Introduction to Deep Learning and Caffe

HE Shuncheng
hsc12@outlook.com

Tsinghua-Seagate Future Robotics Club
Association of Science and Technology of Automation

November 2, 2015

Contents

HE Shuncheng
hsc12@outlook.com

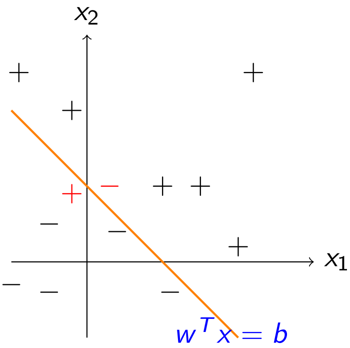
Binary Classification

Figure 1: a cat?

Figure 2: a dog?

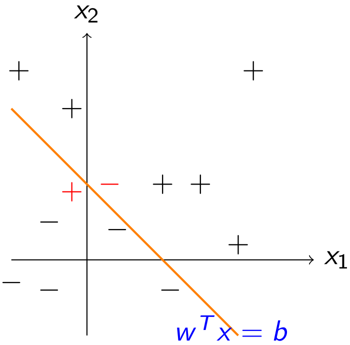
Binary Classification: Given input data x (e.g. a picture), the output of a binary classifier $y = f(x)$ is one label retrieved from a set of two labels $y \in \{\pm 1\}$.

Linear Classifier



Data set $\mathcal{D} = \{(x_1^{(1)}, x_2^{(1)}), \cdots, (x_1^{(n)}, x_2^{(n)})\}$
A linear binary classifier is a hyperplane $w^T x = b$
$f(x) = sgn(w^T x - b)$

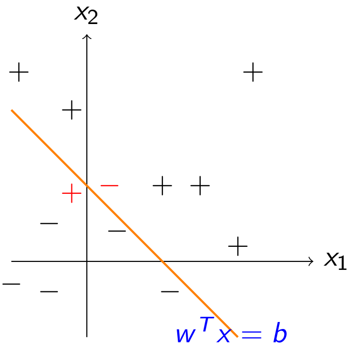
Performance of Linear Classifier



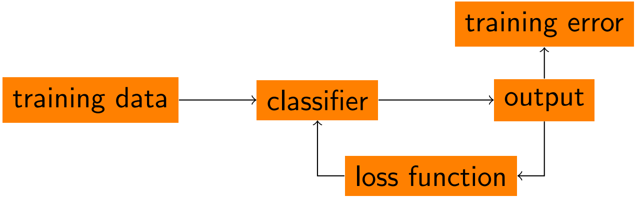
True Positive:
$y = +1, f(x) = +1$
True Negative:
$y = -1, f(x) = -1$
False Positive:
$y = -1, f(x) = +1$
False Negative:
$y = +1, f(x) = -1$
Accuracy:
$\frac{TP+TN}{n}$
Error Rate:
$\frac{FP+FN}{n}$

A good classifier: **minizing** the error rate

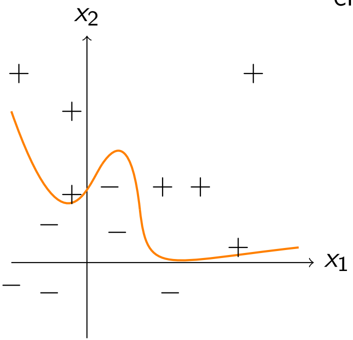
Basic Concepts

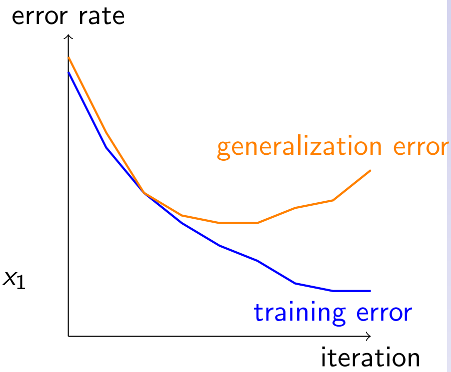


Training Set
Test Set
Training Error
Generalization Error
Overfitting
Loss Function

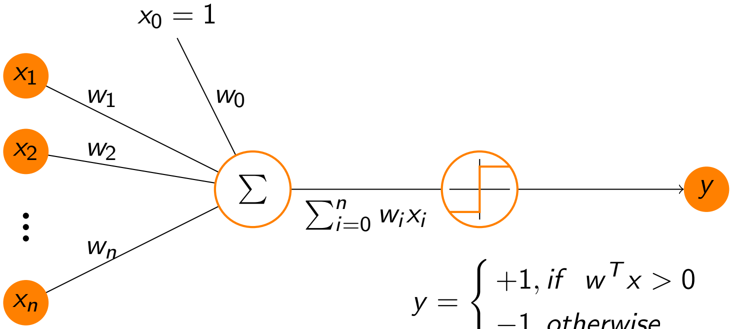


Overfitting



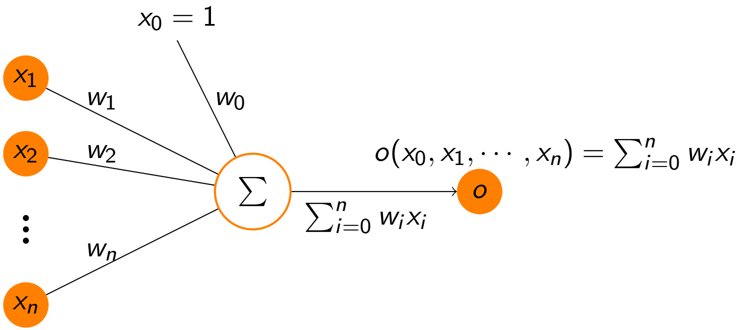


Perceptron



$$y = \begin{cases} +1, if \quad w^T x > 0 \\ -1, otherwise \end{cases}$$

Perceptron



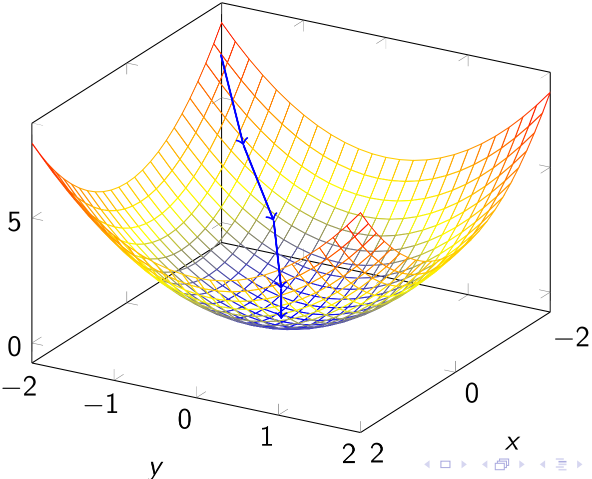
$x_0 = 1$
x_1
x_2
x_n
w_0
w_1
w_2
w_n
$\sum$
o
$\sum_{i=0}^{n} w_i x_i$
$o(x_0, x_1, \cdots, x_n) = \sum_{i=0}^{n} w_i x_i$

Training Algorithm

Define a loss function:

$$E(w) = \frac{1}{2} \sum_{d \in \mathcal{D}} (t_d - o_d)^2$$

How to reduce loss?



Gradient Descent

Gradient w.r.t. w

$$\nabla E(w) = (\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n})^T$$

in where

$$\frac{\partial E}{\partial w_i} = \sum_{d \in \mathcal{D}} (t_d - o_d)(-x_i^{(d)})$$

for every iteration (η denotes learning rate)

$$w_i \leftarrow w_i + \Delta w_i$$

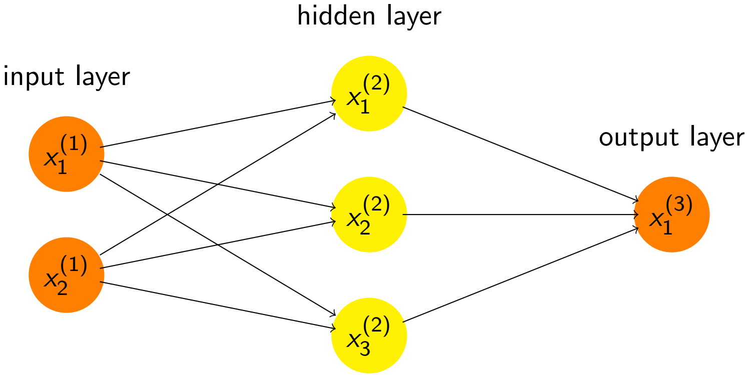
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d \in \mathcal{D}} (t_d - o_d) x_i^{(d)}$$

$$\forall i \in [n]$$

Artificial Neural Network

HE Shuncheng
hsc12@outlook.com

Structure of ANN

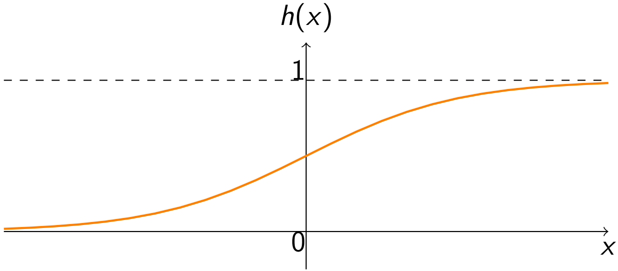


$$x^{l+1} = h(W^{T+1} x^l)$$

h is a non-linear function.

Sigmoid Function

$$h(x) = \frac{1}{1 + e^{-x}}$$



- 1. continuous
- 2. map $[-\infty, +\infty]$ to $[0, 1]$
- 3. nonlinearity
- 4. $h'(x)$ is easy to calculate

$$h'(x) = h(x)(1 - h(x))$$

Back Propagation and Delta Rule

HE Shuncheng
hsc12@outlook.com

Please refer to this page
Mathematical model of ANN

$$x^l = f(u^l), u^l = (W^{l-1})^T x^{l-1} + b^l$$

where l denotes the current layer with the output layer designated to be layer L and the input layer desiganted to ba layer 1. Function $f(\cdot)$ is a nonlinear function (i.e. sigmoid or hyperbolic tangent).
Define loss function as

$$E(x^L, t)$$

where x^L is the network output and t is the target output.

Back Propagation and Delta Rule

Expand the loss function

$$E(x^L, t) = E(f((W^{L-1})^T x^{L-1}), t)$$

Using chain rule, we can write the derivatives w.r.t. W^{L-1}

$$\frac{\partial E}{\partial W^{L-1}} = x^{L-1}(f'(u^L) \star \frac{\partial E}{\partial x^L})^T$$

where $\star$ denotes elementwise multiplication, and if we define

$$\delta^L = f'(u^L) \star \frac{\partial E}{\partial x^L}$$

we get

$$\frac{\partial E}{\partial W^{L-1}} = x^{L-1}(\delta^L)^T$$

HE Shuncheng
hsc12@outlook.com

Back Propagation and Delta Rule

If we calculate the δ term recursively

$$\delta^l = f'(u^l) \star ((W^l)^T \delta^{l+1}), l = L-1, \cdots, 2$$

it is easy to write

$$\frac{\partial E}{\partial W^l} = x^l(\delta^{l+1})^T, l = L-2, \cdots, 1$$

Network Structure

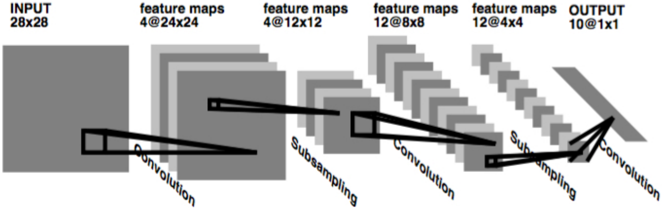
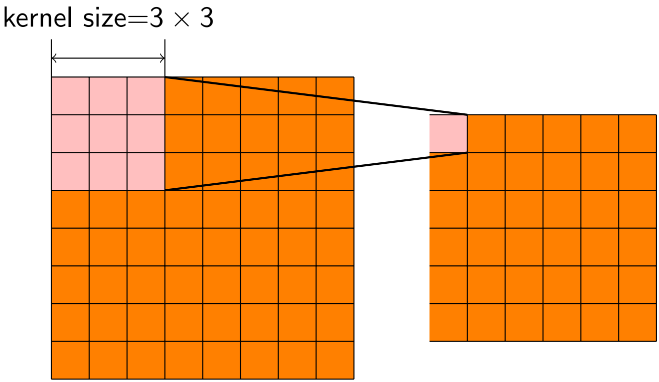


Figure 3: structure of convolutional neural network

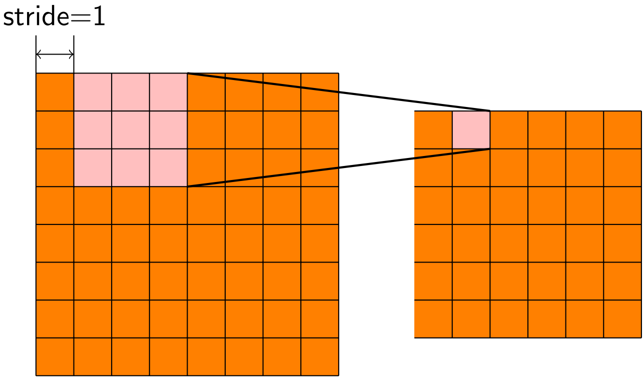
- Convolution Layer
- Pooling Layer (Subsampling)
- Full-connected Layer (Inner-product)
- ReLU Layer
- Softmax Layer

Convolution Layer



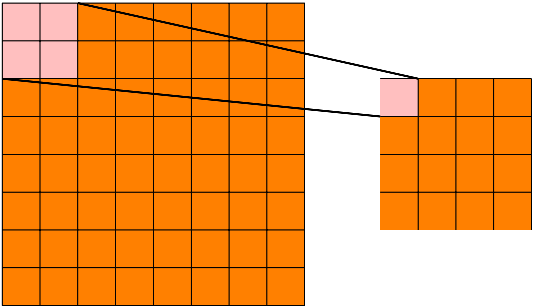
$$g_{ij} = \sum_{s=i}^{i+2} \sum_{t=j}^{j+2} h_{st} k_{st}$$

Convolution Layer



$$g_{ij} = \sum_{s=i}^{i+2} \sum_{t=j}^{j+2} h_{st} k_{st}$$

Pooling Layer



$$g_{ij} = \max\{h_{2i,2j}, h_{2i+1,2j}, h_{2i,2j+1}, h_{2i+1,2j+1}\}$$

No free parameter in pooling layer.

Inner-product

Known as full-connected layer. Weights are designated from every input to every output, namely

$$y = W^T x$$

Rectified Linear Unit

A rectifier

$$y = \max\{0, x\}$$

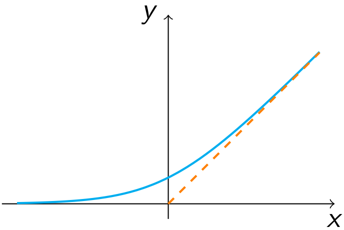
A rectified linear unit

$$y = ln(1 + e^x)$$

with its derivative w.r.t. x

$$\frac{dy}{dx} = \frac{1}{1 + e^{-x}}$$

ReLU improves efficiency of calculating.



Softmax

Derived from softmax regression, extension of logistic regression for multi-label classfication.

$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}, \forall i \in [n]$$

Outputs of softmax layer are probabilities of each label.

MNIST Database

MNIST: Mixed National Institute of Standards and Technology

Figure 4: Handwritten Digits

10 distinguishing classes

LeNet Review

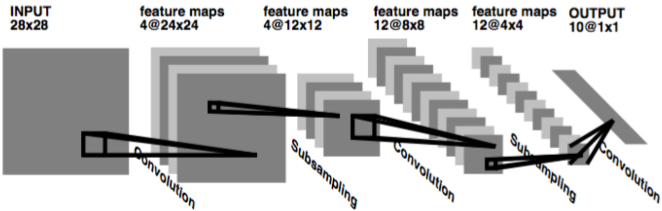


Figure 5: LeNet for MNIST

- input: a picture (size 28×28)
- conv1: 4 kernels (size 5×5)
- pool1: max pooling (size 2×2)
- conv2: 3 kernels (size 5×5)
- pool2: max pooling (size 2×2)
- ip: full-connected ($192 \rightarrow 10$)
- softmax: 10 inputs, 10 prob outputs

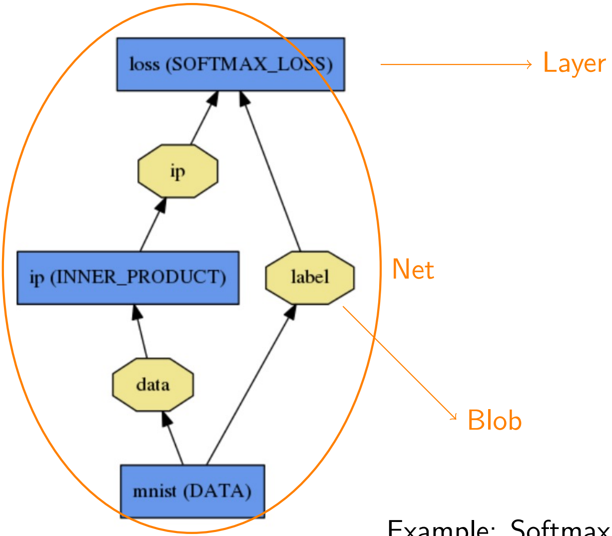
Caffe Tutorial

HE Shuncheng
hsc12@outlook.com

For more information please refer to this page.
Key words:

- Nets, Layers and Blobs
- Forward / Backward
- Loss
- Solver
- Layer Catalogue
- Interfaces
- Data

Nets, Layers and Blobs



Example: Softmax Regression

Forward / Backward

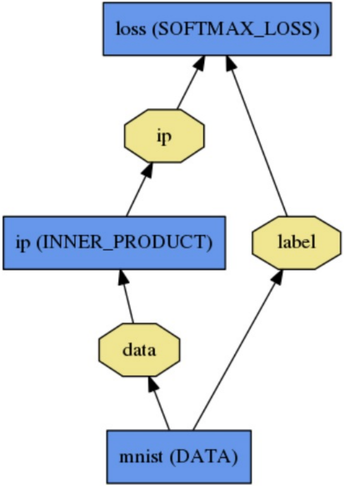
$f(w^T x)$ → $L(f(w^T x))$

$\frac{\partial L}{\partial f}$

$\frac{\partial L}{\partial w}$

$w^T x$

x



Loss

Softmax:

$$y_i(x) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}, \forall i \in [n]$$

Softmax loss function:
let label j be groundtruth, therefore

$$L = -ln(y_j(x)) = -ln(\frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}) = ln(\sum_{k=1}^{n} e^{x_k}) - x_j$$

$$\frac{\partial L}{\partial x_i} = y_i(x) - \delta_{ij}$$

where $\delta_{ij} = 1$ iff $i = j$, and $\delta_{ij} = 0$ otherwise.

Solver

SGD (Stochastic Gradient Descent)

$$
\begin{aligned}
w_{t+1} &= w_t + \Delta w_t \\
\Delta w_{t+1} &= \mu \Delta w_t - \alpha \frac{\partial L}{\partial w_t}
\end{aligned}
$$

α: learning rate

μ: momentum

Solver

Solver parameters (i.e.):

- basic learning rate: $\alpha = 0.01$
- learning rate policy: step (reduce learning rate according to step size)
- step size: 100000
- gamma: 0.1 (multipy learning rate with factor 0.1 after step size)
- momentum: $\mu = 0.9$
- max iteration: 350000 (stop at iteration 350000)

Layer Catalogue

Please refer to this page.

Vision layer:

- convolution
- pooling

Loss layer:

- softmax loss
- Euclidean loss
- cross-entropy

Activation layer:

- sigmoid
- ReLU
- hyperbolic tangent

Layer Catalogue

Data layer:

- datebase
- in-memory
- HDF5 input
- HDF5 output

Common layer:

- inner product
- splitting
- flatening
- reshape
- concatenation

Installation

Prerequisites:

protobuf, CUDA, OpenBLAS, Boost, OpenCV, lmdb, leveldb, cuDNN(optional), Python(optional), numpy(optional), MATLAB(optional)

Install:

git clone git://github.com/BVLC/caffe /your/own/caffe/folder

Go to Caffe root folder

cp Makefile.config.example Makefile.config
make all
make test
make runtest

Hardware:

K40, K20, Titan for ImageNet scale
GTX series or GPU-equipped MacBook Pro for small datasets

LeNet Example

LeNet Structure

1. **Protobuf Protocol**
2. **Run!**

How to be Professional?

1. Figure out theoretical keypoints (read papers)
2. Read Caffe source code
3. Be proficient at programming and debugging skills
4. Take advantage of search engine and community
5. Do it through this pipeline:
   - Experiment design
   - Data preparation (build database with tools)
   - Model selection (including network and solver)
   - Training
   - Analysis and comparison