UMU_54907 DeepLearning

2019VT

Lab1

Mnist classification with CNN

sygr0003 ==Sylvie Grafeuille

60….

sylgrafe@gmail.com

UMEÅ UNIVERSITET

Applied Physics and Electronics

Umeå universitet

Postadress: 901 87 Umeå

Content

Figures

1. Intro

1.1. *About this document*

This document is a laboration rapport about a simple convolutional neural network (CNN) .

The document written in english except for the “uppgift” given by the teacher written in swedish.

Intended audience.

The reader is expected to know python , jupyter , colab and the basic notions about machine learning .

Structure of the document

TODO

About the references

The explanation comes mostly from the book deep_learning in python Francois Cholet

If no ref is given for an explanation the information comes from docString in python or from myself.

1.2. *Uppgift Laboration 1 – CNN*

Labbspec, Deep Learning , Kalle Prorok

Din uppgift är att koda ihop ett Convolutional Neural Net (CNN) som analyserar MNIST-data (du får välja mellan Fashion eller Handskrivna siffror) och ska använda en metod med bättre resultat än det alla-till-alla nätet som presenterades i början på Tensorflow-avsnittet (i Colab).
Se lärplattformen för mer information hur detta kan gå till och/eller läs lämpliga böcker, googla etc, glöm inte att ange referenser i din dokumentation.

Genomförande

Du får använda Python Keras (helst) alternativt Matlab. Du ska ha skrivit all kod själv. Du kan använda och inspireras av exempel du hittar men skriv in allt själv, ändra variabelnamn dokumentera etc, så du lär dig och förstår vad som sker på djupet så du kan lösa andra problem på egen hand efter kursen.
Eftersom det kan vara tidsödande att provköra ditt program är det viktigt att du dokumenterar/presenterar del- och slutresultat tydligt samt även berättar lite kort vad varje kodrad gör samt analyserar/jämför slutresultatet och hur/varför det fungerar som det gör.

Dokumentation och Lagring av kod

För att klasskamraterna och lärare ska kunna nå, och vid behov provköra, koden behöver den lagras på något åtkomligt sätt, t ex på GitHub eller i Google Drive, så att du kan ange lämplig länk till hur och var man kommer åt den.

Redovisning – Peer-review

I uppgiften ingår att kommentera andras lösningar. Tanken med det är att båda parter ska lära sig lite extra, jag hoppas det kan vara kul och lärorikt att se hur andra löst det och på så vis förbättra sig själv.

Uppdatering baserat på feedback
När du erhållt feedback från din(a) kamrat(er)får du justera kod och dokumentation. Ev kan vi behöva köra en andra runda med återkoppling från en uppdaterad version. När genomgångarna är klara tittar läraren igenom kommentarer och resultat och ger båda parter betyg.

Betygskriterier

Utföraren
Grundnivå G: Ett väldokumenterat och förståeligt program som använder sig av CNN. Minst två olika inställningar på nån av parametrarna (t ex strides, fönsterstorlekar eller padding) ska köras och jämföras och slutresultatet kommenteras. Man ska nå minst 98% rätt på testdata i fallet med handskrivna siffor och 90% för Fashion-Mnist.

Mellannivå VG: Användande av metod (t ex Drop-out) för regularisering och även testa Data augmentation. Det är inte säkert att resultatet blir bättre men du ska visa att du känner till och kan beskriva hur metoderna används och vad de kan göra för nytta.
Hög nivå MVG: Filtrens inlärda detektion ska visualiseras på lämpligt sätt. Man ska få en ide om hur nätverket analyserar indata allteftersom i de olika lagren. Efter det använda transfer learning, dvs basera nätet på ett i förväg tränat stort nät och justera till så att det fungerar på dessa nya data.
Om redovisaren använt/utfört delar av MVG-nivån och delar av VG ges VG.
En försenad inlämning kan bara ge G.

Bedömaren
Grundnivå G: kommenterat kort och noterat felaktigheter
Mellannivå VG: gett förbättringsförslag, personliga tips och konstruktiva idéer.

About the DATA for lab1

(Cholet ch2.2)

The problem we’re trying to solve here is to classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9). The MNIST dataset, is a set of 60,000 training images, plus 10,000 test images.

1.3. My environememnt for programming and test

I run python pgm and jupyter pm on my desktop (no GPU) running Ubuntu 18.04

the results gathered ins results.json are produce from the notebook run in Colab with GPU

2. About the files on gitHub

2.1. Introduction

All files related to this lab are put from 2 public repositories on github

https://github.com/SylGrafe/lab1Repo and

https://github.com/SylGrafe/lab1Lib

the files

In https://github.com/SylGrafe/lab1Lib

contains lab1Utils.py imported by all pgm between all pgm

(pgm on local host jupyter notebooks on local host or notebooks in colab)

the file contains routines to dump information a json string or to access a file containing this inoformation.

on https://github.com/SylGrafe/lab1Repo

colab notebooks with differents models

this pdf

a README text file

2.2. about the notebooks

there are 2 kinds of notebooks   on colab

colabReadJSON

this notebook access results.json file and allow the user to choose and print

a summary of all results

a more details presentation for any result

as plot the fit history for any result

example history plot   from colabReadJSON

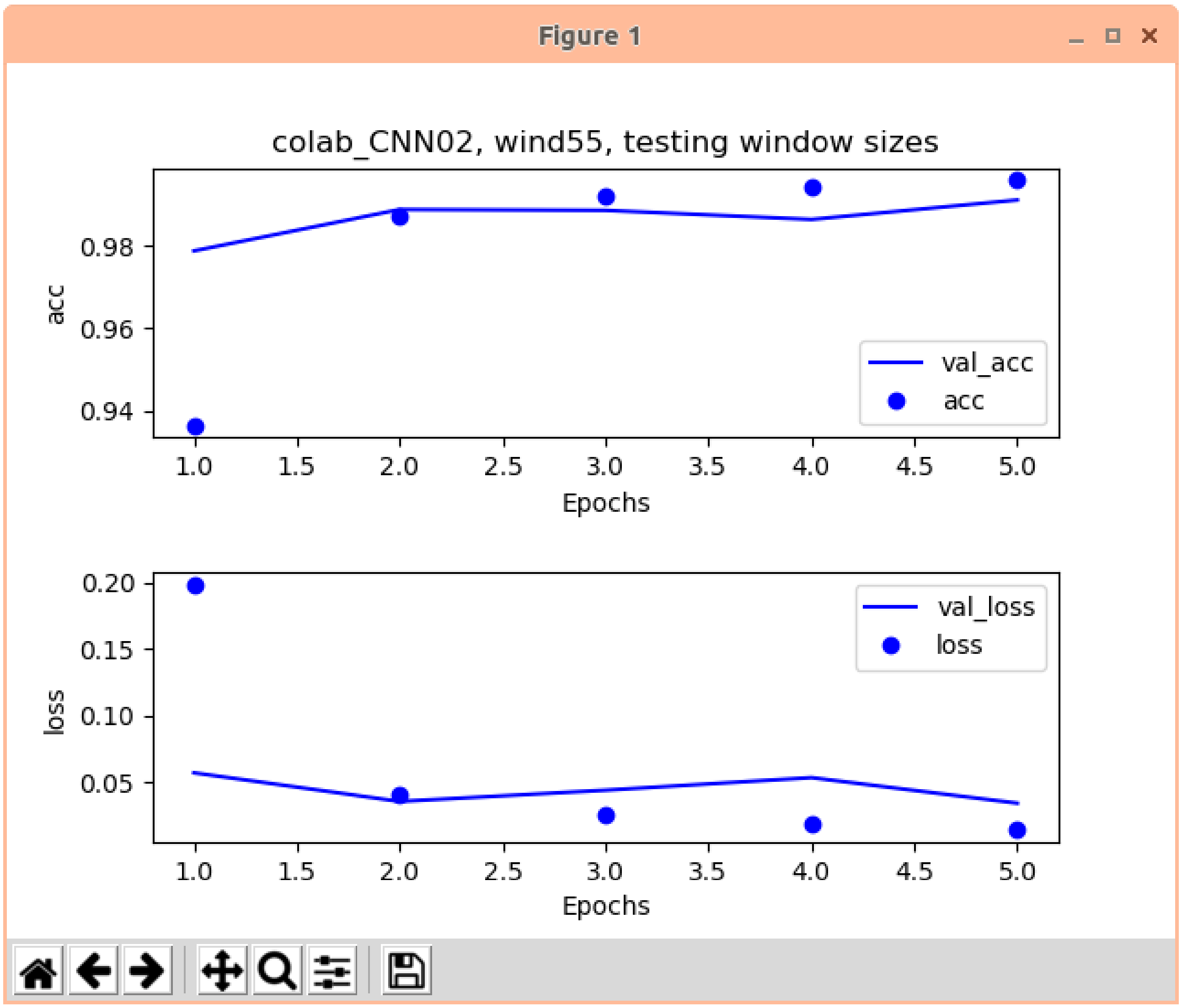


Illustration 1: plot history for colabCNN02 wind55

notes nooks with model

the other notebooks contains the code for differents neural networks.

some test where made with densely connected NN some with CNN.

After each test the results and some more information are dumped in a json file.

2.3. about results.json

As colab is in a virtual environments without easy procedure to read and write files I decided to use Github to make the all pgm results accessibles at once.

each time a pgm is run in colab it may produces a json file.

this file was then downloaded to localhost

All interesting json files were then concatenated into a file: results.json

results.json was then push back into

https://github.com/SylGrafe/lab1Repo

example of information dumped for one test

Here is given the kind of information dumped for one test.

```
{"modelStruct": "s1_valid", "compInfo": "rmsprop, categorical_crossentropy", "histDict":
{"val_loss": [0.0712, 0.054, 0.0492, 0.0373, 0.0257, 0.0259], "val_acc": [0.9795, 0.9842, 0.9843,
0.9882, 0.9928, 0.9918], "loss": [0.2617, 0.0634, 0.0443, 0.0325, 0.0267, 0.0206], "acc": [0.9188,
0.9805, 0.9864, 0.9897, 0.9914, 0.9935]}, "histParams": {"batch_size": 128, "epochs": 6, "steps":
null, "samples": 54000, "verbose": 0, "do_validation": true, "metrics": ["loss", "acc", "val_loss",
"val_acc"]}, "timeStamp": "2303_1652", "info": "testing srides and padding", "h5": "", "testRes":
[0.0294, 0.9917], "codeRef": "colab_CNN01"}
```

explanations

codeRef : the reference to the note book from which the results where produced

"codeRef": "colab_CNN01"

modelStruct a reference to a if block in the code related to the structure of the model

{"modelStruct": "s1_valid"

timestamp : a time stamp at which the pgm was run

"timeStamp": "2303_1652"

info : some more nformation that the user wanted to dump

"info": "testing srides and padding"

the other parts ar enow listed withotu more examplanation

"compInfo": "rmsprop, categorical_crossentropy", "

histDict": {"val_loss": [0.0712, 0.054, 0.0492, 0.0373, 0.0257, 0.0259], "val_acc": [0.9795, 0.9842, 0.9843, 0.9882, 0.9928, 0.9918], "loss": [0.2617, 0.0634, 0.0443, 0.0325, 0.0267, 0.0206], "acc": [0.9188, 0.9805, 0.9864, 0.9897, 0.9914, 0.9935]},

"histParams": {"batch_size": 128, "epochs": 6, "steps": null, "samples": 54000, "verbose": 0, "do_validation": true, "metrics": ["loss", "acc", "val_loss", "val_acc"]},

3. The results

TODO

4. Some Basic concepts for CNN

4.1. General concept

about loading and preparing the data

about the dataset (Cholet ch 2.1)

The MNIST dataset, is a set of 60,000 training images, plus 10,000 test images.

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

about preprocessing the data (Cholet ch2.1)

Before training, we’ll preprocess the data by reshaping it into the shape the network expects and scaling it so that all values are in the [0, 1] interval. Previously, our training images, for instance, were stored in an array of shape (60000, 28, 28) of type uint8 with values in the [0,

255] interval. We transform it into a float32 array of shape (60000, 28 * 28) with values between 0 and 1.

```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
```

about preparing the data (Cholet ch3.5.2)

To vectorize the labels, there are two possibilities: you can cast the label list as an integer tensor, or you can use one-hot encoding. One-hot encoding is a widely used format for categorical data, also called categorical encoding. In this case, one-hot encoding of the labels consists of embedding each label as an all-zero vector with a 1 in the place of the label index.

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

about training a model (Cholet Ch 3.1)

Ch3.1 ANATOMY OF A NEURAL NETWORK

As you saw in the previous chapters, training a neural network revolves around the following objects:

Layers, which are combined into a network (or model)

The input data and corresponding targets

The loss function, which defines the feedback signal used for learning

The optimizer, which determines how learning proceeds

To train a model call the fit function

```
network.fit (train_images , train_labels , epochs=5 , verbose=2,
batch_size=128)
```

Illustration of hyperparameters

Extracted from Practical guide to hyperparameters search for deep learning models

https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

E.g. DNN

Model Design

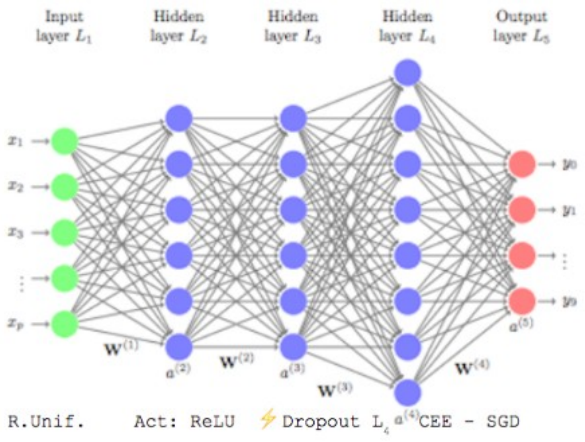
- Weight init.: Random Uniform
- Act.: ReLU
- Loss: CEE
- # Hidden Layers: 3
- # Units per layer {p, p+1, p+1, p+3, 10}
- Optimizer: SGD
- Dropout layer: L_4

Hyperparameters

- Learning rate
- Dropout Rate
- Batch size

Model Parameters

- $W^{(1)}$ => $W^{(4)}$



Variables classification example

Illustration 2: model desing , hyperparameters and parameters

4.2. about Convolutional neural network

Info from wikipedia

https://en.wikipedia.org/wiki/Convolutional_neural_network

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

...

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

about The convolution operation (Cholet 5.1.1).

The fundamental difference between a densely connected layer and a convolution layer is this:

Dense layers learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels), whereas convolution layers learn local patterns : in

the case of images, patterns found in small 2D windows of the inputs. In

this example, these windows are all 3 × 3.

Structure of a basic convnet (Cholet 5.1)

a basic convnet is a stack of Conv2D and MaxPooling2D layers

This is a pattern you'll see in almost all convnets.

about feature map (Cholet 5.1.1)

Convolutions operate over 3D tensors, called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis). For an RGB image, the dimension of the depth axis is 3, because the image has three color channels: red, green, and blue. For a

black-and-white picture, like the MNIST digits, the depth is 1 (levels of gray). The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map. This output feature map is still a 3D tensor: it has a width and a height. Its depth can be arbitrary, because the output depth is a parameter of the layer, and the different channels in that depth axis no longer stand for specific colors as in RGB input; rather, they stand for filters. Filters encode specific aspects of the input data: at a high level, a single filter could encode the concept "presence of a face in the input," for instance.

""""

about the notion of stride (Cholet 5.1.1)

The other factor that can influence output size is the notion of strides. The description of convolution so far has assumed that the center tiles of the convolution windows are all contiguous. But the distance between two successive windows is a parameter of the convolution, called its stride, which defaults to 1. It’s possible to have strided convolutions: convolutions with a stride higher than 1.

4.3. about 2D convolution layer

2D convolution layer (e.g. spatial convolution over images).

Init signature: layers.Conv2D(filters, kernel_size, ... , activation=None,... , **kwargs)

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the sample axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

# Arguments

filters: Integer, the dimensionality of the output space   (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window.

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
```

about The max-pooling operation (Cholet 5.1.2.)

That’s the role of max pooling: to aggressively downsample feature maps. Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel. It’s conceptually similar to convolution, except that instead of transforming local patches via a learned linear transformation (the convolution kernel), they’re transformed via a hardcoded max tensor operation. A big difference from convolution is that max pooling is usually done with 2 × 2 windows and stride 2, in order to downsample the feature maps by a factor of 2. On the other hand, convolution is typically done with 3 × 3 windows and no stride (stride 1).

about max pooling

about model.add(layers.MaxPooling2D((2, 2)))

Max pooling operation for spatial data.

Init signature: layers.MaxPooling2D(pool_size=(2, 2), ... , **kwargs)

# Arguments pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension.

```
model.add(layers.MaxPooling2D((2, 2)))
```

4.4. about dense layers at the end of convolutional neural networks

https://www.quora.com/Why-are-the-often-dense-layers-at-the-end-of-convolutional-neural-networks

So when we talk about convolution and max-pooling layers, the network is learning the relevant features. For example if you have a task to identify whether the image contains a face or not. CNN automatically learns the features in the form of filters from these images. In the final layers, you will have filters which resembles an eye, nose, ears, etc.

...

Now you have all the features required as an input of the neural network. Normal ANN’s work is required, adding dense layers and output nodes at the final layer.

About the descripton of the CNN model

Cholet 5.2.3

The depth of the feature maps progressively increases in the network (from 32 to 64), whereas the size of the feature maps decreases (from 28x28 to 3x3) , see the output from model.summary() : Error: Reference source not found

xxxxxxxxxxxxx end of document