hidden layer*

1 *output layer*

Our activation function is sigmoid

$$sigm = 1./\left(1+e^{-x}\right) = 1./\left(1+\exp(-x)\right)$$

sigmoid Derivative

$$output = 1./\left(1+e^{-x}\right)$$

$$sigmDerivative = output.*(1-output)$$

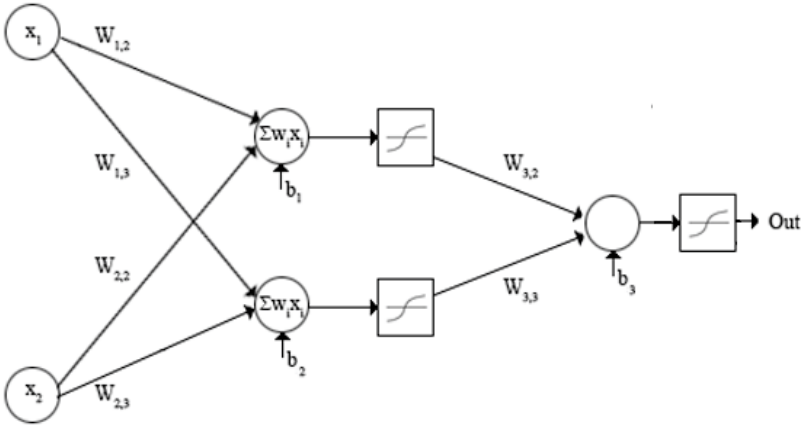
see

week5_ANN_ex1_v1*m*
week5_ANN_ex1_v2*.m*
week5_ANN_ex1_v3*.m*
week5_ANN_ex1_v4*.m*
week5_ANN_ex1_v5*.m*
week5_ANN_ex1_v6*.m*
week5_ANN_ex1_v7*.m*
week5_ANN_ex1_v8*.m*

Coding Example2: Multi-Hidden Layers

inputs		outputs
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Lets help NN to learn the XOR problem



$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$weights = \begin{bmatrix} b_{11} & w_{11} & w_{12} \\ b_{21} & w_{22} & w_{23} \\ b_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 \end{bmatrix}$$

We have
1 input layer
Multi hidden layer
1 output layer

Our activation function is sigmoid

$$sigm = 1./\left(1+e^{-x}\right) = 1./\left(1+\exp(-x)\right)$$

sigmoid Derivative

$$output = 1./\left(1+e^{-x}\right)$$

$$sigmDerivative = output.*(1-output)$$

see examples
week5_ANN_ex2_multi_v1.m
week5_ANN_ex2_multi_v2.m
week5_ANN_ex2_multi_v3.m

Role of Loss Function in NN

- Selecting a proper loss function is important.
- Loss function must be not sensitive to outliers in data.

$predicted = [1 \quad 2 \quad 4]$

$calculated = [4 \quad 3 \quad 50]$

we can mark the 50 *as outlier*

$$L2 = \sum_{i=1}^{3} (4-1)^2 + (3-2)^2 + (50-4)^2$$

$$L2 = 9 + 1 + 2116 = 2126$$

$$L1 = \sum_{i=1}^{3} |4-1| + |3-2| + |50-4|$$

$$L1 = 3 + 1 + 46 = 56$$

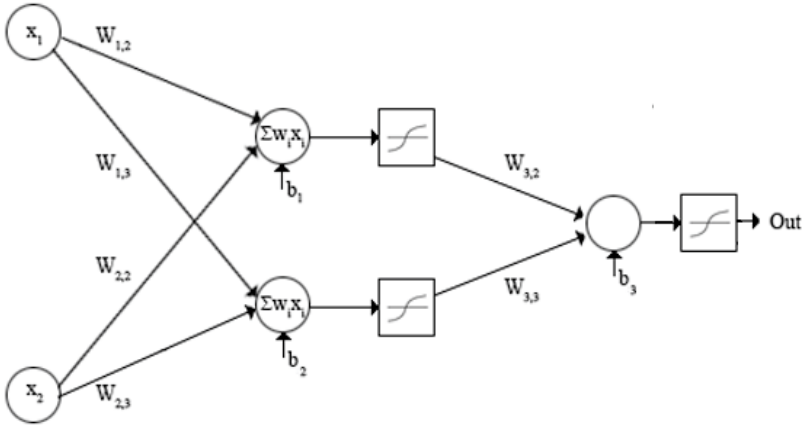
$L2 = 2116$

$L1 = 56$

We can say that the L1 *is more robust to outliers.*
Meanwhile,the L2 *is sensitive to noise*

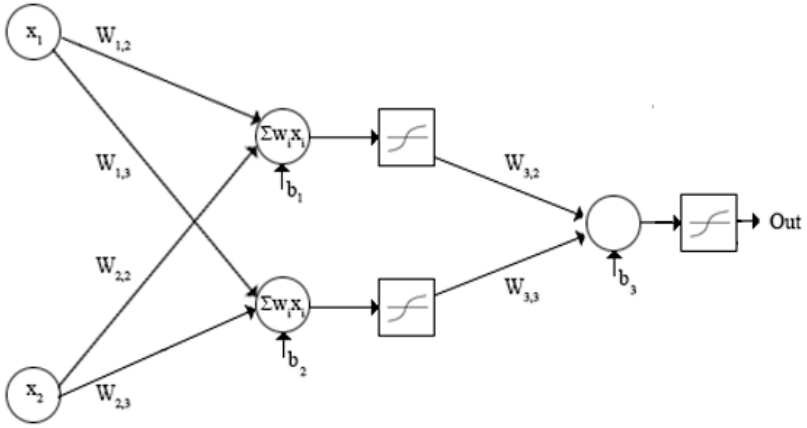
Loss Estimator	Cost ($J(r)$)	Residual ($\psi(r)$)
L-1	$\lvert r \rvert$	$sign(r)$
L-2	$r^2/2$	r
Cauchy	$\frac{c^2}{2}\log\left(1+(r/c)^2\right)$	$\frac{r}{1+(r/c)^2}$
Charbonnier	$\sqrt{r^2+\varepsilon^2}$	$\frac{r}{\sqrt{r^2+\varepsilon^2}}$
Huber	$\begin{cases} r^2/2 & if \lvert r \rvert \le c \\ c(\lvert r \rvert - c/2) & if \lvert r \rvert \ge c \end{cases}$	$\begin{cases} r & if \lvert r \rvert \le c \\ csign(r) & if \lvert r \rvert \ge c \end{cases}$
Pseudo-Huber	$c^2\left(\sqrt{1+r^2/c^2}-1\right)$	$\frac{r}{\sqrt{1+r^2/c^2}}$
Tukey	$\begin{cases} \frac{c^2}{6}\left(1-\left(1-(r/c)^2\right)^3\right) & if \lvert r \rvert \le c \\ 0 & if \lvert r \rvert \ge c \end{cases}$	$\begin{cases} r\left(1-(r/c)^2\right)^2 & if \lvert r \rvert \le c \\ 0 & if \lvert r \rvert \ge c \end{cases}$

Batch Size in SGD



- Batch size 1: 1 input passed through the network.
- Batch size 4: 4 inputs passed through the network.
- Until now, we determined batch size as 1. We have updated the weights after 1 input passed-forward through network.
- What if we consider the batch size is greater than 1. How we update the weights if batchSize>1?
- In this case, average Δw_m is calculated, then the weights are updated.

Batch Size in SGD



Using the larger mini-batch:

- Takes smaller iterations, hence, reducing the computation time.
- Good generalization for classification.
- Requires smaller learning rate.
- Significant degradation in overall accuracy.
- Requires an expensive GPU.

Using the smaller mini-batch:

- Takes larger iterations, hence, increasing the computation time.
- Not good generalization for classification.
- Improves the overall accuracy.
- Requires only CPU and RAM.

see example

week5_ANN_ex3_real_v1_nobatch.m

week5_ANN_ex3_real_v2_batch.m

Stochastic Gradient With Momentum

- Stochastic Gradient With Momentum (SGDM) is more robust and consider the previous velocity when updating the weights.

SGD Learning Rule

$$w_{m+1} = w_m + \Delta w_m$$

Hidden to output

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

Input to Hidden

$$\Delta w_{ji} = \eta \delta_j x_i = \eta \underbrace{\left[\sum_{k=1}^{c} w_{kj} \delta_k \right] f'(net_k)}_{\delta_j} x_i$$

Velocity : v

SGDM Learning Rule

$$v = zeros$$

$$v = \beta v + (1 - \beta) \Delta w_m$$

$$\beta = 0.9$$

$$w_{m+1} = w_m + v$$

Hidden to output

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

Input to Hidden

$$\Delta w_{ji} = \eta \delta_j x_i = \eta \underbrace{\left[\sum_{k=1}^{c} w_{kj} \delta_k \right] f'(net_k)}_{\delta_j} x_i$$

see example

week5_ANN_ex3_real_v1_nobatch_sgdm.m