A tutorial for DNN classifier.

Yosuke Kashiwagi

What is this script ?

- This script provides a simple classifier using Deep Neural Networks (DNNs).
  - Based on KALDI and PDNN.
  - In this script, MNIST database is used.
  - MNIST : Hand written digit recognition task.
    http://yann.lecun.com/exdb/mnist/

What is KALDI ?

- A toolkit for speech recognition.
  - http://www.danielpovey.com/kaldi-docs/index.html
  - written by Dan Povey's team.
  - licensed under the Apache License v2.0.
- You can use KALDI on elf or valkyrie machine.

What is PDNN ?

- A python toolkit for deep learning.
  - https://www.cs.cmu.edu/~ymiao/pdnntk.html
  - written by Yajie Miao.
  - licensed under the Apache License v2.0.

- PDNN is not installed in elf and valkyrie.

How to use ?

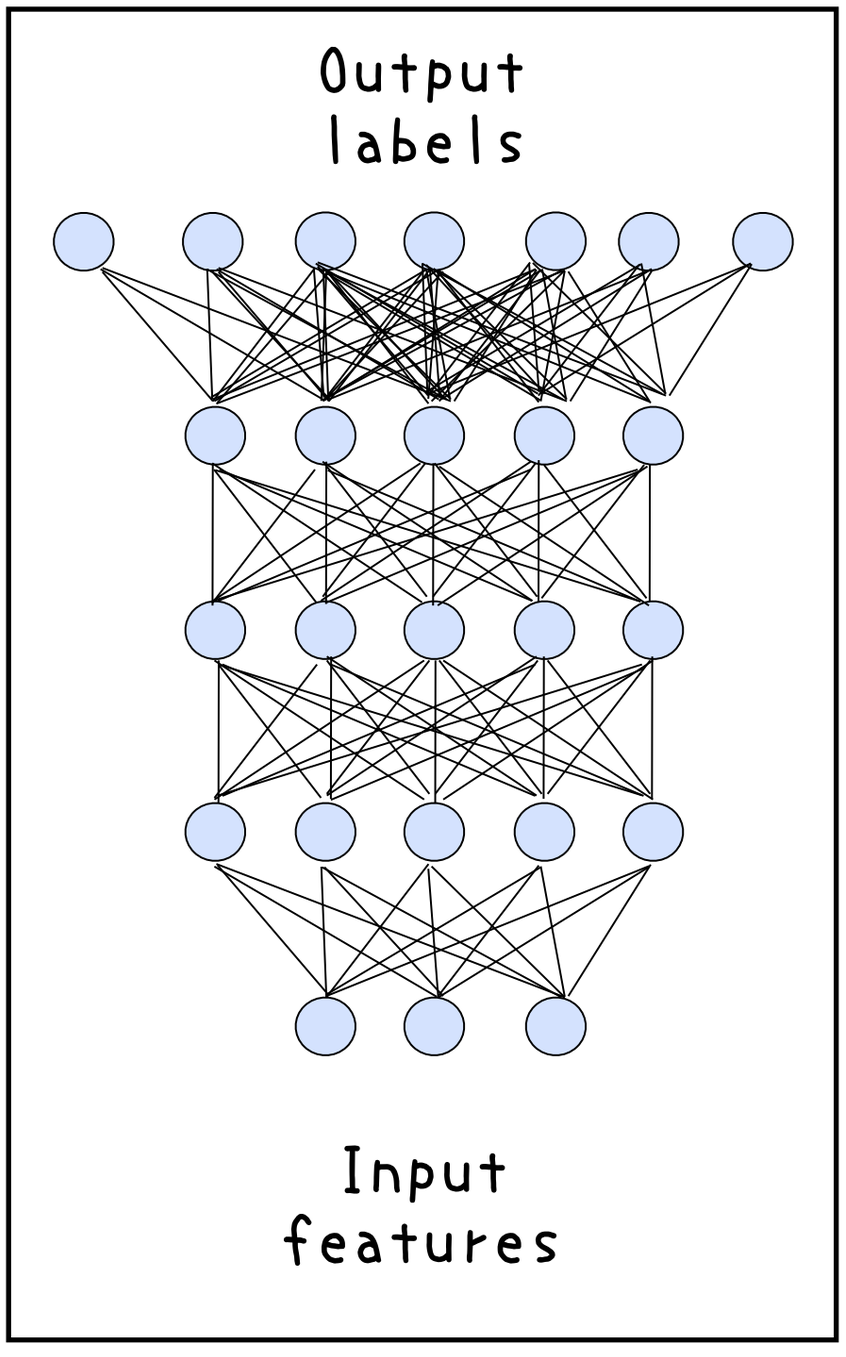
- Get the scripts and type “./run.sh” !

```
$ git clone /home/kashiwagi/share/classifier
$ cd classifier
$ ./run.sh
```

Finish !

A brief introduction of DNN

- Proposed by Hinton in 2006.
  - Multi-layer network.
  - **“Pre-training”** is the key idea.
    - Mitigate serious problems (**local minimum** and **over-fitting**)
- DNN marked the best performance in many challenges.



DNN for speech recognition

- DNN also achieved the best performance in speech recognition tasks.
  - Hybrid model (DNN + Hidden Markov Model)

$$\hat{W} = \operatorname{argmax} \log P(\boldsymbol{O}|W)P(W)$$

$$P(\boldsymbol{O}|W) \stackrel{\text{def}}{=} \sum_{\boldsymbol{s}} \left\{ \prod_{t} Q(s_t|\boldsymbol{o}_t, \boldsymbol{\Lambda}) \right\} P(\boldsymbol{O})P(\boldsymbol{s}|W)$$

$$Q(s_t|\boldsymbol{o}_t, \boldsymbol{\Lambda}) = \frac{p(s_t|\boldsymbol{o}_t, \boldsymbol{\Lambda})}{P(s_t)}$$

This scripts do not use HMMs.

Keywords for recent DNNs

Please Google !

- Activation function
  - ReLU
  - **Maxout**
- Network topology
  - **Convolutional Neural Network**
  - Recurrent Neural Network
    - LSTM

Step 0 : Setup

- **Setup KALDI, PDNN and original MNIST database.**
  - Download and install. Maybe, this step takes a long time :(
    - tools/
    - kaldi-maxout/ : contains kaldi scripts.
      (**nnet-forward** is modified to use maxout)
    - pdnn/ : contains PDNN scripts
    - quicknet/, pfile_utils/ : for using PFile (data format)

Step 1 : Data preparation

- **Prepare the database for training.**
  - Convert the binary data to text data.
    - features : 28x28=784 dim. (0 ~ 255)
    - labels : 1 dim. (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

```
$ head data/train/train-images.txt
$ head data/train/train-labels.txt


$ head data/test/test-images.txt
$ head data/test/test-labels.txt
```

Step 2 : Make PFile

- **Make PFile for DNN training.**
  - Input data format (data/train_tr90/train_tr90.data)
    - 1 dim. : Sample ID
    - 2 dim. : Frame ID (Dummy in this task.)
    - 3~786 dim. : Features
    - 787 dim. : Label

```
$ tools/pfile_utils-v0_51/bin/pfile_create -i data/train_tr90/train_tr90.data -o data/train_tr90/train_tr90.pfile -f 784 -l 1
```

"train_cv10" is a set of subdata for cross-validation

Step 3 : Train DNN

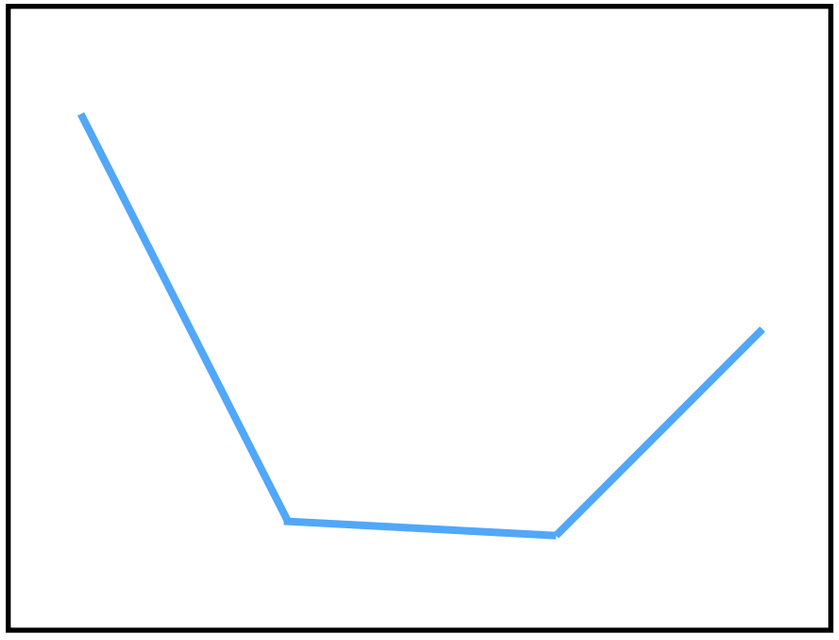
Train DNN.

```
$ python tools/pdnn/cmds/run_DNN.py --train-data data/train_tr90/train_tr90.pfile,partition=10m,random=true,stream=true \
    --valid-data data/train_cv10/train_cv10.pfile,partition=10m,random=true,stream=true \
    --nnet-spec 784:500:500:10 --activation maxout:2 --lrate D:0.0008:0.5:0.01,0.01:8 \
    --wdir dnn/ --kaldi-output-file dnn/nnet
```

- DNN topology (—nnet-spec)
  - 784 : 500 : 500 : 10
    (input feats : hidden layer nodes 1 : hidden layer nodes 2 : output labels)
- Activation function (—activation)
  - maxout:2 (maxout function, 2 represent the # of linear components)
- Learning rate (—lrate)
  - D:0.0008 : 0.5 : 0.001,0.001 : 8
    (Init rate : halving factor : halving start, halving end : minimum epoch)

Step 3 : Train DNN

- **Using maxout activation functions . . .**
  - DNNs can be trained well **without pre-training**.
  - Fast !
  - But they take a large memory.



- **Of course, the final layer is softmax function.**

Step 4 : Calculate posterior probability

- **Calculate posterior probability.**
  - To use **nnet-forward**, we must modify the test data.
    - data/test/test-images.feats

DataID [feats …]
:
:

DataID must be unique !

Step 4 : Calculate posterior probability

- **Posterior probability**
  - data/test/test-images.posterior

```
1  [
  0 2.532356e-14 1.248106e-13 3.016724e-08 2.409788e-16 1.380036e-15 1.44216e-23 1 1.094789e-14 1.118839e-11 ]
2  [
  0 1.76089e-14 1 2.268198e-14 1.964306e-25 7.076257e-15 2.157866e-14 9.311794e-29 1.154093e-22 1.335993e-33 ]
3  [
  0 0.9999992 2.298187e-07 3.534052e-10 2.745042e-07 7.564024e-13 1.853817e-07 1.404711e-07 7.574138e-11 1.100623e-13 ]
4  [
  1 5.902696e-31 3.358515e-21 2.345248e-28 8.341328e-32 5.443353e-26 2.393071e-23 1.087154e-25 3.199901e-31 1.130879e-26 ]
```

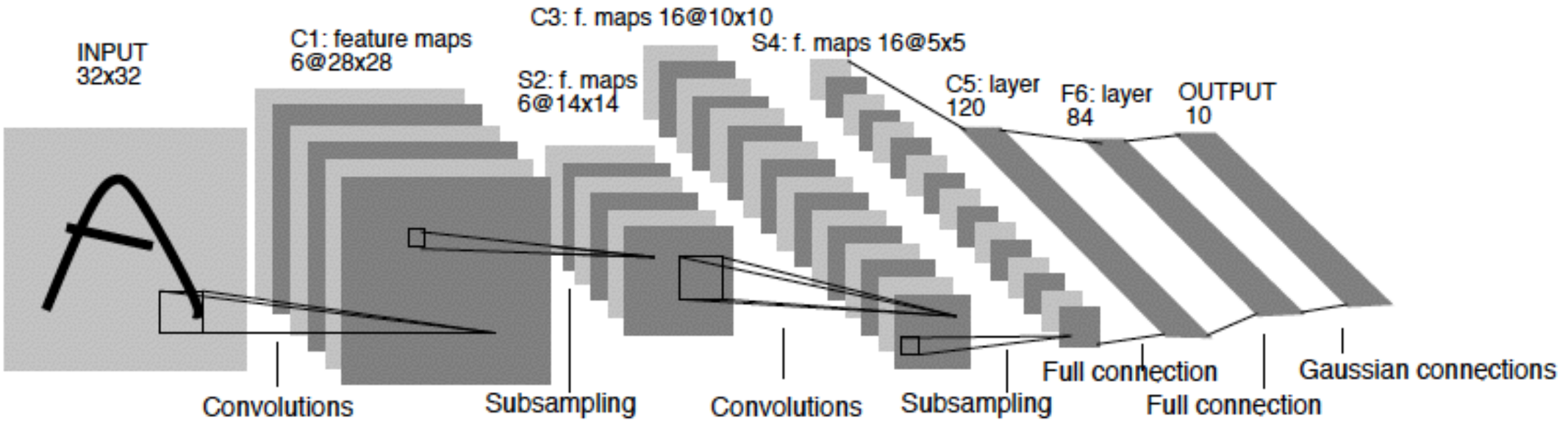
DataID must be unique !

Step 5 : Calculate scores

- **Calculate scores.**
  - Compare the estimated labels and the correct labels.
    - append-feats
      : joint the feats (or labels), like a “paste” in bash
    - calc_score.py
      : calculates the correct rate using the joint input of the posterior probabilities and the correct labels.

Step 6 : Train CNN

- Train a Convolutional Neural Network (CNN).



[LeCun, 1998]

- Complex deep architecture
  (feature maps, convolution filter, pooling layer)

Step 6 : Train CNN

Train a Convolutional Neural Network (CNN).

```
$ python tools/pdnn/cmds/run_CNN.py --train-data data/train_tr90/train_tr90.pfile,partition=10m,random=true,stream=true \
      --valid-data data/train_cv10/train_cv10.pfile,partition=10m,random=true,stream=true \
      --conv-nnet-spec "1x28x28:256,9x9,p2x2,f" \
      --nnet-spec "300:10" \
      --lrate "D:0.08:0.5:0.2,0.2:4" --momentum 0.9 \
      --wdir cnn/ --param-output-file cnn/nnet.param \
      --cfg-output-file cnn/nnet.cfg --kaldi-output-file cnn/cnn.nnet
```

- convolution network topology (—conv-nnet-spec)
  - 1x28x28:256,9x9,p2x2,f
    (input feats : hidden layer nodes, filter size, pooling layers)
- momentum (—momentum)
  - control the training speed

Step 7 : Calculate scores (CNN)

- **Calculate scores (CNN)**
  - Almost same as DNNs.
  - **KALDI does not support CNN.**
    - First, forward through the convolutional layer and create the converted feature files.

```
$ python tools/pdnn/cmds2/run_CnnFeat.py --in-scp-file data/test/test-images.scp --out-ark-file data/test/test-images.conv.forward
  --cnn-param-file cnn/nnet.param --cnn-cfg-file cnn/nnet.cfg
```

To increase the accuracy …

- **The simplest way is to use more powerful networks.**
  - CNN, RNN, …
- Features
  - context expansion, law features,
  - how to set the feature maps in CNNs
- Activation function
  - sigmoid ? maxout ? . . . or Gaussian ?