畳み込みニューラルネットワーク　その２

1　演習の進め方

- 原則として実習です。その回のテキストをみて、各自のペースで演習を行って下さい。早く終ったら、次に進んで結構です。
- その回の演習が全て終ったら、必要に応じて周りの人をアシストして下さい。
- やってみてわからない点は、その場で質問して下さい。
- 前回休んだ人は、「環境設定」のテキストをみて環境設定を行ってから、テキストの演習を行って下さい。
- 途中でプラグラムの解説等を行うことがあります。その場合は、作業をとめて聞いて下さい。
- 演習には、Windows パソコンを持参して下さい。Mac でも動作可能と思いますが、現在のところ未検証です。

2　演習

2.1　実行用のツールのダウンロード

実行用のツールをダウンロードします。授業関連ファイル lesson に入っていますので、ダウンロードします。

```
cd lesson
git clone https://github.com/k1nk/lesson.git
cd ch3/chainer_imagenet_tools/
```

2.2　画像データのダウンロード

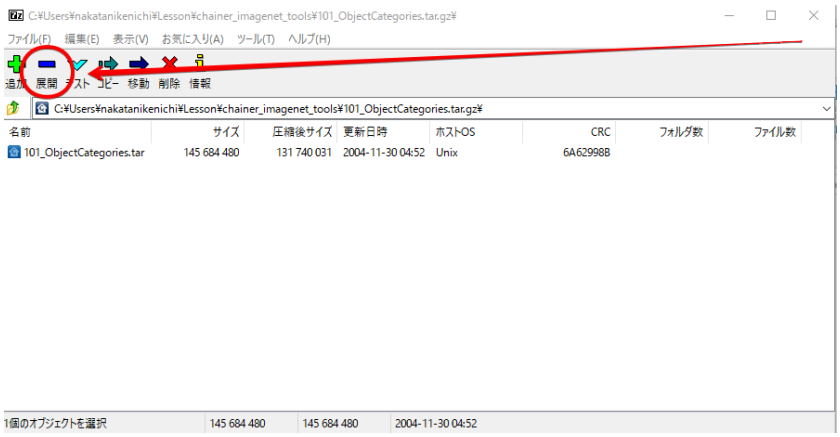
画像データをダウンロードして解凍します。http://www.vision.caltech.edu/Image_Datasets/Caltech101/ のサイトから、「101_ObjectCategories.tar.gz」をダウンロードします。

クリックしてダウンロード

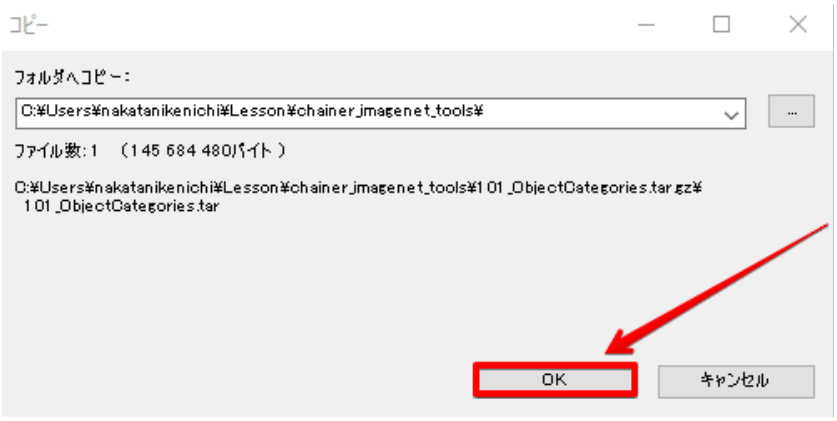
ダウンロードしたファイルを chainer_imagenet_tools の下に置きます。

ファイルを解凍します。

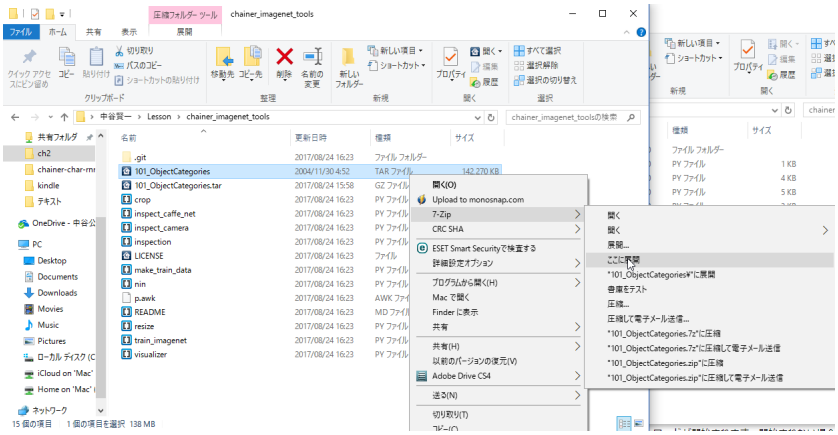
「101_ObjectCategories.tar.gz」を解凍すると、「101_ObjectCategories.tar」が生成されます。それを再度、解凍すると、「101_ObjectCategories」フォルダが作成されます。



クリックして展開



クリック



tar ファイルを右クリックして、「ここに展開」

「101_ObjectCategories」フォルダが作成されます。「101_ObjectCategories」フォルダには、「airplanes」「camera」など、１０１のカテゴリーごとのフォルダーがあります。それぞれのフォルダーの中には、そのカテゴリーに属するものの写真が保存されています。内容を確認してみましょう。

2.3　訓練用・テスト用データの作成

train_imagenet.py で訓練を行うために、訓練用・テスト用のデータが一定の形式に従うようにします。images フォルダの下に、２５６×２５６にデータを整形して配置します。また、訓練用データおよびその区分の一覧と、テスト用のデータおよびその区分の一覧を、それぞれ、train.txt、test.txt として作成します。また、train_imagenet.py を実行するには、訓練用データの画像の平均値のデータ (mean.npy) を前もって準備しておく必要があるため、compute_mean.py を使って、事前に作成しておきます。

```
python make_train_data_win.py 101_ObjectCategories
rename images images_not_cropped
mkdir images
python crop.py images_not_cropped images
python ../../chainer/examples/imagenet/compute_mean.py train.txt
```

2.4 訓練用のプログラムの修正

訓練は、./../chainer/examples/imagenet/train_imagenet.py を使って行います。

Listing 1 train_imagenet.py

```
"""Example code of learning a large scale convnet from ILSVRC2012
    dataset.

Prerequisite: To run this example, crop the center of ILSVRC2012
    training and
validation images, scale them to 256x256 and convert them to RGB, and
    make
two lists of space-separated CSV whose first column is full path to
    image and
second column is zero-origin label (this format is same as that used by
     Caffe's
ImageDataLayer).

"""
from __future__ import print_function
import argparse
import random

import numpy as np

import chainer
from chainer import training
from chainer.training import extensions

import alex
import googlenet
import googlenetbn
import nin
import resnet50


class PreprocessedDataset(chainer.dataset.DatasetMixin):

    def __init__(self, path, root, mean, crop_size, random=True):
        self.base = chainer.datasets.LabeledImageDataset(path, root)
```

```
31          self.mean = mean.astype('f')
32          self.crop_size = crop_size
33          self.random = random
34
35      def __len__(self):
36          return len(self.base)
37
38      def get_example(self, i):
39          # It reads the i-th image/label pair and return a preprocessed
                image.
40          # It applies following preprocesses:
41          #     - Cropping (random or center rectangular)
42          #     - Random flip
43          #     - Scaling to [0, 1] value
44          crop_size = self.crop_size
45
46          image, label = self.base[i]
47          _, h, w = image.shape
48
49          if self.random:
50              # Randomly crop a region and flip the image
51              top = random.randint(0, h - crop_size - 1)
52              left = random.randint(0, w - crop_size - 1)
53              if random.randint(0, 1):
54                  image = image[:, :, ::-1]
55          else:
56              # Crop the center
57              top = (h - crop_size) // 2
58              left = (w - crop_size) // 2
59          bottom = top + crop_size
60          right = left + crop_size
61
62          image = image[:, top:bottom, left:right]
63          image -= self.mean[:, top:bottom, left:right]
64          image *= (1.0 / 255.0)  # Scale to [0, 1]
65          return image, label
66
67
68  def main():
69      archs = {
```

```
70          'alex': alex.Alex,
71          'alex_fp16': alex.AlexFp16,
72          'googlenet': googlenet.GoogLeNet,
73          'googlenetbn': googlenetbn.GoogLeNetBN,
74          'googlenetbn_fp16': googlenetbn.GoogLeNetBNFp16,
75          'nin': nin.NIN,
76          'resnet50': resnet50.ResNet50
77      }
78
79      parser = argparse.ArgumentParser(
80          description='Learning convnet from ILSVRC2012 dataset')
81      parser.add_argument('train', help='Path to training image-label
            list file')
82      parser.add_argument('val', help='Path to validation image-label
            list file')
83      parser.add_argument('--arch', '-a', choices=archs.keys(), default='
            nin',
84                          help='Convnet architecture')
85      parser.add_argument('--batchsize', '-B', type=int, default=32,
86                          help='Learning minibatch size')
87      parser.add_argument('--epoch', '-E', type=int, default=10,
88                          help='Number of epochs to train')
89      parser.add_argument('--gpu', '-g', type=int, default=-1,
90                          help='GPU ID (negative value indicates CPU')
91      parser.add_argument('--initmodel',
92                          help='Initialize the model from given file')
93      parser.add_argument('--loaderjob', '-j', type=int,
94                          help='Number of parallel data loading processes
                                ')
95      parser.add_argument('--mean', '-m', default='mean.npy',
96                          help='Mean file (computed by compute_mean.py)')
97      parser.add_argument('--resume', '-r', default='',
98                          help='Initialize the trainer from given file')
99      parser.add_argument('--out', '-o', default='result',
100                         help='Output directory')
101     parser.add_argument('--root', '-R', default='.',
102                         help='Root directory path of image files')
103     parser.add_argument('--val_batchsize', '-b', type=int, default=250,
104                         help='Validation minibatch size')
105     parser.add_argument('--test', action='store_true')
```

```
106     parser.set_defaults(test=False)
107     args = parser.parse_args()
108
109     # Initialize the model to train
110     model = archs[args.arch]()
111     if args.initmodel:
112         print('Load model from', args.initmodel)
113         chainer.serializers.load_npz(args.initmodel, model)
114     if args.gpu >= 0:
115         chainer.cuda.get_device_from_id(args.gpu).use()  # Make the GPU
                current
116         model.to_gpu()
117
118     # Load the datasets and mean file
119     mean = np.load(args.mean)
120     train = PreprocessedDataset(args.train, args.root, mean, model.
            insize)
121     val = PreprocessedDataset(args.val, args.root, mean, model.insize,
            False)
122     # These iterators load the images with subprocesses running in
            parallel to
123     # the training/validation.
124     train_iter = chainer.iterators.MultiprocessIterator(
125         train, args.batchsize, n_processes=args.loaderjob)
126     val_iter = chainer.iterators.MultiprocessIterator(
127         val, args.val_batchsize, repeat=False, n_processes=args.
                loaderjob)
128
129     # Set up an optimizer
130     optimizer = chainer.optimizers.MomentumSGD(lr=0.01, momentum=0.9)
131     optimizer.setup(model)
132
133     # Set up a trainer
134     updater = training.StandardUpdater(train_iter, optimizer, device=
            args.gpu)
135     trainer = training.Trainer(updater, (args.epoch, 'epoch'), args.out
            )
136
137     val_interval = (10 if args.test else 100000), 'iteration'
138     log_interval = (10 if args.test else 1000), 'iteration'
```

```
139
140     trainer.extend(extensions.Evaluator(val_iter, model, device=args.
            gpu),
141                    trigger=val_interval)
142     trainer.extend(extensions.dump_graph('main/loss'))
143     trainer.extend(extensions.snapshot(), trigger=val_interval)
144     trainer.extend(extensions.snapshot_object(
145         model, 'model_iter_{.updater.iteration}'), trigger=val_interval
                )
146     # Be careful to pass the interval directly to LogReport
147     # (it determines when to emit log rather than when to read
            observations)
148     trainer.extend(extensions.LogReport(trigger=log_interval))
149     trainer.extend(extensions.observe_lr(), trigger=log_interval)
150     trainer.extend(extensions.PrintReport([
151         'epoch', 'iteration', 'main/loss', 'validation/main/loss',
152         'main/accuracy', 'validation/main/accuracy', 'lr'
153     ]), trigger=log_interval)
154     trainer.extend(extensions.ProgressBar(update_interval=10))
155
156     if args.resume:
157         chainer.serializers.load_npz(args.resume, trainer)
158
159     trainer.run()
160
161
162 if __name__ == '__main__':
163     main()
```

2.4.1 修正内容

大きめのデータセットを前提としたプログラムになっているので、val_interval を変更します。

```
137     val_interval = (10 if args.test else 1000), 'iteration'
138     log_interval = (10 if args.test else 1000), 'iteration'
```

訓練中の損失（loss）と正答率 (accuracy) をグラフとして出力するために、以下の行を追加します。

```
149     trainer.extend(extensions.observe_lr(), trigger=log_interval)
150     # Save two plot images to the result dir
151     if extensions.PlotReport.available():
152         trainer.extend(
```

```
153             extensions.PlotReport(['main/loss', 'validation/main/loss'
                    ],'epoch', file_name='loss.png'))
154         trainer.extend(
155             extensions.PlotReport(
156                 ['main/accuracy', 'validation/main/accuracy'],'epoch',
                        file_name='accuracy.png'))
```

2.5 プログラムの動作の仕組み

2.5.1 モデルの作成

```
    # Initialize the model to train
    model = archs[args.arch]()
```

モデルのインスタンスを作成します。args.arch には、指定したモデル（この例では、alex）が入ります。alex.Alex では、損失の計算に、F.softmax_cross_entropy を使っています。予測が答えと違うとそれだけこの関数の誤差は大きくなります。

（参考）softmax_cross_entropy は、ネットの出力を softmax 関数でカテゴリーごとの予測確率に変換したのもと、実際の答え（正解のカテゴリーが 1 でそれ以外は 0 となる配列）との交差エントロピー（cross entropy）を計算します。これは、それぞれのカテゴリごとに、そのカテゴリが正解の時は、p=1、不正解の時は p=0 としたものと、そのカテゴリごとに予測される確率 q を使って、

交差エントロピー誤差関数：$E = -p\log(q) - (1-p)\log(1-q)$

を計算し、その値を全てのカテゴリーについての合計し、カテゴリーの数で割って、平均をとったものです。

交差エントロピー		q					
		0.00001	0.2	0.4	0.6	0.8	0.99999
p	0.00001	5.43429E-05	0.096916034	0.221850511	0.397938248	0.698963984	4.99995
	0.2	1.000003474	0.217322011	0.257067001	0.362721757	0.578558006	4.000000869
	0.4	2.000002606	0.33773401	0.292285253	0.327503505	0.458146008	3.000001737
	0.6	3.000001737	0.458146008	0.327503505	0.292285253	0.33773401	2.000002606
	0.8	4.000000869	0.578558006	0.362721757	0.257067001	0.217322011	1.000003474
	0.99999	4.99995	0.698963984	0.397938248	0.221850511	0.096916034	5.43429E-05

図 1　交差エントロピー

上記の表は、p と q の交差エントロピーを計算したものです。交差エントロピーは、p=0 または、p=1 の時、q の値が正解に近ければ 0 に近い値となり、間違っている場合には急速に大きな値となります。間違っている場合の勾配の大きさも大きくなるという性質があるため、学習が速くなる傾向があります。そのため、交差エントロピーは、クラス分類の損失関数として広く利用されています。

2.5.2 データセットの取得

```
    train = PreprocessedDataset(args.train, args.root, mean, model.
        insize)
    val = PreprocessedDataset(args.val, args.root, mean, model.insize,
        False)
```

トレーニング用と、検証用のデータセットを取得しています。今回は、args.train に text.txt, args.val に test.txt を渡して呼び出しています。PreprocessedDataset は、chainer.dataset.DatasetMixin クラスの get_example(self, i) をオーバーライドしています。これは、データセットの i 番目のデータを返すメソッドです。この中で、

クロッピング

```
        image = image[:, top:bottom, left:right]
```

全データの平均値を差し引く

```
        image -= self.mean[:, top:bottom, left:right]
```

データのスケーリング（元の値が０～２５５であったので２５５分の１のスケールにする）

```
        image *= (1.0 / 255.0)  # Scale to [0, 1]
```

を行っています。

2.5.3 Iteratorn の作成

```
    train_iter = chainer.iterators.MultiprocessIterator(
        train, args.batchsize, n_processes=args.loaderjob)
    val_iter = chainer.iterators.MultiprocessIterator(
        val, args.val_batchsize, repeat=False, n_processes=args.
            loaderjob)
```

データセットから一定のバッチサイズのミニバッチを繰返し取り出す Iterator です。訓練用と検証用のそれぞれについて作成します。

2.5.4 optimizer の設定

```
    # Set up an optimizer
    optimizer = chainer.optimizers.MomentumSGD(lr=0.01, momentum=0.9)
    optimizer.setup(model)
```

確率的勾配降下法（SGD）は、それぞれの訓練データごとに勾配降下法をおこなって重み等のパラメータの更新を行う方法です。SGD に慣性項 (Momentum) を付与したものが、MomentumSGD です。オプティマイザーとして、MomentumSGD を指定し、オプティマイザーとモデルを関連づけます。

2.5.5 学習ループ（Trainer）の作成

```
    # Set up a trainer
    updater = training.StandardUpdater(train_iter, optimizer, device=
        args.gpu)
    trainer = training.Trainer(updater, (args.epoch, 'epoch'), args.out
        )
```

Chainer は学習ループを抽象化する方法を提供しています。Trainer は、学習ループ自体をインプリメントしています。学習ループは２つのパートからなります。２つは Updater です。Updater は、訓練用においてパラメータの更新を実際行います。もう一つは、Extension です。Extension はパラメータの更新以外の任意の機能を Trainer に付加します。

ここでは、まず、データセットが関連づけられている train_iter と、モデルや最適化の方法が関連付けられている optimizer から、updater を作成しています。そして、その updater を使って trainer を作成しています。

2.5.6 学習ループ（Trainer）の設定

Listing 2　Evaluator

```
    trainer.extend(extensions.Evaluator(val_iter, model, device=args.
        gpu), trigger=val_interval)
```

trainer に extensions.Evaluator を付加します。Evaluator は、val_iter に関連付けられたデータセットを使って、model を評価します。Evaluator は、val_intervaln の iteration ごとに呼び出されます。

Listing 3　モデルの計算の構造をアウトプット

```
trainer.extend(extensions.dump_graph('main/loss'))
```

trainer に extensions.dump_graph を付加します。dump_graph は、モデルの計算の構造を表す computational graph （図 2）を出力します。computational graph は、DOT 言語 (cg.dot) でアウトプットされます。

Listing 4　trainer のスナップショットを保存

```
    trainer.extend(extensions.snapshot(), trigger=val_interval)
```

trainer に extensions.snapshot を付加します。extensions.snapshot は、trainer のスナップショットを保存します。スナップショットを保存することにより、学習ループを保存したところから再開することができます。

Listing 5　指定したオブでジェクとのスナップショットを保存

```
    trainer.extend(extensions.snapshot_object(
        model, 'model_iter_{.updater.iteration}'), trigger=val_interval
            )
```

trainer に extensions.snapshot_object を付加します。snapshot_object は、指定したオブでジェクとのスナップショットを保存します。ここでは、model のスナップショットを保存するように指定しています。

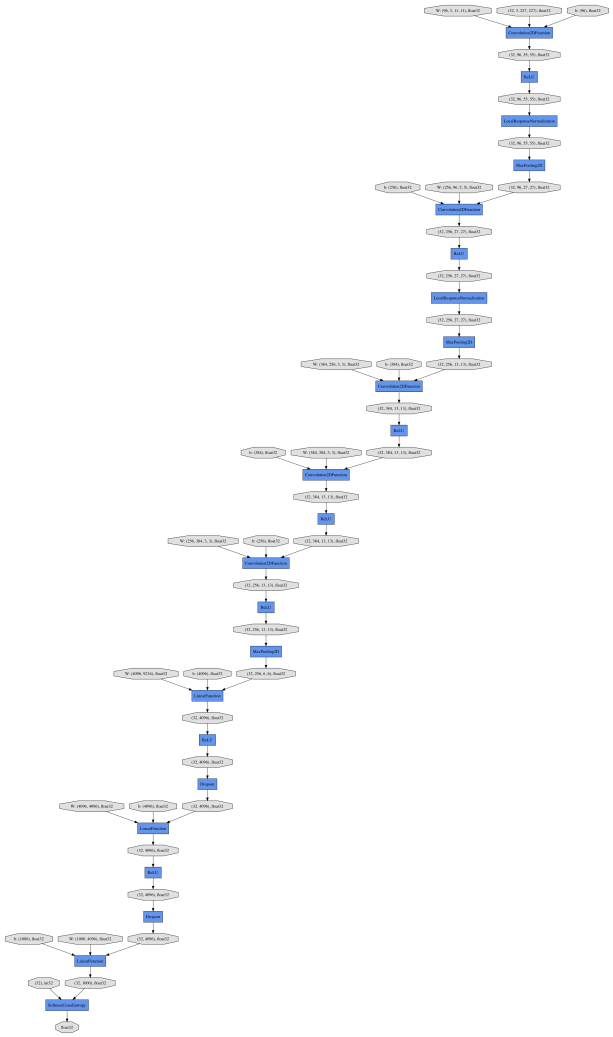


図 2　computational graph

Listing 6　計算結果を出力

```
    trainer.extend(extensions.LogReport(trigger=log_interval))
```

trainer に extensions.LogReport を付加します。LogReport は、累積された計算結果をログファイル (Listing 7) に出力します。ログファイル名はデフォルトで”log”です。

Listing 7　log

```
[
    {
        "main/loss": 3.3944456577301025,
        "validation/main/loss": 2.4045562744140625,
        "iteration": 1000,
        "elapsed_time": 245.2878770828247,
```

```
        "epoch": 4,
        "lr": 0.01,
        "validation/main/accuracy": 0.45733150839805603,
        "main/accuracy": 0.2798125147819519
    },...
   ,
    {
        "main/loss": 0.10925810784101486,
        "validation/main/loss": 1.8571486473083496,
        "iteration": 15000,
        "elapsed_time": 4276.536968946457,
        "epoch": 69,
        "lr": 0.01,
        "validation/main/accuracy": 0.6915444731712341,
        "main/accuracy": 0.968874990940094
    }
]
```

Listing 8　learning rate を記録”

```
    trainer.extend(extensions.observe_lr(), trigger=log_interval)
```

trainer に extensions.observe_lr を付加します。observe_lr は、オプティマイザーの learning rate を記録します。ログに”lr”が含まれます。

Listing 9　ログの内容を表示

```
    trainer.extend(extensions.PrintReport([
        'epoch', 'iteration', 'main/loss', 'validation/main/loss',
        'main/accuracy', 'validation/main/accuracy', 'lr'
    ]), trigger=log_interval)
```

trainer に extensions.PrintReport を付加します。PrintReport は、LogReport で出力されるログの内容を人の読みやすい形式で標準出力へ出力します。

Listing 10　プログレスバーを表示

```
    trainer.extend(extensions.ProgressBar(update_interval=10))
```

trainer に extensions.ProgressBar を付加します。ProgressBar は、訓練の進行状況を示すプログレスバーを標準出力へ出力します。

2.5.7　学習を開始

```
    trainer.run()
```

設定された内容で学習を開始します。

2.6 訓練の実行

訓練は、以下のコマンドで実行します。

```
python ../../chainer/examples/imagenet/train_imagenet.py -E 100 -a alex
    train.txt test.txt
```

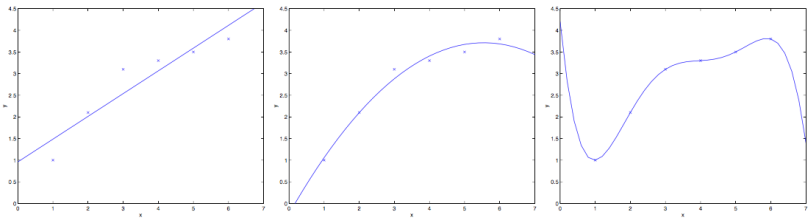
-a　のパラメータを変えると、訓練するモデルを変更できます。

ノートパソコン等で実行する場合、場合によっていは実行してから最初のエポックが開始されるまでに１０分〜１５分程度かかることがあります。それまで、画面に何も表示されませんが、気長に待って下さい。モデルの作成には、多くの計算が必要なため、GPU を使って計算を行うことが一般的です。訓練を行うと、result ディレクトリの下にモデル（result/model_iter_****）、損失のグラフ（loss.png）、正答率のグラフ（accuracy.png）が作成されます。

2.7 過学習

一般に、データセットは、「トレーニング用（訓練用）」、「バリデーション用（検証用）」（利用しないこともある）、「テスト用」のデータセットに分かれます。まず、訓練用データセットを用いて、モデルの訓練を行います。そしてバリデーション用のデータセットをつかって、例えばモデルの層の数などのモデルのハイパーパラメータの検証やチューニングを行います。

完成した最終的なモデルに対して、テスト用のデータを使ってモデルのテストを行います。トレーニング用やバリデーション用のデータは、そのデータを使ってモデルを作成しているので、モデルはそのデータに最適化されています。そのため、それら以外のデータを使ってモデルの善し悪しをテストする必要があります。これらの値の



右：直線
左：５次曲線

図３　過学習

モデルのパラメータが多いと、学習に用いたデータに対してはフィットするが、一般的なデータにはフィットしなくなることがあります。これを過学習といいます。このようなことを避けるために、学習に用いたデータとは別のデータを用いて、損失（loss）や正答率（accuracy）を測定します。これにより、学習したデータでは別の一般的なデータに対してモデルがどの程度フィットするかを確認することが可能となります。一般的に、

- モデルのデータが少ない
- モデルが問題に対して複雑すぎる

場合に、過学習がおきます。この状況は、学習するデータの数に対して、訓練用データを用いた正答率と、検証用データを用いた正答率をグラフにすると分かりやすいです。

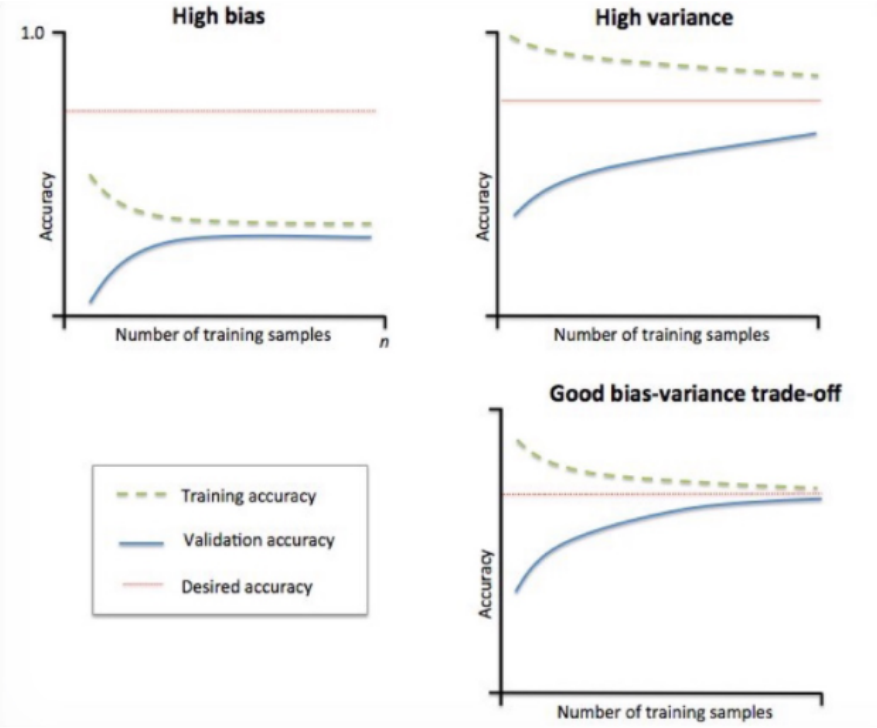


図 4　High Bias / High Variance

上記のグラフで、赤い直線は、望ましい正答率を表します。青い曲線は検証用データを用いた正答率を、緑の点線は訓練用データを用いた正答率を表します。いずれの場合でも、望ましい正答率に達しない場合（左上の場合、High bias)、モデルをより複雑なものにする必要があります。

訓練用データを用いた場合には、望ましい正答率よりも、よくなるが、検証用データでは望ましい正答率に達せず、その差が大きい場合、過学習になっているといえます（右上の場合、High variance)。この場合には、データ量を増やす、あるいはモデルが複雑すぎるのでもっと簡単なものを使うこととなります。

モデルが適切な場合、右下のようなグラフになります。

図 5 は、今回のケースでグラフプロットしてみたものです。今回は大規模なデータセットを前提とした、Alexnet を使っています。一方、学習に使っているデータの数は、Alexnet がコンペティションで使ったものより少ない状態です。今回の図 5 学習結果をみると、仮に望ましい正答率を 9 割とすると訓練用データでは 9 割を超えていますが、検証用データでは約 7 割となっています。これは、典型的な過学習の状態です。今回のデータセットを分類するには、モデルが複雑過ぎるのでもっとパラメータの少ないモデルを使う、あるいは、Alexnet をコンペティションで訓練した時のようにデータをもっと増やす、のいずれかの方法が必要であると思います。

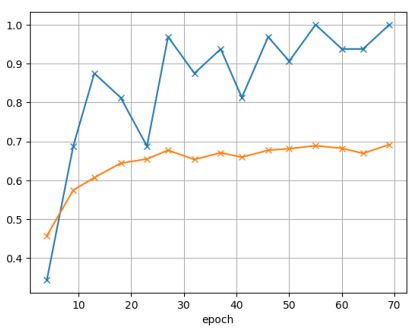


図 5　accuracy

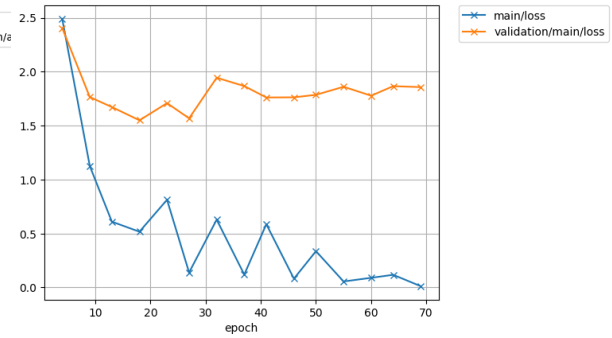


図 6　loss

2.8　モデルを使った予測

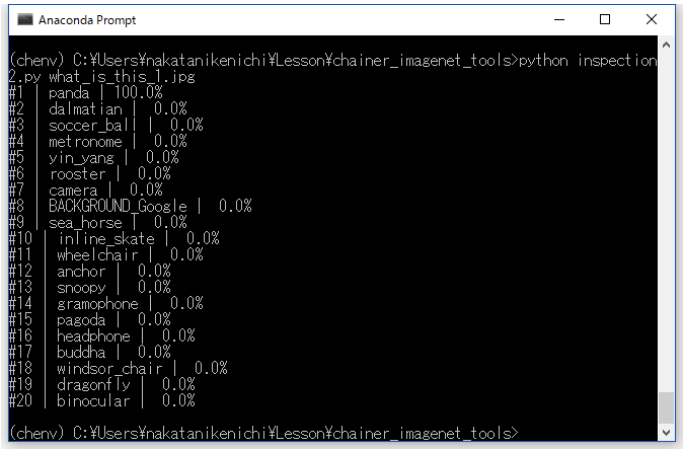
2.8.1　予測

訓練には時間がかかるため、授業時間中に完了しないと思います。上記の方法で事前に訓練した学習済みモデル (result_alex/model_iter_15000) を用いて予測を行います。学習済みデータは、github のファイル上限１００M byte を超えるため、USB で配布します。result_alex の下にコピーして下さい。

what_is_this_1.jpg は、２５６×２５６にクロップされた画像ファイルです。このファイルを認識させてみましょう。

```
python inspection2.py what_is_this_1.jpg
```

what_is_this_1.jpg



2.8.2　任意のファイルの予測

```
python resize.py <filename1> <filename2> …
```

とすると、jpg のファイルを２５６×２５６にクロップします。クロップされたファイルは、resized フォルダの下に保存されます。自分で任意のファイルを認識させて、１０１のカテゴリーのどれに分類されるか試して

みましょう。

```
python resize.py <filename.jpg>
python inspection2.py resized/<filename.jpg>
```

2.9 予測用プログラム

Listing 11 inspection2.py

```
1  from __future__ import print_function
2  import argparse
3  import datetime
4  import json
5  import multiprocessing
6  import random
7  import sys
8  import threading
9  import time
10
11 import numpy as np
12 from PIL import Image
13
14
15 import six
16 #import six.moves.cPickle as pickle
17 import cPickle as pickle
18 from six.moves import queue
19
20 import chainer
21 import matplotlib.pyplot as plt
22 import numpy as np
23 import math
24 import chainer.functions as F
25 import chainer.links as L
26 from chainer.links import caffe
27 from matplotlib.ticker import *
28 from chainer import serializers
29
30 parser = argparse.ArgumentParser(
31     description='Image inspection using chainer')
32 parser.add_argument('image', help='Path to inspection image file')
```

```
33 parser.add_argument('--model','-m',default='model', help='Path to model
       file')
34 parser.add_argument('--mean', default='mean.npy',
35                     help='Path to the mean file (computed by
                           compute_mean.py)')
36 args = parser.parse_args()
37
38
39 def read_image(path, center=False, flip=False):
40   image = np.asarray(Image.open(path)).transpose(2, 0, 1)
41   if center:
42     top = left = cropwidth / 2
43   else:
44     top = random.randint(0, cropwidth - 1)
45     left = random.randint(0, cropwidth - 1)
46   bottom = model.insize + top
47   right = model.insize + left
48   image = image[:, top:bottom, left:right].astype(np.float32)
49   image -= mean_image[:, top:bottom, left:right]
50   image /= 255
51   if flip and random.randint(0, 1) == 0:
52     return image[:, :, ::-1]
53   else:
54     return image
55
56 import nin
57 import alex
58
59 #mean_image = pickle.load(open(args.mean, 'rb'))
60 mean_image = np.load(args.mean)
61
62 #model = nin.NIN()
63 model = alex.Alex()
64
65 #serializers.load_hdf5("gpu1out.h5", model)
66 #serializers.load_hdf5("cpu1out.h5", model)
67 serializers.load_npz("result_alex/model_iter_15000", model)
68 cropwidth = 256 - model.insize
69 model.to_cpu()
70
```

```
71 def predict(net, x):
72     h = F.max_pooling_2d(F.relu(net.mlpconv1(x)), 3, stride=2)
73     h = F.max_pooling_2d(F.relu(net.mlpconv2(h)), 3, stride=2)
74     h = F.max_pooling_2d(F.relu(net.mlpconv3(h)), 3, stride=2)
75     #h = net.mlpconv4(F.dropout(h, train=net.train))
76     h = net.mlpconv4(F.dropout(h))
77     h = F.reshape(F.average_pooling_2d(h, 6), (x.data.shape[0], 1000))
78     return F.softmax(h)
79 
80 def predict_alex(net,x):
81     h = F.max_pooling_2d(F.local_response_normalization(
82             F.relu(net.conv1(x))), 3, stride=2)
83     h = F.max_pooling_2d(F.local_response_normalization(
84             F.relu(net.conv2(h))), 3, stride=2)
85     h = F.relu(net.conv3(h))
86     h = F.relu(net.conv4(h))
87     h = F.max_pooling_2d(F.relu(net.conv5(h)), 3, stride=2)
88     h = F.dropout(F.relu(net.fc6(h)))
89     h = F.dropout(F.relu(net.fc7(h)))
90     h = net.fc8(h)
91 
92     return F.softmax(h)
93 
94 #setattr(model, 'predict', predict)
95 
96 img = read_image(args.image)
97 x = np.ndarray(
98         (1, 3, model.insize, model.insize), dtype=np.float32)
99 x[0]=img
100 #x = chainer.Variable(np.asarray(x), volatile='on')
101 x = chainer.Variable(np.asarray(x))
102 with chainer.using_config('train', False):
103     with chainer.no_backprop_mode():
104         #score = predict(model,x)
105         score = predict_alex(model,x)
106         #score=cuda.to_cpu(score.data)
107 
108 categories = np.loadtxt("labels.txt", str, delimiter="\t")
109 
110 top_k = 20
```

```
111 prediction = zip(score.data[0].tolist(), categories)
112 prediction.sort(cmp=lambda x, y: cmp(x[0], y[0]), reverse=True)
113 for rank, (score, name) in enumerate(prediction[:top_k], start=1):
114     print('#%d | %s | %4.1f%%' % (rank, name, score * 100))
```

2.10 プログラムの動作の仕組み

Listing 12　イメージの読み込み

```
img = read_image(args.image)
```

指定されたイメージを読み込みます。read_image では、指定されたイメージのクロッピングやスケーリングなどを行って、その結果を返します。

Listing 13　入力用変数の作成

```
x = np.ndarray((1, 3, model.insize, model.insize), dtype=np.float32)
x[0]=img
x = chainer.Variable(np.asarray(x))
```

イメージから、chainer.Variable である x を作成します。

Listing 14　予測

```
with chainer.using_config('train', False):
    with chainer.no_backprop_mode():
        score = predict_alex(model,x)
```

訓練やバックプロパゲーションを行わない状態で、model を使って、score を予測します。model の中で Dropout などを使っている場合、訓練時とそうでない時とで動作が違うので、訓練中でない状態であることを指定する必要があります。

Listing 15　predict_alex

```
def predict_alex(net,x):
    h = F.max_pooling_2d(F.local_response_normalization(
            F.relu(net.conv1(x))), 3, stride=2)
    h = F.max_pooling_2d(F.local_response_normalization(
            F.relu(net.conv2(h))), 3, stride=2)
    h = F.relu(net.conv3(h))
    h = F.relu(net.conv4(h))
    h = F.max_pooling_2d(F.relu(net.conv5(h)), 3, stride=2)
    h = F.dropout(F.relu(net.fc6(h)))
    h = F.dropout(F.relu(net.fc7(h)))
    h = net.fc8(h)
```

```
    return F.softmax(h)
```

predict_alex では、指定した net の学習したパラメータと alexnet の構造を使って、x から、それぞれのカテゴリーに分類される確率を計算します。

Listing 16　カテゴリーの表示

```
categories = np.loadtxt("labels.txt", str, delimiter="\t")

top_k = 20
prediction = zip(score.data[0].tolist(), categories)
prediction.sort(cmp=lambda x, y: cmp(x[0], y[0]), reverse=True)
for rank, (score, name) in enumerate(prediction[:top_k], start=1):
    print('#%d | %s | %4.1f%%' % (rank, name, score * 100))
```

予測された確率とカテゴリー名を対にして、予測された確率の大きい順にソートします。そして、予測されるカテゴリについて、トップ２０までランキング、カテゴリ名、スコア（確率× 100）を出力します。※ cmp( ) は a と b の二つの引数を取り、a ¡ b なら-1、a == b なら 0、a ¿ b なら 1 を返します。