



Caffe V1 Introduction

- Convolutional Architecture for Fast Feature Embedding -

2019 - 2020

Ando Ki, Ph.D.

adki@future-ds.com

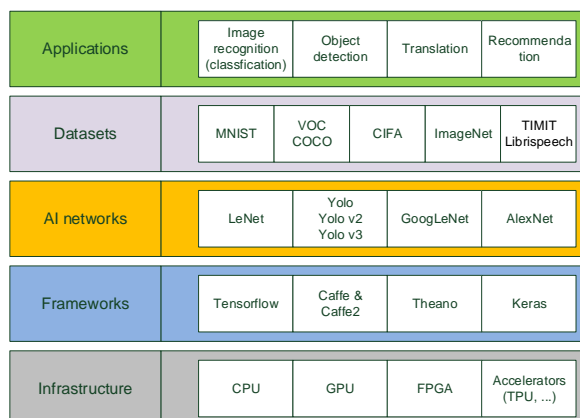
Contents

- Deep learning hierarchy
- Deep learning frameworks

- Related topics
 - ▶ Google protocol buffer (protobuf)
 - ▶ LMDB

- What is Caffe
- Caffe: blob, layer, net, solver

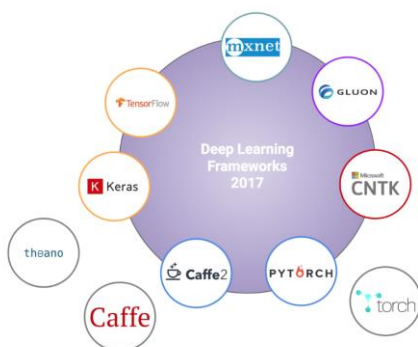
Deep Learning Hierarchy



- **Data set**
 - ▶ A collection of data to be used for AI training, validation, and testing.
- **AI networks**
 - ▶ Artificial neural network
- **Frameworks and libraries**
- **Infrastructure**

(3)

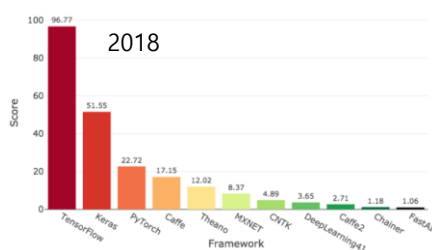
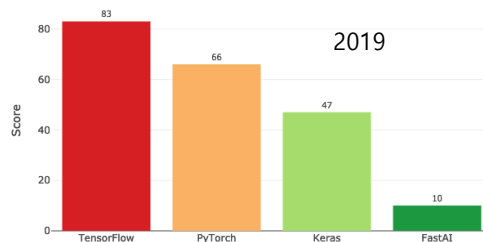
Deep learning frameworks



- **Caffe V1**
 - ▶ Berkeley / BVLC (Berkeley Vision and Learning Center)
 - ▶ BAIR (Artificial Intelligence Research)
 - ▶ C++, Python, Matlab
- **TensorFlow**
 - ▶ Google Brain
 - ▶ C++, Python
- **Caffe V2 → PyTorch**
 - ▶ Facebook
- **theano**
 - ▶ U. Montreal
 - ▶ Python
- **torch**
 - ▶ Facebook / NUU
 - ▶ C, C++, Lua
- **CNTK**
 - ▶ Microsoft
- **MXNet**
 - ▶ Carnegie Mellon University / DMLC (Distributed Machine Learning Community)

(4)

Deep learning frameworks



(5)

Related topics

■ Protocol buffer (protobuf)

- ▶ Protobuf is a data structure format in short.
- ▶ Caffe uses protobuf to describe network (i.e., layer, net, solver)

■ Lightning memory-mapped database (LMDB)

- ▶ database in the form of a key-value store.
- ▶ Caffe uses LMDB to store training data, which is huge.

■ HDF

- ▶ Hierarchical Data Format (HDF) is an open source file format for storing huge amounts of numerical data.
- ▶ HDF5 data requires two files.
 - .h5 file containing data and label
 - .txt file specifying path to the .h5 file(s)

(6)

Protocol buffer (Protobuf)

- There are many ways to capture structured data to serialize.

- ▶ XML (eXtensible Markup Language)
- ▶ JSON (JavaScript Object Notation)
- ▶ Protobuf (Protocol Buffer)

```
<!-- XML example -->
<person>
  <name>Kil-Dong Hong</name>
  <age>30</age>
  <contacts>
    <email>kdhong@email.com</email>
    <phone>111-222-333</phone>
  </contacts>
</person>

// JSON example
person {
  name: "Kil-Dong Hong"
  age: 30
  contacts: {
    email: kdhong@email.com
    phone: "111-222-333"
  }
}
```

(7)

Protocol buffer (Protobuf)

■ Protobuf

- ▶ A [message](#) is just an aggregate containing a set of [typed fields](#). (bool, int32, float, double, and string) and structures.
- ▶ Each element has its unique identifier, i.e., '[tag](#)'. ('=1', '=2', ...)
- ▶ Field modifiers (required, optional, repeated)
- ▶ Accessor methods (for each field)
 - ➡ foo(), set_foo(), get_foo(), has_foo(), clear_foo(), ...
- ▶ comment: '//'

```
// protobuf example (person.proto)
message Person {
  required string name = 1;
  optional string email = 2;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

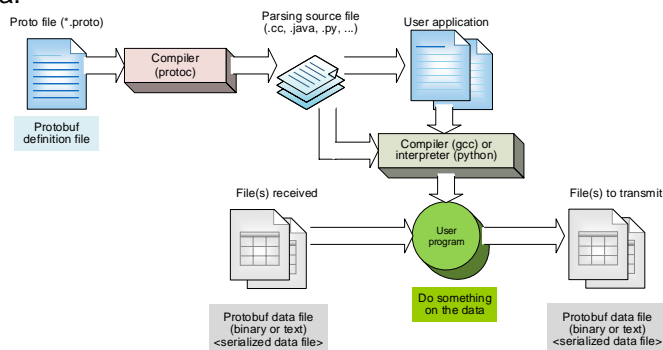
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 3;
}
```

(8)

Protobuf

- Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.



- The **.proto** file is used to describe the structure (the 'protocol') of the data to be serialized. The protobuf compiler can turn this file into python/or C++/or Java code to serialize and deserialize data with that structure. (use `//` for comments.)
- The **.prototxt** file is serialized file in text instead of binary. (use `#` for comments.)
- The **.binaryproto** or **.protobin** file is serialized file in binary format.

(9)

Protobuf

```

adki@AndoUbuntu: ~/work/seminars/20180110_DeepLe
[adki@AndoUbuntu] ls
main.cpp Makefile person.proto*
[adki@AndoUbuntu] make
protoc --cpp_out=out person.proto
g++ -o test main.cpp out/person.pb.cc \
    -Iout `pkg-config --cflags --libs protobuf`
[adki@AndoUbuntu] ls
main.cpp Makefile out/ person.proto* test*
[adki@AndoUbuntu] ls out
person.pb.cc person.pb.h
[adki@AndoUbuntu] make run
./test
Name: Kil-Dong Hong
E-mail: kdhong@email.com
[adki@AndoUbuntu]
  
```

- refer to following program and files

- ▶ `protoc`
 - ⇒ Google protocol buffer compiler
- ▶ `out/person.pb.{h,cc}`
 - ⇒ parsing program
- ▶ `myfile`
 - ⇒ serialized data file

To install protocol buffer compiler (**protoc**).

\$ `sudo apt-get install libprotobuf-dev libleveldb-dev libsnapappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler`

```

#include <iostream>
#include <iostream>
#include "person.pb.h"

using namespace std;

int main(int argc, char *argv[])
{
    Person personA;
    personA.set_name("Kil-Dong Hong");
    personA.set_email("kdhong@email.com");

    //write binary
    fstream output("myfile.pb.bin", ios::out | ios::binary);
    personA.SerializeToOstream(&output);
    output.close();

    Person personB;
    //read binary
    fstream input("myfile.pb.bin", ios::in | ios::binary);
    personB.ParseFromStream(&input);
    cout << "Name: " << personB.name() << endl;
    cout << "E-mail: " << personB.email() << endl;
    input.close();

    return 0;
}
  
```

It handles binary form of protobuf file.

(10)

Running protobuf example

■ This example shows how to use compile Tiny-Dnn program

- ▶ Step 1: go to your project directory
 - ➔ [user@host] cd \$(PROJECT)/codes/caffe_v1-projects/protobuf_person
- ▶ Step 2: see the codes
- ▶ Step 3: compile
 - ➔ [user@host] make
- ▶ Step 4: run
 - ➔ [user@host] make run

```
[user@host] cd $(PROJECT)/codes/caffe_v1-projects/protobuf_person
[user@host] make
[user@host] make run
```

(11)

Protobuf: handling .prototxt

```
#include <fstream>
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <google/protobuf/io/coded_stream.h>
#include <google/protobuf/io/zero_copy_stream_impl.h>
#include <google/protobuf/text_format.h>
#include "person.pb.h"
```

It handles textual form of protobuf file.

```
using namespace std;
using google::protobuf::io::FileInputStream;
using google::protobuf::io::FileOutputStream;
using google::protobuf::io::ZeroCopyInputStream;
using google::protobuf::io::CodedInputStream;
using google::protobuf::io::ZeroCopyOutputStream;
using google::protobuf::io::CodedOutputStream;
```

```
int main(int argc, char *argv[])
{
    GOOGLE_PROTOBUF_VERIFY_VERSION;
```

```
    Person personA;
    personA.set_name("Kil-Dong Hong");
    personA.set_email("kdhong@email.com");
```

```
//write textual file
int fd = open("myfile.prototxt",
O_WRONLY|O_CREAT|O_TRUNC, 0644);
FileOutputStream *output = new FileOutputStream(fd);
google::protobuf::TextFormat::Print(personA, output);
delete output;
close(fd);
```

```
Person personB;
//read textual file
fd = open("myfile.prototxt", O_RDONLY);
FileInputStream *input = new FileInputStream(fd);
google::protobuf::TextFormat::Parse(input, &personB);
delete input;
close(fd);
cout << "Name: " << personB.name() << endl;
cout << "E-mail: " << personB.email() << endl;
```

```
    return 0;
}
```

12

Running protobuf example

■ This example shows how to use compile Tiny-Dnn program

- ▶ Step 1: go to your project directory
 - ➔ [user@host] `cd $(PROJECT)/codes/caffe_v1-projects/protobuf_person_text`
- ▶ Step 2: see the codes
- ▶ Step 3: compile
 - ➔ [user@host] `make`
- ▶ Step 4: run
 - ➔ [user@host] `make run`

```
[user@host] cd $(PROJECT)/codes/caffe_v1-projects/protobuf_person_text
[user@host] make
[user@host] make run
```

(13)

LMDB

■ LMDB

- ▶ Lightning memory-mapped database
- ▶ database in the form of a key-value store.
- ▶ A tiny database with fast search and cheap read transactions with concurrent reads
- ▶ Memory mapped, allowing for zero copy lookup and iteration

■ Recall that

- ▶ Training involves huge number of iterations for the same data.
- ▶ So read-efficient database is required

■ LMDB creates two files

- ▶ *lock.mdb*
 - ➔ to synchronize data between different readers
- ▶ *data.mdb*

■ Caffe also uses Google **LevelDB**.

- ▶ LevelDB is an open-source on-disk key-value store.

■ Caffe also uses **HDF5** file format.

- ▶ Hierarchical Data Format (HDF) is an open source file format for storing huge amounts of numerical data.
- ▶ HDF5 data requires two files.
 - ➔ .h5 file containing data and label
 - ➔ .txt file specifying path to the .h5 file(s)

(14)

Contents

- Deep learning hierarchy
- Deep learning frameworks
- Google protocol buffer (protobuf)
- What is Caffe
- Caffe: blob, layer, net, solver

(15)

What is Caffe

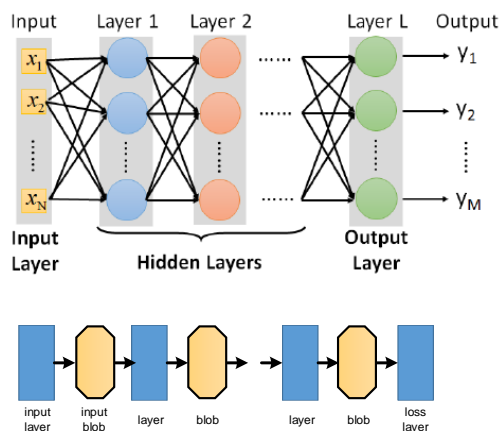
- Caffe is a deep learning framework.
 - ▶ Convolutional Architecture for Fast Feature Embedding
 - ▶ It is written in C++, with a Python interface.
 - ▶ models and optimizations are defined as plaintext schema instead of code. (i.e., prototxt)
- Yangqing Jia created the project during his PhD at Berkeley AI Research (BAIR).
- BSD 2-Clause license

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Main classes
 - ▶ **Blob** is the standard array and unified memory interface for the framework.
 - ▶ **Layer** is the foundation of both model and computation.
 - ▶ **Net** is the collection and connection of layers.
 - ▶ **Solver** is the model optimization.
- Caffe develop steps
 - ▶ *Step 1: Data preparation*
 - ▶ *Step 2: Model definition* - parameters in a configuration file with extension .prototxt.
 - ▶ *Step 3: Solver definition* - solver parameters in a configuration file with extension .prototxt.
 - ▶ *Step 4: Model training* - trained model in a file with extension .caffemodel.
 - ▶ *Step 5: Prediction* (inference, deploy) – use trained model (.caffemodel) to make predictions of new data.

(16)

Caffe

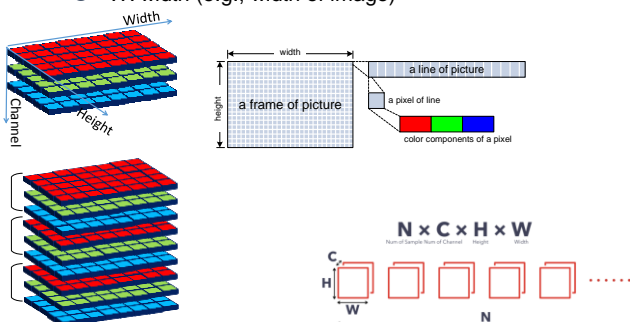


(17)

Caffe: blob

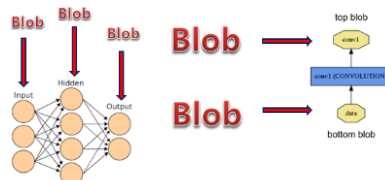
Store data in memory

- C-contiguous fashion (row-major)
- CxHxW convention
 - ⇒ C: channel (e.g., R, G, B)
 - ⇒ H: height (e.g., height of image)
 - ⇒ W: width (e.g., width of image)



Blob is the standard array and unified memory interface for the framework.

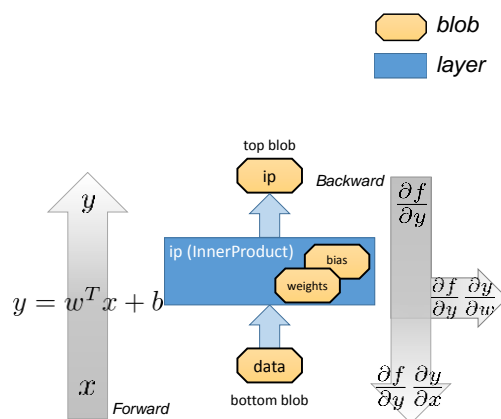
- Caffe stores and communicates data using blobs.
 - ⇒ Caffe 내에서 데이터 저장과 레이어 간의 통신을 위한 구조체
- N-Dimensional arrays for stored in a C-contiguous fashion (row-major) and communicating data
- Blobs provide a unified memory interface holding data
- Blob size: $N \times C \times H \times W$



(18)

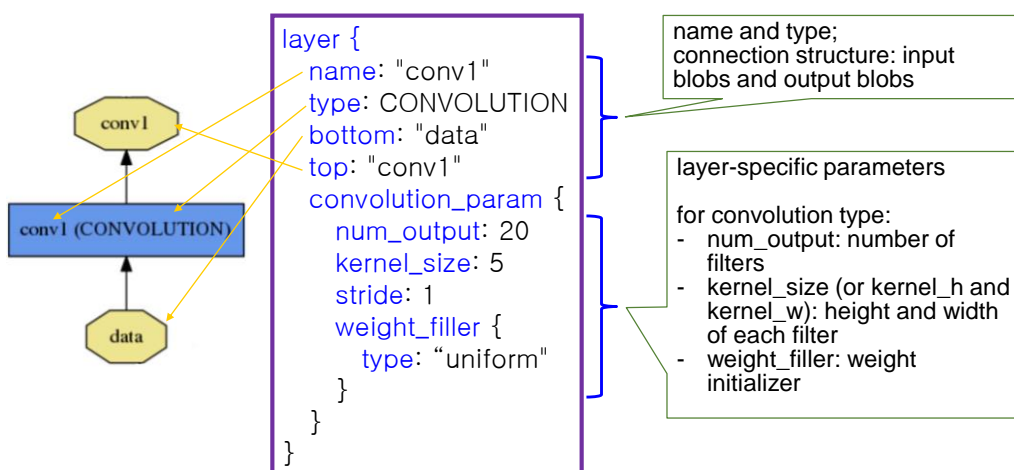
Caffe: layer

- **Layer** is the foundation of both model and computation.
 - ▶ Caffe's fundamental unit of computation
 - ▶ Transforms bottom blobs to top blobs
 - ▶ 네트워크 구성에 필요한 각종 요소들이 미리 준비되어 있음
 - ➡ Data access, convolution, pooling, activation functions, loss functions, drop out, and so on
 - ▶ Each layer type defines three computations: setup, forward, and backward.
 - ▶ Blob should be used between layers.



(19)

Caffe: layer



(20)

Caffe: layers

■ Data Layers

- ▶ Data can come from efficient databases (LevelDB or LMDB), directly from memory, or, when efficiency is not critical, from files on disk in HDF5 or common image formats.
- ▶ Has common input preprocessing (mean subtraction, scaling, random cropping, and mirroring)

■ Common Layers

- ▶ Various commonly used layers, such as: Inner Product, Reshape, Concatenation, Softmax, ...

■ Neuron Layers

- ▶ Neuron layers are element-wise operators, taking one bottom blob and producing one top blob of the same size.
- ▶ ReLU, Sigmoid, tanh, Dropout, ...

■ Vision Layers

- ▶ Vision layers usually take images as input and produce other images as output.
- ▶ convolutional layer & deconvolutional (convolution transpose) layer
- ▶ Pooling layer

■ Loss Layers

- ▶ Loss drives learning by comparing an output to a target and assigning cost to minimize. The loss is computed by the forward pass.
- ▶ SoftmaxWithLoss layer, EuclideanLoss layer

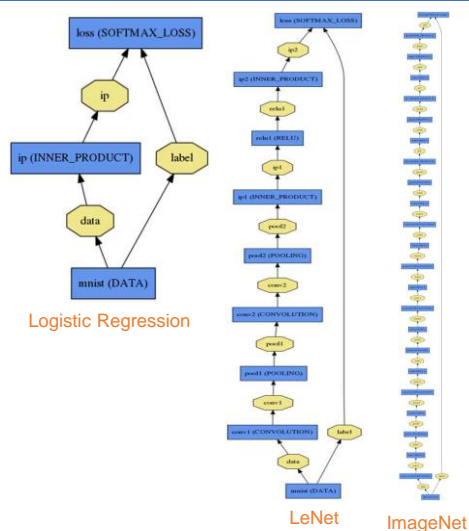
(21)

Caffe: net

■ Net is the collection and connection of layers.

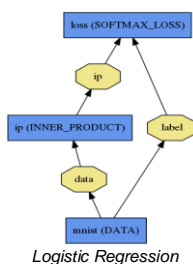
- ▶ 레이어로 구성된 네트워크 (Directed Acyclic Graph)
- ▶ Many layers
- ▶ computes gradients via Forward / Backward

```
name: "mynetwork"
layers { name: "data" ... }
layers { name: "conv" ... }
layers { name: "pool" ... }
... more layers ...
layers { name: "loss" ... }
```

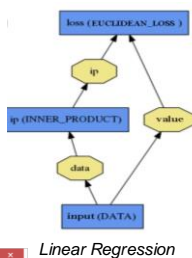


(22)

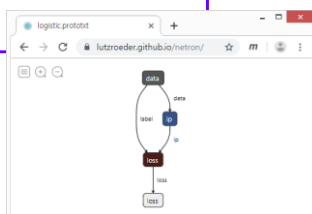
Caffe: net



```
name: "simple logistic regression classifier"
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include { phase: TRAIN }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param { num_output: 2 }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```



```
name: "LinearReg"
layer {
  name: "input"
  type: "Data"
  top: "data"
  top: "value"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param { num_output: 1 }
}
layer {
  name: "loss"
  type: "EuclideanLoss"
  bottom: "ip"
  bottom: "value"
  top: "loss"
}
```



(23)

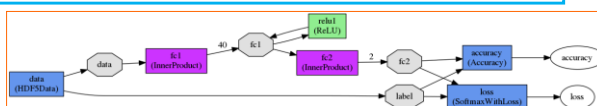
Visualizing Caffe network

- Visualize Caffe network prototxt file as DAGs
- Install dependencies
 - ▶ \$ sudo pip install pydot
 - ▶ \$ sudo apt-get install graphviz
- Add following to your bash startup (.bashrc) at the home

```
if [ -n "${PATH}" ]; then
  export PATH=${HOME}/caffe_v1/caffe/build/tools:${HOME}/caffe_v1/caffe/python:${PATH}
else
  export PATH=${HOME}/caffe_v1/caffe/build:${HOME}/caffe_v1/caffe/python
fi
```

- Run as follows

```
$ draw_net.py xor_net.prototxt xor_net.png
$ display xor_net.png
```



(24)

Caffe: solver

■ Solver is the model optimization

- ▶ Orchestrated model optimization by coordinating the network's forward interface and backward gradients
- ▶ Uses gradients to update weights
 - ⇒ Calls forward/backward and updates net parameters
- ▶ Calls forward/backward and updates net parameters
- ▶ Periodically evaluates model on the test network(s)
- ▶ Snapshots model and solver state
- ▶ Solvers available
 - ⇒ SGD, AdaDelta, AdaGrad, Adam, Nesterov, RMSprop

Solver specifies the net to apply optimization for train.

```
# solver prototxt file example
train_net: "lenet_train.prototxt"
base_lr: 0.01 # begin training at a learning rate of 0.01
lr_policy: "step"
gamma: 0.1 # drop the learning rate
stepsize: 100000 # drop the learning rate every 100K iterations
max_iter: 350000
momentum: 0.9
snapshot_prefix: "lenet_snapshot"
solver_mode: CPU
```

(25)

Caffe: solver

- | | |
|--|---|
| ■ net: Proto filename for the train net, possibly combined with test net | ■ base_lr: initial learning rate |
| ■ display: the number of iterations between displaying info | ■ lr_policy: "fixed" = always 'base_lr' |
| ■ max_iter : The maximum number of iterations | ■ lr_policy: "step" = start at 'base_lr' and after each 'stepsize' iterations reduce learning rate by 'gamma' |
| ■ solver_mode: the mode solver will use: CPU or GPU | ■ lr_policy: "inv" = start at 'base_lr' and after each iteration reduced learning rate some equation |
| ■ test_iter: The number of iterations for each test net | ■ momentum: weight of the previous update |
| ■ test_interval: The number of iterations between two testing phases | ■ weight_decay |
| | ■ snapshot: |
| | ■ snapshot_prefix: |

(26)

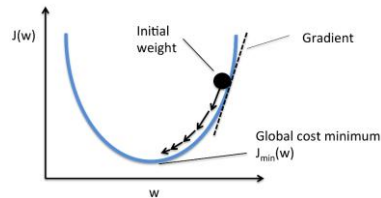
Solver methods

- The solver orchestrates model optimization by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss.

- ▶ Stochastic Gradient Descent (type: "SGD"),
- ▶ AdaDelta (type: "AdaDelta"),
- ▶ Adaptive Gradient (type: "AdaGrad"),
- ▶ Adam (type: "Adam"),
- ▶ Nesterov's Accelerated Gradient (type: "Nesterov") and
- ▶ RMSprop (type: "RMSProp")

- SGD: Stochastic Gradient Descent

- ▶ 확률적 경사하강법



$$V_{t+1} = \mu V_t - \eta \nabla L(W_t)$$

$$W_{t+1} = W_t + V_{t+1}$$

W_t : Weight at time t (i.e., at iteration t)

V_t : Weight updated at time t

$\nabla L(W_t)$: negative gradient of Weight at time t

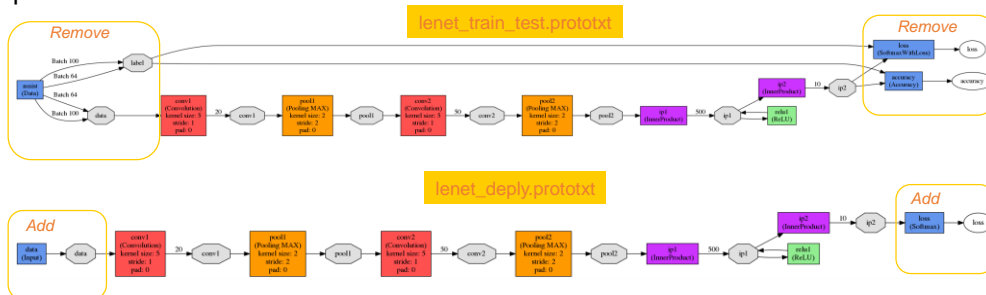
η : Learning rate (weight of negative gradient)

μ : Momentum (weight of previous update)

(27)

Deploy prototxt

- Remove input data layer and replace with a description of input data dimension
 - ▶ Remove the data layer that was used for training, as for in the case of classification we are no longer providing labels for our data.
 - ▶ Remove any layer that is dependent upon data labels.
 - ▶ Set the network up to accept data.
- Remove "loss" and "accuracy" layers and replace with an appropriate layer to have the network output the result.



(28)

(주)퓨처디자인시스템

34051 대전광역시 유성구 문지로 193, KAIST 문지캠퍼스, F723호
(042) 864-0211~0212 / contact@future-ds.com / www.future-ds.com

Future Design Systems, Inc.

Faculty Wing F723, KAIST Munji Campus, 193 Munji-ro, Yuseong-gu, Daejeon 34051, Korea
+82-042-864-0211~0212 / contact@future-ds.com / www.future-ds.com



FUTURE
Design Systems