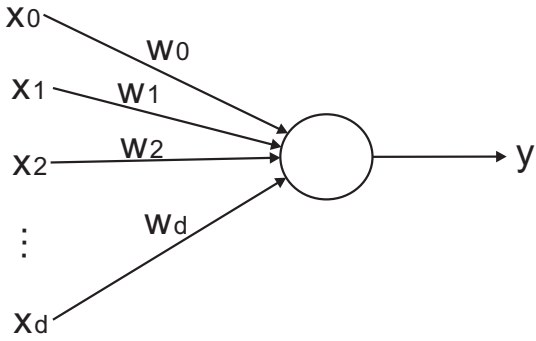
パターン認識
Pattern Recognition

佐藤真一
Shin'ichi Satoh

国立情報学研究所
National Institute of Informatics

May 30, 2023



$$g(x) = w_0 + \sum_{j=1}^{d} w_j x_j.$$

Summary of linear discriminant functions

Classifier	Criterion function	Solver
Perceptron	$J = \sum_{x \in \tilde{\mathcal{X}}} (-w^T x)$	SGD
MSE	$J_p = \frac{1}{2}(\hat{w}^T \hat{x_p} - b_p)^2$	Pinv
Widrow-Hoff	$J_p = \frac{1}{2}(\hat{w}^T \hat{x_p} - b_p)^2$	SGD
Neural network	$J_p = \frac{1}{2}(f(\hat{w}^T \hat{x_p}) - b_p)^2$	SGD

The Perceptron (recap)

Rosenblatt, 1957[1]
We want to determine w so that $w^T x > 0$ for all training samples.
The Perceptron criterion function:

$$J(w) = \sum_{x \in \tilde{\mathcal{X}}} (-w^T x)$$

where $\tilde{\mathcal{X}}$ is the set of samples misclassified.
J is never negative, and we want it to be zero ($\tilde{\mathcal{X}}$ to be empty).

[1] F. Rosenblatt, The perceptron - A perceiving and recognizing automaton, Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January, 1957.

Since the gradient of J is

$$\nabla J = \sum_{x \in \tilde{\mathcal{X}}} (-x)$$

we can update the weight vector based on Gradient Descent (aka batch Gradient Descent) as

$$w(k+1) = w(k) - \rho \nabla J = w(k) + \rho \sum_{x \in \tilde{\mathcal{X}}} x.$$

ρ is called learning rate.

Minimum Squared-Error and the Pseudoinverse (recap)

We assume

$$X = [\hat{x_1}\ \hat{x_2}\ \cdots\ \hat{x_n}]^T$$
$$b_i = [b_{i1}\ b_{i2}\ \cdots\ b_{in}]^T\ (i = 1, \cdots, c)$$

then the criterion function will be

$$J(\hat{w_1}, \hat{w_2}, \cdots, \hat{w_c}) = \frac{1}{2}\sum_{i=1}^{c} \|X\hat{w_i} - b_i\|^2$$
$$\frac{\partial J}{\partial \hat{w_i}} = X^T(X\hat{w_i} - b_i) = 0$$
$$X^T X\hat{w_i} = X^T b_i$$
$$\hat{w_i} = (X^T X)^{-1} X^T b_i$$

This gives MSE solution of $\|Xw_i - b_i\|^2$.

The Widrow-Hoff or LMS Procedure (recap)

We now consider gradient descent:

$$\hat{w}_i(k+1) = \hat{w}_i(k) - \rho \frac{\partial J}{\partial \hat{w}_i} = \hat{w}_i(k) - \rho \nabla_i J$$
$$\Delta \hat{w}_i = -\rho \nabla_i J$$

We can also consider stochastic gradient descent:

$$\Delta \hat{w}_i = -\rho \frac{\partial J_p}{\partial \hat{w}_i}$$

The Widrow-Hoff or LMS Procedure (recap)

Here we denote $g_i(x_p)$ as g_{ip}.

$$\frac{\partial J_p}{\partial \hat{w}_i} = \frac{\partial J_p}{\partial g_{ip}} \frac{\partial g_{ip}}{\partial \hat{w}_i}$$

where

$$\frac{\partial J_p}{\partial g_{ip}} = g_{ip} - b_{ip} = \varepsilon_{ip} \quad (\because J_p = \frac{1}{2} \sum_{i=1}^{c} (\hat{w_i}^T \hat{x_p} - b_{ip})^2)$$

$$\frac{\partial g_{ip}}{\partial \hat{w}_i} = \hat{x_p} \quad (\because g_{ip} = g_i(x_p) = \hat{w_i}^T \hat{x_p})$$

therefore

$$\frac{\partial J_p}{\partial \hat{w}_i} = (g_{ip} - b_{ip})\hat{x_p} = \varepsilon_{ip}\hat{x_p}$$

The Widrow-Hoff or LMS Procedure (recap)

The update rule becomes

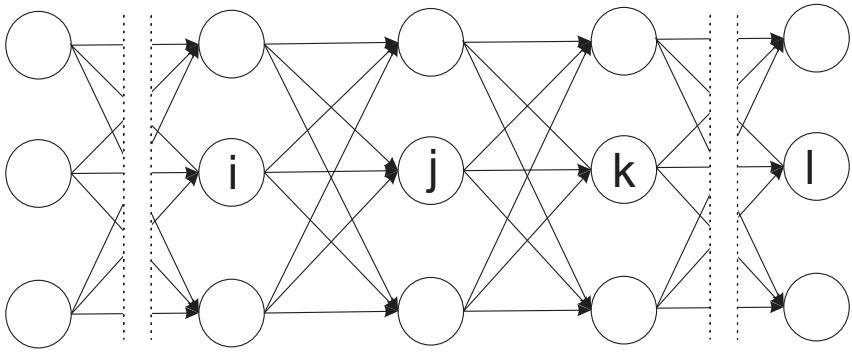
$$\begin{aligned}\Delta \hat{w}_i &= \frac{\partial J_p}{\partial \hat{w}_i} \\ &= -\rho \varepsilon_{ip} \hat{x_p} \\ &= -\rho (g_{ip} - b_{ip}) \hat{x_p} \\ &= -\rho (\hat{w_i}^T \hat{x_p} - b_{ip}) \hat{x_p}.\end{aligned}$$

This is the Widrow-Hoff or LMS rule (least-mean-squared).

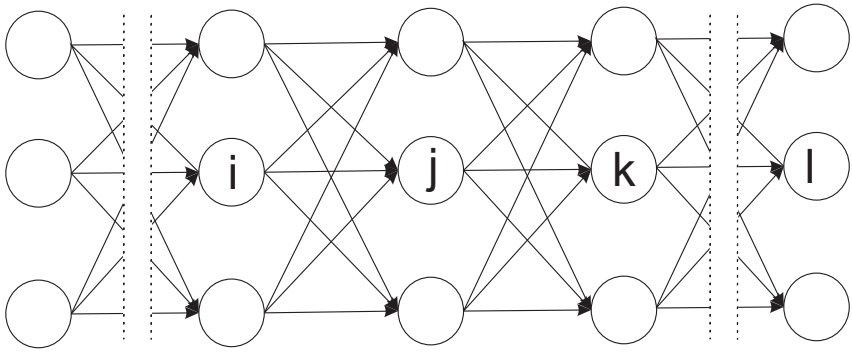
Neural Network

- A brain is a information processing system composed of the network of huge number of neural cells, and can flexibly perform intelligent processes such as pattern recognition
- An (artificial) neural network is a computation model for pattern recognition and other intelligent processes by artificially implemented neural cells and networks with software and so on
- One aim is to explore the mechanism of neurons and brains, i.e., biological aspect, and another aim is to explore the basic principle of parallel and distributed processes by neural network, i.e., computer science aspect
- McCulloch-Pitts model (1943), Hebb rule (1949)
- Feed forward network, recurrent neural network
- supervised learning, unsupervised learning

Multilayer Neural Network



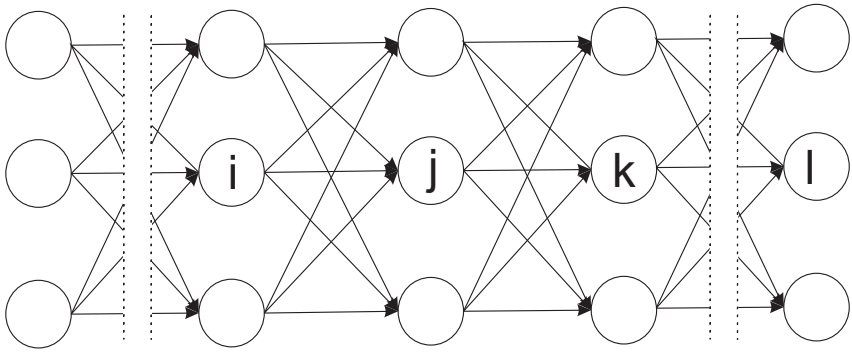
Multilayer Neural Network



$$y = [w_{kl}]\,[w_{jk}]\,[w_{ij}]\,x$$

$$= \left[\sum_{j,k} w_{ij}\,w_{jk}\,w_{kl}\right] x$$

Multilayer Neural Network



$$y = f\left([w_{kl}]\, f\left([w_{jk}]\, f\left([w_{ij}]\, x\right)\right)\right)$$

Multilayer Neural Network

Simply stacking linear discriminant functions does not work: combination of linear functions is equal to a linear function.
So we introduce non-linear function $f(\cdot)$.

$$h_{jp} = \sum_i w_{ij} g_{ip}$$
$$g_{jp} = f(h_{jp})$$

Then the criterion function

$$J_p = \frac{1}{2} \sum_l (g_{lp} - b_{lp})^2$$
$$J = \sum_p J_p$$

We can solve this by gradient descent.

$$\frac{\partial J_p}{\partial w_{ij}} = \frac{\partial J_p}{\partial h_{jp}} \cdot \frac{\partial h_{jp}}{\partial w_{ij}}$$

$$\varepsilon_{jp} \stackrel{\text{def}}{=} \frac{\partial J_p}{\partial h_{jp}}$$

$$\frac{\partial h_{jp}}{\partial w_{ij}} = g_{ip} \quad (\because h_{jp} = \sum_i w_{ij} g_{ip})$$

$$\frac{\partial J_p}{\partial w_{ij}} = \varepsilon_{jp} g_{ip}$$

$$\Delta w_{ij} = -\rho \frac{\partial J_p}{\partial w_{ij}} = -\rho \varepsilon_{jp} g_{ip}$$

How to determine ε_{jp} for each unit?

$$\begin{aligned}\varepsilon_{jp} &= \frac{\partial J_p}{\partial h_{jp}} = \frac{\partial J_p}{\partial g_{jp}} \cdot \frac{\partial g_{jp}}{\partial h_{jp}} \\ &= \frac{\partial J_p}{\partial g_{jp}} \cdot f'(h_{jp}) \quad (\because g_{jp} = f(h_{jp}))\end{aligned}$$

If the unit j is in the output layer,

$$\frac{\partial J_p}{\partial g_{jp}} = g_{jp} - b_{jp} \quad (\because J_p = \frac{1}{2}\sum_l (g_{lp} - b_{lp})^2).$$

If the unit j is in the intermediate layers,

$$\frac{\partial J_p}{\partial g_{jp}} = \sum_k \frac{\partial J_p}{\partial h_{kp}} \cdot \frac{\partial h_{kp}}{\partial g_{jp}} \quad (\because h_{kp} = \sum_j w_{jk} g_{jp}).$$

where

$$\frac{\partial J_p}{\partial h_{kp}} = \varepsilon_{kp}$$

$$h_{kp} = \sum_j w_{jk} g_{jp}$$

$$\frac{\partial h_{kp}}{\partial g_{jp}} = w_{jk}$$

therefore

$$\frac{\partial J_p}{\partial g_{jp}} = \sum_k \frac{\partial J_p}{\partial h_{kp}} \cdot \frac{\partial h_{kp}}{\partial g_{jp}}$$

$$= \sum_k \varepsilon_{kp} w_{jk}$$

How about the non-linear function $f \cdot$?
We can consider a threshold function but it's not differentiable.
Instead we use the following sigmoid function

$$f(u) = \frac{1}{1 + e^{-u}}$$
$$f'(u) = f(u)(1 - f(u))$$

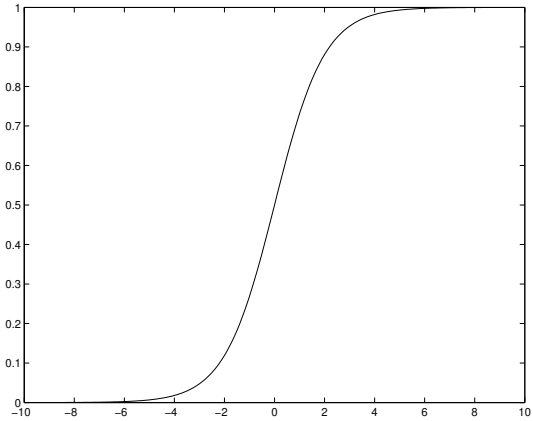
So,

$$f'(h_{jp}) = g_{jp}(1 - g_{jp}) \quad (\because f(h_{jp}) = g_{jp})$$

In summary,

$$\varepsilon_{jp} = \begin{cases} (g_{jp} - b_{jp})g_{jp}(1 - g_{jp}) & (j \text{ is in the output layer}) \\ (\sum_k \varepsilon_{kp} w_{jk})g_{jp}(1 - g_{jp}) & (j \text{ is in the intermediate layers}) \end{cases}$$

The sigmoid function



The Backpropagation

How to train overall network?

- When a training vector is input, based on the difference between the training signals and output in the output layer, ε_{jp}s are computed.
- The weights between the output layer and the previous layer are then updated.
- Based on them ε_{jp}s of the previous layer are then computed, then weights of the previous layer are updated.
- The process is continued up to the input layer.
- The overall process is further continued for all data.
- Again, the overall process is continued multiple times visiting all data (one such step is called epoch).

Since the error will be propagated from the output layer to the input layer, this algorithm is called the backpropagation.

Deep Learning

Deep learning is a kind of multilayer neural network especially having huge number of layers.
Theoretically multilayer neural network is equivalent to piece-wise linear classifier, and by using large number of layers, arbitrarily complex decision boundary can be obtained.
However, the training is getting very hard.
Couple of recent innovations enabled deep learning

- Pre-training, autoencoder, restricted Boltzmann machine (RBM)
- Dropout
- Fast computers, GPUs

Deep convolutional neural network is also very popular.

- Convolutional layer
- Pooling layer
- Fully connected layer

Two key innovations (SGD, convolutional layer) were made by Japanese.

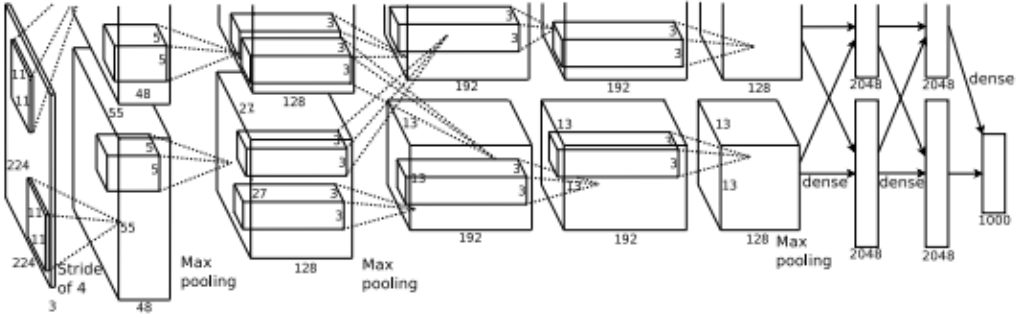


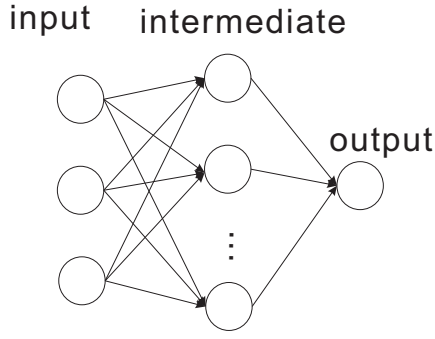
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Krizhevsky, Sutskever, and Hinton, NIPS 2012.

Assignment

- Programming project and non-programming project are imposed.
- You are expected to solve either programming project OR non-programming project.
- Programming project is recommended.
- Of course you are most welcomed to solve both.
- Due on June 13.

Programming Project 1. Implement multilayer neural network with the following architecture.



Train the network with the three data sets provided last week using backpropagation. How many intermediate units needed?

Note: don't use neural network tools or libraries. Implement computing responses of units, sigmoid functions, etc., such as `outp = sigmoid(sigmoid(inp.dot(Wim)).dot(Wmo))`.

Programming Project 2. (mid level) Implement MSE classifier for MNIST. How is the accuracy?
(FYI. My implementation achieved 86%.)

Programming Project 3. (Optional) Implement neural network classifier for MNIST, with the same architecture to the assignment 1. How is the accuracy?
(97.7%)

Assignment

Non-Programming Project Explain the following technical terms related to deep learning. Consider how we can incorporate them into the explained multilayer neural network in the course.

- dropout
- momentum
- data augmentation

Assignment 1

```
import numpy as np
import matplotlib.pyplot as plt
from nonlinear import *

inp = 3  # input vector dim
mid = 10  # num of units of intermediate layer
out = 1  # num of units of output layer
rho = 1  # learning rate
wim = np.random.rand(inp, mid) * 2 - 1  # weights from input to intermediate
wmo = np.random.rand(mid, out) * 2 - 1  # weights from intermediate to output
ax = np.concatenate((np.ones((1, n)), x))  # extended vector
sigmoid = lambda x: 1. / (1. + np.exp(-x))
```

```
xx, yy = np.meshgrid(np.linspace(-1, 1), np.linspace(-1, 1))
xf = xx.flatten()[:, np.newaxis]
yf = yy.flatten()[:, np.newaxis]
axy = np.concatenate((np.ones(xf.shape[0])[:, np.newaxis], xf, yf),
                     axis=1).T
l = (l == 1)
plt.figure()
plt.ion()
```

Assignment 1

```
while True:  # loop for each epoch
    for i in range(n):  # loop for each data
        gm = <<compute output of the intermediate layer>>
        go = <<compute output of the output layer>>
        eo = <<compute error of the output layer>>
        em = <<compute error of the intermediate layer>>
        wim = wim - rho * <<update W_{im}>>
        wmo = wmo - rho * <<update W_{mo}>>
```

```
plt.clf()
plt.xlim([-1, 1])
plt.ylim([-1, 1])
plt.plot(x[0, np.where(l == 1)], x[1, np.where(l == 1)], 'bo')
plt.plot(x[0, np.where(l == 0)], x[1, np.where(l == 0)], 'bx')
p = sigmoid(axy.T.dot(wim)).dot(wmo)  # compute classification results
cs = plt.contour(xx, yy, np.reshape(p, xx.shape),
                 levels=[-5, 0, 5], colors='g')
plt.clabel(cs)
# plt.show()
plt.pause(0.01)
```