# Lec9 - Neural Networks

## CS 783 - Visual Recognition

Vinay P. Namboodiri

IIT Kanpur

$5^{th}$ February 2019

# Contents

# Outline

# How to Improve Representation further

While spatial pyramid representation was performing well, however, the results for object recognition was around 65% accuracies on one of the easy datasets i.e. Caltech 101.

# How to Improve Representation further

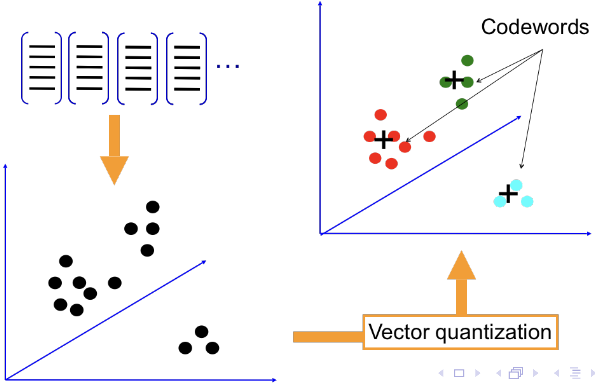
While spatial pyramid representation was performing well, however, the results for object recognition was around 65% accuracies on one of the easy datasets i.e. Caltech 101.

One of the ways to improve the accuracy was to consider the problem of vector quantization and the loss due to that.

# How to Improve Representation further

While spatial pyramid representation was performing well, however, the results for object recognition was around 65% accuracies on one of the easy datasets i.e. Caltech 101.

One of the ways to improve the accuracy was to consider the problem of vector quantization and the loss due to that.

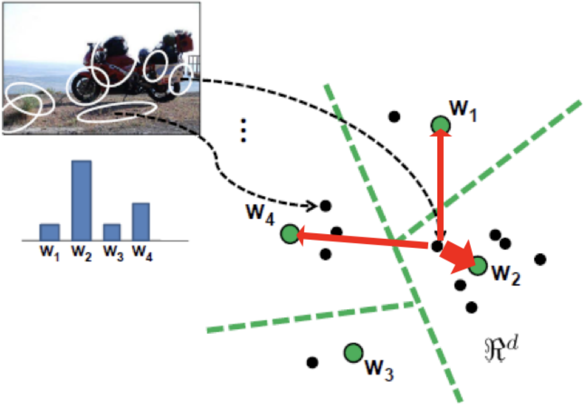


# Kernel Codebook Encoding

An approach proposed to improve this was to consider weighted matching of a word to the nearest related words that allowed for soft assignment of sift features to the centroids

# Kernel Codebook Encoding

An approach proposed to improve this was to consider weighted matching of a word to the nearest related words that allowed for soft assignment of sift features to the centroids



# VLAD

A better approach was to consider the variance of the sift features after considering the mean. This allowed to aggregate the quantization error in each dimension and explicitly represent it

# VLAD

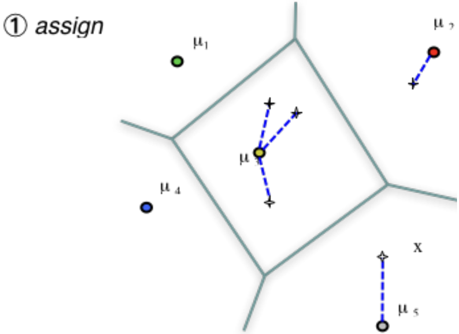
A better approach was to consider the variance of the sift features after considering the mean. This allowed to aggregate the quantization error in each dimension and explicitly represent it

It consisted of the following steps. Initially the centroids $\{\mu_i, i = 1, ..., N\}$ is estimated from the set of sift feature vectors $X = \{x_j, j = 1, \ldots T\}$.

# VLAD

A better approach was to consider the variance of the sift features after considering the mean. This allowed to aggregate the quantization error in each dimension and explicitly represent it

It consisted of the following steps. Initially the centroids $\{\mu_i, i = 1, ..., N\}$ is estimated from the set of sift feature vectors $X = \{x_j, j = 1, \ldots T\}$.



# VLAD

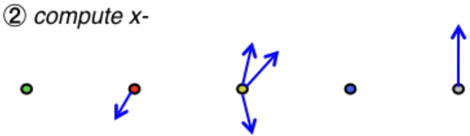
Next for each feature vector we associate the nearest neighbor centroid obtained by $NN(x_t) = argmin_{\mu_i} ||x_t - \mu_i||$

# VLAD

Next for each feature vector we associate the nearest neighbor centroid obtained by $NN(x_t) = argmin_{\mu_i} ||x_t - \mu_i||$
After associating it with the centroid the difference $x_t - \mu_i$ is calculated

# VLAD

Next for each feature vector we associate the nearest neighbor centroid obtained by $NN(x_t) = argmin_{\mu_i} ||x_t - \mu_i||$
After associating it with the centroid the difference $x_t - \mu_i$ is calculated

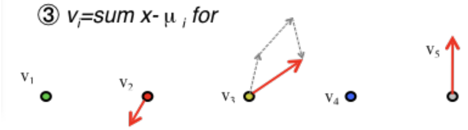


# VLAD

Next for centroid the difference is aggregated around each dimension of the SIFT vector from the set of centroids mapped to the centroid. each feature vector we associate the nearest neighbor centroid obtained by $v_i = \sum_{x_t : NN(x_t) = \mu_i} x_t - \mu_i$

# VLAD

Next for centroid the difference is aggregated around each dimension of the SIFT vector from the set of centroids mapped to the centroid. each feature vector we associate the nearest neighbor centroid obtained by $v_i = \sum_{x_t:NN(x_t)=\mu_i} x_t - \mu_i$



# VLAD

These are the steps for VLAD summarized as follows

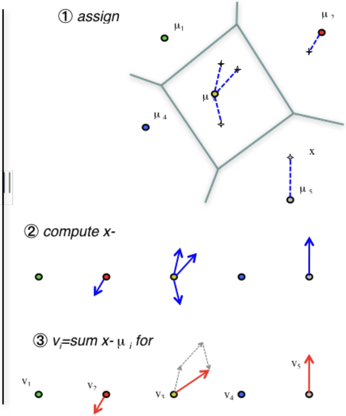
$$\{\mu_i, i = 1 \dots N\}$$

$$X = \{x_t, t = 1 \dots T\}$$

$$\mathrm{NN}(x_t) = \arg\min_{\mu_i} ||x_t - \mu_i||$$

$$v_i = \sum_{x_t : \mathrm{NN}(x_t) = \mu_i} x_t - \mu_i$$

$$\ell_2$$



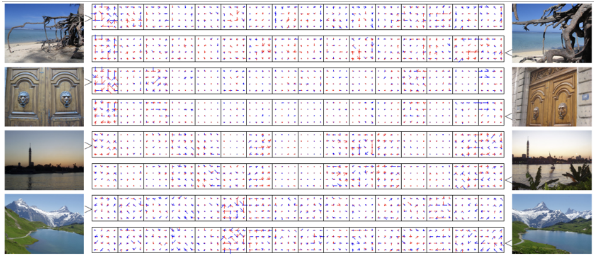
# VLAD

This can be understood by visualizing the variance for the case where there are 16 centroids and we consider the variance for these 16 centroids. This bag of words representation will have a length of $16 + 16 \times 128$ as its length for the case of 16 centroids

# VLAD

This can be understood by visualizing the variance for the case where there are 16 centroids and we consider the variance for these 16 centroids. This bag of words representation will have a length of 16+ 16 × 128 as its length for the case of 16 centroids

$$v_i = \sum_{x_t : \mathrm{NN}(x_t) = \mu_i} x_t - \mu_i$$

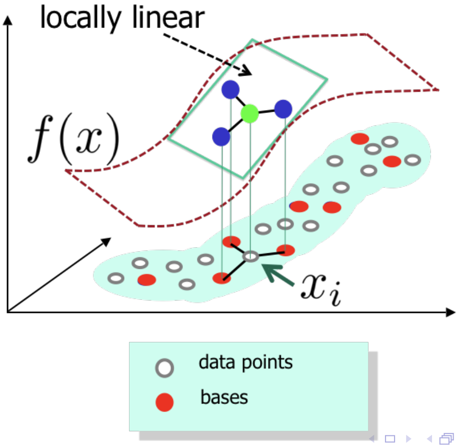


# Locally linear coordinate representation

VLAD feature representation is expensive in terms of size of the feature vector. A better approach was to reduce the quantization error using locally linear coordinate representation

# Locally linear coordinate representation

VLAD feature representation is expensive in terms of size of the feature vector. A better approach was to reduce the quantization error using locally linear coordinate representation



# Locally linear coordinate formulation

The formulation for loally linear coordinate coding is given as follows

# Locally linear coordinate formulation

The formulation for loally linear coordinate coding is given as follows

- Coding for x, to obtain its sparse representation a

  Step 1 – ensure locality: find the K nearest bases

$$[\phi_j]_{j \in J(x)}$$

  Step 2 – ensure low coding error:

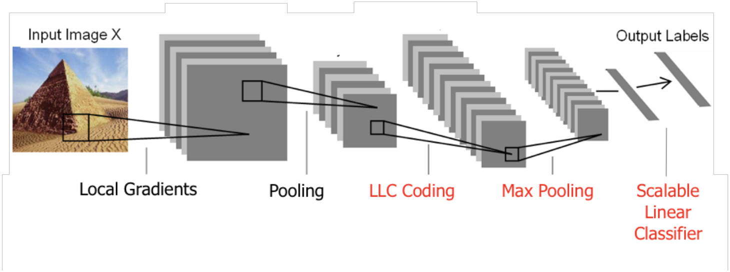
$$\min_a \left\| x - \sum_{j \in J(x)} a_{i,j} \phi_j \right\|^2, \quad \text{s.t.} \sum_{j \in J(x)} a_{i,j} = 1$$

# Locally linear coordinate Overview

The pipeline for the whole framework is presented as follows. This appears to be very similar to the conventional deep learning framework that consists of convolution and pooling operations

# Locally linear coordinate Overview

The pipeline for the whole framework is presented as follows. This appears to be very similar to the conventional deep learning framework that consists of convolution and pooling operations



# Outline

1. Earlier Representations
2. Overview: DL for Categorization
3. Learning Perceptrons
4. Multi Layer Perceptron

# AlexNet

The field was revolutionized in December 2012 with the following seminal work that was presented in NIPS 2012. The work showed that object categorization could be solved with much higher accuracy than was previously thought possible.

# AlexNet

The field was revolutionized in December 2012 with the following seminal work that was presented in NIPS 2012. The work showed that object categorization could be solved with much higher accuracy than was previously thought possible.

## ImageNet Classification with Deep Convolutional Neural Networks

**Alex Krizhevsky**
University of Toronto
kriz@cs.utoronto.ca

**Ilya Sutskever**
University of Toronto
ilya@cs.utoronto.ca

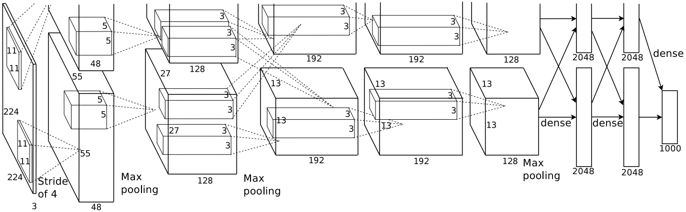
**Geoffrey E. Hinton**
University of Toronto
hinton@cs.utoronto.ca

# AlexNet Architecture

The architecture adopted by Alex Krizhevsky et al is as shown in the figure below. This architecture consists of various convolution and pooling layers with the parameters being learned. As can be observed these are conceptually similar to the architectures being followed by researchers before deep learning approach was introduced.

# AlexNet Architecture

The architecture adopted by Alex Krizhevsky et al is as shown in the figure below. This architecture consists of various convolution and pooling layers with the parameters being learned. As can be observed these are conceptually similar to the architectures being followed by researchers before deep learning approach was introduced.



# AlexNet Results

The results obtained by AlexNet were far superior to those by earlier methods in the ImageNet 2010 dataset

# AlexNet Results

The results obtained by AlexNet were far superior to those by earlier methods in the ImageNet 2010 dataset

| **Model** | **Top-1** | **Top-5** |
|---|---|---|
| *Sparse coding [2]* | *47.1%* | *28.2%* |
| *SIFT + FVs [24]* | *45.7%* | *25.7%* |
| CNN | **37.5%** | **17.0%** |

# AlexNet Results

The authors also won the ImageNet 2012 challenge and again the results obtained were far superior. Further, current state of the art using this approach obtains less than 5% error for this dataset.

# AlexNet Results

The authors also won the ImageNet 2012 challenge and again the results obtained were far superior. Further, current state of the art using this approach obtains less than 5% error for this dataset.

| **Model** | **Top-1 (val)** | **Top-5 (val)** | **Top-5 (test)** |
|---|---|---|---|
| *SIFT + FVs [7]* | — | — | *26.2%* |
| 1 CNN | 40.7% | 18.2% | — |
| 5 CNNs | 38.1% | 16.4% | **16.4%** |
| 1 CNN* | 39.0% | 16.6% | — |
| 7 CNNs* | 36.7% | 15.4% | **15.3%** |

# AlexNet Qualitative Results

The results from the network were also qualitatively very good with even the errors being understandable

# AlexNet Qualitative Results

The results from the network were also qualitatively very good with even the errors being understandable

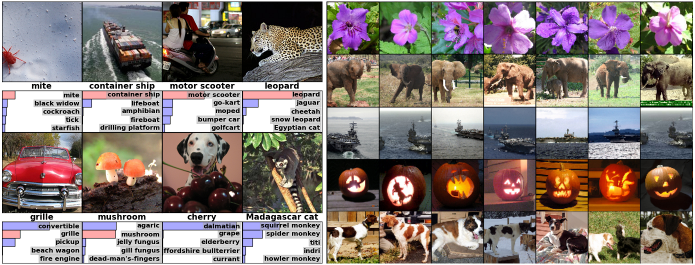



Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

# Outline

# A Perceptron

- Consider a model of a linear perceptron that receives an input vector $\mathbf{x}$ of dimension $n$ i.e. $x_1, x_2, \cdots, x_n$. The output for such a model can be obtained through the following equation

$$o = f(\sum_{i=1}^{n} w_i x_i + \theta) \tag{1}$$

where $w_i$ is the weight applied to each input.

# A Perceptron

- Consider a model of a linear perceptron that receives an input vector $\mathbf{x}$ of dimension $n$ i.e. $x_1, x_2, \cdots, x_n$. The output for such a model can be obtained through the following equation

$$o = f(\sum_{i=1}^{n} w_i x_i + \theta) \tag{1}$$

  where $w_i$ is the weight applied to each input.
- The bias parameter $\theta$ can be included in the weight vector by assuming a constant input $x_0$ with value 1 allowing the use of $n+1$ weights. In general we therefore consider an input vector $\mathbf{x}$ that is multiplied by a parameter vector $\mathbf{w}$

# A Perceptron

- Consider a model of a linear perceptron that receives an input vector $\mathbf{x}$ of dimension $n$ i.e. $x_1, x_2, \cdots, x_n$. The output for such a model can be obtained through the following equation

$$o = f(\sum_{i=1}^{n} w_i x_i + \theta) \tag{1}$$

  where $w_i$ is the weight applied to each input.
- The bias parameter $\theta$ can be included in the weight vector by assuming a constant input $x_0$ with value 1 allowing the use of $n+1$ weights. In general we therefore consider an input vector $\mathbf{x}$ that is multiplied by a parameter vector $\mathbf{w}$
- The function $f()$ is a function that assigns for instance 0 for negative argument and 1 for positive argument

# Illustration of a Perceptron

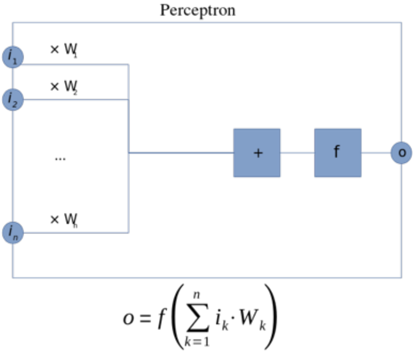



Figure: Perceptron

# The perceptron learning algorithm

- The question then arises as to how to learn the perceptron weights

# The perceptron learning algorithm

- The question then arises as to how to learn the perceptron weights
- A naive algorithm would involve generating various random perceptron weight vectors and checking them till we find a weight vector that separates the training samples.

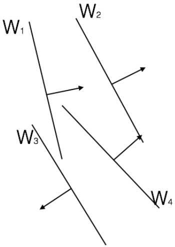



Figure: Naive algorithm for training a perceptron that consists of just sampling random weight vectors

# The perceptron learning algorithm

The perceptron algorithm proposed by Rosenblatt is as follows

1. Initialise the weights and threshold to a small random numbers.

# The perceptron learning algorithm

The perceptron algorithm proposed by Rosenblatt is as follows

1. Initialise the weights and threshold to a small random numbers.
2. Present a vector **x** and calculate the output

# The perceptron learning algorithm

The perceptron algorithm proposed by Rosenblatt is as follows

1. Initialise the weights and threshold to a small random numbers.
2. Present a vector **x** and calculate the output
3. Update the weights according to the following equation:

$$w_j(t+1) = w_j(t) + \eta(d-y)x \tag{2}$$

where
$d$ is the desired output,
$t$ is the iteration number, and
$\eta$ is the gain or step size, where $0.0 < \eta < 1.0$

# The perceptron learning algorithm

The perceptron algorithm proposed by Rosenblatt is as follows

1. Initialise the weights and threshold to a small random numbers.
2. Present a vector **x** and calculate the output
3. Update the weights according to the following equation:

$$w_j(t+1) = w_j(t) + \eta(d-y)x \tag{2}$$

   where
   $d$ is the desired output,
   $t$ is the iteration number, and
   $\eta$ is the gain or step size, where $0.0 < \eta < 1.0$
4. Repeat steps 2 and 3 until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

# Illustration of the Perceptron learning algorithm

Step 1

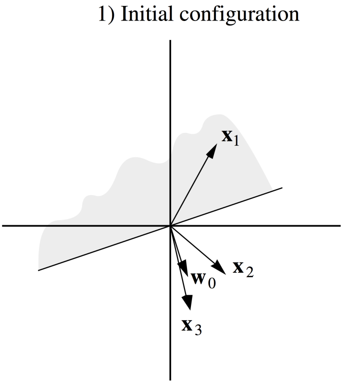



Figure: The different steps for the Perceptron learning algorithm. (fig. credit Rojas 1996)

# Illustration of the Perceptron learning algorithm

Step 2

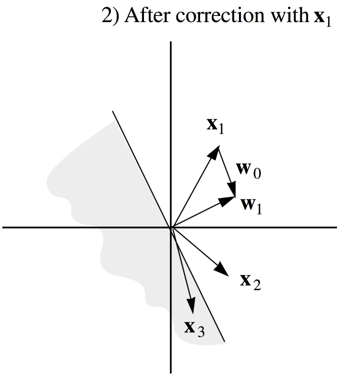



Figure: The different steps for the Perceptron learning algorithm. (fig. credit Rojas 1996)

# Illustration of the Perceptron learning algorithm

Step 3

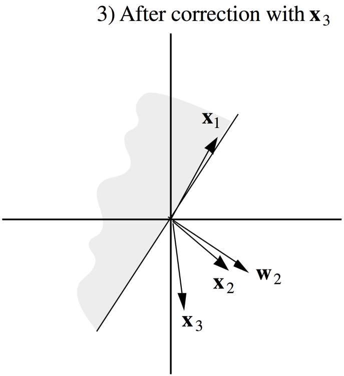



Figure: The different steps for the Perceptron learning algorithm. (fig. credit Rojas 1996)

# Illustration of the Perceptron learning algorithm

Step 4

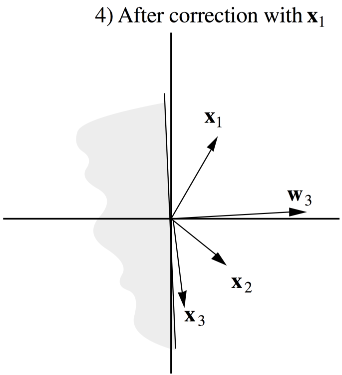



Figure: The different steps for the Perceptron learning algorithm. (fig. credit Rojas 1996)

# Illustration of the Perceptron learning algorithm

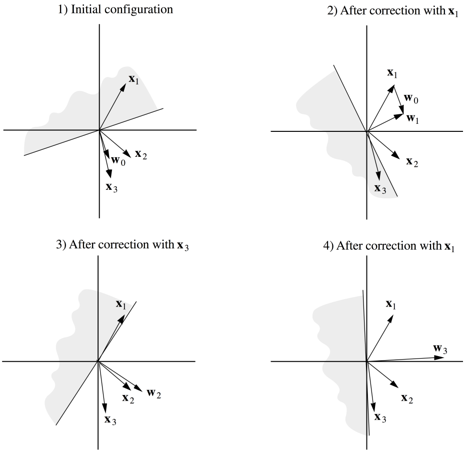



Figure: The different steps for the Perceptron learning algorithm. (fig. credit Rojas 1996)

# Limitations of the Perceptron

## Linear Classifier

# Limitations of the Perceptron

## Linear Classifier

- The perceptron was shown to be limited in its power as it is a linear classifier

# Limitations of the Perceptron

## Linear Classifier

- The perceptron was shown to be limited in its power as it is a linear classifier

## Distinguishing capability

# Limitations of the Perceptron

## Linear Classifier

- The perceptron was shown to be limited in its power as it is a linear classifier

## Distinguishing capability

- Minsky and Papert in their book showed that perceptrons proposed then could not classify XOR functions or distinguish between one or two spirally connected regions

# Limitations of the Perceptron

## Linear Classifier

- The perceptron was shown to be limited in its power as it is a linear classifier

## Distinguishing capability

- Minsky and Papert in their book showed that perceptrons proposed then could not classify XOR functions or distinguish between one or two spirally connected regions

## Next Steps

# Limitations of the Perceptron

## Linear Classifier

- The perceptron was shown to be limited in its power as it is a linear classifier

## Distinguishing capability

- Minsky and Papert in their book showed that perceptrons proposed then could not classify XOR functions or distinguish between one or two spirally connected regions

## Next Steps

- The perceptron of optimal stability together with the kernel trick provide the conceptual foundations of the support vector machine. We next consider the model of Multi-layer Perceptrons

# Outline

# Multi-layer Perceptron

- The multi-layer perceptron addressed the drawbacks and limitations of the perceptron.

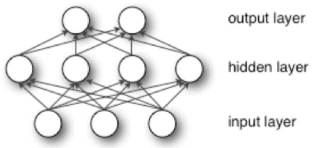



Figure: Multi-Layer Perceptron

# Multi-layer Perceptron

- The multi-layer perceptron addressed the drawbacks and limitations of the perceptron.
- Though it is termed Multi-layer perceptron, it does not use the perceptron algorithm for learning

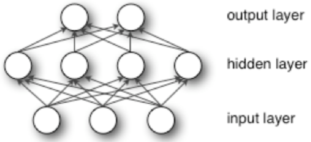



Figure: Multi-Layer Perceptron

# Multi-layer Perceptron

- The multi-layer perceptron addressed the drawbacks and limitations of the perceptron.
- Though it is termed Multi-layer perceptron, it does not use the perceptron algorithm for learning
- For a single perceptron the method used is the delta learning rule and the method used for training the whole network is the backpropagation algorithm

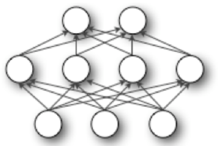



Figure: Multi-Layer Perceptron

# The delta learning rule

- Assuming there exists a feasible separable region, the perceptron algorithm moves the weight vector based on each mistake towards that separable region (away from the mistake)

# The delta learning rule

- Assuming there exists a feasible separable region, the perceptron algorithm moves the weight vector based on each mistake towards that separable region (away from the mistake)
- An alternative approach is to obtain the error across *all samples* and move the weight vector iteratively to minimize the error

# The delta learning rule

- Assuming there exists a feasible separable region, the perceptron algorithm moves the weight vector based on each mistake towards that separable region (away from the mistake)
- An alternative approach is to obtain the error across *all samples* and move the weight vector iteratively to minimize the error
- Suppose we have to guess the price of two items $A$ *and* $B$ from a shopkeeper who just gives the total price

# The delta learning rule

- Assuming there exists a feasible separable region, the perceptron algorithm moves the weight vector based on each mistake towards that separable region (away from the mistake)
- An alternative approach is to obtain the error across *all samples* and move the weight vector iteratively to minimize the error
- Suppose we have to guess the price of two items $A$ *and* $B$ from a shopkeeper who just gives the total price
- We do this by buying multiple items of A and B and summing the error from the total price. We then adjust the price based on the total error that we observe. This updation based on error is called the delta learning rule

# Delta learning rule: Example

Consider that we go to a shop and buy some object and observe the error in obtaining the price per sample


Figure: Example for delta learning rule

# Delta learning rule: Example

Consider that we go to a shop and buy some object and observe the error in obtaining the price per sample

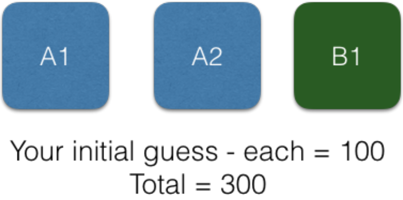



Figure: Example for delta learning rule

# Delta learning rule: Example

Consider that we go to a shop and buy some object and observe the error in obtaining the price per sample

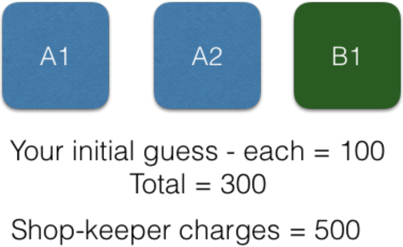



Figure: Example for delta learning rule

# Delta learning rule: Example

Consider that we go to a shop and buy some object and observe the error in obtaining the price per sample

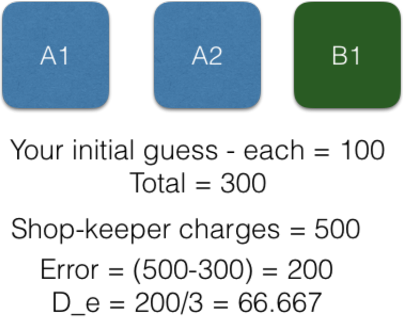



Figure: Example for delta learning rule

# Delta learning rule: Example

Consider that we go to a shop and buy some object and observe the error in obtaining the price per sample

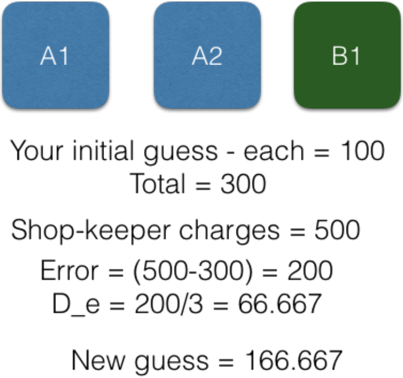



Figure: Example for delta learning rule

# The delta learning rule - formulation

- Obtain error $E$ given by

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2 \tag{3}$$

where $t_j$ is the desired output and $y_j$ is the predicted output for sample $j$

# The delta learning rule - formulation

- Obtain error $E$ given by

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2 \tag{3}$$

where $t_j$ is the desired output and $y_j$ is the predicted output for sample $j$

- The contribution of the weight vector $w_{ji}$ to the error can be used to update the weight vector. This is obtained by taking the gradient of the error $E$ with respect to the weight vector.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial \left( \sum_j \frac{1}{2}(t_j - y_j)^2 \right)}{\partial w_{ji}} \tag{4}$$

# The delta learning rule - formulation

- Obtain error $E$ given by

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2 \tag{3}$$

where $t_j$ is the desired output and $y_j$ is the predicted output for sample $j$

- The contribution of the weight vector $w_{ji}$ to the error can be used to update the weight vector. This is obtained by taking the gradient of the error $E$ with respect to the weight vector.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial \left( \sum_j \frac{1}{2}(t_j - y_j)^2 \right)}{\partial w_{ji}} \tag{4}$$

- The equation 4 can be solved using chain rule as

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial \left( \sum_j \frac{1}{2}(t_j - y_j)^2 \right)}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} \tag{5}$$

# The delta learning rule - formulation

- We note that $y_j = g(h_j)$ where $h_j = \sum_i x_i w_{ji}$

# The delta learning rule - formulation

- We note that $y_j = g(h_j)$ where $h_j = \sum_i x_i w_{ji}$
- Therefore equation 5 can be further simplified as:

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - y_j)\frac{\partial y_j}{\partial h_j}\frac{\partial h_j}{\partial w_{ji}} \quad (6)$$

# The delta learning rule - formulation

- We note that $y_j = g(h_j)$ where $h_j = \sum_i x_i w_{ji}$
- Therefore equation 5 can be further simplified as:

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - y_j)\frac{\partial y_j}{\partial h_j}\frac{\partial h_j}{\partial w_{ji}} \tag{6}$$

- As $\frac{\partial h_j}{\partial w_{ji}}$ is just $x_i$ we obtain the following final expression

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - y_j)g'(h_j)x_i \tag{7}$$

where $g'(h_j)$ is $\frac{\partial y_j}{\partial h_j}$

# The delta learning rule - formulation

- We note that $y_j = g(h_j)$ where $h_j = \sum_i x_i w_{ji}$
- Therefore equation 5 can be further simplified as:

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - y_j)\frac{\partial y_j}{\partial h_j}\frac{\partial h_j}{\partial w_{ji}} \tag{6}$$

- As $\frac{\partial h_j}{\partial w_{ji}}$ is just $x_i$ we obtain the following final expression

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - y_j)g'(h_j)x_i \tag{7}$$

  where $g'(h_j)$ is $\frac{\partial y_j}{\partial h_j}$
- Based on this we can update the weight vector $w_{ji}$ using the following update rule

$$w_{ji}^{t+1} = w_{ji}^{t} + \eta \triangle w_{ji} \tag{8}$$

  where $\triangle w_{ji}$ is $\frac{\partial E}{\partial w_{ji}}$ and $\eta$ is the learning rate parameter

# The delta learning rule

- The learning procedure iteratively approaches the target output and by adapting the learning rate one can approach close to the target output

# The delta learning rule

- The learning procedure iteratively approaches the target output and by adapting the learning rate one can approach close to the target output
- Learning can be slow if the samples are correlated (we purchase similar number of items all the time)

# The delta learning rule

- The learning procedure iteratively approaches the target output and by adapting the learning rate one can approach close to the target output
- Learning can be slow if the samples are correlated (we purchase similar number of items all the time)
- In perceptron rule, we change the weight vector if we make a mistake, in delta rule we change the weights iteratively based on the cumulative error

# The delta learning rule

- The learning procedure iteratively approaches the target output and by adapting the learning rate one can approach close to the target output
- Learning can be slow if the samples are correlated (we purchase similar number of items all the time)
- In perceptron rule, we change the weight vector if we make a mistake, in delta rule we change the weights iteratively based on the cumulative error
- The delta rule is a specialisation of the back-propagation algorithm

# Sigmoidal Activation function

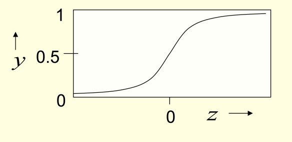



Figure: Sigmoidal Activation function

- So far we have not specified the activation function $g(h_j)$ that is used. We have $y = g(h_j)$ where $h_j = \sum_i x_i w_{ji}$. The learning procedure iteratively approaches the target output and by adapting the learning rate one can approach close to the target output

# Sigmoidal Activation function

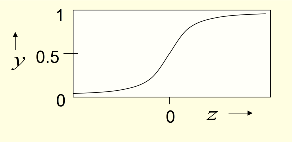



Figure: Sigmoidal Activation function

- So far we have not specified the activation function $g(h_j)$ that is used. We have $y = g(h_j)$ where $h_j = \sum_i x_i w_{ji}$. The learning procedure iteratively approaches the target output and by adapting the learning rate one can approach close to the target output
- One such activation function that can be used is sigmoidal activation function (to approximate the discontinuous Heaviside function used in perceptrons).

# Sigmoidal Activation function

- The sigmoidal activation function is given by

$$y_j = \frac{1}{1 + e^{-h_j}} \tag{9}$$

# Sigmoidal Activation function

- The sigmoidal activation function is given by

$$y_j = \frac{1}{1 + e^{-h_j}} \tag{9}$$

- The derivative of the sigmoid is obtained as

$$\frac{\partial y_j}{\partial h_j} = y_j(1 - y_j) \tag{10}$$

as $\frac{\partial y_j}{\partial h_j} = \frac{-1}{(1+e^{-h_j})^2}(-e^{-h_j}) = (\frac{1}{1+e^{-h_j}})(\frac{e^{-h_j}}{1+e^{-h_j}}) = y_j(1 - y_j)$

# Sigmoidal Activation function

- The sigmoidal activation function is given by

$$y_j = \frac{1}{1 + e^{-h_j}} \tag{9}$$

- The derivative of the sigmoid is obtained as

$$\frac{\partial y_j}{\partial h_j} = y_j(1 - y_j) \tag{10}$$

as $\frac{\partial y_j}{\partial h_j} = \frac{-1}{(1+e^{-h_j})^2}(-e^{-h_j}) = (\frac{1}{1+e^{-h_j}})(\frac{e^{-h_j}}{1+e^{-h_j}}) = y_j(1 - y_j)$

- The equation 7 can now be obtained by substituting the gradient of the function as

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - y_j)y_j(1 - y_j)x_i \tag{11}$$

# Learning with many layers

- We have so far obtained through the delta learning rule the ability to perturb the weights of one perceptron with respect to the error.

# Learning with many layers

- We have so far obtained through the delta learning rule the ability to perturb the weights of one perceptron with respect to the error.
- In a multi-layer perceptron, there are many sets of weights with hidden units as well. How do we modify the weights for such a network?

# Learning with many layers

- We have so far obtained through the delta learning rule the ability to perturb the weights of one perceptron with respect to the error.
- In a multi-layer perceptron, there are many sets of weights with hidden units as well. How do we modify the weights for such a network?
- Should we perturb the weights one at a time? Should we perturb the weights randomly?

# Learning with many layers

- We have so far obtained through the delta learning rule the ability to perturb the weights of one perceptron with respect to the error.
- In a multi-layer perceptron, there are many sets of weights with hidden units as well. How do we modify the weights for such a network?
- Should we perturb the weights one at a time? Should we perturb the weights randomly?
- This is obtained through the backpropagation algorithm that propagates the errors and updates all weights based on the error.

# Learning with many layers

- We have so far obtained through the delta learning rule the ability to perturb the weights of one perceptron with respect to the error.
- In a multi-layer perceptron, there are many sets of weights with hidden units as well. How do we modify the weights for such a network?
- Should we perturb the weights one at a time? Should we perturb the weights randomly?
- This is obtained through the backpropagation algorithm that propagates the errors and updates all weights based on the error.
- It relies on the computation of gradient of error $E$ with respect to any weight vector in the network.

# Backpropagation Algorithm

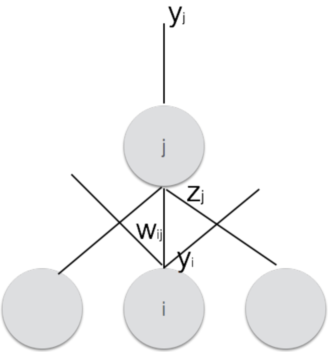



Figure: Backpropagation algorithm

- $E = \frac{1}{2}\sum_j (t_j - y_j)^2$ Therefore we have

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j). \tag{12}$$

# Backpropagation Algorithm

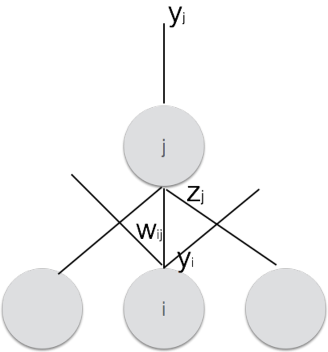



Figure: Backpropagation algorithm

- $E = \frac{1}{2}\sum_j (t_j - y_j)^2$ Therefore we have

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j). \tag{12}$$

- Next considering the gradient with respect to $z_j$ we have

$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j}\frac{\partial E}{\partial y_j}. \tag{13}$$

which is obtained as

$$\frac{\partial E}{\partial z_j} = y_j(1 - y_j)\frac{\partial E}{\partial y_j}. \tag{14}$$

# Backpropagation Algorithm

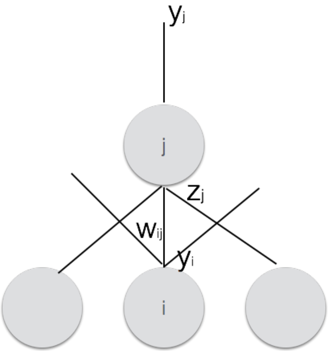



Figure: Backpropagation algorithm

- The expression for error with respect to $y_i$ is obtained as

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i} \frac{\partial E}{\partial z_j}. \qquad (15)$$

that is given by

$$\frac{\partial E}{\partial y_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}. \qquad (16)$$

as the output $y_i$ is being sent to $j$ nodes with the weight vector $w_{ij}$ between nodes $i$ and $j$

# Backpropagation Algorithm

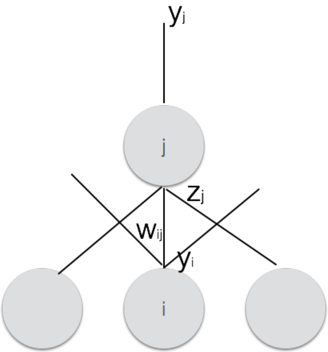



Figure: Backpropagation algorithm

- The expression for error with respect to $w_{ij}$ is obtained as

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j}. \tag{17}$$

that is given by

$$\frac{\partial E}{\partial w_{ij}} = y_i \frac{\partial E}{\partial z_j}. \tag{18}$$

as the output $y_i$ is being sent to $j$ nodes with the weight vector $w_{ij}$ between nodes $i$ and $j$

# Backpropagation algorithm

- Used to update all the weights based on the update rule.

# Backpropagation algorithm

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate $\eta$

# Backpropagation algorithm

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate $\eta$
- There are different modes of update

# Backpropagation algorithm

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate $\eta$
- There are different modes of update
  - Online: After seeing each training sample

# Backpropagation algorithm

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate $\eta$
- There are different modes of update
  - Online: After seeing each training sample
  - Full: After seeing all the training samples

# Backpropagation algorithm

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate $\eta$
- There are different modes of update
  - Online: After seeing each training sample
  - Full: After seeing all the training samples
  - mini-batch: After seeing a small set of training samples. This is the most common case.

# Backpropagation algorithm

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate $\eta$
- There are different modes of update
  - Online: After seeing each training sample
  - Full: After seeing all the training samples
  - mini-batch: After seeing a small set of training samples. This is the most common case.
- Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum

# The End