Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

PROC. OF THE IEEE, NOVEMBER 1998

Student: Gimmy Wu

Advisor: Jenn-Jier James Lien (連震杰)

Robotics Lab, CSIE NCKU

Outline ??

0. Abstract and Nomenclature
1. Introduction
   1) Learning from data
   2) Gradient-Based Learning
   3) Gradient Back-Propagation
   4) Learning in Real Handwriting Recognition System
   5) Globally Trainable Systems
2. Convolutional neural networks for isolated character recognition
   1) Convolutional Networks
   2) LeNet-5
   3) Loss Function
3. Result and comparison with other methods
   1) Database: the Modified NIST set
   2) Results
   3) Comparison with Other Classifiers
   4) Discussion
   5) Invariance and Noise Resistence
4. Multi-module system and graph transformer networks
5. Multiple object recognition: Heuristic over-segmentation
6. Global training for graph transformer networks
7. Multiple object recognition: Space displacement neural network
8. Graph transformer networks and transducers
9. An on-line handwriting recognition system
10. A check reading system
11. Conclusion

0. Background knowledge (1/2) ??

Loss function

$$\min E = \frac{1}{N}\sum_{i=1}^{N}(b_i - b_i')$$

MSE

Estimation

GT: Groundtruth

- Linear Algebra
- Probability

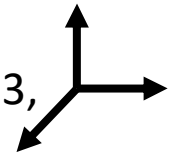
→ Signal Processing (cos, sin) → Image Processing (wavelets) → Computer Vision Machine Learning → AI → Deep Learning

- Eigen vector: e (basic function: cos, sin)
  - Orthonormal ??
- Eigen value: λ (parameter)

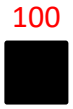
$$b_M = \lambda_1 e_1 + \ \dots + \lambda_N e_N$$

N = 10000 (100x100)

If N = 3,

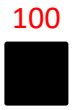


- C_{NxM} =

b1 =  100 image – mean()

100 Face

...

b_M =  100 image

100

M [Image ... Image] M = 2000

- ??$C = UWVT$ SVD

Affine transform??

$$= \begin{matrix} e_1 \\ \\ e_N \end{matrix}\begin{bmatrix} \boxed{} \\ \dots \\ \boxed{} \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \lambda_N \end{bmatrix}\begin{bmatrix} & \dots & \end{bmatrix}$$

- Homogenous Matrix: A

Estimate Data Para

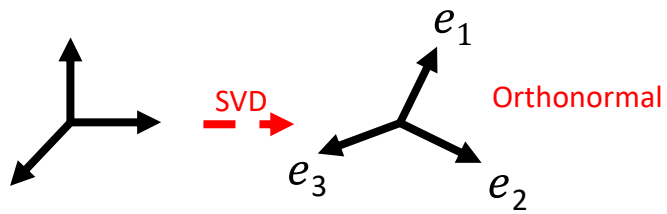
$$b = Ax$$

Solve this

- Linear Combination:

$$\begin{aligned} b_1 &= a_1 x_{11} + \ \dots + \ a_N x_{1N} \\ \dots \quad & \quad \dots \qquad\qquad \dots \\ b_M &= a_1 x_{M1} + \ \dots + \ a_N x_{MN} \end{aligned}$$

$$\begin{bmatrix} b_1 \\ \dots \\ b_M \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1N} \\ \dots & \dots & \dots \\ x_{M1} & \dots & x_{MN} \end{bmatrix}\begin{bmatrix} a_1 \\ \dots \\ a_N \end{bmatrix}$$



After SVD -> transform to "eigen space"

0. Background knowledge: Orthonormal (2/2) ??

0. Abstract and Nomenclature

Abstract—

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called Graph Transformer Networks (GTN), allows such multi-module systems to be trained globally using Gradient-Based methods so as to minimize an overall performance measure.

- Multilayer neural network use gradient-based learning can be used for classification.
- This paper reviews various methods applied to handwritten character recognition and compares to standard ones.
- Convolutional neural network (CNN) outperforms all other techniques.

Two systems for on-line handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of Graph Transformer Networks.

A Graph Transformer Network for reading bank check is also described. It uses Convolutional Neural Network character recognizers combined with global training techniques to provides record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

Keywords— Neural Networks, OCR, Document Recognition, Machine Learning, Gradient-Based Learning, Convolutional Neural Networks, Graph Transformer Networks, Finite State Transducers.

0. Abstract and Nomenclature

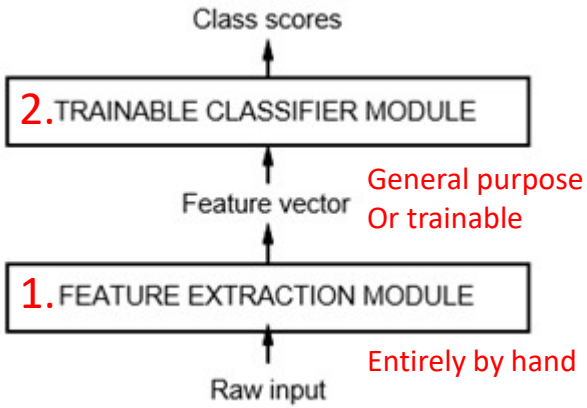
- GT Graph transformer.
- GTN Graph transformer network.
- HMM Hidden Markov model.
- HOS Heuristic oversegmentation.
- K-NN K-nearest neighbor.
- NN Neural network.
- OCR Optical character recognition.
- PCA Principal component analysis.
- RBF Radial basis function.
- RS-SVM Reduced-set support vector method.
- SDNN Space displacement neural network.
- SVM Support vector method.
- TDNN Time delay neural network.
- V-SVM Virtual support vector method.

1. Introduction

A Graph Transformer Network for reading bank check is also described. It uses Convolutional Neural Network character recognizers combined with global training techniques to provides record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

Classify using feature vectors

Transform the input patterns into low-dimensional vectors or short string.



General purpose Or trainable

Entirely by hand

Fig. Usual method of recognizing individual patterns

- Variability + Richness (features of an image)
  => Impossible to build an accurate recognition system entirely by hand.
- Most pattern recognition systems are built:
  1. Automatic learning techniques
  2. Hand-crafted algorithms.

- 3 factors: (change the vision that it is necessary to have an appropriate feature extractor)
  - Low-cost machines
  - Large dataset
  - Powerful machine learning techniques

Historically, the need for appropriate feature extractors was due to the fact that the learning techniques used by the classifiers were limited to low-dimensional spaces with easily separable classes [1]. A combination of three factors have changed this vision over the last decade. First, the availability of low-cost machines with fast arithmetic units allows to rely more on brute-force "numerical" methods than on algorithmic refinements. Second, the availability of large databases for problems with a large market and wide interest, such as handwriting recognition, has enabled designers to rely more on real data and less on hand-crafted feature extraction to build recognition systems. The third and very important factor is the availability of powerful machine learning techniques that can handle high-dimensional inputs and can generate intricate decision functions when fed with these large data sets.

Noise:
1. White noise (Gaussian noise)
2. Salt-and-pepper noise

1. Introduction

1) Learning from data (1/3)

Object: learning model (find out W) to minimize Ep

There are several approaches to automatic machine learning, but one of the most successful approaches, popularized in recent years by the neural network community, can be called "numerical" or *gradient-based learning*.

learning machine computes a function $Y^p = F(Z^p, W)$

Z^p is the p-th input pattern

Exp, 5x5 input pattern, pattern is as output of previous layer, such as raw input or feature

W represents the collection of adjustable parameters in the system

the output Y^p may be inter- preted as the recognized class label of pattern Z^p.

2. Loss function

Ground-truth

Output of system

$$E^p = D(D^p, F(Z^p, W))$$

D^p: the desired output for pattern Z^p

Z^p: p-th input pattern (ex: 28X28 grayscale image)

W: adjustable parameters(ex: weight, bias)

E^p: the discrepancy between desired output and the system output

D: function of error (ex: MSE)

Regularization

$$E(w) = \frac{1}{N} \min \sum (D^P - F(W, Z^P))^2 + N(m = 0, \text{var})$$

(White noise)

A loss function $E^p = \mathcal{D}(D^p, F(W, Z^p))$, measures the discrepancy between D^p, the "correct" or desired output for pattern Z^p, and the output produced by the system. The average loss function $E_{train}(W)$ is the average of the errors E^p over a set of labeled examples called the training set $\{(Z^1, D^1), \ldots (Z^P, D^P)\}$. In the simplest setting, the learning problem consists in finding the value of W that minimizes $E_{train}(W)$.

the gap between the expected error rate on the test set E_{test} and the error rate on the training set E_{train} decreases with the number of training samples approximately

as $E_{test} - E_{train} = k(h/P)^\alpha$

P is the number of training samples

h is a measure of "effective capacity" or complexity of the machine

α is a number between 0.5 and 1.0

k is a constant

Ex : hidden layer number, neuron number in one hidden layer

$$(E_{test} - E_{train})^2 = \frac{h}{P}$$

J: If P=10000 total training samples, that is, the inconsistent result of ($E_{Test} - E_{Train}$) will be only h samples

Purpose is to find best W_{test}, which can approximate the best E_{test}

1. Neural network function:

?? ??

$$Y^p = F(Z^p, W)$$

Z^p: p-th input pattern (ex: 28X28 grayscale image)

W: adjustable parameters(ex: weight, bias)

Y^p: recognized class label of pattern Z^p (ex: the digit)

F: Transfer function

1. Introduction

1) Learning from data (2/3)

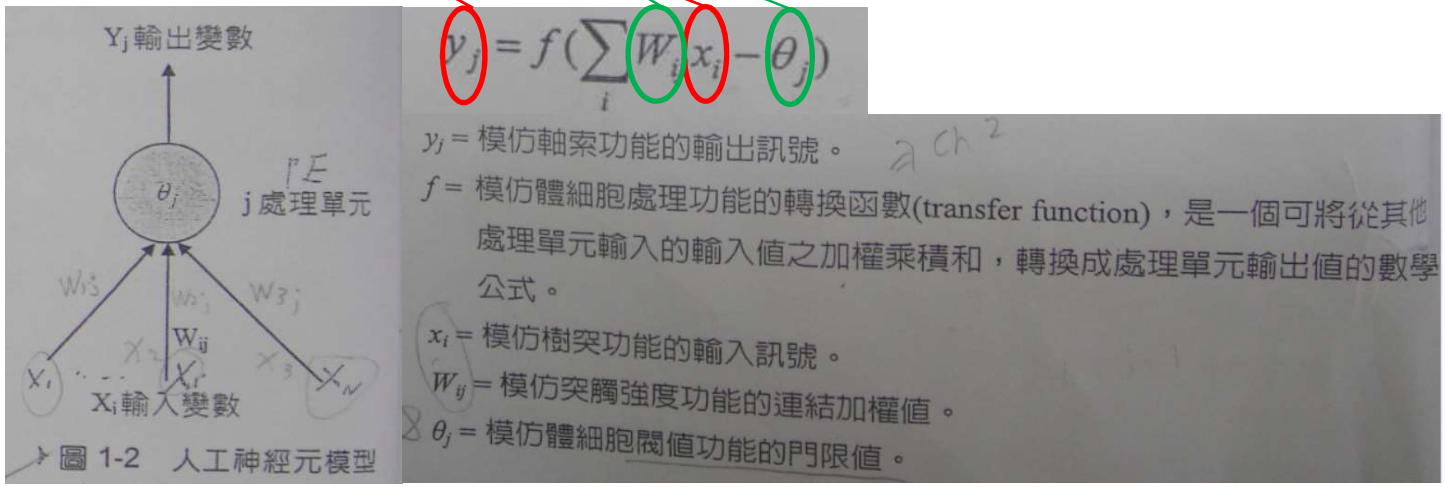
a) Detail of transfer function F:

Neural network

Z^p: p-th input pattern (ex: 28X28 grayscale image)
W: adjustable parameters(ex: global weight, bias)
Y^p: recognized class label of pattern Z^p(ex: the digit)
F: Transfer function

?? ??

$$Y^p = F(Z^p, W)$$



b) Learning problem illustration:

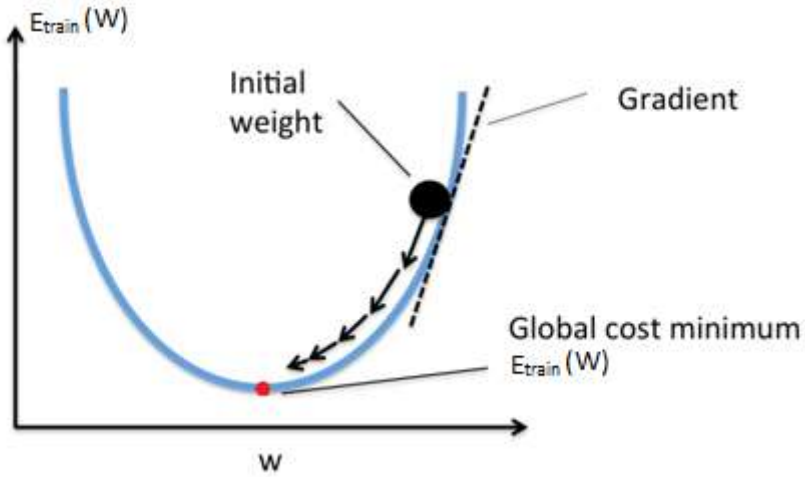


Fig. Learning problem: Find the W to minimize the $E_{train}(W)$

1. Introduction

1) Learning from data (3/3)

Object of slide: Introduce Etest – a system performance evaluation

The gap between test set and training set:

E_{test}: To evaluate the system

$$E_{test}(W) - E_{train}(W) = k(h/P)^{\alpha}$$

Disjoint from dataset

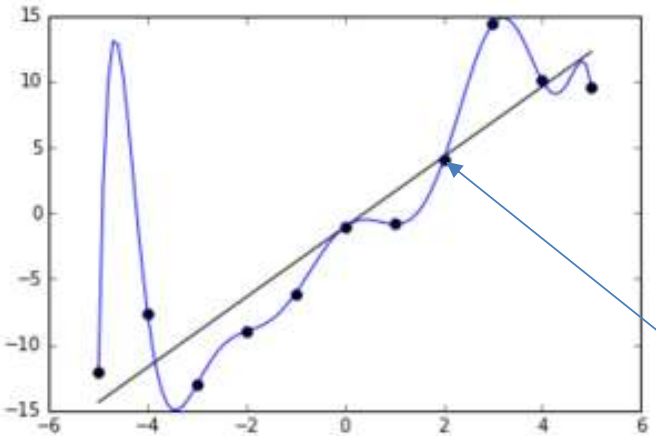
$E_{train}(W)$: the average errors E^p over the training set $\{(Z^1, D^1),\ldots(Z^p, D^p)\}$
$E_{test}(W)$: the average errors E^q over the test set $\{(Z^1, D^1),\ldots(Z^q, D^q)\}$
P: the number of training samples(ex: 60000 training samples)
?? h: complexity of machine(ex: hidden layer number, neuron number in one hidden layer)
α: number between 0.5 and 1.0
k: a constant

P: We try to minimize the gap between training data set error and test data set error.
If we try to minimize the training data set as small as possible, we may encounter the overfitting problem.
So there is a trade-off between minimizing training data set and avoiding the gap becoming bigger. (may encounter overfitting)

P↑, gap↓
h↑, E_{train}(W) ↓,gap↑
trade-off

P: L1 and L2 regularization(deep learning one day - weight decay)

Overfitting:



Training samples

$E_{train} \cong 0$ but gap is big (E_{test} is big)

Structural Risk Minimization:

$$(\text{Min})\ E_{train} + \beta H(W)$$

H(W): Regularization function
Beta: Constant

P: This is the graph which explain the overfitting problem. We hope to find the black line by training our model. The black points are the training set to represent the line but some of them have noise. If our model tries to fit these training set, we finally find the curve which perfectly fit these point. This curve is not the ideal solution because it lacks generalization.

1. Introduction

2) Gradient-Based Learning (1/3)

Minimize E(w): estimating the impact of W

?? ??

$$E(w) = \min \sum (D^P - F(W, Z^P))^2 + N(m = 0, \text{var})$$

Y' - Y

Gradient-Based Learning algorithms can use one of two classes of methods to update the parameters. The first method, dubbed "Batch Gradient", is the classical one: the gradients are accumulated over the entire training set, and the parameters are updated after the exact gradient has been so computed. In the second method, called "Stochastic Gradient", a partial, or noisy, gradient is evaluated on the basis of one single training sample (or a small number of samples), and the parameters are updated using this approximate gradient. The training samples can be selected randomly or according to a properly randomized sequence. In the stochastic version, the gradient estimates are noisy, but the parameters are updated much more often than with the batch version. An empirical result of considerable practical importance is that on tasks with large, redundant data sets, the stochastic version is considerably faster than the batch version, sometimes by orders of magnitude [117]. Although the reasons for this are not totally understood theoretically, an intuitive explanation can be found in the following extreme example. Let us take an example where the training database is composed of two copies of the same subset. Then accumulating the gradient over the whole set would cause redundant computations to be performed. On the other hand, running Stochastic Gradient once on this training set would amount to performing two complete

a) Common adjusting weights W formula:

Epsilon: weight for current k situation

K-1 previous situation

$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}.$$

W_k: the kth update of W

W: parameter(weight, bias)

ε: *scalar constan*

ϵ is a scalar constant. More sophisticated procedures use variable ϵ, or substitute it for a diagonal matrix, or substitute it for an estimate of the inverse Hessian matrix as in Newton or Quasi-Newton methods.

b) Minimization procedures:

1. Stochastic gradient algorithm (on-line update):

belong to

??

$$W_k = W_{k-1} - \epsilon \frac{\partial E^{P_k}(W)}{\partial W}$$

It consists in updating the parameter vector using a noisy, or approximated, version of the average gradient. In the most common instance of it, W is updated on the basis of a single sample:

Ex: 60000 training samples, 看完一個sample更新一次參數 Wk,
Stochastic: Random order for 60000, That is, each run, the input order will be different because of random. Why random, because different order, the result/performance will be different.

[Phu]: Random: To change the order of input data, in case running the model multiples times, the same order data on each epoch will lead to problem like finding local minima or slower convergence.

2. Batch gradient algorithm:

$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}.$$

Ex: 60000 training samples, 看完60000個sample在更新一次參數 Wk

3. Mini-batch gradient algorithm: Better for big data

Phu: Also need random dataset before training

Ex: 60000 training samples, 看完100個sample更新一次參數 Wk

disadvantage of SGD

- require much manual tuning of optimization parameters
  - Such as learning rate
  - If not know the task well, difficult to find a good learning rate
- difficult to parallelize them using GPUs
- may go far from the batch gradient
  - based on one random data point

Le, Quoc V., et al. "On optimization methods for deep learning." *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011.

1. Introduction

2) Gradient-Based Learning (2/3)

Object of slide: Particular example of mini-batch minimization procedure

- Batch size: the number of training instances used to compute the gradient of the parameters in each step.
- Iteration: Weight update step. When model have seen all data in a mini batch.
- Epoch: is a full turn when all data have been seen by the model.

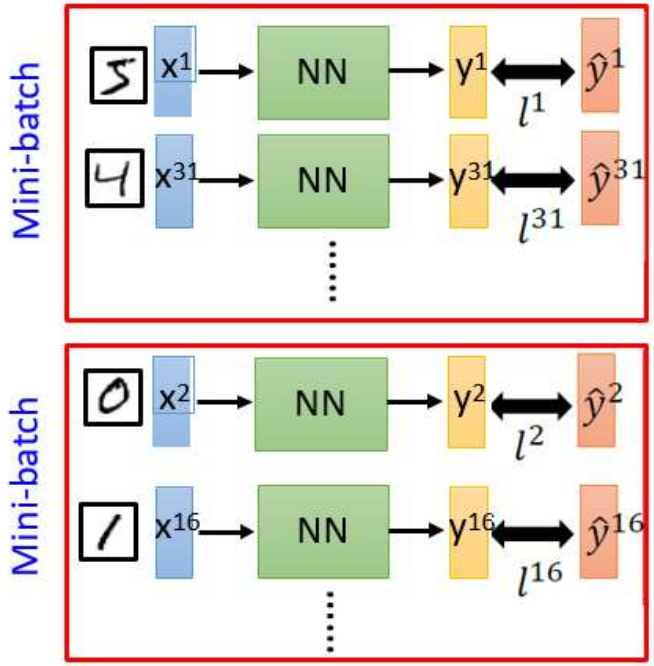
Assume: 60,000 training samples.
In 1 epoch:
- Batch-size: 100 samples
- Iteration: 600

We do not really minimize total loss!

Mini-batch

100 sample (batch size)

100 sample (batch size)



- Randomly initialize network parameters
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once
  ⋮
- Until all mini-batches have been picked

1 iteration

1 iteration

one epoch

Repeat the above process

※擷取至一天搞懂Deep Learning

1. Introduction

2) Gradient-Based Learning (3/3)

Object of slide: effect of learning rate on the optimization process.

補充: $W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}$.

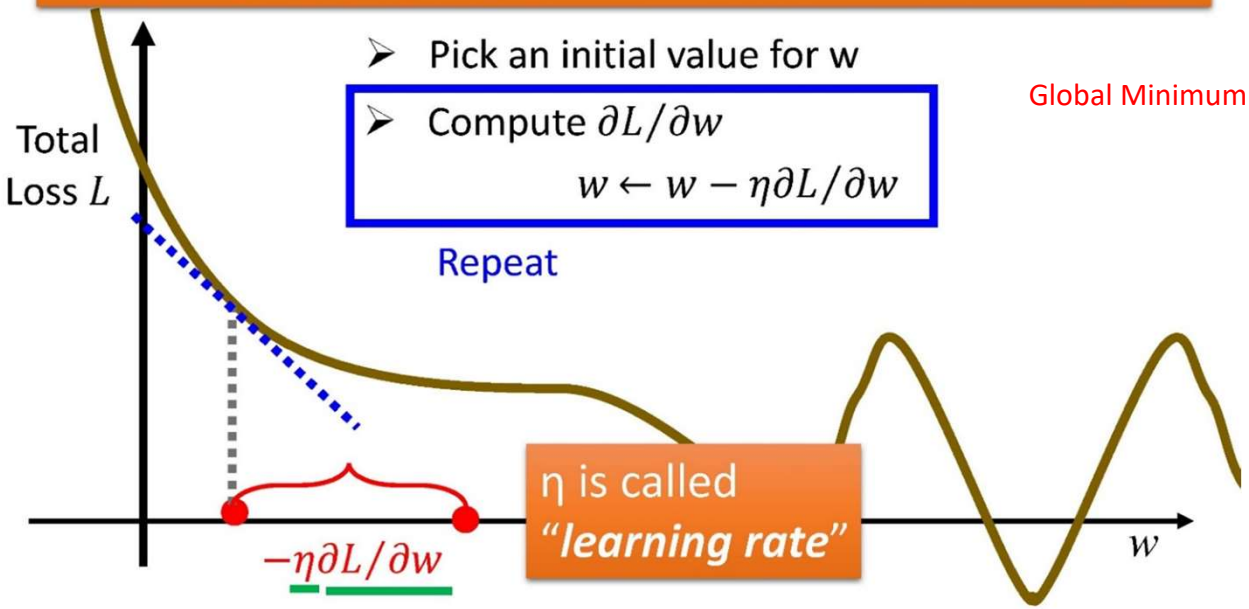
Previously: scala (fixed)

Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Learning Target

Find the network parameters θ^* that minimize total loss



- Pick an initial value for w
- Compute $\partial L/\partial w$

  $w \leftarrow w - \eta\partial L/\partial w$

Repeat

η is called "*learning rate*"

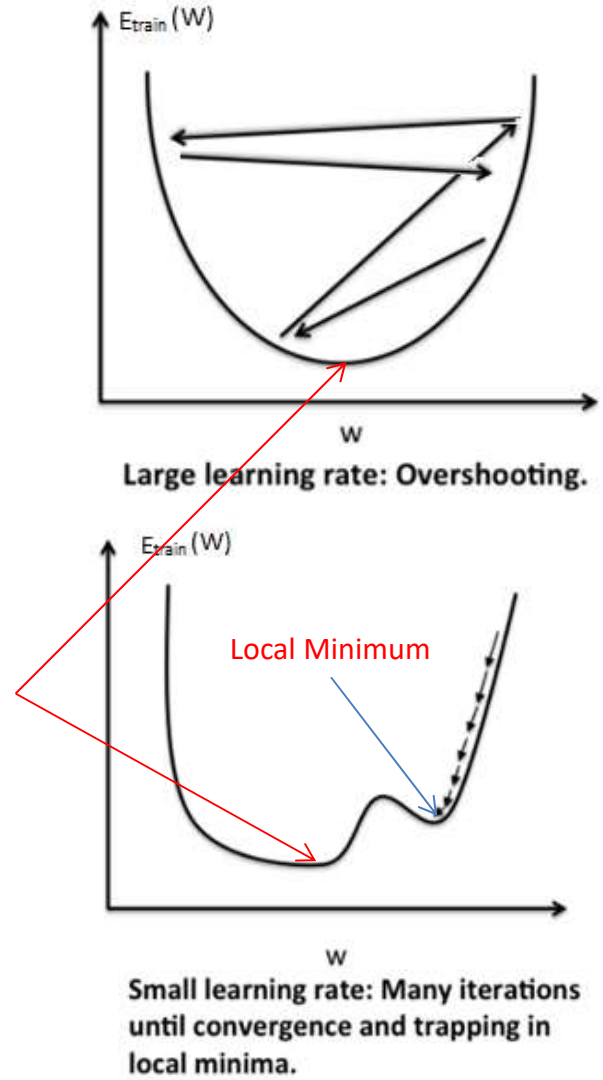


Fig: Some problems of the learning rate

Phu: Sol: change the learning rate after several iterations (decay)

※擷取至一天搞懂Deep Learning

1. Introduction

3) Gradient Back-Propagation

Gradient-Based Learning procedures have been used since the late 1950's, but they were mostly limited to linear systems [1]. The surprising usefulness of such simple gradient descent techniques for complex machine learning tasks was not widely realized until the following three events occurred. The first event was the realization that, despite early warnings to the contrary [12], the presence of local minima in the loss function does not seem to be a major problem in practice. This became apparent when it was noticed that local minima did not seem to be a major impediment to the success of early non-linear gradient-based Learning techniques such as Boltzmann machines [13], [14]. The second event was the popularization by Rumelhart, Hinton and Williams [15] and others of a simple and efficient procedure, the back-propagation algorithm, to compute the gradient in a non-linear system composed of several layers of processing. The third event was the demonstration that the back-propagation procedure applied to multi-layer neural networks with sigmoidal units can solve complicated learning tasks. The basic idea of back-propagation is that gradients can be computed efficiently by propagation from the output to the input. This idea was described in the control theory literature of the early sixties [16], but its application to machine learning was not generally realized then.

Three events to realize the usefulness of Gradient Descent techniques:

1. Local minima acceptance.
2. Popularization of Back Propagation algorithm.
3. Multi-layer neural network using BP can solve complicated problem .

Basic idea of Back-propagation: Gradients can be computed efficiently by Back propagation from output to input.

1. Introduction

4) Learning in Real Handwriting Recognition System

One of the most difficult problems in handwriting recognition, however, is not only to recognize individual characters, but also to separate out characters from their neighbors within the word or sentence, a process known as segmentation. The technique for doing this that has become the "standard" is called *Heuristic Over-Segmentation*. It consists in generating a large number of potential cuts between characters using heuristic image processing techniques, and subsequently selecting the best combination of cuts based on scores given for each candidate character by the recognizer. In such a model, the accuracy of the system depends upon the quality of the cuts generated by the heuristics, and on the ability of the recognizer to distinguish correctly segmented characters from pieces of characters, multiple characters, or otherwise incorrectly segmented characters. Training a recognizer to perform this task poses a major challenge because of the difficulty in creating a labeled database of incorrectly segmented characters. The simplest solution consists in running the images of character strings through the segmenter, and then manually labeling all the character hypotheses. Unfortunately, not only is this an extremely tedious and costly task, it is also difficult to do the labeling consistently. For example, should the right half of a cut up 4 be labeled as a 1 or as a non-character? should the right half of a cut up 8 be labeled as a 3?

Object of slide: Introduce the real Hand-writing recognition system.

- In recognizing individual character:
  - Neural network trained with Gradient Based learning performs better than others.
  - CNN is the best model.
- In multiple character recognition:
  - Separate out characters from neighbors.

  => **Heuristic Over-Segmentation method**

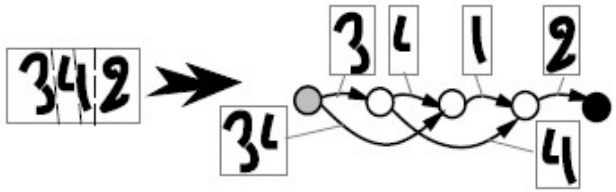
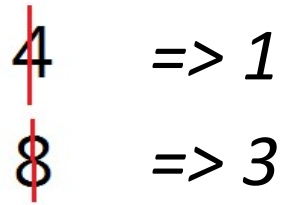


Fig. 16. Building a segmentation graph with Heuristic Over-Segmentation.

Problem:



=> 1 or nothing

=> 3

1. Introduction

5) Globally Trainable Systems(1/3)

As stated earlier, most practical pattern recognition systems are composed of multiple modules. For example, a document recognition system is composed of a field locator, which extracts regions of interest, a field segmenter, which cuts the input image into images of candidate characters, a recognizer, which classifies and scores each candidate character, and a contextual post-processor, generally based on a stochastic grammar, which selects the best grammatically correct answer from the hypotheses generated by the recognizer. In most cases, the information carried from module to module is best represented as graphs with numerical information attached to the arcs. For example, the output of the recognizer module can be represented as an acyclic graph where each arc contains the label and the score of a candidate character, and where each path represent a alternative interpretation of the input string. Typically, each module is manually optimized, or sometimes trained, outside of its context. For example, the character recognizer would be trained on labeled images of pre-segmented characters. Then the complete system is assembled, and a subset of the parameters of the modules is manually adjusted to maximize the overall performance. This last step is extremely tedious, time-consuming, and almost certainly suboptimal.

Traditional method:

- Most practical system composed of multiple modules:
  - Locator.
  - Segmenter.
  - Recognizer.
  - Contextual post-processor.
- Each module is manually optimized (or trained).

Example:

Training character recognizer system.

1. The system is trained on labeled images of pre-segmented characters.
2. Complete system is assembled, and a subset of the parameters of the modules is manually adjusted to maximize the overall performance.

Drawback:

- Extremely tedious
- Time-consuming
- Suboptimal.

Method:

- Train the entire system so as to minimize the global error measure.

1. Introduction

5) Globally Trainable Systems(2/3)

Object: Find a good minimum of the global loss function with respect to all parameters in the system.

$\frac{\partial E^p}{\partial W_n}$ $\frac{\partial E^p}{\partial X_{n-1}}$

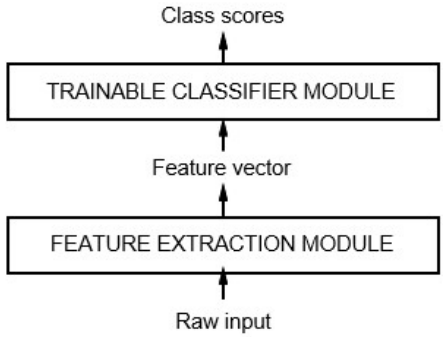


Fig. Document recognition system

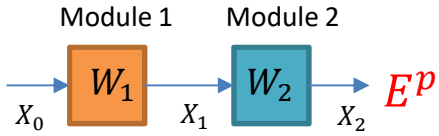


Fig. System with cascade module

Global loss function:

※$\min E^p = \mathrm{D}(D^p, \mathrm{F}(Z^p, W))$

Phu: If the loss function E measuring the performance can be made differentiable with respect to the system's tunable parameter W => We can **find a local minimum of E using Gradient-Based Learning**.

$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}.$$

Let consider system with cascade modules

which implements a function $X_n = F_n(W_n, X_{n-1})$, where X_n is a vector representing the output of the module, W_n is the vector of tunable parameters in the module (a subset of W), and X_{n-1} is the module's input vector (as well as the previous module's output vector). The input X_0 to the first module is the input pattern Z^p. If the partial derivative of E^p with respect to X_n is known, then the partial derivatives of E^p with respect to W_n and X_{n-1} can be computed using the backward recurrence

Back Propagation

$$\frac{\partial E^p}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1}) \frac{\partial E^p}{\partial X_n}$$

Chain rule

$$\frac{\partial E^p}{\partial X_{n-1}} = \frac{\partial F}{\partial X}(W_n, X_{n-1}) \frac{\partial E^p}{\partial X_n}$$

Known $\frac{\partial E^p}{\partial X_n}$

where $\frac{\partial F}{\partial W}(W_n, X_{n-1})$ is the Jacobian of F with respect to W evaluated at the point (W_n, X_{n-1}), and $\frac{\partial F}{\partial X}(W_n, X_{n-1})$ is the Jacobian of F with respect to X. The Jacobian of a vector function is a matrix containing the partial derivatives of all the outputs with respect to all the inputs. The first equation computes some terms of the gradient of $E^p(W)$, while the second equation generates a backward recurrence, as in the well-known back-propagation procedure for neural networks.

Multi-Layer

$$X_n = F_n(W_n, X_{n-1})$$

n: the module number
X_n: the output of the module
W_n: tunable parameters in the module
X_{n-1}: module's input vector
F_n: Function to calculate **X_n**

X_0: 1st module input pattern Z^p

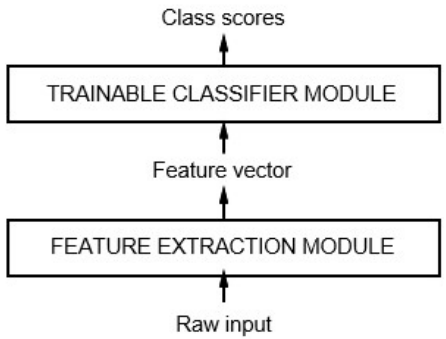
Jacobian matrices

$$\frac{\partial F}{\partial W} = \begin{bmatrix} \frac{\partial F_1}{\partial W_1} & \cdots & \frac{\partial F_1}{\partial W_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial W_1} & \cdots & \frac{\partial F_n}{\partial W_n} \end{bmatrix}$$

$$\frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial F_1}{\partial X_1} & \cdots & \frac{\partial F_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial X_1} & \cdots & \frac{\partial F_n}{\partial X_n} \end{bmatrix}$$

1. Introduction

5) Globally Trainable Systems(3/3)



Input: X_0

Outputs: X=$\{X_1, X_2\}$

Weights: W=$\{W_1, W_2\}$

Functions: F=$\{F_1, F_2\}$

$$X_n = F_n(W_n, X_{n-1})$$

$$E^p = \mathrm{D}(D^p, \mathrm{F}(Z^p, W))$$

n: the module number
$\mathbf{X_n}$: the output of the module
$\mathbf{W_n}$: tunable parameters in the module
$\mathbf{X_{n-1}}$: module's input vector
$\mathbf{F_n}$: Function to calculate $\mathbf{X_n}$

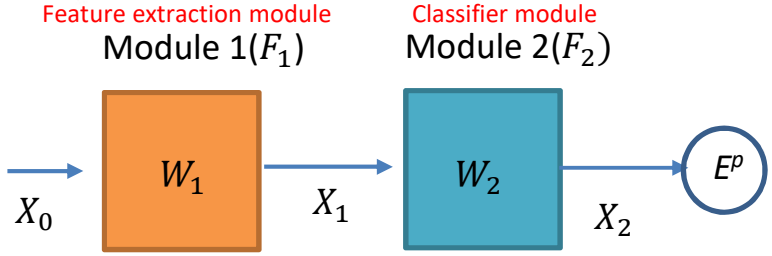
X_0: 1st module input pattern Z^p

Backward Recurrence

$$\frac{\partial E^p}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1})\frac{\partial E^p}{\partial X_n}$$

Chain rule

$$\frac{\partial E^p}{\partial X_{n-1}} = \frac{\partial F}{\partial X}(W_n, X_{n-1})\frac{\partial E^p}{\partial X_n}$$



$$X_0 = Z^p \qquad X_1 = F(W_1, X_0) \qquad X_2 = F(W_2, X_1)$$

$$E^p = \mathrm{D}(D^p, X_2) = \mathrm{D}(D^p, \mathrm{F}(W_2, X_1))$$

Known $\frac{\partial E^p}{\partial X_n}$

Jacobian:

$$\frac{\partial F}{\partial W} = \begin{bmatrix} \frac{\partial F_1}{\partial W_1} & \frac{\partial F_1}{\partial W_2} \\ \frac{\partial F_2}{\partial W_1} & \frac{\partial F_2}{\partial W_2} \end{bmatrix}$$

$$\frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial F_1}{\partial X_1} & \frac{\partial F_1}{\partial X_2} \\ \frac{\partial F_2}{\partial X_1} & \frac{\partial F_2}{\partial X_2} \end{bmatrix}$$

By chain rule

??

$$\frac{\partial E^p}{\partial W_2} = \frac{\partial F}{\partial W}(W_2, X_1)\boxed{\frac{\partial E^p}{\partial X_2}}$$

$$\boxed{\frac{\partial E^p}{\partial X_2}} = \frac{\partial D(D^p, X_2)}{\partial X_2}$$

??

$$\frac{\partial E^p}{\partial W_1} = \frac{\partial F}{\partial W}(W_1, X_0)\boxed{\frac{\partial E^p}{\partial X_1}}$$

$$\boxed{\frac{\partial E^p}{\partial X_1}} = \frac{\partial F}{\partial X}(W_2, X_1)\frac{\partial E^p}{\partial X_2}$$

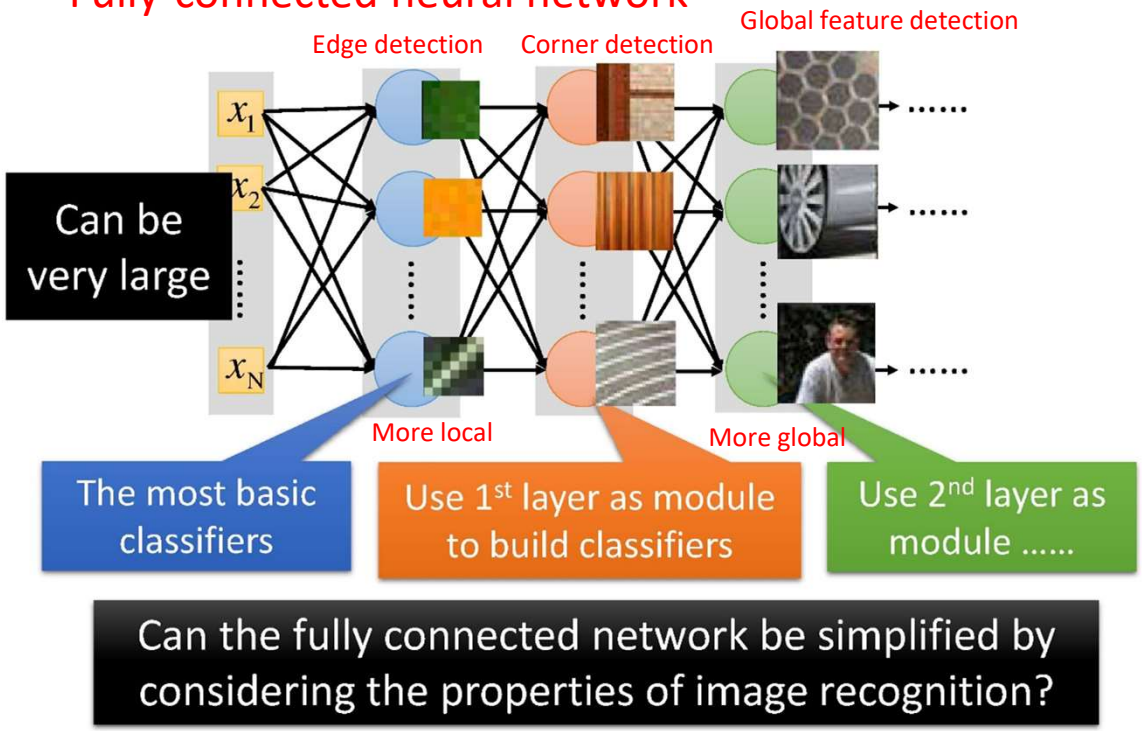
Update weight:

$$W_k = W_{k-1} - \epsilon\frac{\partial E(W)}{\partial W}.$$

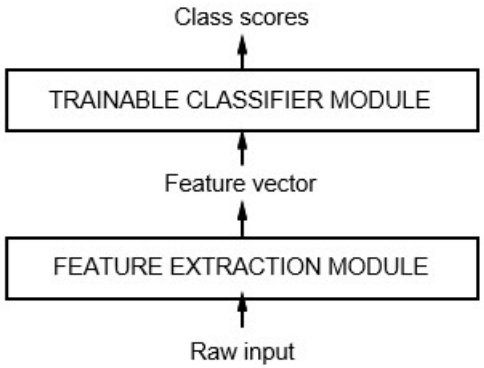
2. Convolutional neural networks for isolated character recognition ??

1) Convolutional Networks(1/5)

Fully-connected neural network



※擷取至一天搞懂Deep Learning



Problems of fully-connected NN:

1. Require large memory
2. No built-in invariant with respects to translations (such as R, T.)
3. Only work with raw data.

Why need convolutional neural network?

a) Local receptive fields
b) Shared weight
c) Sub-sampling

2. Convolutional neural networks for isolated character recognition ??

1) Convolutional Networks(2/5)

Why need convolutional neural network (reduce para.)??

a) Local receptive fields

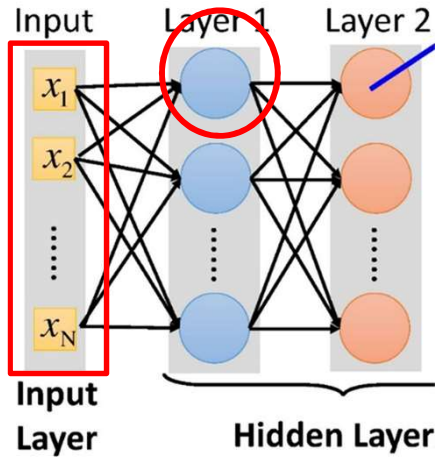
b) Shared weight

c) Sub-sampling

P:not need global(whole image and fully connect), only see the local field

Local receptive field: Reduce layer nodes from full-connection to local connection

Fully-connected neural network

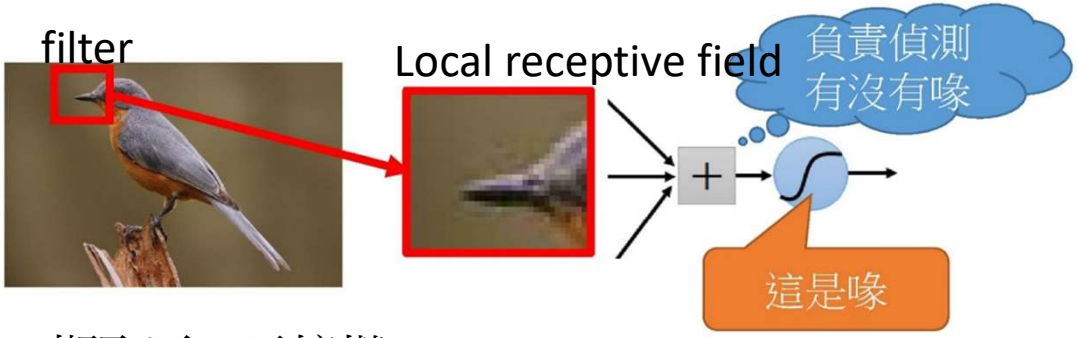


Local receptive field: Reduce layer nodes from full-connection to local connection

- Some patterns are much smaller than the whole image

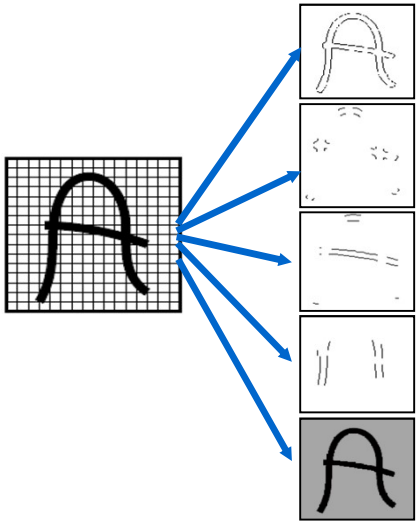
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



※擷取至一天搞懂Deep Learning

features



2. Convolutional neural networks for isolated character recognition ??

1) Convolutional Networks(3/5)

Why need convolutional neural network?

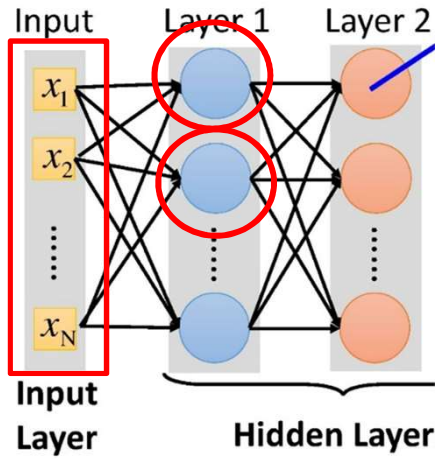
a) Local receptive fields

b) Shared weight

c) Sub-sampling

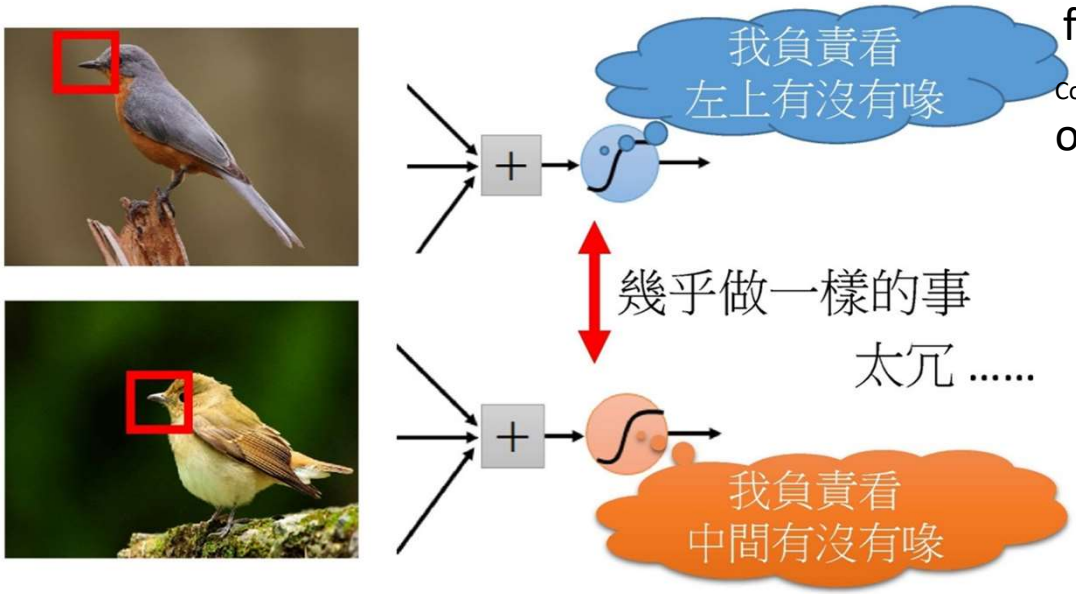
※weight就是filter的值
weight share the same filter value
-Exp. Red line has the same final coeff. w=1.0: During BP learning or updating coeff., each red line has its own coeff. w, after updating, the final w will be the average value of 3 red line w. So final red line coeff w will be the same. So the para number can be reduced.

Fully-connected neural network



Shared weight: Reduce parameters (weight) number from 6 parameters (weights) to 2 weights

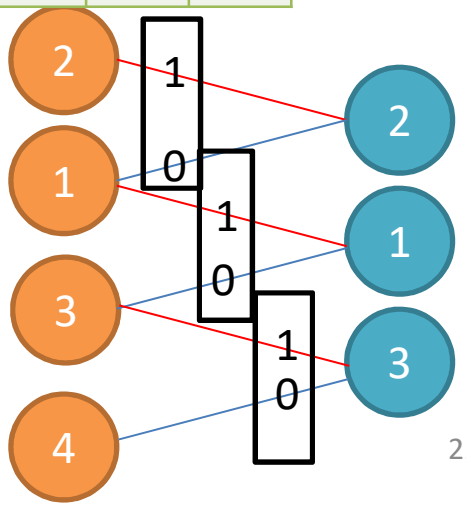
- The same patterns appear in different regions.



EX

input	2	1	3	4
filter	1	0		
Convolution output	2	1	3	

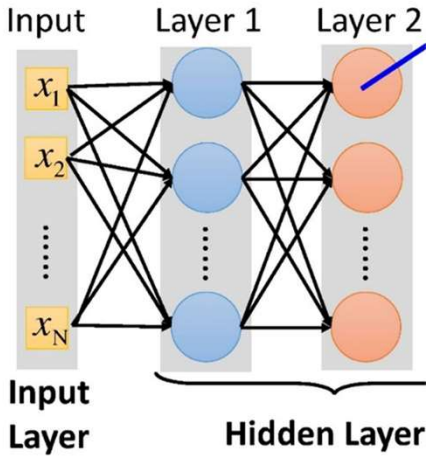
Shift (position)



※擷取至一天搞懂Deep Learning

2. Convolutional neural networks for isolated character recognition ??

Fully-connected neural network



1) Convolutional Networks(4/5)

Why need convolutional neural network?

a) Local receptive fields

b) Shared weight

c) Sub-sampling

- Subsampling the pixels will not change the object

bird

subsampling

bird

We can subsample the pixels to make image smaller

➡ Less parameters for the network to process the image

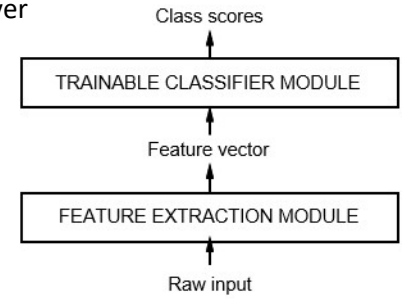
※擷取至一天搞懂Deep Learning

2. Convolutional neural networks for isolated character recognition ??

156 parameters: (5x5)x6=150, 6 bias

C_x: convolution layer
S_x: sub-sampling layer
F_x: fully-connected layer

x is the layer index



1) Convolutional Networks(5/5)

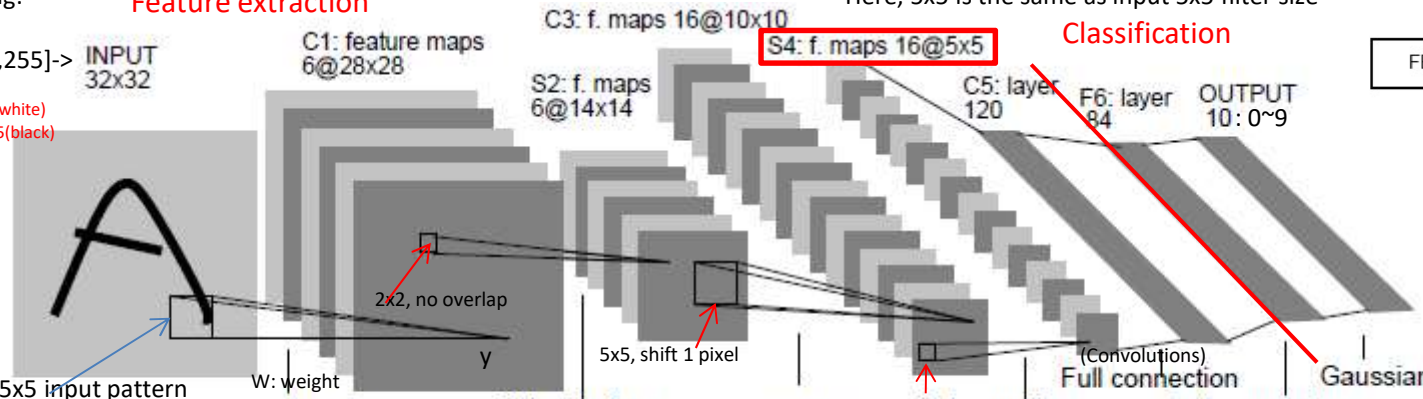
Preprocessing: normalize grayvalue [0,255]-> N(m=0,v=1.0)
Background: -0.1(white)
Foreground: 1.175(black)

Convolution/ shift pixel by pixel from left-top to right-bottom

Feature extraction

Here, 5x5 is the same as input 5x5 filter size

Classification



※Convolution filter (C1,C3,C5) size is 5X5

Shared weight

Zp=5x5 input pattern

Filter: 5x5 window size

6 filters (coeff??)

Why 3 layers here??

Affine transformation

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Convolutional Networks combine three architectural ideas to ensure some degree of shift, scale, and distortion invariance: *local receptive fields*, *shared weights (or weight replication)*, and spatial or temporal *sub-sampling*. A typical convolutional network for recognizing characters, dubbed LeNet-5, is shown in figure 2.

in the previous layer. The idea of connecting units to local receptive fields on the input goes back to the Perceptron in the early 60s, and was almost simultaneous with Hubel and

own feature extractor. The weight sharing technique has the interesting side effect of reducing the number of free parameters, thereby reducing the "capacity" of the machine and reducing the gap between test error and training error [34]. The network in figure 2 contains 340,908 con-

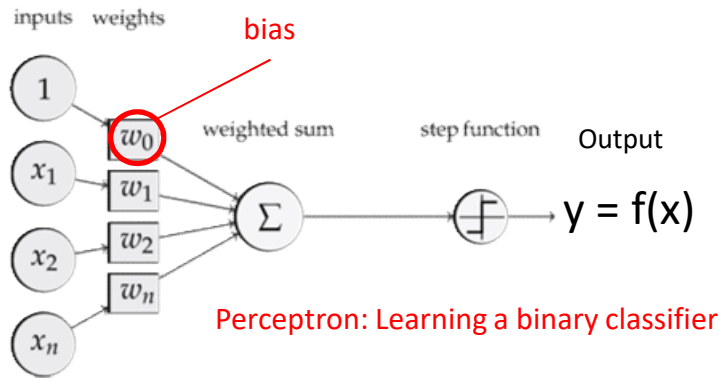
Once a feature has been detected, its exact location becomes less important. Only its approximate position relative to other features is relevant. For example, once

tive features are encoded in a feature map is to reduce the spatial resolution of the feature map. This can be achieved with a so-called *sub-sampling layers* which performs a local

2. Convolutional neural networks for isolated character recognition

1.1) Perceptron

In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers:
functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another.



Perceptron: Learning a binary classifier

Definition:

Result of f(x)

Threshold

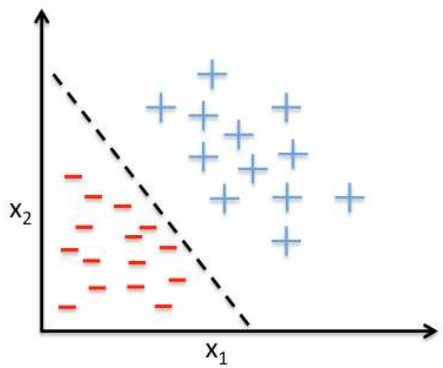
$$y = f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

x: Each pixel at input image

f(x):1 – foreground pixel
0 – background pixel

where w is a vector of real-valued weights, $w \cdot x$ is the dot product $\sum_{i=0}^{m} w_i x_i$,

where m is the number of inputs to the perceptron and b is the *bias*.



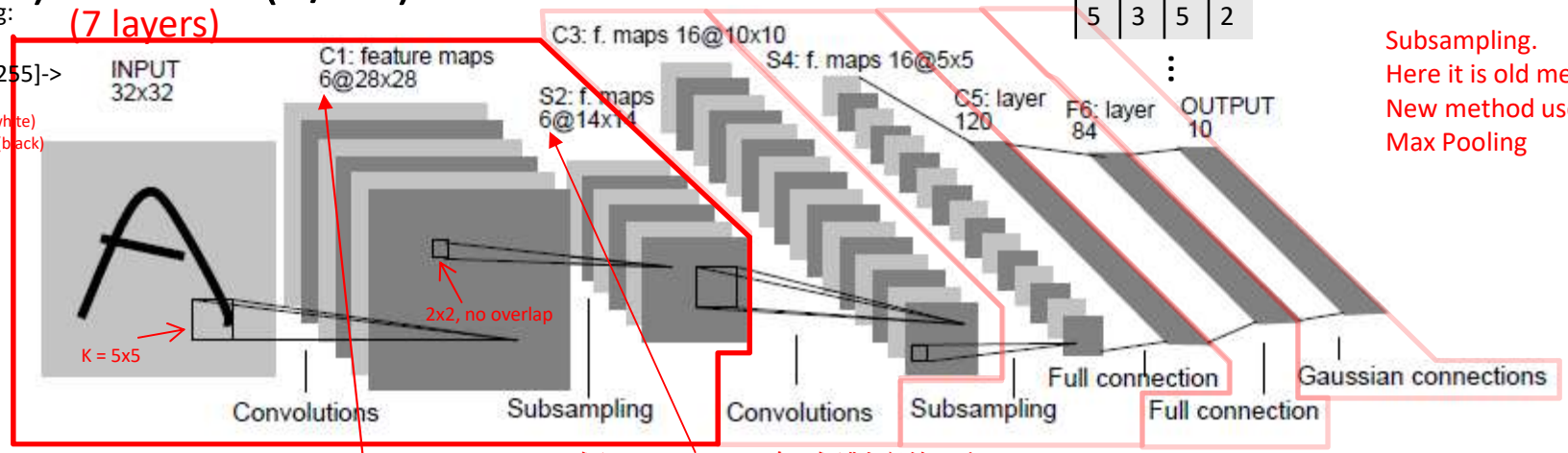
Example of a linear decision boundary for binary classification.

2. Convolutional neural networks for isolated character recognition ??

2) LeNet-5(1/18)

Preprocessing:
normalize
grayvalue [0,255]->
~~N(m=0,v=1.0)~~
Background: -0.1(white)
Foreground: 1.175(black)

(7 layers)



6個filter(5*5)→產生6個feature map

1個feature map 有1個對應的w和b

Here, 5x5 filter parameters (value w) are learned to have them

Convolution layer is used for extracting features.

Layer C1 is a convolutional layer with 6 feature maps. Each unit in each feature map is connected to a 5x5 neighborhood in the input. The size of the feature maps is 28x28 which prevents connection from the input from falling off the boundary. C1 contains 156 trainable parameters, and 122,304 connections.

(5*5+1)*6

(28*28)*(5*5+1)*6

Convolution
activation function
(squashing function)

(*) If is was fully connected network
we had: (32*32+1)*(28*28)*6
= **4,821,600** connections.
(CNN reduces parameters)

28

5	1	1	7
3	4	0	6
3	4	0	5
5	3	5	2

28

…

Trainable parameter

$w_1 = 0.9\,, b_1 = 0.5$

Sigmoid((5+1+3+4)*0.9+0.5)

Subsampling.
Here it is old method.
New method uses
Max Pooling

Pooling layer is used for selecting features.

Layer S2 is a sub-sampling layer with 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The four inputs to a unit in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias. The result is passed through a sigmoidal function. The 2x2 receptive fields are non-overlapping, therefore feature maps in S2 have half the number of rows and column as feature maps in C1. Layer S2 has 12 trainable parameters and 5,880 connections.

(w:6, b:6) 6+6

(14*14)*(2*2+1)*6

Stride is 2

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(2/18): Convolutional layer

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

1x1 + 1x0 + 1x1 +
0x0 + 1x1 + 1x0 +
0x1 + 0x0 + 1x1
= 4

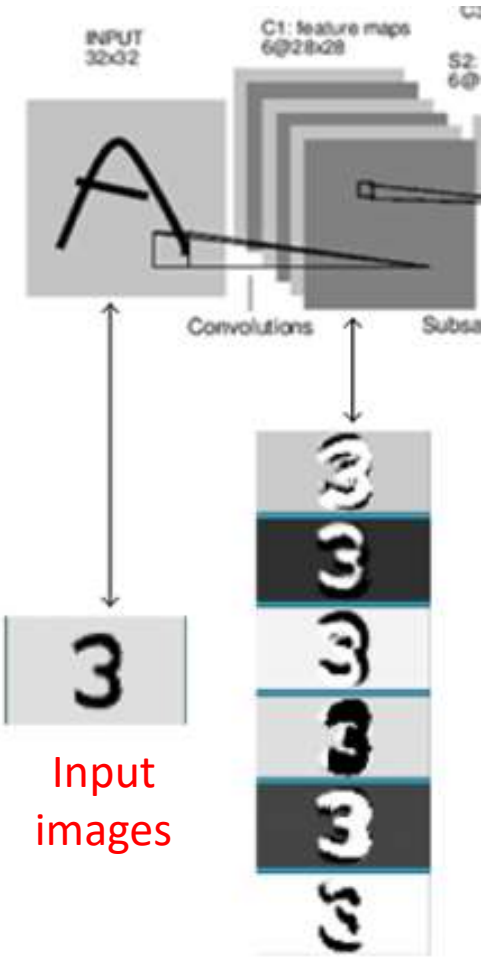
1	1	1	0	0
0	1	1	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
$0_{\times 0}$	$0_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0

Image

4	3	4
2	4	3
2		

Convolved Feature

0x1 + 0x0 + 1x1 +
0x0 + 0x1 + 1x0 +
0x1 + 1x0 + 1x1
= 2



Input images

6 Feature maps

How convolutional layer works and its result

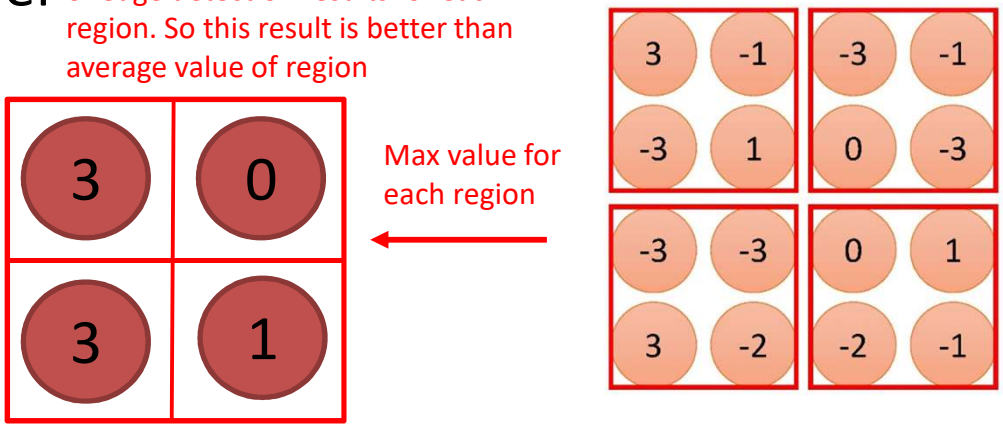
2. Convolutional neural networks for isolated character recognition ??

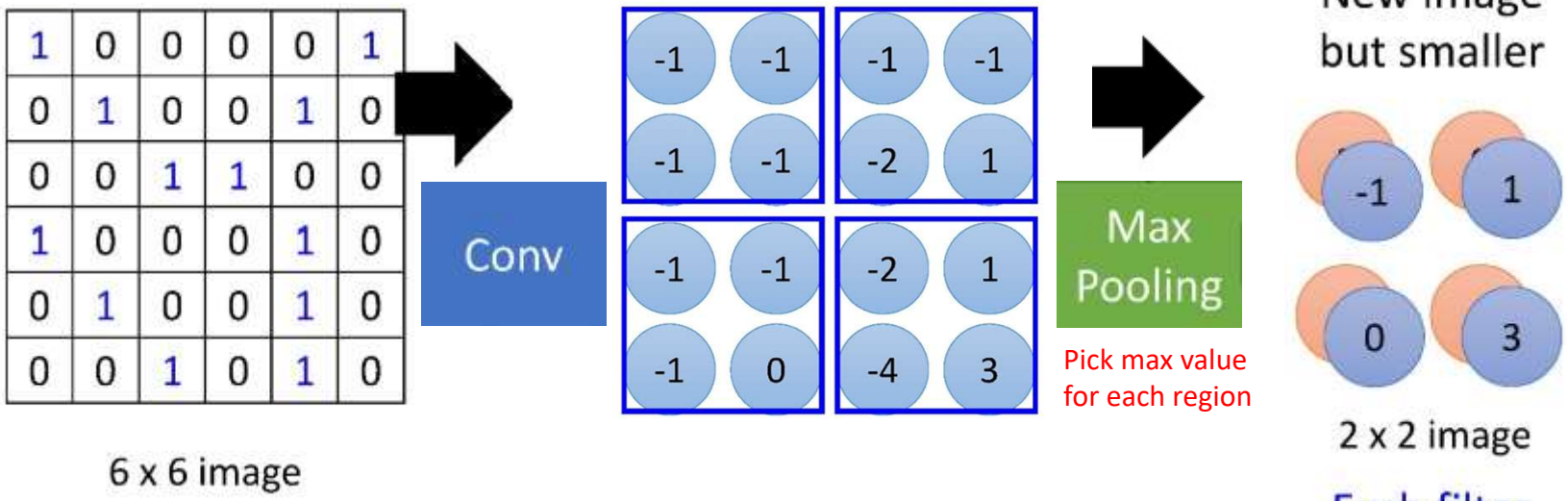
2) LeNet-5(3/18): Pooling layer

Max pooling

Feature map: Pooling ?

J: Max pooling is like max magnitude of edge detection results for each region. So this result is better than average value of region



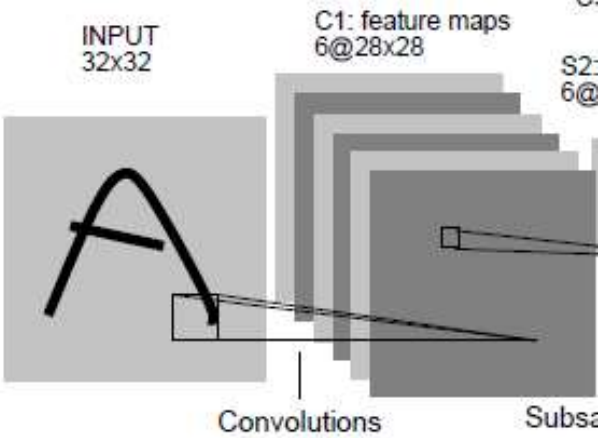


※擷取至一天搞懂Deep Learning

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(4/18)

補充:



32X32 → 28X28

$$O = \frac{(W - K + 2P)}{S} + 1$$

O: output length
W: input length(ex: 32)
K: filter length(ex: 5)
P: padding(ex: 0)
S: stride(ex: 1)

Layer C1:
Input:32X32
Filter:5X5
Stride:1
Padding:0

$$\frac{(32 - 5 + 2 * 0)}{1} + 1 = 28$$

No padding:
Input: 6x6
Filter: 3x3
Stride: 1
Padding: 0

$$\frac{(6 - 3 + 2 * 0)}{1} + 1 = 4$$

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

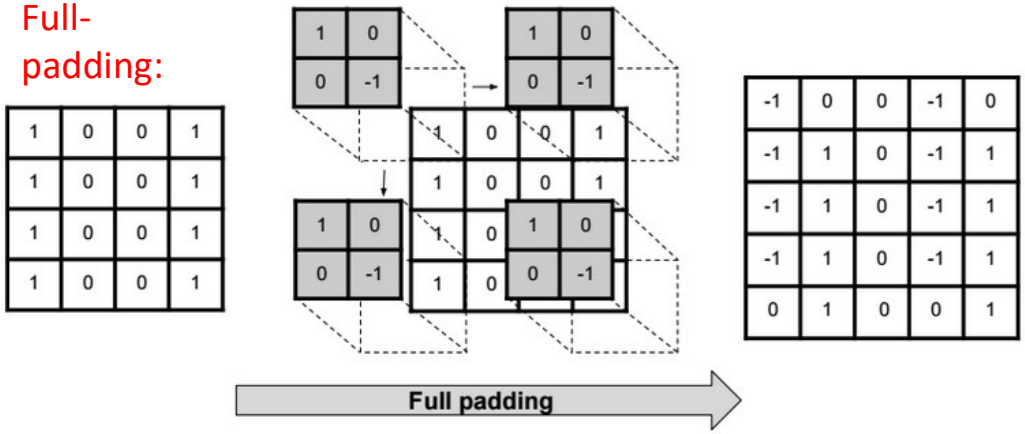
Zero-padding:
Input: 6X6
Filter: 3X3
Stride: 1
Padding: 1

$$\frac{(6 - 3 + 2 * 1)}{1} + 1 = 6$$

0	0	0				
0	1	0	0	0	0	1
0	0	1	0	0	1	0
	0	0	1	1	0	0
	1	0	0	0	1	0
	0	1	0	0	1	0
	0	0	1	0	1	0
					0	0
					0	0
					0	0

6 x 6 image

Full-padding:



※擷取至一天搞懂Deep Learning

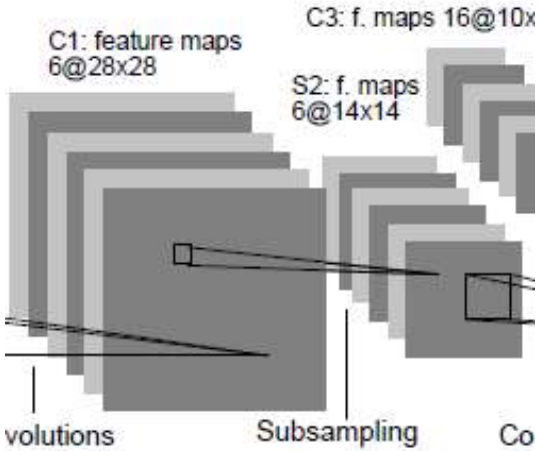
[Reference] http://sqlml.azurewebsites.net/2017/09/12/convolutional-neural-network/

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(5/18)

補充:

??28X28 → 14X14



$$O = \frac{(W - K + 2P)}{S} + 1$$

O: output length
W: input length(ex: 28)
K: filter size(ex: 2)
P: padding(ex: 0)
S: stride(ex:2)

Sub-sampling: 4 in a group

S2:
Input:28X28
Filter:2X2
Stride:2
Padding:0

1	1
1	1

Non overlapping

$$\frac{(28 - 2 + 2 * 0)}{2} + 1$$
$$= 14$$

2. Convolutional neural networks for isolated character recognition ??

2) LeNet-5(6/18)

補充：convolution simple example

Input:3X3

2	4	5
1	0	2
3	2	4

Filter:2X2

1	0
0	1

Bias

-2

Output:2X2

0	4
1	2

Stride =1

Padding = 0

Output length

$$\frac{(3-2+2*0)}{1}+1=2$$

Output:

(2*1+4*0+1*0+1*1) + (-2) = 0

(4*1+5*0+0*0+2*1) + (-2) = 4

(1*1+0*0+3*0+2*1) + (-2) = 1

(0*1+2*0+2*0+4*1) + (-2) = 2

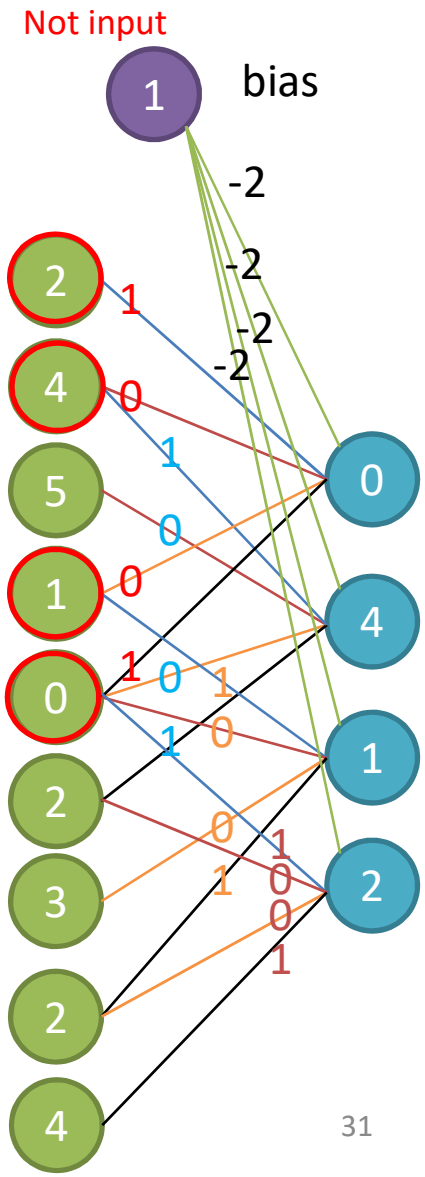
Contains 5 trainable parameters, and 20 connections

↓ (2*2+1)*1

↓ (2*2+1)*(2*2)*1

*fully connected, there would be (3*3+1)*(2*2)*1= 40 connections.

※The same color line represent the shared weights

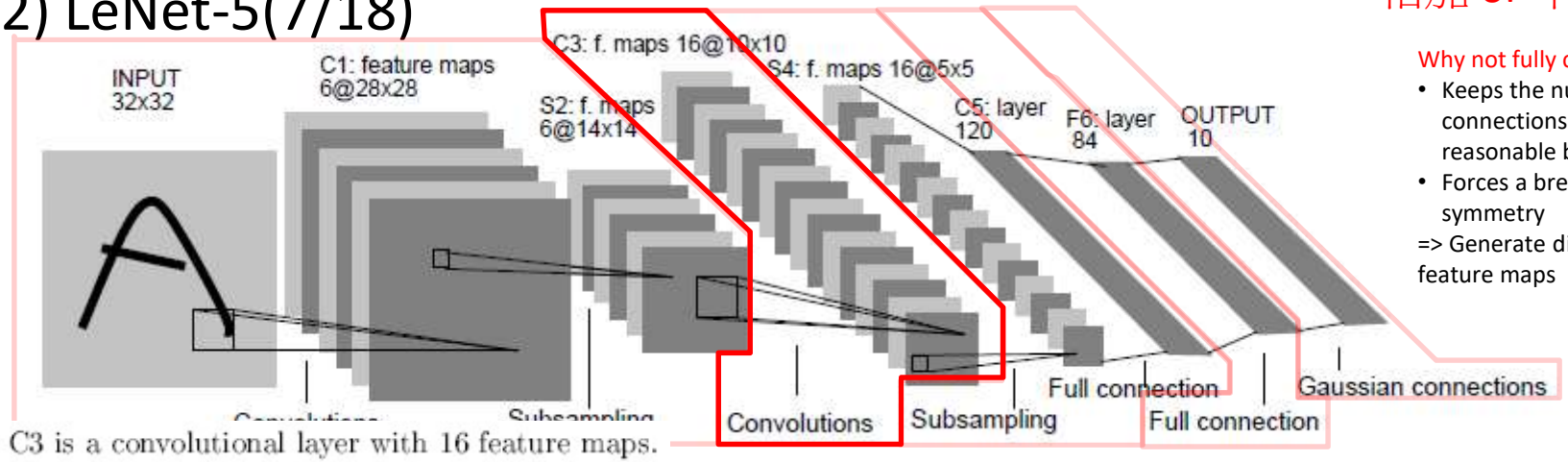


2. Convolutional neural networks for isolated character recognition

60個filter(5*5)→依下方表格方式combine
→產生16個feature map

??Combine:
相加 or 平均

2) LeNet-5(7/18)



Why not fully connect:
- Keeps the number of connections within reasonable bounds
- Forces a break of symmetry

=> Generate different feature maps

Layer C3 is a convolutional layer with 16 feature maps. Each unit in each feature map is connected to several 5x5 neighborhoods at identical locations in a subset of S2's feature maps. Table I shows the set of S2 feature maps combined by each C3 feature map. Why not connect every S2 feature map to every C3 feature map? The reason is twofold. First, a non-complete connection scheme keeps the number of connections within reasonable bounds. More importantly, it forces a break of symmetry in the network. Different feature maps are forced to extract different (hopefully complementary) features because they get different sets of inputs. The rationale behind the connection scheme in table I is the following. The first six C3 feature maps take inputs from every contiguous subsets of three feature maps in S2. The next six take input from every contiguous subset of four. The next three take input from some discontinuous subsets of four. Finally the last one takes input from all S2 feature maps. Layer C3 has 1,516 trainable parameters and 151,600 connections.

C3 feature maps combine with some sets of S2 feature map
→將第一層的一些feature 組成更複雜的feature
(ex: semicircle = curve + straight edge)

C3 feature maps Cross or temporal domain

S2 feature maps

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

3個連續為一組,
But 3 個 filters
(5x5)x3

4個連續為一組

4個不連續為一組

6個為一組

60個filter

→ (5*5)*(3*6+4*6+4*3+6*1)+16 = 1516

→ (1516)*(10*10) = 151600

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(8/18)

O/P C3 feature maps

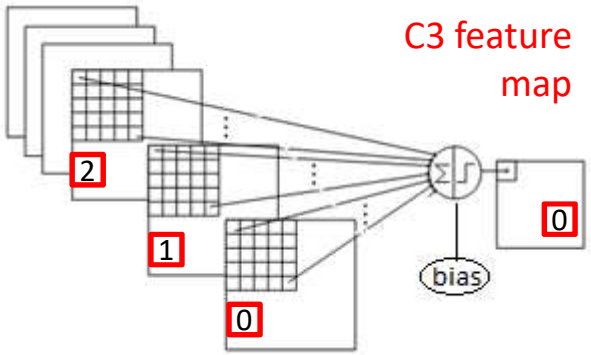
I/P S2 feature maps

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

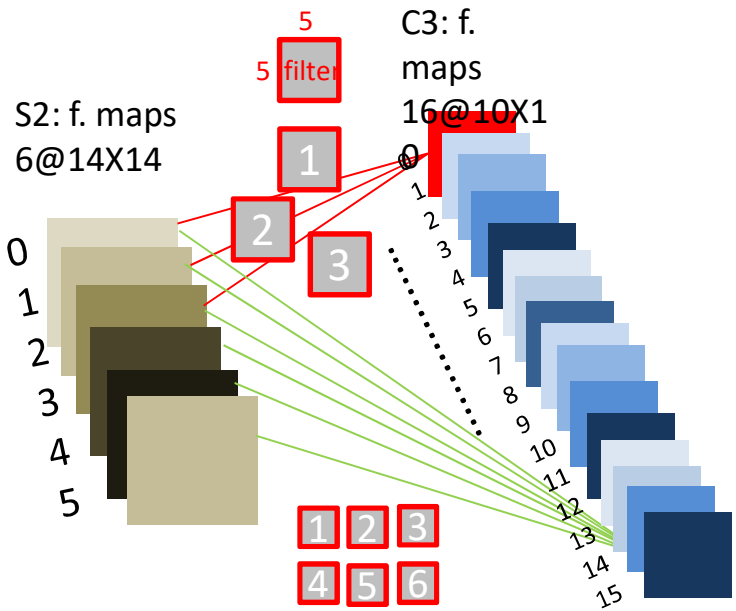
Contiguous subsets of 3 feature maps | Contiguous subsets of 4 feature maps | Dis-contiguous subsets of 4 feature maps | All feature maps

S2 feature maps

C3 feature map



Demonstration of how S2 is connect to C3



(5*5)*(3*6+4*6+4*3+6)+16
=25*(60)+16
↑

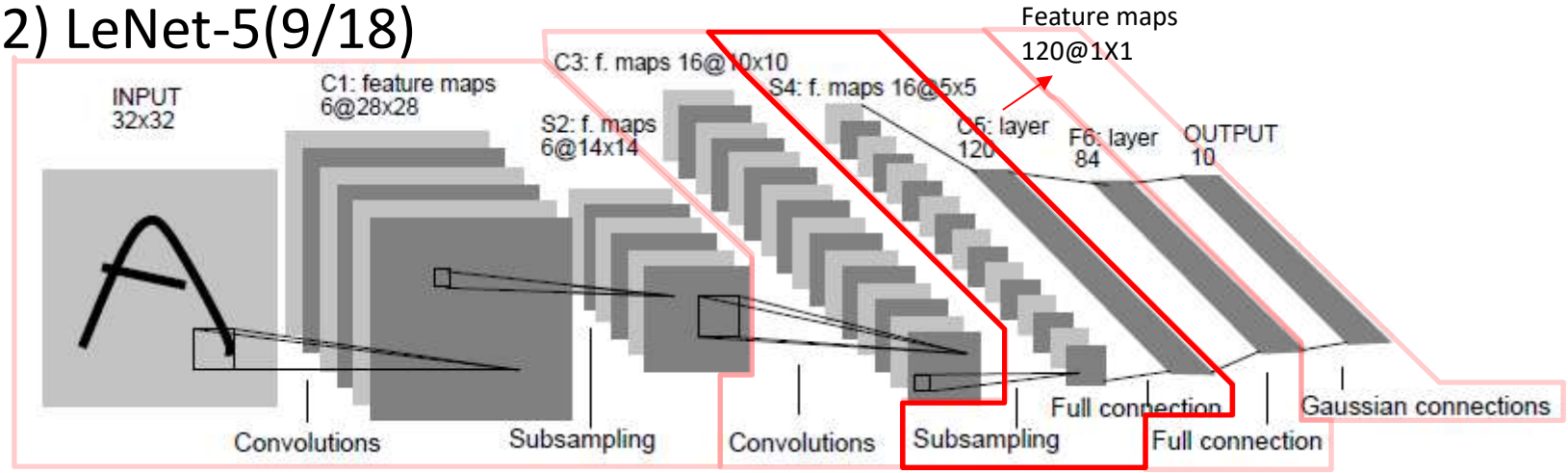
Layer C3 has 1,516 trainable parameters and 151,600 connections

↓

1516*(10*10)

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(9/18)



16*120個filter(5*5)→16為一組combine
→產生120個feature map

Layer S4 is a sub-sampling layer with 16 feature maps of size 5x5. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C3, in a similar way as C1 and S2. Layer S4 has 32 trainable parameters and 2,000 connections.

(2*2+1)*16*(5*5)

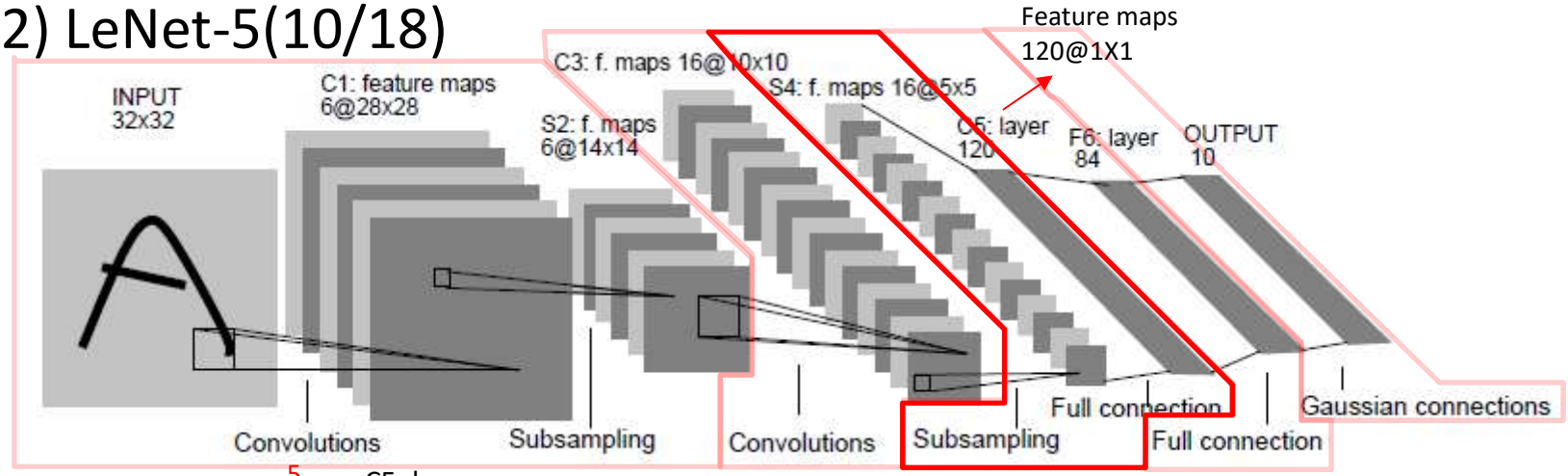
16+16
f(x)=wx+b =>w: 1, b:1

Layer C5 is a convolutional layer with 120 feature maps. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Here, because the size of S4 is also 5x5, the size of C5's feature maps is 1x1: this amounts to a full connection between S4 and C5. C5 is labeled as a convolutional layer, instead of a fully-connected layer, because if LeNet-5 input were made bigger with everything else kept constant, the feature map dimension would be larger than 1x1. This process of dynamically increasing the size of a convolutional network is described in the section Section VII. Layer C5 has 48,120 trainable connections.

16*(5*5)*120+120

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(10/18)



S4: f. maps
16@5X5

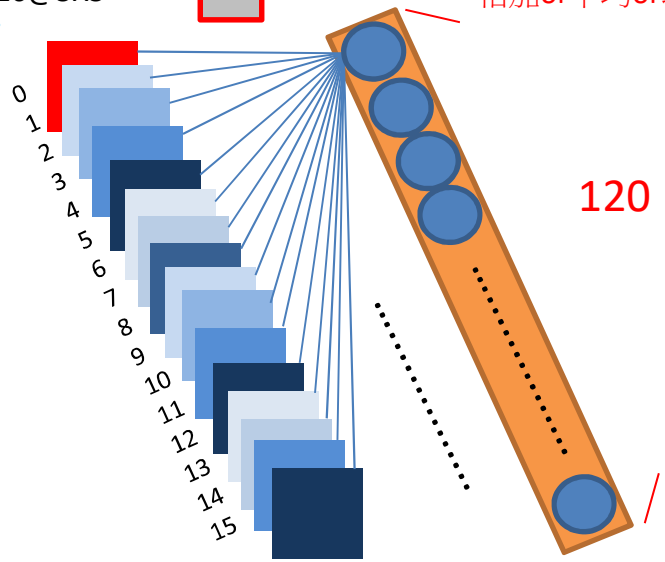
5
5 filter

C5: layer
120@1X1

??如何combine
相加or平均or取最大值

16*120個filter(5*5)→16為一組combine
→產生120個feature map

Each filter
is 5x5x16.
There are
120 filters



Full connection

Layer C5 is a convolutional layer with 120 feature maps. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Here, because the size of S4 is also 5x5, the size of C5's feature maps is 1x1: this amounts to a full connection between S4 and C5. C5 is labeled as a convolutional layer, instead of a fully-connected layer, because if LeNet-5 input were made bigger with everything else kept constant, the feature map dimension would be larger than 1x1. This process of dynamically increasing the size of a convolutional network is described in the section Section VII. Layer C5 has 48,120 trainable connections.

↓

16*(5*5)*120+120

Background knowledge

Turning FC layer into convolutional layers

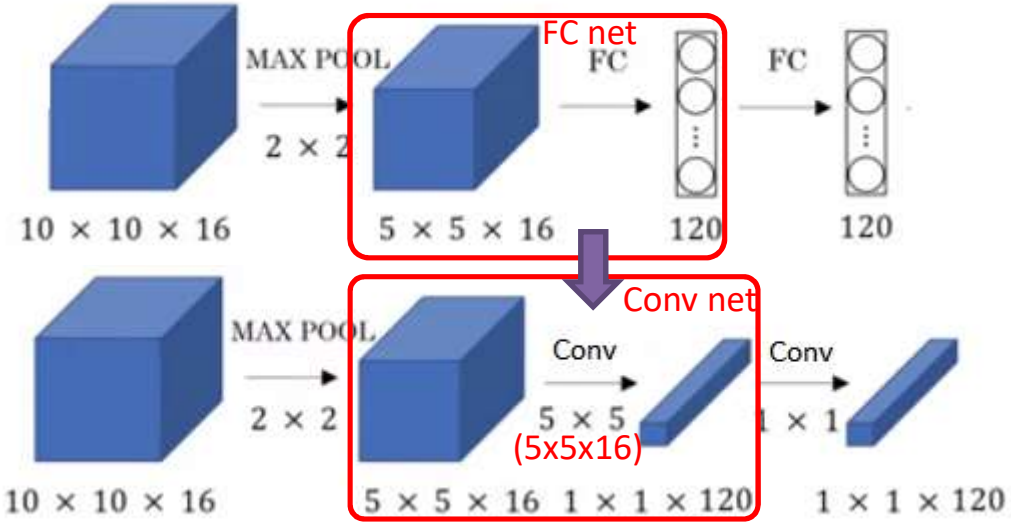


Fig. Turning FC layer to Conv layer

- Implement fully connected layer as a convolutional layer with 5x5 filter.
  - For 1 (5x5) filter:
    - We take 5x5x16 maps and convolve it with a 5x5 filter.
    - Each 5x5 filter is implemented as 5x5x16 because our convention is that the filter looks across all 16 channels.
    - The output will be 1x1.
  - Use 120 (5x5) filters:
    - The output will be 1x1x120
- Rather than viewing these 120 (FC) as just a set of nodes, we are going to view this as a 1x1x120 (Conv) volume.
- Each of those 120 values is arbitrary linear function of these 5x5x16 activations from the previous layer.

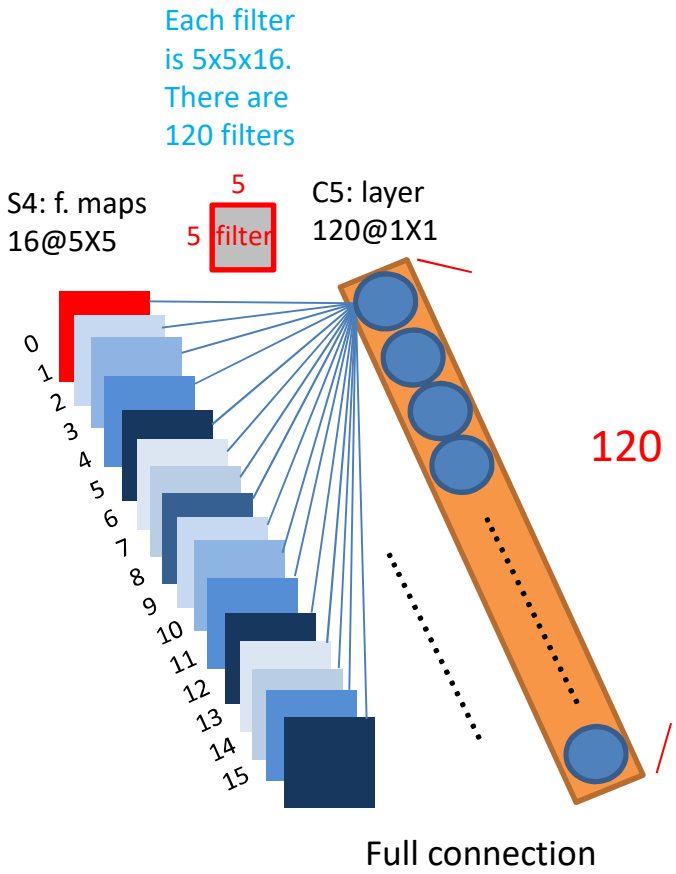
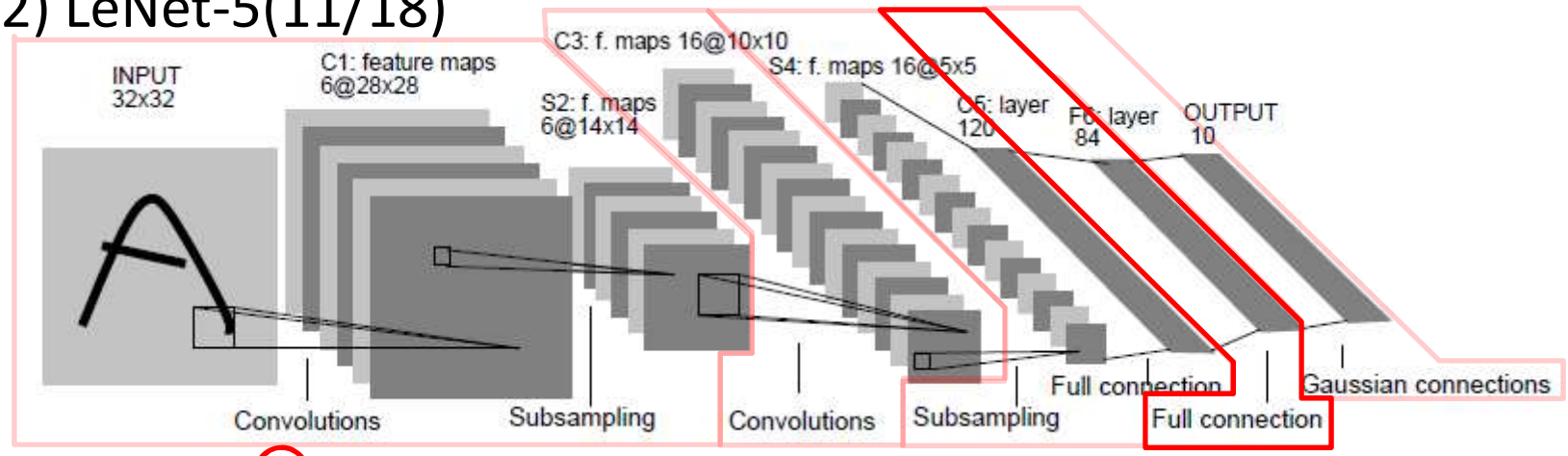


Fig. Layer S4 to C5 in LeNet 5

Source 1: DeepLearn_RecogDoc_Ieee1998_20180314_V3Beta6_Phu
Source 2: https://www.coursera.org/learn/convolutional-neural-networks/lecture/6UnU4/convolutional-implementation-of-sliding-windows

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(11/18)



Layer F6, contains 84 units (the reason for this number comes from the design of the output layer, explained below) and is fully connected to C5. It has 10,164 trainable parameters.

120*84+84

Why this number? discuss later

As in classical neural networks, units in layers up to F6 compute a dot product between their input vector and their weight vector, to which a bias is added. This weighted sum, denoted a_i for unit i, is then passed through a ~~sigmoid~~ squashing function to produce the state of unit i, ~~denoted~~ by x_i:

$$x_i = f(a_i) \tag{5}$$

The squashing function is a scaled hyperbolic tangent:

$$f(a) = A\tanh(Sa) \tag{6}$$

a: input

A: amplitude of the function(+A~-A) (chosen to be 1.7159)

S: Angle (0~2Pi => 0~pi if S=2, so slope change bigger) slope at the origin

where A is the amplitude of the function and S determines its slope at the origin. The function f is odd, with horizontal asymptotes at $+A$ and $-A$. The constant A is chosen

x_i = A*tanh(S*α_i)

α_i = sum(X.*w) + b

α_i: dot product between input vector X (from C5) and weight w vector and add bias b for unit i

f: sigmoid squashing function

x_i: state of unit i, output i=0~83

2. Convolutional neural networks for isolated character recognition

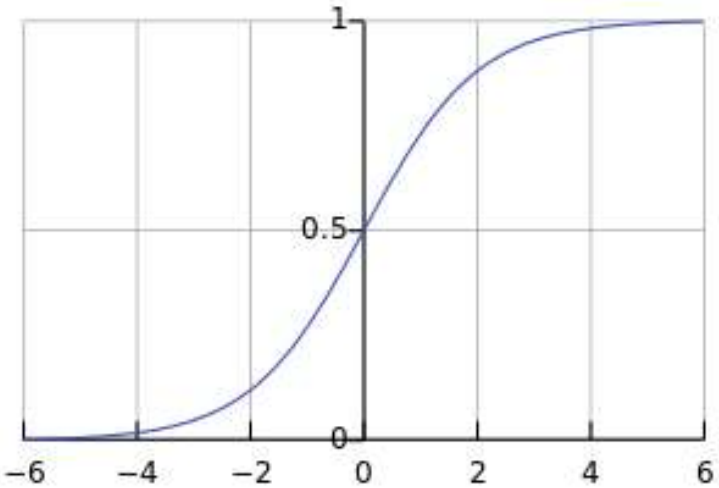
2) LeNet-5(12/18)

補充:

Sigmoid function

Equation:

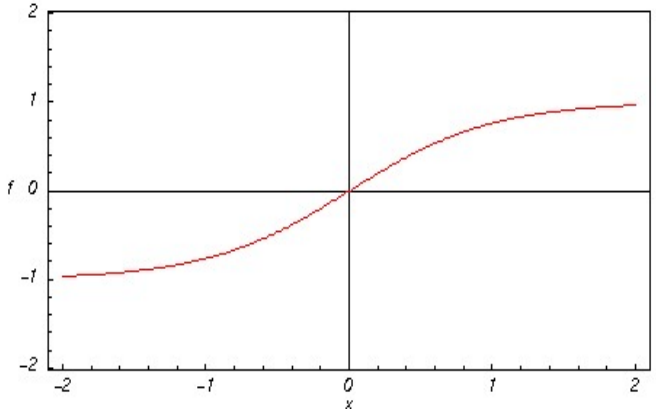
$$S(t) = \frac{1}{1+e^{-t}}$$

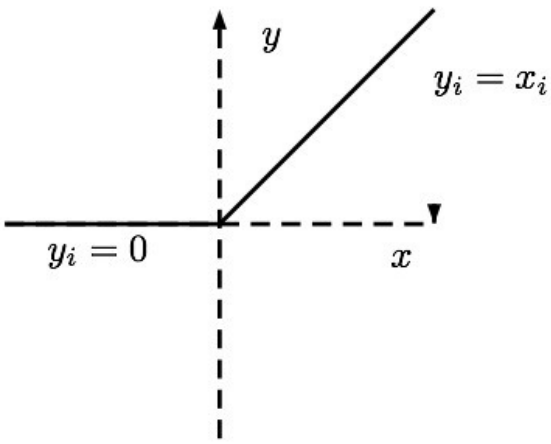


Hyperbolic tangent

Equation:

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$





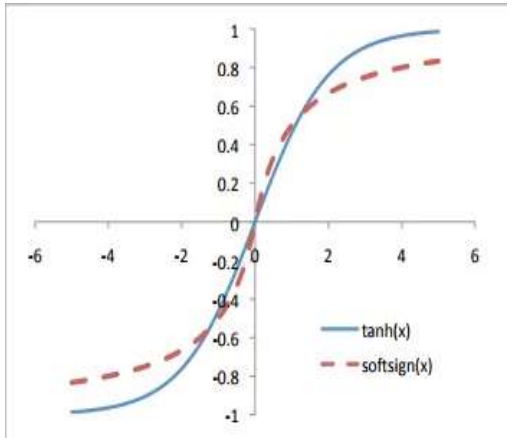
ReLU

$$f(x) = x^+ = \max(0, x),$$

Squash function is not good(gradient vanish problem),so nowadays often use ReLu (Rectified Linear Unit)

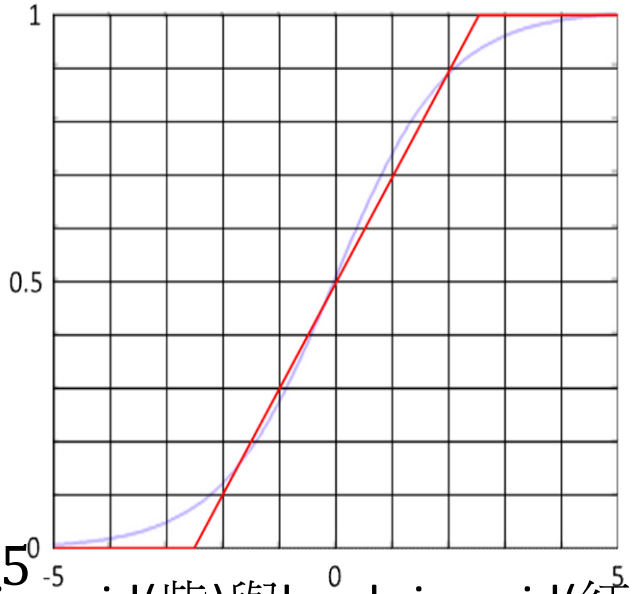
Activation function

- Softsigh
  - 更平坦的曲線與更慢的微分導數下降
  - 代表可以有更高的效率學習
  - $f(x) = \frac{x}{1+|x|}$



softsigh與tanh

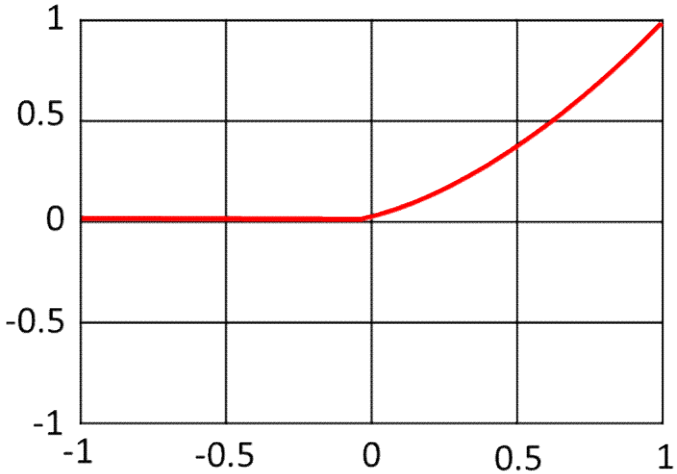
- hard sigmoid
  - 計算上會比sigmoid更快
  - 不適合用在回歸
  - $f(x) = \begin{cases} 0 & if\ x < -2.5 \\ 0.2x + 0.5 & if\ -2.5 < x < -2.5 \\ 1 & if\ 2.5 < x \end{cases}$



sigmoid(紫)與hard sigmoid(紅)

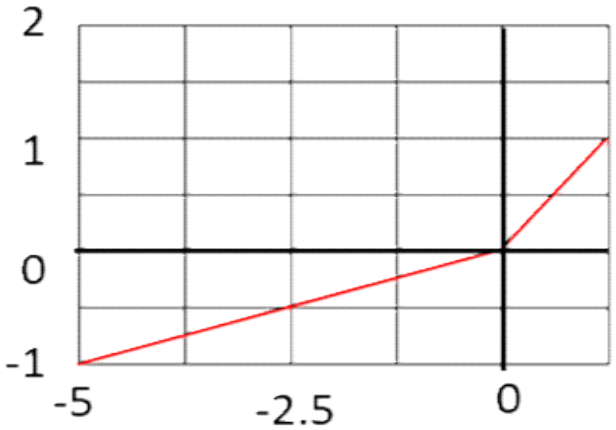
Activation function

- Softplus
  - 更加接近生物學中的人腦神經元
  - $\mathrm{f(x)} = \log(1 + \exp(\mathrm{x}))$



Softplus

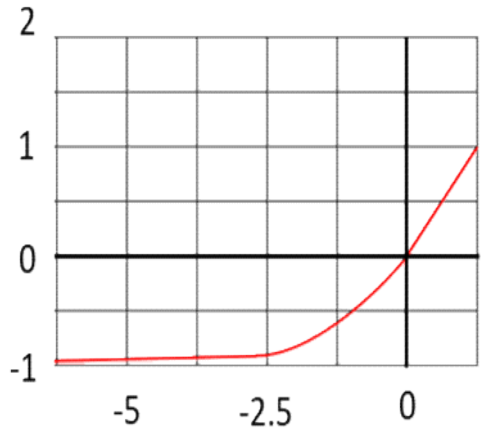
- Prelu
  - 為了解決relu在輸入為負值時權重不會更新的問題
  - $f(x) = \lambda \begin{cases} x \ \ if \ x > 0 \\ \alpha x \ if \ x \leq 0 \end{cases}$



PRelu

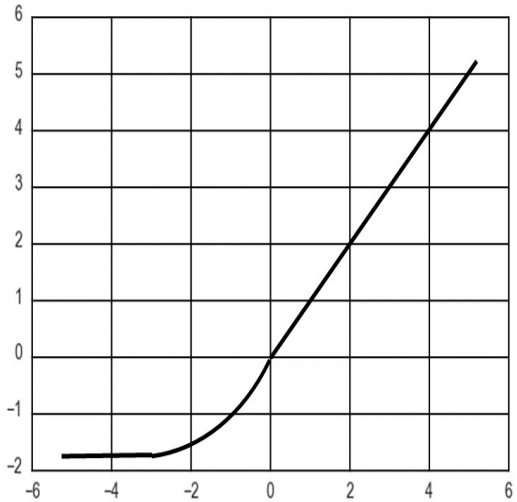
Activation function

- Elu
  - 負值並且平均值更接近零
  - 卷積神經網路學習的更快
  - 具有抗雜訊的能力
  - $f(x) = \begin{cases} x & if\ x > 0 \\ \alpha(\exp(x) - 1) & if\ x \leq 0 \end{cases}$



Elu

- Selu
  - 能使網路層數更多
  - $f(x) = \lambda \begin{cases} x & if\ x > 0 \\ \alpha(\exp(x) - \alpha) & if\ x \leq 0 \end{cases}$



Selu

2. Convolutional neural networks for isolated character recognition

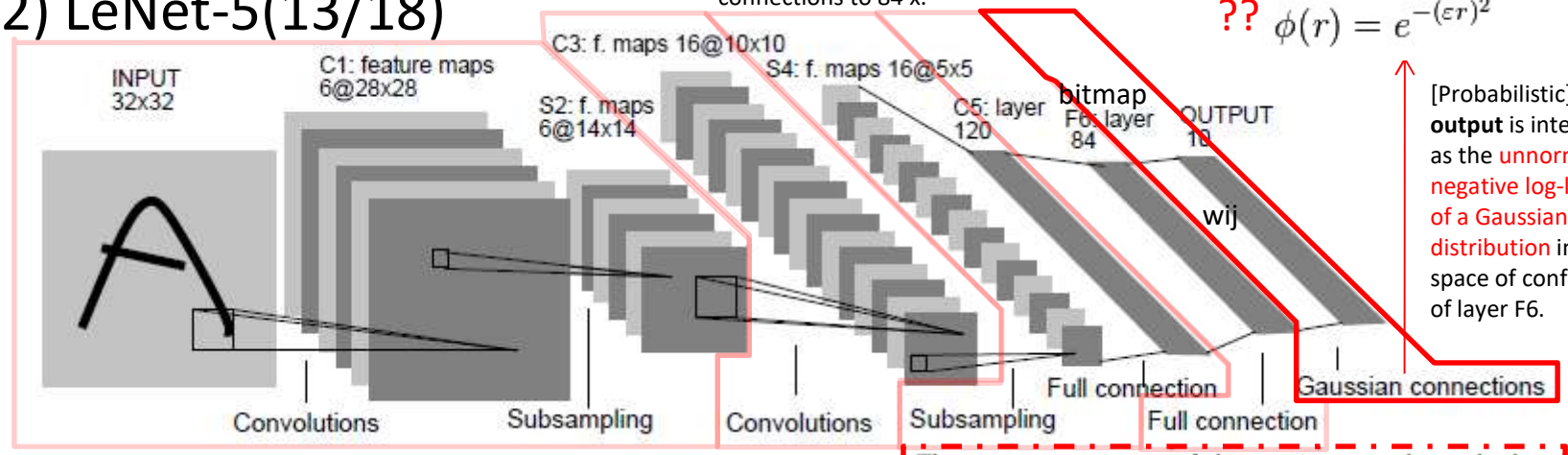
2) LeNet-5(13/18)

-For 10 digits, 84*10 connections,
Ex. For 1st digit 0, it
Has 84 w connections to 84 x.
For 2nd digit 1, it also has 84
connections to 84 x.

-For A,B.. 26, 84*26 connections,
Ex. For 1st alphabet A, it
Has 84 w connections to 84 x.
For 2nd alphabet B, it also has 84 connections to 84 x.

?? $\phi(r) = e^{-(\varepsilon r)^2}$

[Probabilistic], the **RBF output** is interpreted as the unnormalized negative log-likelihood of a Gaussian distribution in the space of configuration of layer F6.



Finally, the output layer is composed of Euclidean Radial Basis Function units (RBF), one for each class, with 84 inputs each. The outputs of each RBF unit y_i is computed as follows:

input vector

Euclidean distance $y_i = \sum_{j=1}^{84} (x_j - w_{ij})^2, \; i = 1, \ldots, 10.$

target vector
From F6

7x12 bitmap

still confuse

→ Use linguistic post-processor to solve

Fig. 3. Initial parameters of the output RBFs for recognizing the full ASCII set.

As template, initial -1 (no ink) or +1 (ink), after learning, it becomes gray value btw -1~+1.

The parameter vectors of these units were chosen by hand and kept fixed (at least initially). The components of those parameters vectors were set to -1 or +1. While they could have been chosen at random with equal probabilities for -1 and +1, or even chosen to form an error correcting code as suggested by [47], they were instead designed to represent a stylized image of the corresponding character class drawn on a 7x12 bitmap (hence the number 84). Such a representation is not particularly useful for recognizing isolated digits, but it is quite useful for recognizing strings of characters taken from the full printable ASCII set. The rationale is that characters that are similar, and therefore confusable, such as uppercase O, lowercase O, and zero, or lowercase l, digit 1, square brackets, and uppercase I, will have similar output codes. ?? This is particularly useful if the system is combined with a linguistic post-processor that can correct such confusions. Because the codes for confusable classes are similar, the output of the corresponding RBFs for an ambiguous character will be similar, and the post-processor will be able to pick the appropriate interpretation. Figure 3 gives the output codes for the full ASCII set.

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(14/18)

補充:

Finally, the output layer is composed of Euclidean Radial Basis Function units (RBF), one for each class, with 84 inputs each. The outputs of each RBF unit y_i is computed as follows:

input vector

Euclidean distance $$y_i = \sum_j (\boxed{x_j} - \boxed{w_{ij}})^2. \quad (7)$$

target vector

P:如果output layer某Unit的output越小，則input pattern越接近某Unit所代表的類別 (ex: $y_1 = 0,\ label\ is\ 1$)

- The radial basic function is a real-valued function whose ***value*** depends only on the ***distance*** from the ***point*** to the ***origin*** (or some other points, called a centers).
- The norm is usually Euclidean distance.

一、集成函數（Summation function）

集成函數是用以將從其他處理單元輸出 x_i 透過網路連結 W_{ij} 傳來的訊息綜合，常見者有二種：

(1) 加權乘積和 $net_j = \sum_i W_{ij} x_i$

(2) 歐氏距離 $net_j = \sum_i (x_i - W_{ij})^2$

Euclidean distance
Often used in supervised learning

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(15/18)

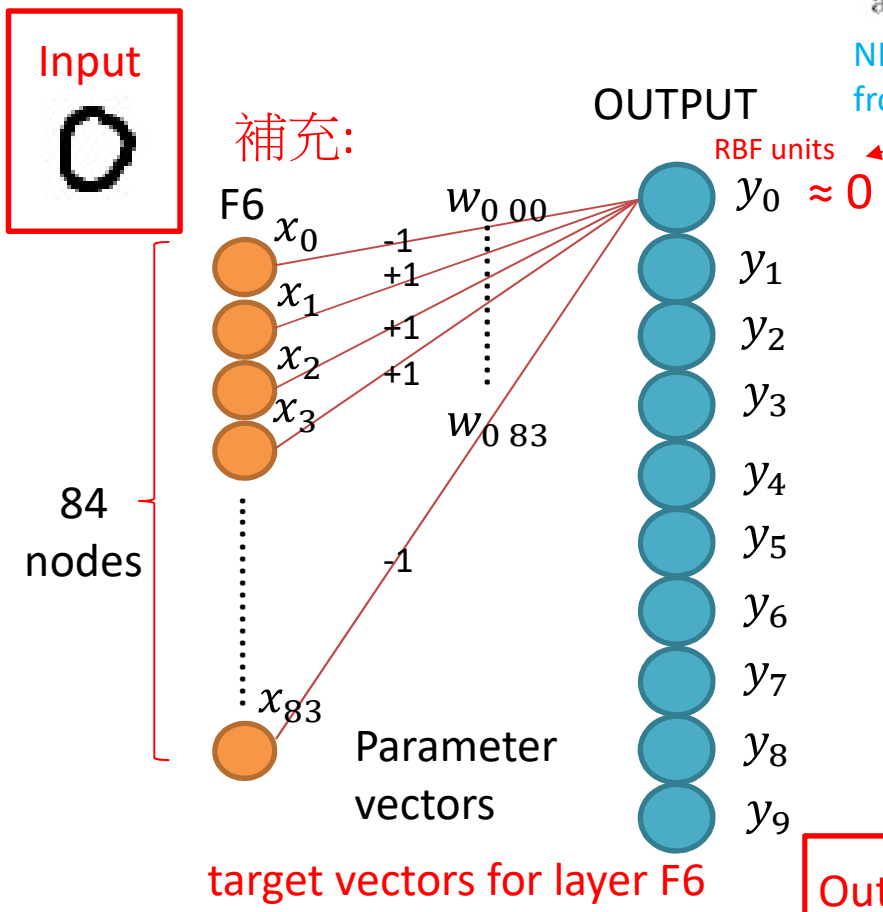
Finally, the output layer is composed of Euclidean Radial Basis Function units (RBF), one for each class, with 84 inputs each. The outputs of each RBF unit y_i is computed as follows: 歐式距離

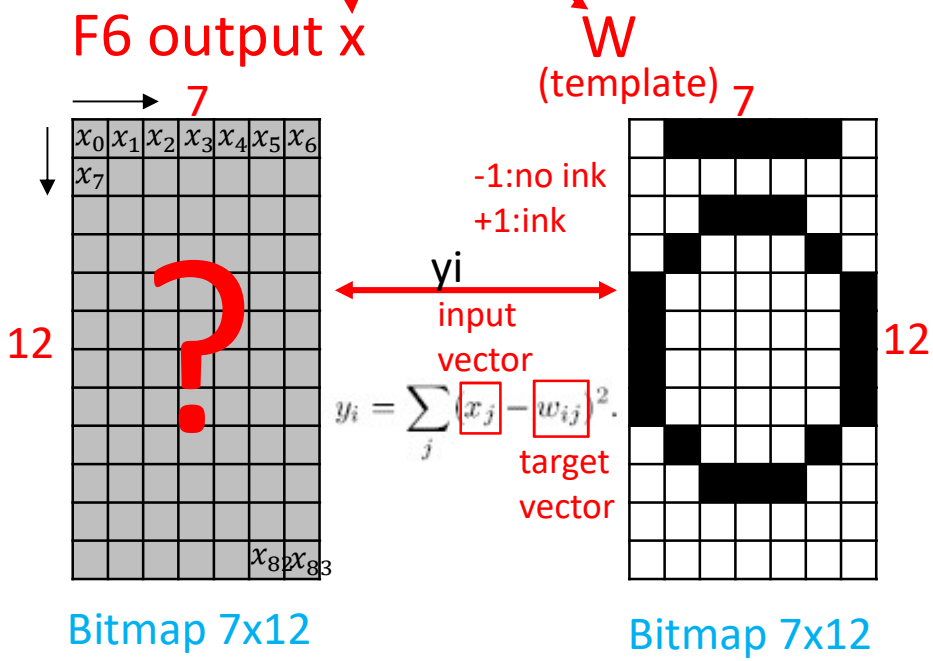
$$y_i = \sum_j (x_j - w_{ij})^2. \quad (7)$$

input vector

target vector

NN: SOM, Different from BPNN





P:如果output layer某Unit的output越小，則input pattern越接近某Unit所代表的類別 (ex: $y_1 = 0,\ label\ is\ 1$)

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(16/18)

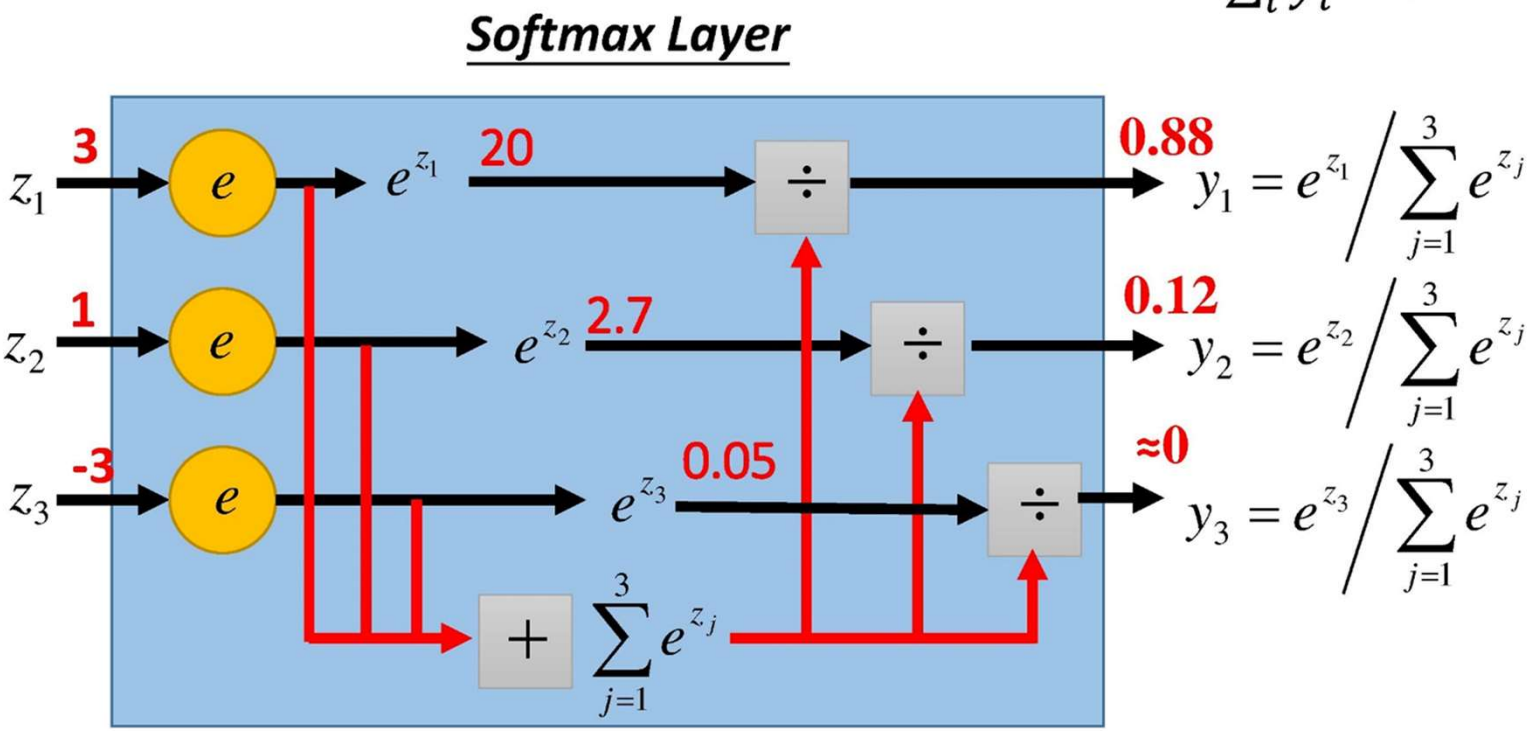
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

補充: 最近常用Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

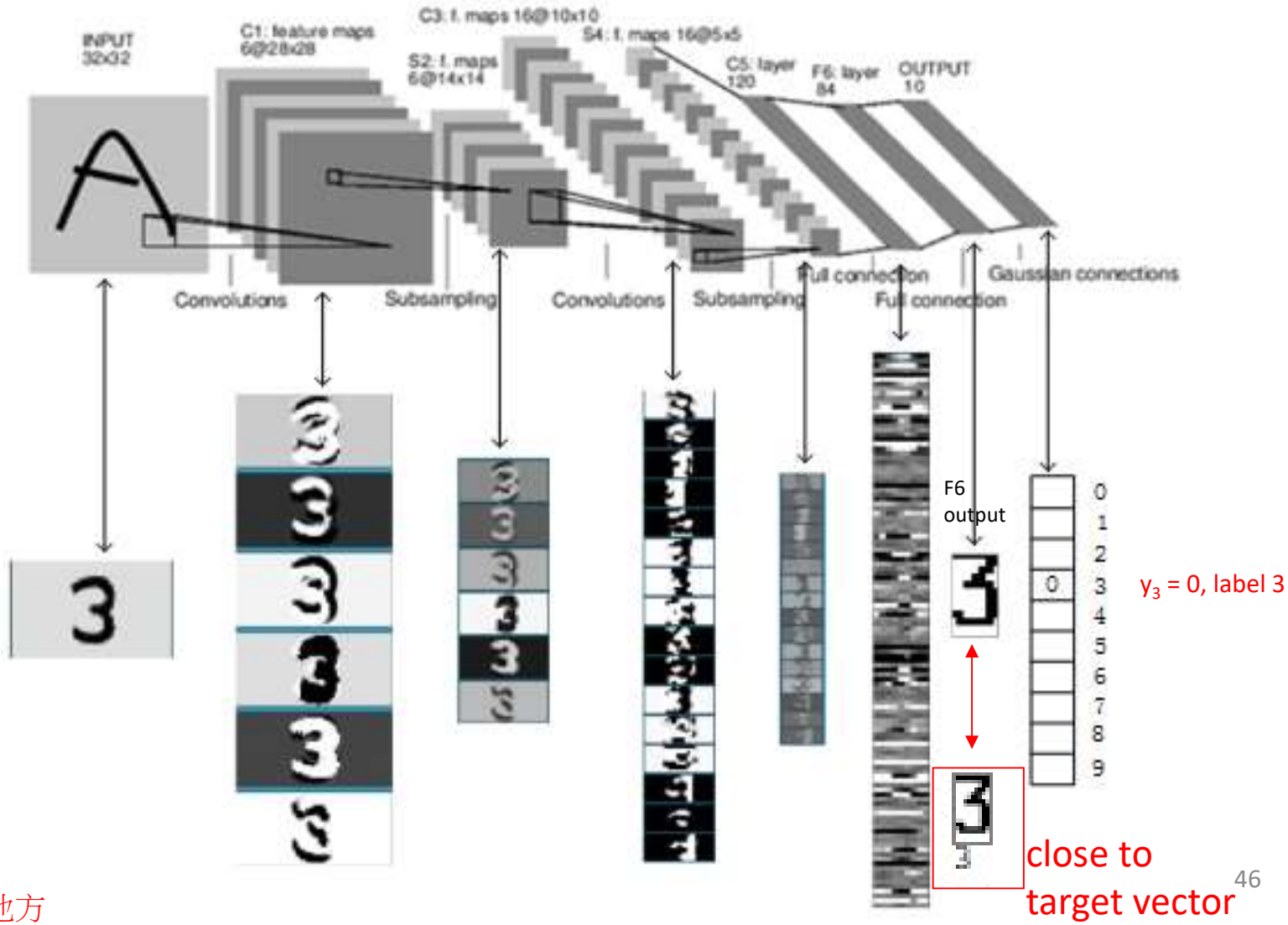


※擷取至一天搞懂Deep Learning

2. Convolutional neural networks for isolated character recognition

2) LeNet-5(17/18)

Ex:



※黑色為值大的地方

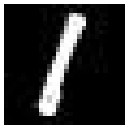
2. Convolutional neural networks for isolated character recognition

2) LeNet-5(18/18)

(Activation function use ReLu)
(Output layer use Softmax)

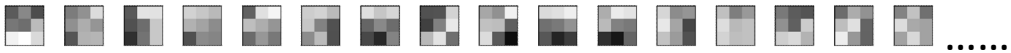
Ex:

INPUT
28X28



3X3 filter

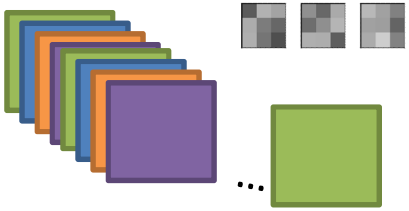
C1:feature maps
32@26X26

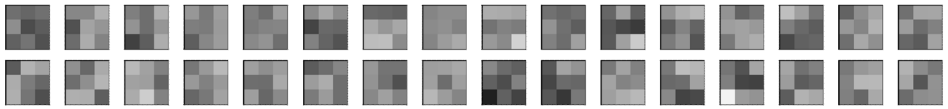


3x3 32 filters

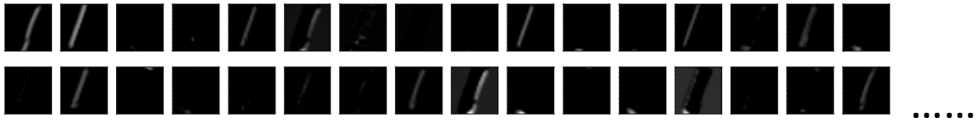
3X3 filter

C2:feature maps
64@24X24



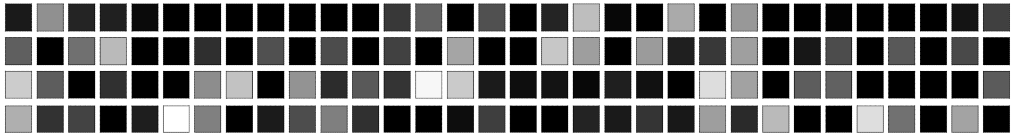


3x3 64 filters



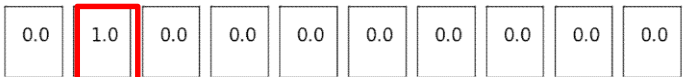
S3:feature maps
64@12X12

Flatten



F4:layer 128

OUTPUT 10



2. Convolutional neural networks for isolated character recognition

2) LeNet-5(18/18)

(Activation function use ReLu)
(Output layer use Softmax)

Ex:

INPUT
28X28

3X3 filter

C1:feature maps
32@26X26

3X3 filter

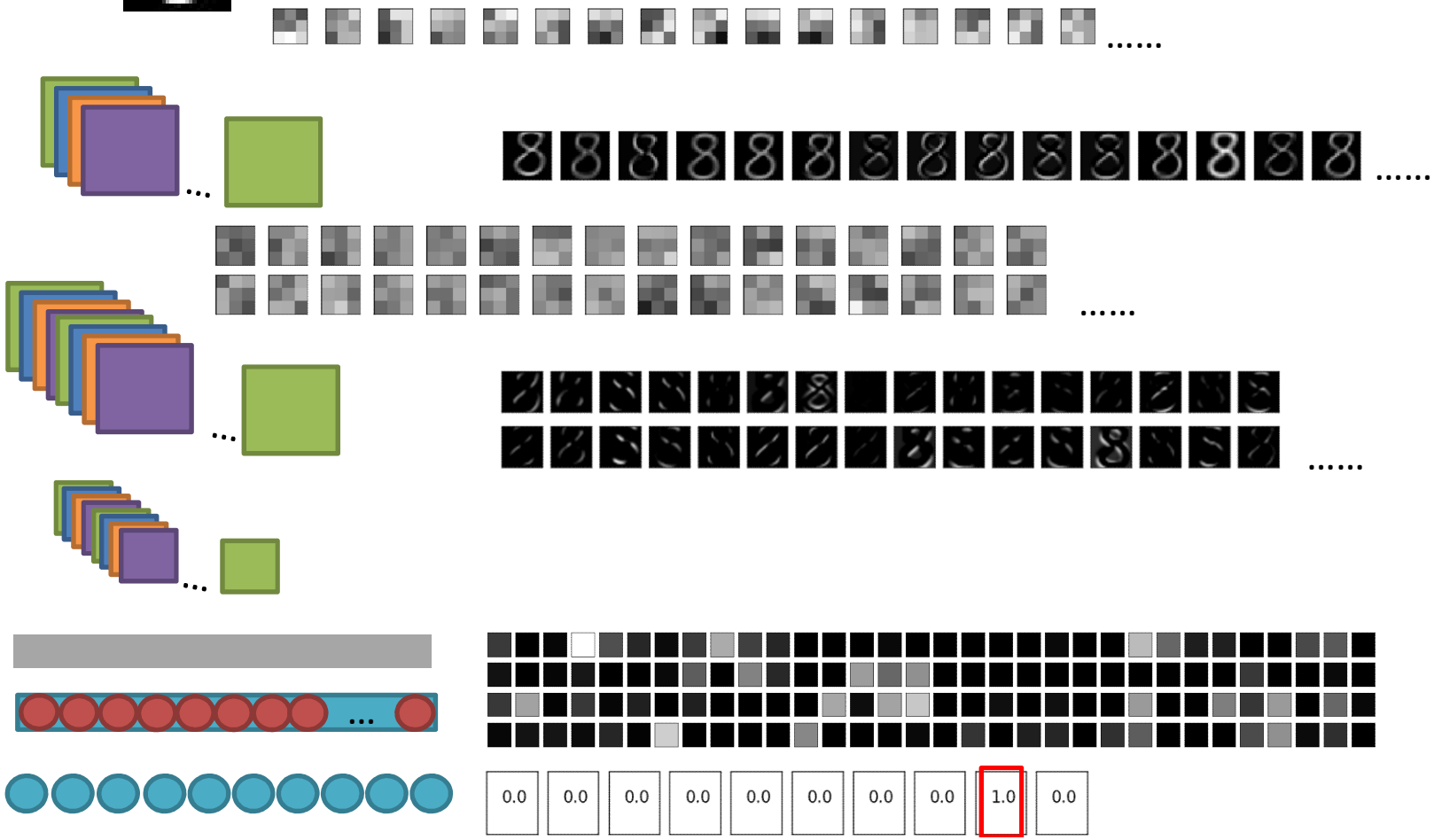
C2:feature maps
64@24X24

S3:feature maps
64@12X12

Flatten

F4:layer 128

OUTPUT 10



2. Convolutional neural networks for isolated character recognition

3) Loss Function

$$E^p = \overset{\text{difference}}{\mathrm{D}}(D^p, \mathrm{F}(Z^p, W))$$

Collapsing phenomenon
→if RBF parameter vectors can adapt, RBF所有 parameter vector皆相同,
不管input 為何,所有RBF outputs 皆為0

P:找到一組W，使得F6的output皆為constant，與RBF的parameter vector 差距為0，RBF output皆為0

Z^p. While this cost function is appropriate for most cases, it lacks three important properties. First, if we allow the parameters of the RBF to adapt, $E(W)$ has a trivial, but totally unacceptable, solution. In this solution, all the RBF parameter vectors are equal, and the state of F6 is constant and equal to that parameter vector. In this case the network happily ignores the input, and all the RBF outputs are equal to zero. This collapsing phenomenon does not occur if the RBF weights are not allowed to adapt. The second problem is that there is no competition between the classes. Such a competition can be obtained by using a more discriminative training criterion, dubbed the MAP (maximum a posteriori) criterion, similar to Maximum Mutual Information criterion sometimes used to train HMMs [48], [49], [50]. It corresponds to maximizing the posterior probability of the correct class D_p (or minimizing the logarithm of the probability of the correct class), given that the input image can come from one of the classes or from a background "rubbish" class label. In terms of

Class之間沒有競爭關係

The simplest output loss function that can be used with the above network is the Maximum Likelihood Estimation criterion (MLE), which in our case is equivalent to the Minimum Mean Squared Error (MSE). The criterion for a set of training samples is simply:

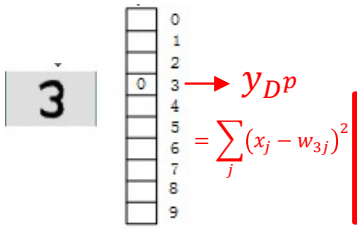
越小越好

$$y_i = \sum_j (x_j - w_{ij})^2. \qquad E(W) = \frac{1}{P}\sum_{p=1}^{P} y_{D^p}(Z^p, W) \qquad (8)$$

Not good

where y_{D^p} is the output of the D_p-th RBF unit, i.e. the one that corresponds to the correct class of input pattern Z^p.

Ex:



input RBF output units

p: p-th training data
P: training data number(ex: 60000)
Z^p: p-th input pattern/data
W: parameter(weight, bias)
y_{D^p}: output of the D^p-th RBF unit
D^p: p-th input pattern corresponds to the correct class(ex: RBF 1-unit)

j: constant(positive)

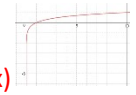
penalties, it means that in addition to pushing down the penalty of the correct class like the MSE criterion, this criterion also pulls up the penalties of the incorrect classes:

Penalty of the incorrect class

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}\left(y_{D^p}(Z^p, W) + \log\left(e^{-j} + \sum_i e^{-y_i(Z^p, W)}\right)\right)$$

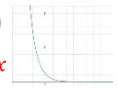
Penalty of the correct class

log(x)



??

e^{-x}



2. Convolutional neural networks for isolated character recognition

3) Loss Function (new)

Minimize ??

$$E^p = \mathrm{D}(D^p, \mathrm{F}(Z^p, W))$$

p: p-th training data
P: training data number(ex: 60000)
Z^p: p-th input pattern/data
W: parameter(weight, bias)
D^p: p-th input pattern corresponds to the correct class(ex: RBF 1-unit)

The simplest output loss function that can be used with the above network is the Maximum Likelihood Estimation criterion (MLE), which in our case is equivalent to the Minimum Mean Squared Error (MSE). The criterion for a set of training samples is simply:

predictions Ground truth

越小越好

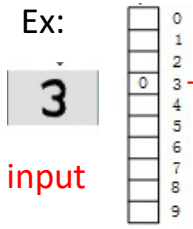
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \qquad E(W) = \frac{1}{P}\sum_{p=1}^{P} y_{D^p}(Z^p, W) \tag{8}$$

where y_{D^p} is the output of the D_p-th RBF unit, i.e. the one that corresponds to the correct class of input pattern Z^p.

$$y_i = \sum_j (x_j - w_{ij})^2.$$

x_j = sum(Z^p.*W) + b

Ex:



input

RBF output units

$$y_{D^p} = \sum_j (x_j - w_{ij})^2$$

j: j-th input (0,1,..,84)
i: i-th output (0,1,...,9)

Only check the zero or closer zero term.
Here consider only this term without considering other teams.
So right side equation, it needs to consider additional penalty team to include other teams above.

Class之間沒有競爭關係

Collapsing phenomenon
→if RBF parameter vectors can adapt, RBF所有 parameter vector皆相同,
不管input 為何,所有RBF outputs 皆為0

P:找到一組W，使得F6的output皆為constant，與RBF的parameter vector 差距為0，RBF output皆為0

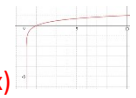
Z^p. While this cost function is appropriate for most cases, it lacks three important properties. First, if we allow the parameters of the RBF to adapt, $E(W)$ has a trivial, but totally unacceptable, solution. In this solution, all the RBF parameter vectors are equal, and the state of F6 is constant and equal to that parameter vector. In this case the network happily ignores the input, and all the RBF outputs are equal to zero. This collapsing phenomenon does not occur if the RBF weights are not allowed to adapt. The second problem is that there is no competition between the classes. Such a competition can be obtained by using a more discriminative training criterion, dubbed the MAP (maximum a posteriori) criterion, similar to Maximum Mutual Information criterion sometimes used to train HMMs [48], [49], [50]. It corresponds to maximizing the posterior probability of the correct class D_p (or minimizing the logarithm of the probability of the correct class), given that the input image can come from one of the classes or from a background "rubbish" class label. In terms of

penalties, it means that in addition to pushing down the penalty of the correct class like the MSE criterion, this criterion also pulls up the penalties of the incorrect classes:

Penalty of the incorrect class

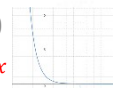
$$E(W) = \frac{1}{P}\sum_{p=1}^{P}\left(y_{D^p}(Z^p, W) + \log\left(e^{-j} + \sum_i e^{-y_i(Z^p, W)}\right)\right)$$

Penalty of the correct class



log(x)

??

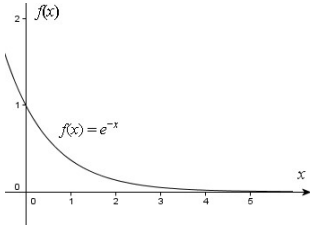


e^{-x}

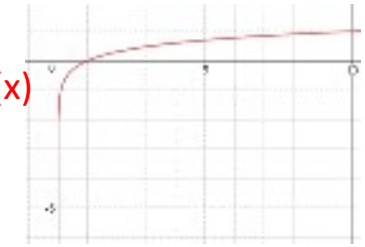
[Phu and Peter]:

penalties, it means that in addition to pushing down the penalty of the correct class like the MSE criterion, this criterion also pulls up the penalties of the incorrect classes:

e^{-x}



log(x)



Penalty of the correct class

Penalty of the incorrect class

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}\left(\boxed{y_{D^p}(Z^p, W)} + \boxed{\log(e^{-j} + \sum_i e^{-y_i(Z^p,W)})}\right)$$

Case 1: (Incorrect case) more than 1 class have RBF output = 0 $y_{D^p}(Z^p, W)$ = 0 Other incorrect classes RBF output = 0	Case 2: (Correct case) only 1 class have RBF output = 0 $y_{D^p}(Z^p, W)$ = 0 Other incorrect classes RBF output != 0
$y_i(Z^p,W)$ = 0 $e^{-y_i(Z^p,W)}$ = 1 $\sum_i e^{-y_i(Z^p,W)}$ > 1 $\log(e^{-j} + \sum_i e^{-y_i(Z^p,W)})$ Big ; E(W): big Not good (because e^{-j} always be greater than 0)	$y_i(Z^p,W)$!= 0 $e^{-y_i(Z^p,W)}$ = Approximately 0 $\sum_i e^{-y_i(Z^p,W)}$ = Approximately 0 $\log(e^{-j} + \sum_i e^{-y_i(Z^p,W)})$ Small ; E(W): small Good (add e^{-j}: avoid: log(z) -> negative infinitive)

3. Result and comparison with other methods

1) Database: the Modified NIST set

The original black and white (bilevel) images were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as result of the anti-aliasing (image interpolation) technique used by the normalization algorithm. Three versions of the database were used. In the first version, the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. In some instances, this 28x28 field was extended to 32x32 with background pixels. This version of the database will be referred to as the *regular* database. In the second version of the database, the character images were deslanted and cropped down to 20x20 pixels images. The deslanting computes the second moments of inertia of the pixels (counting a foreground pixel as 1 and a background pixel as 0), and shears the image by horizontally shifting the lines so that the principal axis is vertical. This version of the database will be referred to as the *deslanted* database. In the third version of the database, used in some early experiments, the images were reduced to 16x16 pixels. The regular database (60,000 training examples, 10,000 test examples size-normalized to 20x20, and centered by center of mass in 28x28 fields) is available at `http://www.research.att.com/yann/ocr/mnist`. Figure 4 shows examples randomly picked from the test set.

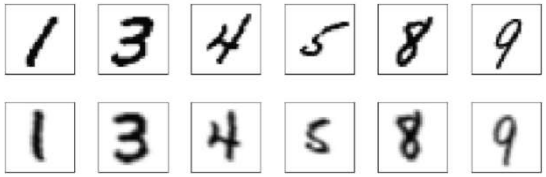
MNIST database(Size-normalized)

60000 training examples, 10000 test examples

Three version of the database:
1. Regular database (28X28)
2. De-slanted database (20X20)
3. Reduced to 16X16 pixels

Regular database

De-slanted database

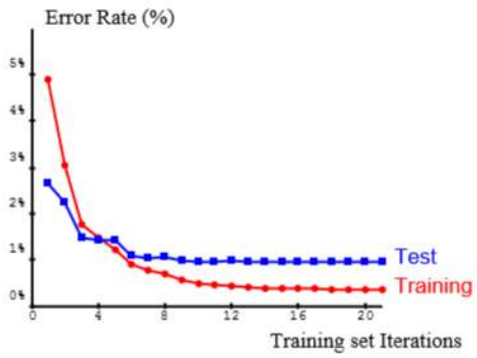


3. Result and comparison with other methods

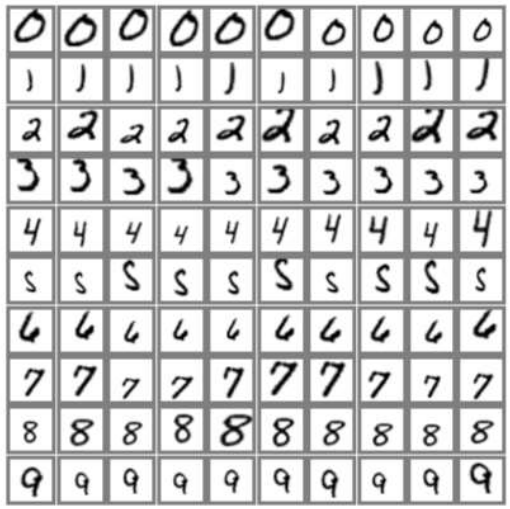
2) Results

Size-normalized examples from the MNIST database

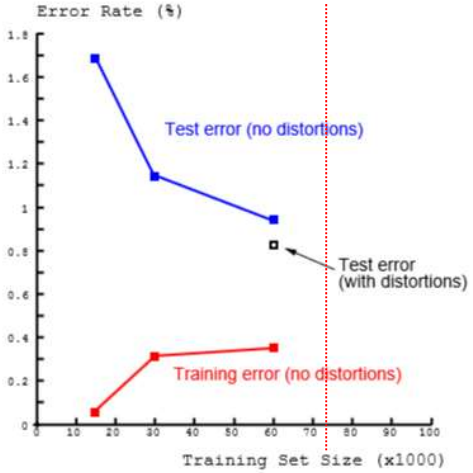
LeNet-5
60000 training set
10000 test set



Examples of distortion of ten training pattern



P↑, gap↓



82 test patterns misclassified by LeNet-5

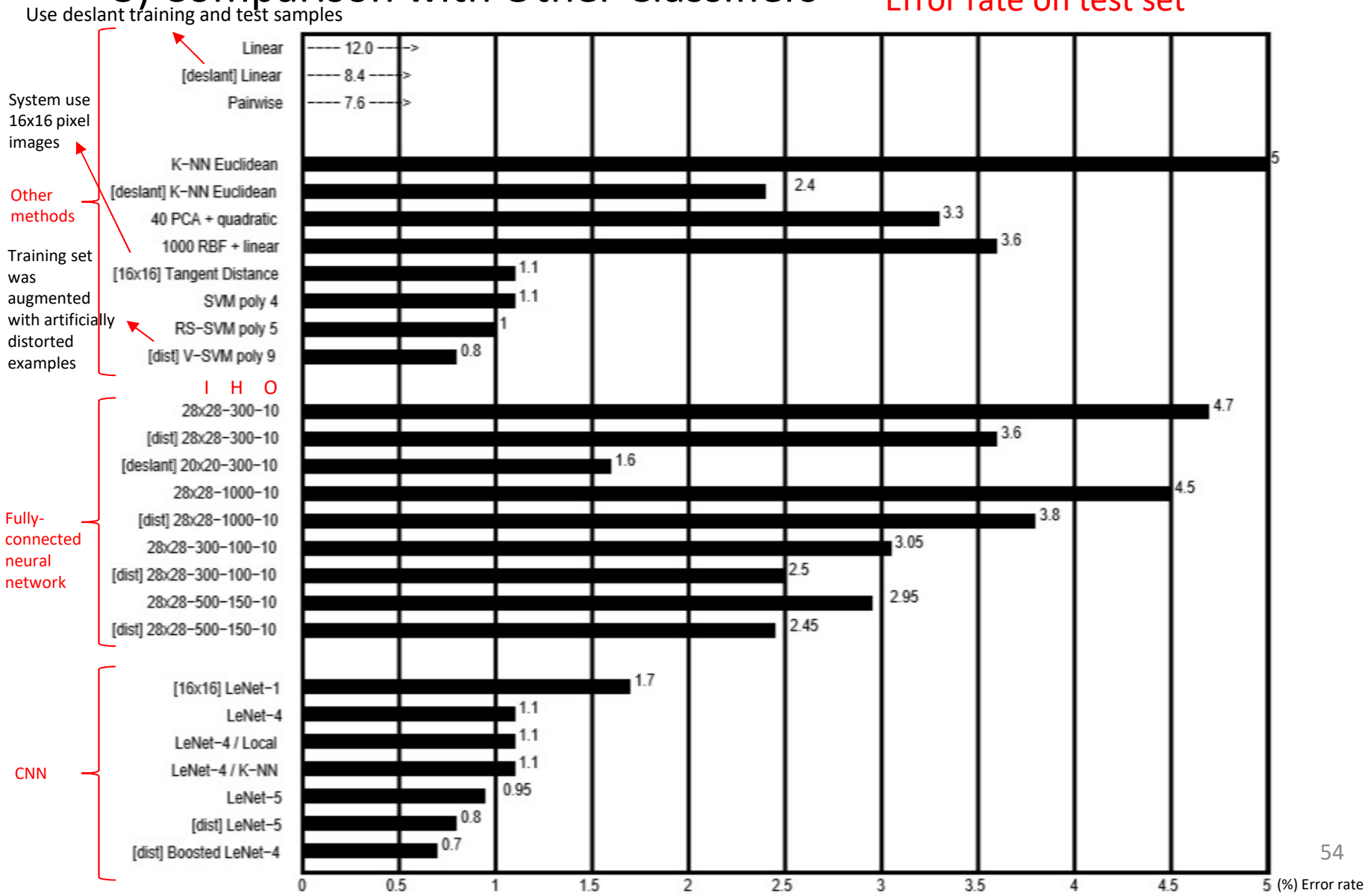
0.0005	0.0002	0.0001	0.00005	0.00001
				....

Learning rate is decreased after iterations

3. Result and comparison with other methods

3) Comparison with Other Classifiers

Error rate on test set



3. Result and comparison with other methods

3) Comparison with Other Classifiers

No hidden layer
neural network

C.1 Linear Classifier, and Pairwise Linear Classifier

Possibly the simplest classifier that one might consider is a linear classifier. Each input pixel value contributes to a weighted sum for each output unit. The output unit with the highest sum (including the contribution of a bias constant) indicates the class of the input character. On the regular data, the error rate is 12%. The network has 7850 free parameters. On the deslanted images, the test error rate is 8.4% The network has 4010 free parameters. The deficiencies of the linear classifier are well documented [1] and it is included here simply to form a basis of comparison for more sophisticated classifiers. Various combinations of sigmoid units, linear units, gradient descent learning, and learning by directly solving linear systems gave similar results. ??

A simple improvement of the basic linear classifier was tested [52]. The idea is to train each unit of a single-layer network to separate each class from each other class. In our case this layer comprises 45 units labeled 0/1, 0/2,...0/9, 1/2,...8/9. Unit i/j is trained to produce +1 on patterns of class i, -1 on patterns of class j, and is not trained on other patterns. The final score for class i is the sum of the outputs all the units labeled i/x minus the sum of the output of all the units labeled y/i, for all x and y. The error rate on the regular test set was 7.6%.

Regular size(28x28); Deslanted size (20x20)
10 output classes
Free parameters:

- *Regular: 28*28*10 +10 = 7850*
- *Deslanted: 20*20*10+10 = 4010*

C.2 Baseline Nearest Neighbor Classifier

Another simple classifier is a K-nearest neighbor classifier with a Euclidean distance measure between input images. This classifier has the advantage that no training time, and no brain on the part of the designer, are required. However, the memory requirement and recognition time are large: the complete 60,000 twenty by twenty pixel training images (about 24 Megabytes at one byte per pixel) must be available at run time. Much more compact representations could be devised with modest increase in error rate. On the regular test set the error rate was 5.0%. On the deslanted data, the error rate was 2.4%, with $k = 3$. Naturally, a realistic Euclidean distance nearest-neighbor system would operate on feature vectors rather than directly on the pixels, but since all of the other systems presented in this study operate directly on the pixels, this result is useful for a baseline comparison.

i / j (+1 on patterns of class i, -1 on patterns of class j)

Final score for class i = i / x – y / i

Ex:

Final score for class 8: 8/9 – (0/8+1/8+3/8+4/8+5/8+6/8+7/8)

Final score for class 3: 3/4 – (0/3+1/3+2/3)

3. Result and comparison with other methods

3) Comparison with Other Classifiers

C.3 Principal Component Analysis (PCA) and Polynomial Classifier

Following [53], [54], a preprocessing stage was constructed which computes the projection of the input pattern on the 40 principal components of the set of training vectors. To compute the principal components, the mean of each input component was first computed and subtracted from the training vectors. The covariance matrix of the resulting vectors was then computed and diagonalized using Singular Value Decomposition. The 40-dimensional feature vector was used as the input of a second degree polynomial classifier. This classifier can be seen as a linear classifier with 821 inputs, preceded by a module that computes all products of pairs of input variables. The error on the regular test set was 3.3%.

C.4 Radial Basis Function Network

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

Following [55], an RBF network was constructed. The first layer was composed of 1,000 Gaussian RBF units with 28x28 inputs, and the second layer was a simple 1000 inputs / 10 outputs linear classifier. The RBF units were divided into 10 groups of 100. Each group of units was trained on all the training examples of one of the 10 classes using the adaptive K-means algorithm. The second layer weights were computed using a regularized pseudo-inverse method. The error rate on the regular test set was 3.6%

C.5 One-Hidden Layer Fully Connected Multilayer Neural Network

Another classifier that we tested was a fully connected multi-layer neural network with two layers of weights (one hidden layer) trained with the version of back-propagation described in Appendix C. Error on the regular test set was 4.7% for a network with 300 hidden units, and 4.5% for a network with 1000 hidden units. Using artificial distortions to generate more training data brought only marginal improvement: 3.6% for 300 hidden units, and 3.8% for 1000 hidden units. When deslanted images were used, the test error jumped down to 1.6% for a network with 300 hidden units.

It remains somewhat of a mystery that networks with such a large number of free parameters manage to achieve reasonably low testing errors. We conjecture that the dynamics of gradient descent learning in multilayer nets has a "self-regularization" effect. Because the origin of weight space is a saddle point that is attractive in almost every direction, the weights invariably shrink during the first few epochs (recent theoretical analysis seem to confirm this [56]). Small weights cause the sigmoids to operate in the quasi-linear region, making the network essentially equivalent to a low-capacity, single-layer network. As the learning proceeds, the weights grow, which progressively increases the effective capacity of the network. This seems to be an almost perfect, if fortuitous, implementation of Vapnik's "Structural Risk Minimization" principle [6]. A better theoretical understanding of these phenomena, and more empirical evidence, are definitely needed.

3. Result and comparison with other methods

3) Comparison with Other Classifiers

C.6 Two-Hidden Layer Fully Connected Multilayer Neural Network

To see the effect of the architecture, several two-hidden layer multilayer neural networks were trained. Theoretical results have shown that any function can be approximated by a one-hidden layer neural network [57]. However, several authors have observed that two-hidden layer architectures sometimes yield better performance in practical situations. This phenomenon was also observed here. The test error rate of a 28x28-300-100-10 network was 3.05%, a much better result than the one-hidden layer network, obtained using marginally more weights and connections. Increasing the network size to 28x28-1000-150-10 yielded only marginally improved error rates: 2.95%. Training with distorted patterns improved the performance somewhat: 2.50% error for the 28x28-300-100-10 network, and 2.45% for the 28x28-1000-150-10 network.

C.7 A Small Convolutional Network: LeNet-1

Convolutional Networks are an attempt to solve the dilemma between small networks that cannot learn the training set, and large networks that seem over-parameterized. LeNet-1 was an early embodiment of the Convolutional Network architecture which is included here for comparison purposes. The images were down-sampled to 16x16 pixels and centered in the 28x28 input layer. Although about 100,000 multiply/add steps are required to evaluate LeNet-1, its convolutional nature keeps the number of free parameters to only about 2600. The LeNet-1 architecture was developed using our own version of the USPS (US Postal Service zip codes) database and its size was tuned to match the available data [35]. LeNet-1 achieved 1.7% test error. The fact that a network with such a small number of parameters can attain such a good error rate is an indication that the architecture is appropriate for the task.

3. Result and comparison with other methods

3) Comparison with Other Classifiers

??

C.8 LeNet-4

Experiments with LeNet-1 made it clear that a larger convolutional network was needed to make optimal use of the large size of the training set. LeNet-4 and later LeNet-5 were designed to address this problem. LeNet-4 is very similar to LeNet-5, except for the details of the architecture. It contains 4 first-level feature maps, followed by 8 subsampling maps connected in pairs to each first-layer feature maps, then 16 feature maps, followed by 16 subsampling map, followed by a fully connected layer with 120 units, followed by the output layer (10 units). LeNet-4 contains about 260,000 connections and has about 17,000 free parameters. Test error was 1.1%. In a series of experiments, we replaced the last layer of LeNet-4 with a Euclidean Nearest Neighbor classifier, and with the "local learning" method of Bottou and Vapnik [58], in which a local linear classifier is retrained each time a new test pattern is shown. Neither of those methods improved the raw error rate, although they did improve the rejection performance.

C.9 Boosted LeNet-4

Following theoretical work by R. Schapire [59], Drucker et al. [60] developed the "boosting" method for combining multiple classifiers. Three LeNet-4s are combined: the first one is trained the usual way. the second one is trained on patterns that are filtered by the first net so that the second machine sees a mix of patterns, 50% of which the first net got right, and 50% of which it got wrong. Finally, the third net is trained on new patterns on which the first and the second nets disagree. During testing, the outputs of the three nets are simply added. Because the error rate of LeNet-4 is very low, it was necessary to use the artificially distorted images (as with LeNet-5) in order to get enough samples to train the second and third nets. The test error rate was 0.7%, the best of any of our classifiers. At first glance, boosting appears to be three times more expensive as a single net. In fact, when the first net produces a high confidence answer, the other nets are not called. The average computational cost is about 1.75 times that of a single net.

3. Result and comparison with other methods

3) Comparison with Other Classifiers

??

C.10 Tangent Distance Classifier (TDC)

The Tangent Distance classifier (TDC) is a nearest-neighbor method where the distance function is made insensitive to small distortions and translations of the input image [61]. If we consider an image as a point in a high dimensional pixel space (where the dimensionality equals the number of pixels), then an evolving distortion of a character traces out a curve in pixel space. Taken together, all these distortions define a low-dimensional manifold in pixel space. For small distortions, in the vicinity of the original image, this manifold can be approximated by a plane, known as the tangent plane. An excellent measure of "closeness" for character images is the distance between their tangent planes, where the set of distortions used to generate the planes includes translations, scaling, skewing, squeezing, rotation, and line thickness variations. A test error rate of 1.1% was achieved using 16x16 pixel images. Prefiltering techniques using simple Euclidean distance at multiple resolutions allowed to reduce the number of necessary Tangent Distance calculations.

3. Result and comparison with other methods

3) Comparison with Other Classifiers

C.11 Support Vector Machine (SVM)

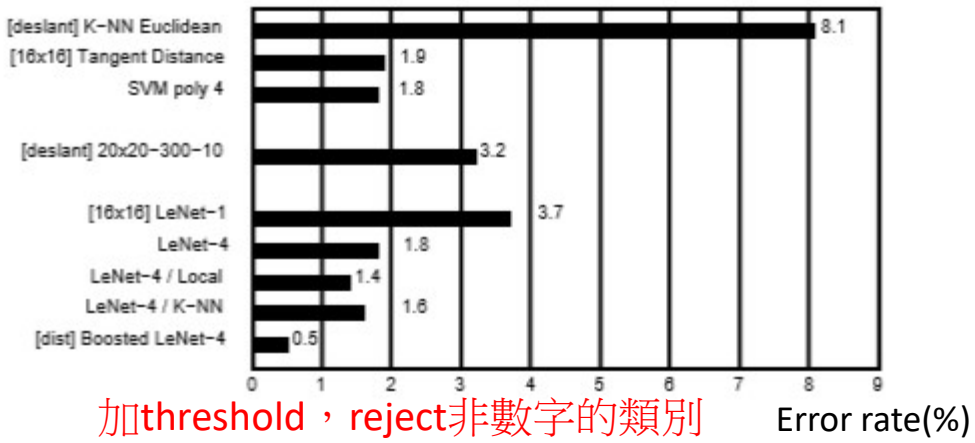
Polynomial classifiers are well-studied methods for generating complex decision surfaces. Unfortunately, they are impractical for high-dimensional problems, because the number of product terms is prohibitive. The Support Vector technique is an extremely economical way of representing complex surfaces in high-dimensional spaces, including polynomials and many other types of surfaces [6].

A particularly interesting subset of decision surfaces is the ones that correspond to hyperplanes that are at a maximum distance from the convex hulls of the two classes in the high-dimensional space of the product terms. Boser, Guyon, and Vapnik [62] realized that any polynomial of degree k in this "maximum margin" set can be computed by first computing the dot product of the input image with a subset of the training samples (called the "support vectors"), elevating the result to the k-th power, and linearly combining the numbers thereby obtained. Finding the support vectors and the coefficients amounts to solving a high-dimensional quadratic minimization problem with linear inequality constraints. For the sake of comparison, we include here the results obtained by Burges and Schölkopf reported in [63]. With a regular SVM, their error rate on the regular test set was 1.4%. Cortes and Vapnik had reported an error rate of 1.1% with SVM on the same data using a slightly different technique. The computational cost of this technique is very high: about 14 million multiply-adds per recognition. Using Schölkopf's Virtual Support Vectors technique (V-SVM), 1.0% error was attained. More recently, Schölkopf (personal communication) has reached 0.8% using a modified version of the V-SVM. Unfortunately, V-SVM is extremely expensive: about twice as much as regular SVM. To alleviate this problem, Burges has proposed the Reduced Set Support Vector technique (RS-SVM), which attained 1.1% on the regular test set [63], with a computational cost of only 650,000 multiply-adds per recognition, i.e. only about 60% more expensive than LeNet-5.

3. Result and comparison with other methods

3) Comparison with Other Classifiers

On test set



加threshold，reject非數字的類別

Fig. 10. Rejection Performance: percentage of test patterns tha must be rejected to achieve 0.5% error for some of the systems.

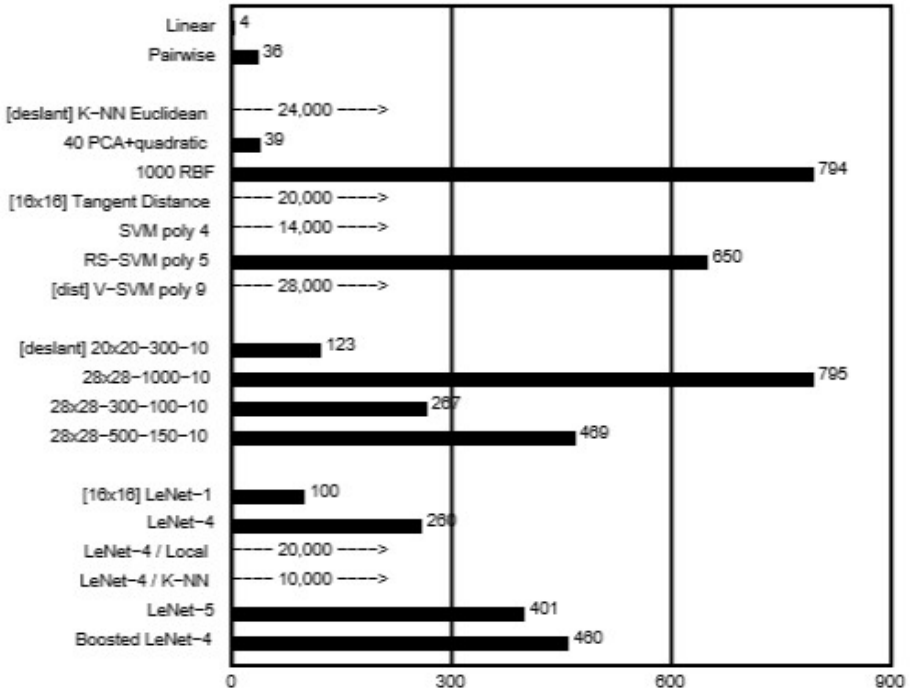
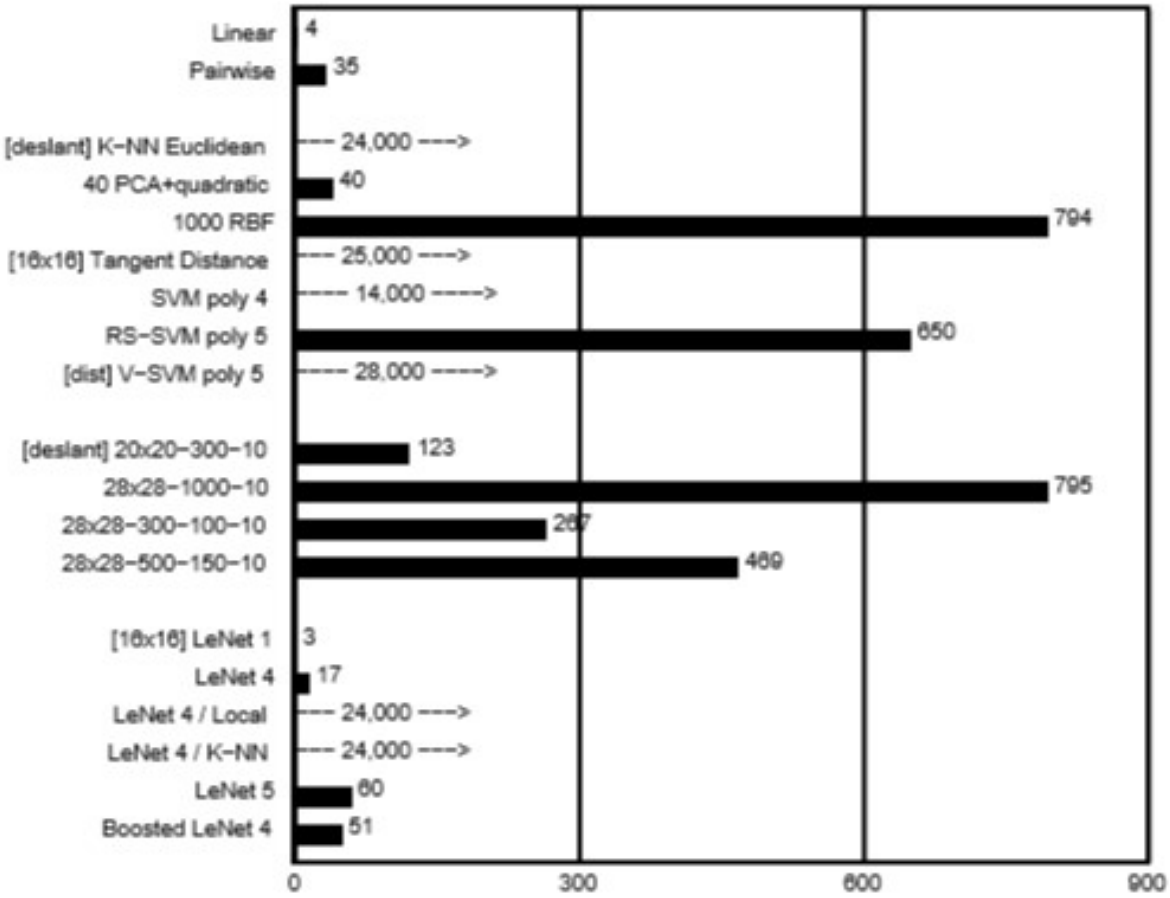


Fig. 11. Number of multiply-accumulate operations for the recognition of a single character starting with a size-normalized image.

3. Result and comparison with other methods

4) Discussion



number of variables

Fig. 12. Memory requirements, measured in number of variables, for each of the methods. Most of the methods only require one byte per variable for adequate performance.

3. Result and comparison with other methods

5) Invariance and Noise Resistance

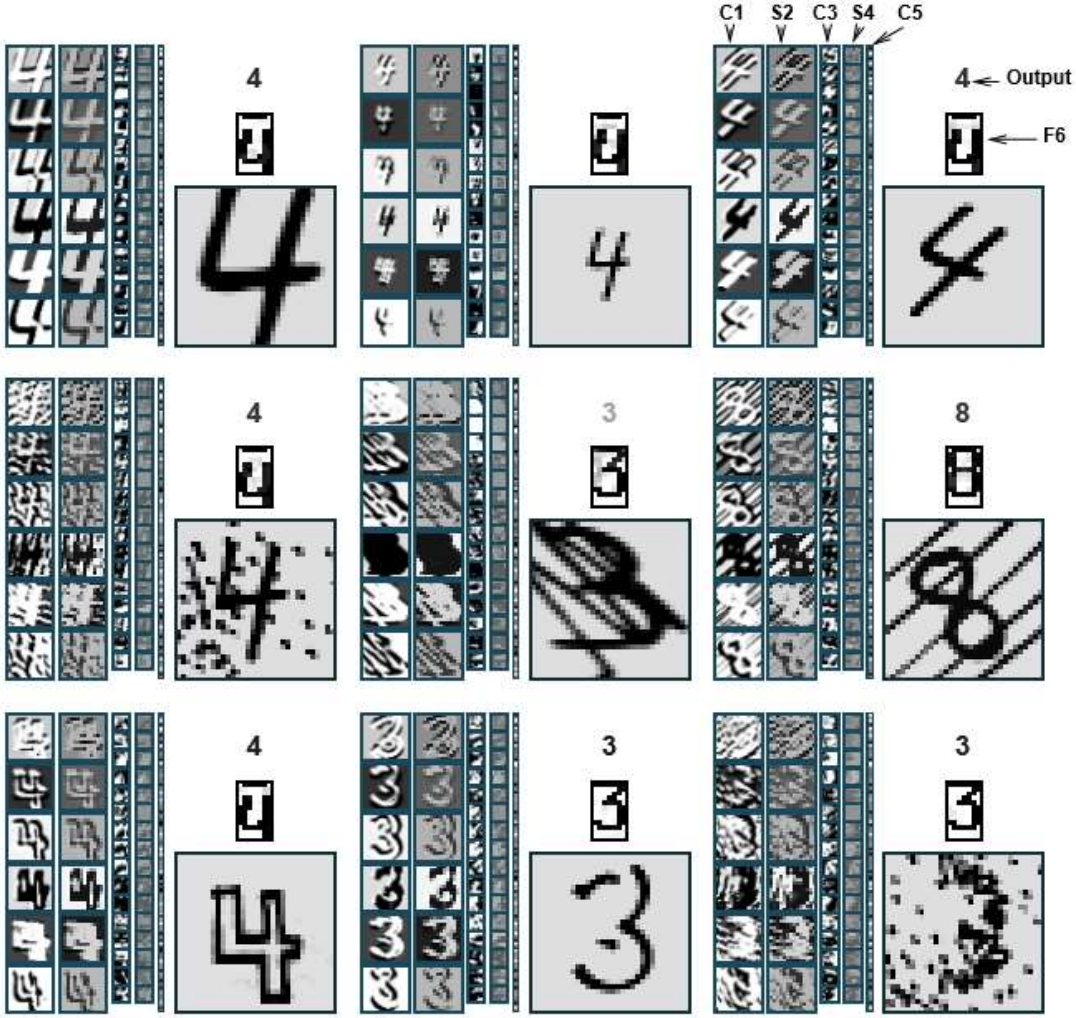


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).

9. Conclusions:

- Convolutional Neural Network: eliminate the need for hand-crafted feature extractors.
- Graph Transformer Networks: reduce the need for hand-crafted heuristics, manual labeling, and manual parameter tuning in document recognition systems.
- The back propagation algorithm: solved the credit assignment problem in multi-layer neural networks.
- The gradient-based learning procedure for GTN: solves the credit assignment problem in systems whose functional architecture dynamically changes with each new input.
- All steps of a document analysis system can be formulated as graph transformers.
- Gradient based learning applied to global discriminative loss function guarantees optimal classification and rejection.
- Feature extraction: Application of Gradient-Based Learning to Convolutional Neural Networks allows to learn appropriate features from examples.
- Segmentation and recognition: using Heuristic Over-Segmentation to generate and evaluate a large number of hypotheses.
- Training multi-module systems to optimize a global measure of performance.

MLE of a Gaussian random variable

The parameters of a Gaussian distribution are the mean (μ) and variance (σ^2). Given observations $x_1, \ldots, x_N$, the likelihood of those observations for a certain μ and σ^2 (assuming that the observations came from a Gaussian distribution) is

$$p(x_1, \ldots, x_N | \mu, \sigma^2) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ \frac{-(x_n - \mu)^2}{2\sigma^2} \right\} \tag{2}$$

and the log likelihood is

$$\mathcal{L}(\mu, \sigma) = -\frac{1}{2} N \log(2\pi\sigma^2) - \sum_{n=1}^{N} \frac{(x_n - \mu)^2}{2\sigma^2} \tag{3}$$

We can then find the values of μ and σ^2 that maximize the log likelihood by taking derivative with respect to the desired variable and solving the equation obtained. By doing so, we find that the MLE of the mean is

$$\hat{\mu} = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{4}$$

and the MLE of the variance is

$$\hat{\sigma^2} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \hat{\mu})^2 \tag{5}$$

http://www.cs.princeton.edu/courses/archive/spr08/cos424/scribe_notes/0214.pdf