**CZ4042 Assignment2 Report**

Li Bingzi

U1722793H

Part A: Object Recognition

# *Abstract-*The aim of this project is to build a CNN model to classify the a sample of the CIFAR-10 dataset, which contains RGB color images of size 32x32 and their corresponding labels from 0 to 9.

**Keywords:** Image dataset, CNN, Classification, Keras

# Introduction

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.[1]

In this experiment, ConvNet is built and combination of channels at the Conv layers is found using grid search. Different optimizers are tested. Dropout is applied.

# Methodology

## Neural Network and Keras

Neural networks are series of algorithm and data structure to recognize underlying relationships in the set of data by adjusting the weights according to the errors like a human learning from mistakes.

Keras is a well-known deep learning API in Python. It wraps frequently used methods and algorithms in deep learning to enable easy model building and access to the history of model training. It also contains some data pre-processing API, but not as many as sklearn. Therefore, sklearn will be used as well in this project alongside Keras.

## Data and Pre-processing

In this project, a part of the whole CIFAR-10 dataset is taken, which contains 10,000 training samples and 2,000 test samples. There are ten classes, including bird, cat, airplanes and so on.

The dataset is a pickle object. Each entry is an array of 32x32x3=3072 pixels values, ranged from 0 to 255. The data is firstly normalised to [0,1] by dividing 255. Then it goes though the reshape layer in the model, before Conv layers. The input to Conv layers is the shape (32, 32, 3).

## ConvNet

When processing dataset like images, ConvNet is able to capture the Spatial and Temporal dependencies in an image through relevant filters.

In this experiment, a simple and commonly used architecture of ConvNet is designed: two Conv layers and Pooling layers, and two fully connected layers.

In the experiment, different combinations of channels in the two Conv layer is searched to find an optimal combination. Then experiments are done on different optimizers.

A picture containing diagram

Description automatically generated

Fig 1: Convolutional Network [from lecture slides]

## Optimizers

The following is some key points on optimizers used in the experiement.

**Mini-batch Gradient Descent**

Stochastic gradient descent (SGD) performs a parameter update for a size of training data points, usually from 50 to 256. Mini-batch GD is a common optimizer, more efficient than Gradient Descent, and does not have problem when converging to global minima like SGD.

**Momentum**

Ravines are areas where the surface curves much more steeply in one dimension than in another. It usually happens with local optima. SGD will only make hesitant steps towards the optima. SGD with momentum will accelerate this process.

**RMSProp**

RMSProp is an adaptive learning rate method, which means it will change the learning rate in the process, by dividing the learning rate by an exponentially decaying average of squared gradients. It discards the history from extreme past so that it can converge rapidly after finding a convex region.[from lecture slides]

**Adam**

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. It combines the RMSProp and Momentum methods. Adam is very robust and very commonly-used.

## Grid Search

Grid search is a exhaustive search on the combinations of hyperparameters to compute the best one. In this experiment, the search space is the combination of channels at the Conv layers. C1∈{10, 30, 50, 70, 90}, C2∈{20, 40, 60, 80, 100}.

The metrics used for selecting the optimal values are testing accuracies of the models. Since the models may have different converging points, the max testing accuracy in the 1000 epochs are selected. It is assumed that each model got early stopped at the maximum accuracy.

## Dropout

Dropout is a popular regularizer technique used to reduce overfitting, especially in neural networks which contains a large number of neurons in the hidden layers. It drop the neuron at random in each epoch, with a probability of our choice. In this experiment, the dropout probability is 0.2. According to Srivastava et al, the dropout makes the presence of other neuron units unreliable, thus the network cannot generate the complex co-adaptations that remember the seen data but not generalize well on unseen data.[2]

Keras API has a dropout layer which takes in the dropout probability, noise shape and seed. The latter two are not used in our experiment.[3]

# experiments and Results

## A.1 Mini-batch GD

**Cost and Acc**



Fig 2: Acc [C1=50, C2=60, Mini-batch GD, no dropout]



Fig 3: Cost [C1=50, C2=60, Mini-batch GD, no dropout]

**Feature Maps**

**Test Image 1**

A close up of a building

Description automatically generated

Fig 4: Conv layer1 [C1=50, C2=60, Mini-batch GD, no dropout]

A close up of a building

Description automatically generated

Fig 5: Max Pooling layer1 [C1=50, C2=60, Mini-batch GD, no dropout]

A large building

Description automatically generated

Fig 6: Conv layer2 [C1=50, C2=60, Mini-batch GD, no dropout]

A large building

Description automatically generated

Fig 7: Max Pooling layer2 [C1=50, C2=60, Mini-batch GD, no dropout]

**Test Image 2**

A gate in front of a building

Description automatically generated

Fig 8: Conv layer1 [C1=50, C2=60, Mini-batch GD, no dropout]

A close up of a building

Description automatically generated

Fig 9: Max Pooling layer1 [C1=50, C2=60, Mini-batch GD, no dropout]

A large building

Description automatically generated

Fig 10: Conv layer2 [C1=50, C2=60, Mini-batch GD, no dropout]

A large building

Description automatically generated

Fig 11: Max Pooling layer2 [C1=50, C2=60, Mini-batch GD, no dropout]

As observed from the feature maps, the features become more prominent after max pooling, and abstract after a Conv layer.

## A.2 Grid Search

The results are sorted from in descending order.

|  |  |  |
| --- | --- | --- |
| C1 | C2 | Acc |
| 50 | 100 | 0.5405 |
| 70 | 60 | 0.5395 |
| 70 | 100 | 0.5380 |
| 90 | 100 | 0.5345 |
| 90 | 60 | 0.5340 |
| 50 | 80 | 0.5305 |
| 90 | 20 | 0.5260 |
| 30 | 100 | 0.5255 |
| 90 | 80 | 0.5245 |
| 30 | 80 | 0.5225 |
| 90 | 40 | 0.5225 |
| 30 | 60 | 0.5200 |
| 10 | 80 | 0.5195 |
| 50 | 40 | 0.5195 |
| 70 | 80 | 0.5195 |
| 50 | 20 | 0.5185 |
| 30 | 40 | 0.5170 |
| 70 | 20 | 0.5150 |
| 70 | 40 | 0.5150 |
| 30 | 40 | 0.5135 |
| 10 | 60 | 0.5125 |
| 10 | 100 | 0.5120 |
| 10 | 40 | 0.4980 |
| 10 | 20 | 0.4940 |
| 50 | 60 | 0.3400 |

Table 1: Grid search accuracies [Mini-batch GD, no dropout]

For the experiment onwards, C1=50, C1=100 is used.



Fig 12: Acc [C1=50, C2=100, Mini-batch GD, no dropout]

## A.3 Experiment on Optimizers

**Momentum**



Fig 13: Acc [C1=50, C2=100, Momentum, no dropout]



Fig 14: Loss [C1=50, C2=100, Momentum, no dropout]

**RMSProp**

****

Fig 15: Acc [C1=50, C2=100, RMSProp, no dropout]



Fig 16: Loss [C1=50, C2=100, RMSProp, no dropout]

**Adam**

****

Fig 17: Acc [C1=50, C2=100, Adam, no dropout]



Fig 18: Loss [C1=50, C2=100, Adam, no dropout]

**Dropout**

****

Fig 19: Acc [C1=50, C2=100, Mini-batch GD, dropout]



Fig 20: Loss [C1=50, C2=100, Mini-batch GD, dropout]

# Conclusion

**Grid Search**

Comparing models in A.1 (C1=50, C2=60) and A.2 (C1=50, C2=100), we could see the grid search has found a better hyperparameter so the model is improved. The accuracy is improved from 0.3400 to 0.5405.

These are the accuracies within 1000 epochs. As observed from the plots of A.1 model (C1=50, C2=60), the curve has not yet converged. If trained for more epochs, the performance of the model might be better than A.2 (C1=50, C2=100) model. However, the dataset is a lot larger in practice. To ensure the efficiency, we need fast converging in the training process. Therefore (C1=50, C2=100) is a set of improved hyperparameters as the model has higher accuracy and converges faster.

**Momentum**

It is observed from the plots of accuracy and loss that the model converged at about 600 epochs. The maximum converging accuracy is 0.5250, a bit less than Mini-batch GD. The curve is very similar to only Mini-batch GD.

**RMSProp**

The model with RMSProp converges faster than any other model built in this experiment. The converging accuracy is 0.4825, not good compared to other models using the optimal (C1, C2) parameters. The reason is it divides the learning rate by an exponentially decaying average of squared gradients, so it can converge very rapidly after finding a convex region.

After the convergence, we can see serious overfitting from the plots.

This model can be recommended if efficiency and fast convergence is prioritised.

**Adam**

The converging accuracy with Adam as optimizer is 0.4745.

Spikes can be observed from the accuracy and cost, which is unavoidable consequence of Mini-Batch GD with Adam. There are some batches which contains data bad for optimization. These “unlucky” batches are causes of the spike.

**Dropout**

The converging accuracy with mini-batch GD and dropout is 0.5470, which is a improvement from the model without dropout. It can be clearly observed from the gap between training and testing accuracy in the models without dropout, which indicates severe over-fitting. Dropout solved this issue and improved the model.

# References

[1] Cs.toronto.edu. 2020. CIFAR-10 And CIFAR-100 Datasets. [online] Available at: <https://www.cs.toronto.edu/~kriz/cifar.html> [Accessed 1 November 2020].

[2] Dl.acm.org. 2020. Dropout: A Simple Way To Prevent Neural Networks From Overfitting: The Journal Of Machine Learning Research: Vol 15, No 1. [online] Available at: <https://dl.acm.org/doi/10.5555/2627435.2670313> [Accessed 1 November 2020].

[3] Team, K., 2020. Keras Documentation: Dropout Layer. [online] Keras.io. Available at: <https://keras.io/api/layers/regularization\_layers/dropout/> [Accessed 1 November 2020].

Part B: Text Classification

# *Abstract-*The aim of this experiment is to build a model to classify each entry as a paragraph from Wikipedia to 15 categories such as people, company, schools.

**Keywords:** Classification, Neural Networks, Keras, Text Embedding, CNN, RNN

# INTRODuction

The dataset contains the first paragraphs collected from Wikipage entries and the corresponding labels about their category. There are 15 categories.

The data is converted into a vector on word level or character level. The maximum length of the characters or word inputs is restricted to 100.

The combinations of two ways of embedding (word level and char level) and two neural networks (CNN and RNN) are used to build the models. Dropout are experimented on the four networks. The Dropout is applied on the dense layer.

For RNN networks, the experiment tried using different layers (GRU, vanilla RNN, LSTM), increasing number of RNN layers, and add gradient clipping to RNN training.

# Methodology

## Char and Word Embedding

Data scaling is discussed in part A. Something to add in part B is that data exploration could be done in a visual way to give a better idea of which scaling method to use in part B. The reason is that part B have far fewer features compared to part A. This enable us to do a detailed analysis.

## RNN

RNN is a class of neural networks that is powerful for modelling sequence data such as time series or natural language. The reason is that it captures dependency among the data points in the sequence.[from lecture slides]

## GRU, vanilla RNN and LSTM

There are many types of RNN layers. Ones used in the experiment are GRU, vanilla RNN and LSTM.

**Vanilla RNN**

In Vanilla RNN, the hidden-layer activation at time t − 1 is kept by the delay unit and fed to the hidden layer at time t together with the raw input x(t).

A close up of a clock

Description automatically generated

Fig 21: Vanilla RNN [from lecture slide]

**GRU and LSTM**

Vanishing and exploding gradients in backward propagation in Vanilla RNN make it difficult to learn long term dependencies. Therefore Gated RNN are introduced.

GRU and LSTM are two different types of Gated RNN.

LSTM incorporate memory cell state c(t) so it learn when to forget previous hidden states and when to update hidden states.

Diagram, schematic

Description automatically generated

Fig 22: LSTM cell [from lecture slide]

The main difference between GRU and LSTM is that GRU uses a single gating unit to control forgetting factor and the decision to update the hidden unit at the same time, while LSTM uses separate units.

Diagram

Description automatically generated

Fig 23: GRU cell [from lecture slide]

## Gradient Clipping

As mentioned in 7.3, phenomenon called vanishing and exploding gradients happen in backward propagation in Vanilla RNN. Large gradients can move the parameters away from true minimum with gradient descent. Gradient Clipping is applied to solve the issue.

It clip the gradient when it exceeds a threshold. In the experiment, the threshold is set to 2.

# EXperiments and results

## B.1-B.4

**CharCNN**

****

Fig 24: Test Acc [Adam, no dropout]

****

Fig 25: Train Cost [Adam, no dropout]

**WordCNN**

****

Fig 26: Test Acc [Adam, no dropout]

****

Fig 27: Train Cost [Adam, no dropout]

**CharRNN**

****

Fig 28: Test Acc [Adam, no dropout]****

Fig 29: Train Cost [Adam, no dropout]

**WordRNN**

****

Fig 30: Test Acc [Adam, no dropout]



Fig 31: Train Cost [Adam, no dropout]

|  |  |
| --- | --- |
| Network( no dropout) | Converging Test Acc |
| CharCNN | 0.6843 |
| WordCNN | 0.8086 |
| CharRNN | 0.7186 |
| WordRNN | 0.8686 |

Table 2: Converging Acc of the four networks

|  |  |
| --- | --- |
| Network | Avg Runing time (s/epoch) |
| CharCNN | 0.583 |
| WordCNN | 1.129 |
| CharRNN | 7.009 |
| WordRNN | 7.742 |

Table 3: Time per epoch for the four networks

## B.5 Dropout

**CharCNN**

****

Fig 32: Testing Acc [Adam, dropout]

****

Fig 33: Train Cost [Adam, dropout]

**WordCNN**

****

Fig 34: Testing Acc [Adam, dropout]

****

Fig 35: Train Cost [Adam, dropout]

**CharRNN**

****

Fig 36: Testing Acc [Adam, dropout]

****

Fig 37: Train Cost [Adam, dropout]

**WordRNN**

****

Fig 38: Testing Acc [Adam, dropout]

****

Fig 39: Train Cost [Adam, dropout]

|  |  |
| --- | --- |
| Network(dropout) | Converging Test Acc |
| CharCNN | 0.7043 |
| WordCNN | 0.8171 |
| CharRNN | 0.7343 |
| WordRNN | 0.8786 |

Table 4: Converging Acc of the four networks with dropout

## B.6 Model Improvement

**CharRNN**

|  |  |
| --- | --- |
| Network(dropout) | Converging Test Acc |
| Vanilla RNN | 0.1657 |
| LSTM | 0.7086 |
| 2 Layers | 0.7500 |
| Gradient Clipping | 0.7257 |

Table 4: Converging Acc of the four improvement attempts, without dropout, CharRNN

**WordRNN**

|  |  |
| --- | --- |
| Network(dropout) | Converging Test Acc |
| Vanilla RNN | 0.1114 |
| LSTM | 0.8029 |
| 2 Layers | 0.8371 |
| Gradient Clipping | 0.8657 |

Table 5: Converging Acc of the four improvement attempts, without dropout, WordRNN

# conclusion

start

# References

[1] TensorFlow. 2020. Tf.Keras.Callbacks.Earlystopping | Tensorflow Core V2.3.0. [online] Available at: <https://www.tensorflow.org/api\_docs/python/tf/keras/callbacks/EarlyStopping> [Accessed 13 October 2020].