CIS 8392
Topics in Big Data Analytics

#1D Convnets

Yu-Kai Lin

Agenda

- Understanding 1D convolution for sequence data
- 1D pooling for sequence data
- Implementing a 1D convnet

Question: Any issues when training RNNs?

Prerequisites

```
#install.packages("tidyverse")
#install.packages("keras")

library(tidyverse)
library(keras)
#install_keras() # will take a while
```

Some of the deep neural networks are too large to train on a normal laptop, you may use the ones that I have already trained:

```
model <- load_model_hdf5("some_model_filename.h5")
```

Download pretrained models (and their histories) here:
https://www.dropbox.com/s/hpdhvzrx9p9etnl/deeplearning-pretrained.zip?dl=0

If you are using the VM, the pretrained model and history files are stored in C:/CIS8392/data/deeplearning-pretrained/

Recap: convnets

In last week, we learned about convolutional neural networks (convnets) and how they perform particularly well on computer vision problems, due to their ability to operate *convolutionally*, extracting features from local input patches and allowing for representation modularity and data efficiency.

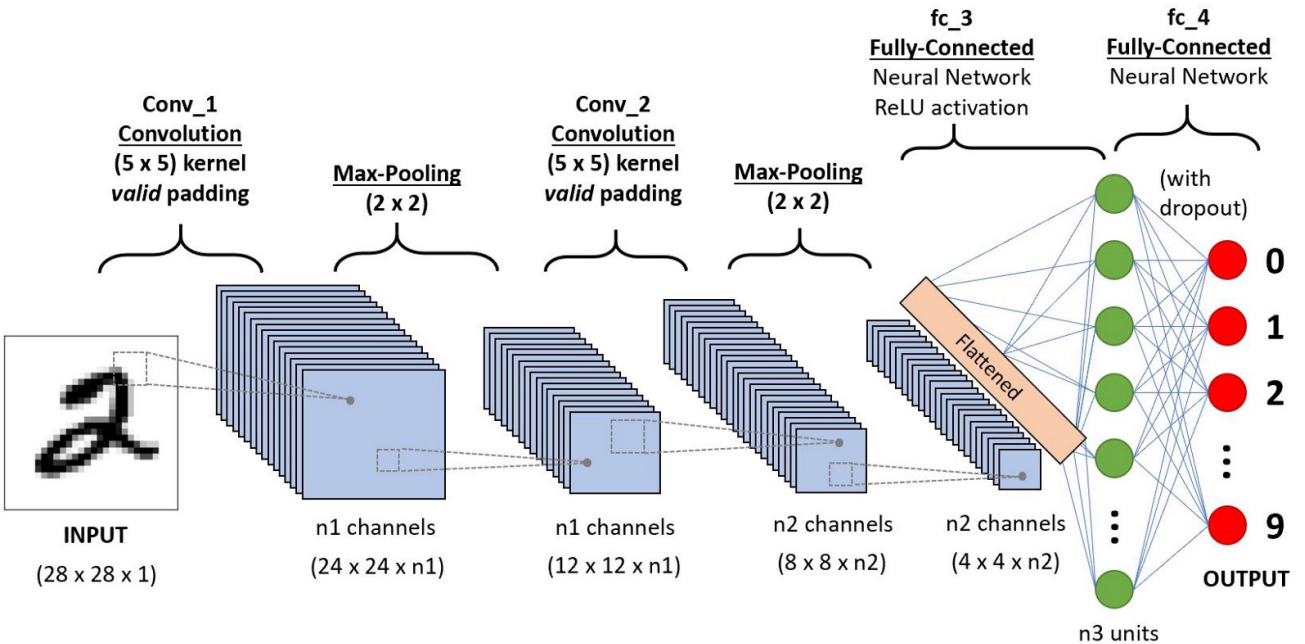


Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

1D convnets

Time can be treated as a spatial dimension, like the height or width of a 2D image.

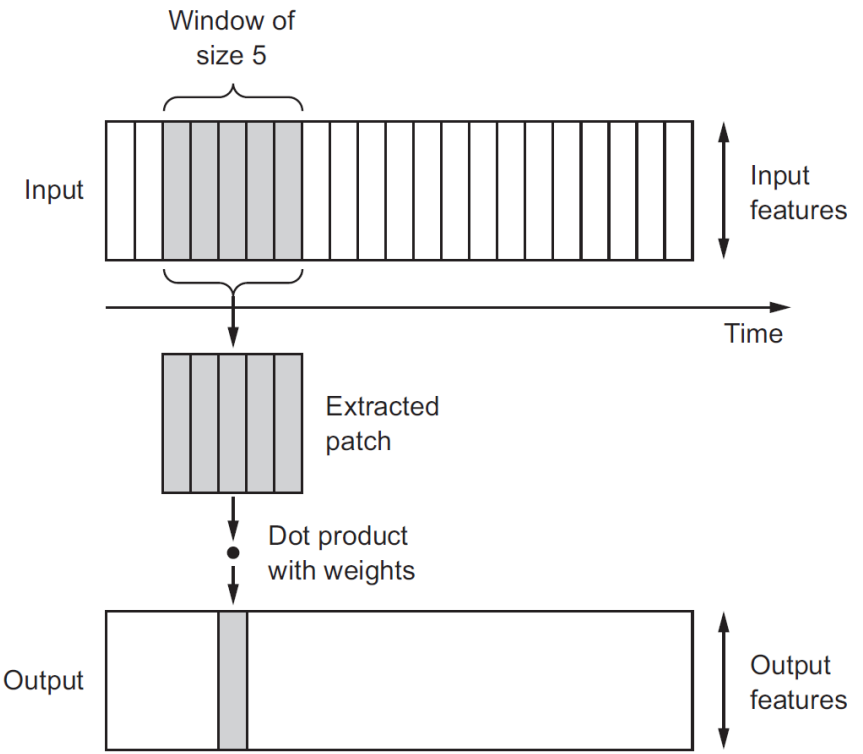
Such 1D convnets can be competitive with RNNs on certain sequence-processing problems, usually at a considerably cheaper computational cost.

Recently, 1D convnets have been used with great success for audio generation and machine translation. In addition to these specific successes, it has long been known that small 1D convnets can offer a fast alternative to RNNs for simple tasks such as text classification and timeseries forecasting.

Understanding 1D convolution for sequence data

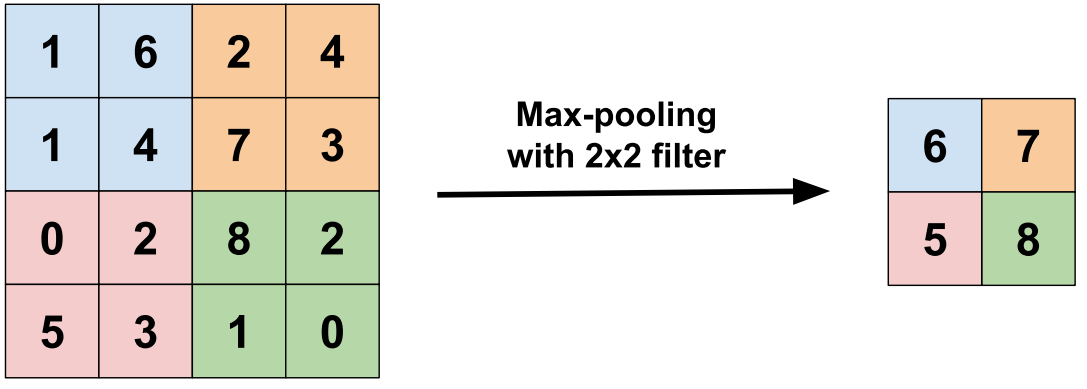
The convolution layers introduced previously were 2D convolutions, extracting 2D patches from image tensors and applying an identical transformation to every patch.

In the same way, we can use 1D convolutions, extracting local 1D patches (subsequences) from sequences.



1D pooling for sequence data

You're already familiar with 2D pooling operations, such as 2D average pooling and 2D max pooling, used in convnets to spatially downsample image tensors.



The 2D pooling operation has a 1D equivalent: extracting 1D patches (subsequences) from an input and outputting the maximum value (max pooling) or average value (average pooling).

Just as with 2D convnets, this is used for reducing the length of 1D inputs (*subsampling*).

Implementing a 1D convnet

In Keras, we use a 1D convnet via the `layer_conv_1d` function, which has an interface similar to `layer_conv_2d`. It takes as input 3D tensors with shape `(samples, time, features)` and returns similarly shaped 3D tensors.

The convolution window is a 1D window on the temporal axis: the second axis in the input tensor.

1D convnets are structured in the same way as their 2D counterparts, which we used in last week: they consist of a stack of `layer_conv_1d` and `layer_max_pooling_1d`, ending in either a `layer_global_max_pooling_1d` or `layer_flatten`, that turn the 3D outputs into 2D outputs, allowing us to add one or more dense layers to the model for classification or regression.

Read what is the difference between Keras' MaxPooling1D and GlobalMaxPooling1D functions?

1D convnets vs 2D convnets

The structure and implementation of 1D/2D convents are very similar.

The main difference is that we can afford to use larger convolution windows with 1D convnets.

With a 2D convolution layer, a 3 × 3 convolution window contains 3 × 3 = 9 feature vectors; but with a 1D convolution layer, a convolution window of size 3 contains only 3 feature vectors. We can thus easily afford 1D convolution windows of size 7 or 9.

Let’s build a simple two-layer 1D convnet and apply it to the IMDB sentiment classification task we’re already familiar with.

Preparing the IMDB data (skip this part if you have done it earlier)

```
max_features <- 10000
max_len <- 500
imdb <- dataset_imdb(num_words = max_features)
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb
cat(length(x_train), "train sequences\n")
```

```
## 25000 train sequences
```

```
cat(length(x_test), "test sequences")
```

```
## 25000 test sequences
```

```
x_train <- pad_sequences(x_train, maxlen = max_len)
x_test <- pad_sequences(x_test, maxlen = max_len)
cat("x_train shape:", dim(x_train), "\n")
```

```
## x_train shape: 25000 500
```

```
cat("x_test shape:", dim(x_test), "\n")
```

```
## x_test shape: 25000 500
```

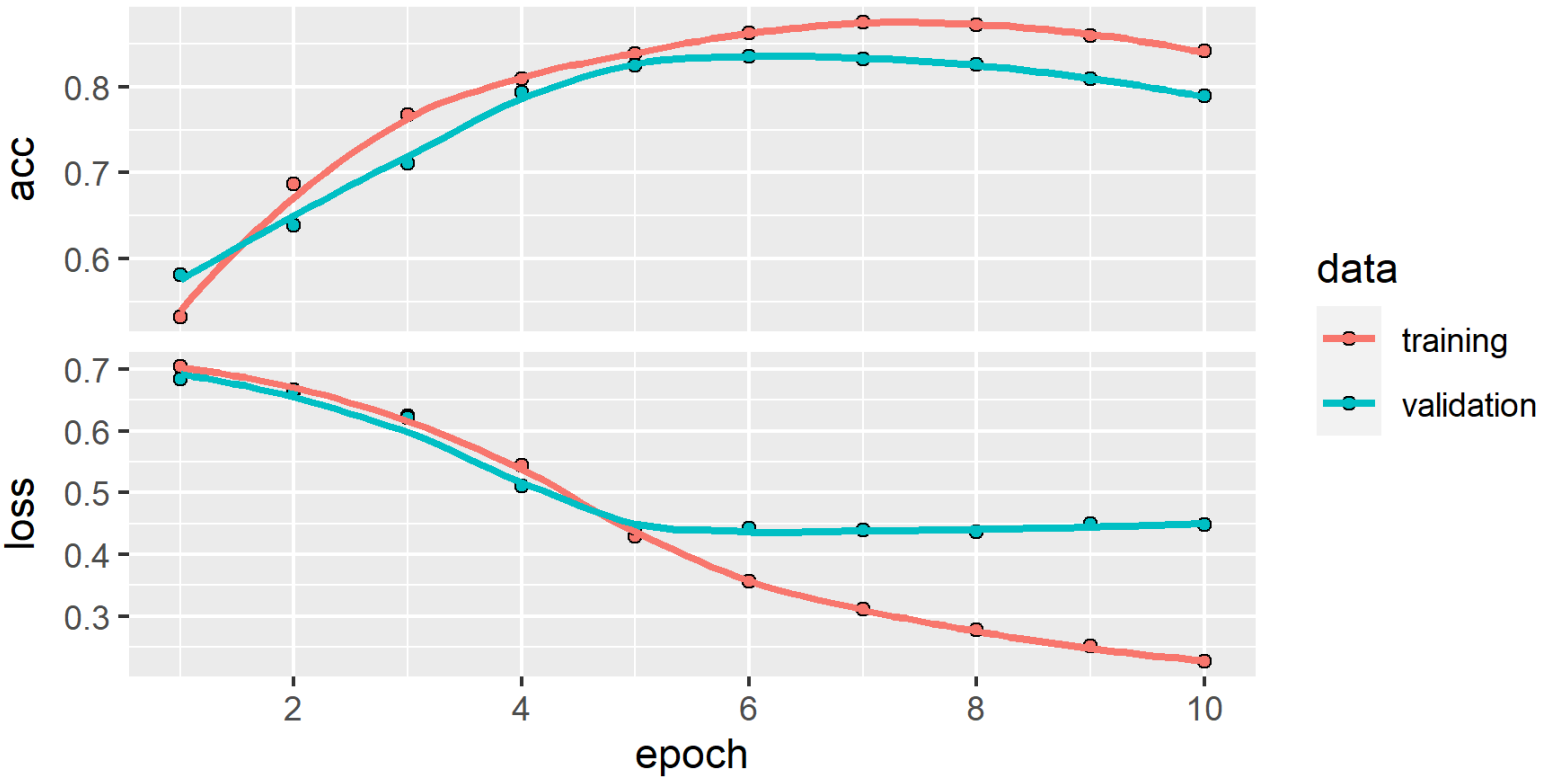
Training and evaluating a simple 1D convnet on the IMDB data

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 128,
                  input_length = max_len) %>%
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%
  layer_max_pooling_1d(pool_size = 5) %>%
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%
  layer_global_max_pooling_1d() %>%
  layer_dense(units = 1)

summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## embedding (Embedding)               (None, 500, 128)                1280000
## ________________________________________________________________________________
## conv1d_1 (Conv1D)                   (None, 494, 32)                 28704
## ________________________________________________________________________________
## max_pooling1d (MaxPooling1D)        (None, 98, 32)                  0
## ________________________________________________________________________________
## conv1d (Conv1D)                     (None, 92, 32)                  7200
## ________________________________________________________________________________
## global_max_pooling1d (GlobalMaxPool (None, 32)                      0
## ________________________________________________________________________________
## dense (Dense)                       (None, 1)                       33
```

```
model %>% compile(
  optimizer = optimizer_rmsprop(lr = 1e-4),
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history <- model %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2
)
```

```
plot(history)
```



Validation accuracy is somewhat less than that of the LSTM, but runtime is faster. This is a convincing demonstration that a 1D convnet can offer a fast, cheap alternative to as RNN on a sentiment-classification task.

Combine CNNs and RNNs for long sequences

Because RNNs are extremely expensive for processing very long sequences, but 1D convnets are cheap, it sometimes can be a good idea to use a 1D convnet as a preprocessing step before an RNN, shortening the sequence and extracting useful representations for the RNN to process.

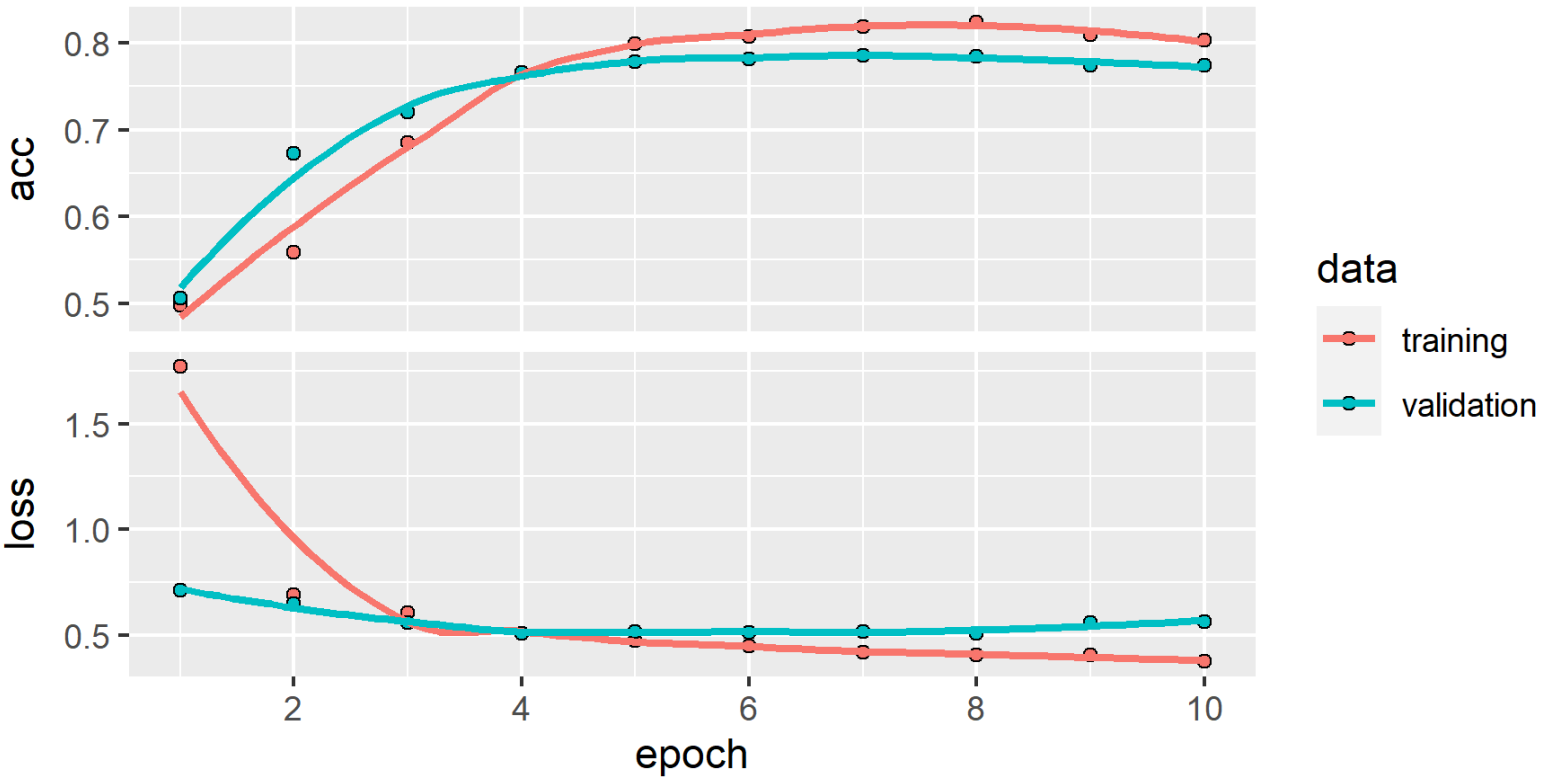
```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 128,
                  input_length = max_len) %>%
  layer_conv_1d(filters = 32, kernel_size = 5, activation = "relu",
                input_shape = list(NULL, dim(data)[[-1]])) %>%
  layer_max_pooling_1d(pool_size = 3) %>%
  layer_conv_1d(filters = 32, kernel_size = 5, activation = "relu") %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 1)
```

```
summary(model)
```

```
## Model: "sequential_1"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## embedding_1 (Embedding)      (None, 500, 128)          1280000
## _________________________________________________________________________
## conv1d_3 (Conv1D)            (None, 496, 32)           20512
## _________________________________________________________________________
## max_pooling1d_1 (MaxPooling1D)  (None, 165, 32)        0
## _________________________________________________________________________
## conv1d_2 (Conv1D)            (None, 161, 32)           5152
## _________________________________________________________________________
## lstm (LSTM)                  (None, 32)                8320
## _________________________________________________________________________
## dense_1 (Dense)              (None, 1)                 33
## =========================================================================
## Total params: 1,314,017
## Trainable params: 1,314,017
## Non-trainable params: 0
## _________________________________________________________________________
```

```
model %>% compile(
  optimizer = optimizer_rmsprop(lr = 1e-4),
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history <- model %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2
)
```

```
plot(history)
```



Your turn

We have learned how to implement 1D CNNs and how to combine 1D CNNs with RNNs.

1. Evaluate the two models using the test data. Which of these models has a better accuracy on test data?

2. Modify the parameters and network structures on both models. Can you get an even better accuracy on the test data?