CQS Summer Institute: Machine Learning and Statistics in Data Science

Matthew S. Shotwell, Ph.D.

Department of Biostatistics
Vanderbilt University Medical Center
Nashville, TN, USA

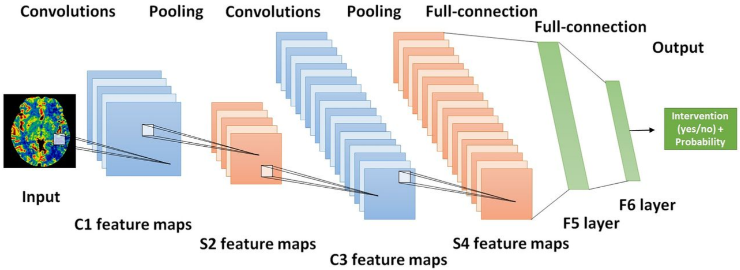
August 8, 2019

Course Overview

- Syllabus and R code:
- https://github.com/biostatmatt/cqs-ml-stat-r
- Monday: Intro and Data Management
- Tuesday: Supervised Learning Part 1
- Wednesday: Supervised Learning Part 2
- Thursday: Supervised Learning Part 3
- Friday: Unsupervised Learning

Deep learning

- Deep learning uses deep NNs
- Deep NNs are simply NNs with many layers, complex connectivity, and processing steps between layers:

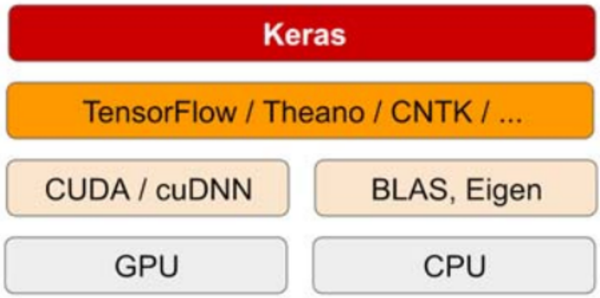


Complex NNs in R

- No (good) native R libraries for complex NNs
- R can interface to good libraries, notably Keras
- See https://keras.rstudio.com/
- "Deep Learning with R" by François Chollet and J. J. Allaire is a great reasource, and provided a basis for this presentation, including the examples many of the figures.

Keras in R

- Keras is a high-level model-building library for NNs; legos for NNs
- Keras can be used in R, Python, and other software
- Low-level computing handled by a “backend” library
- Backend libraries include TensorFlow, Theano, CNTK



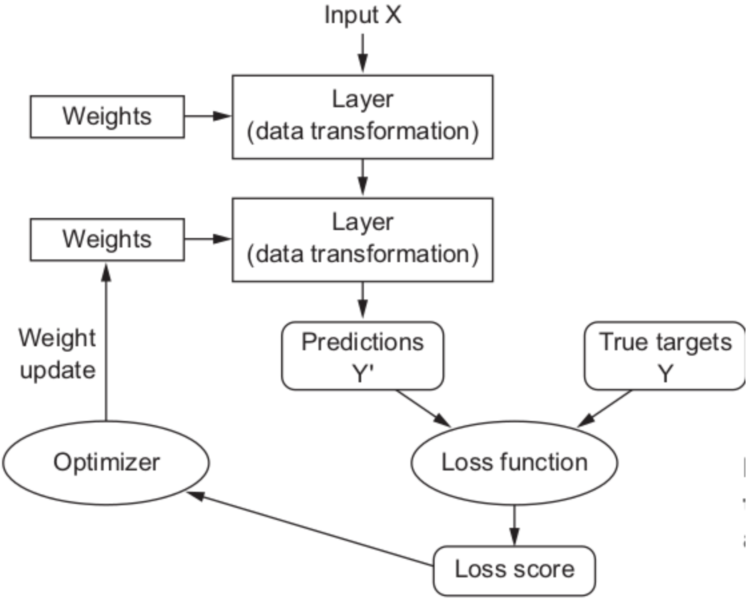
Alternative method to set up Keras workstation

- set up a cloud computer on Amazon AWS
- get one that's already set up with Keras
- take advantage of GPU computing
- GPU gives 5 to 10 times speed improvement
- costs about $0.90/hr
- https://tensorflow.rstudio.com/tools/cloud_gpu

Modeling steps in Keras overview

1. define training data: input and target tensors
2. define network of layers mapping inputs to targets
3. choose loss function, optimizer and metrics (e.g., accuracy)
4. do optimization

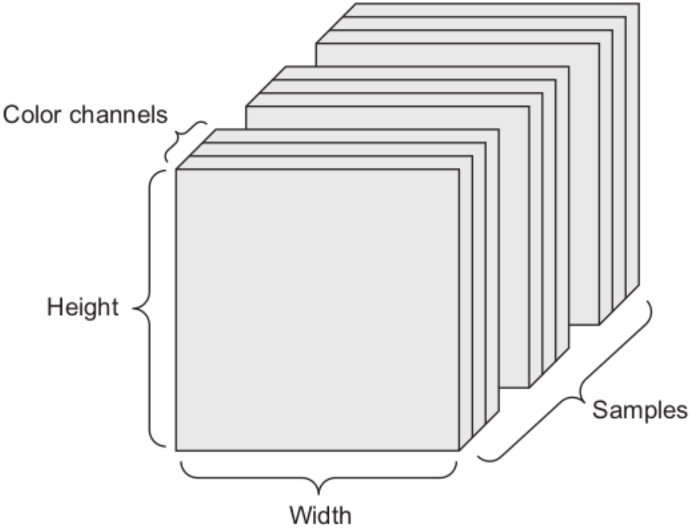
NN in Keras Overview



Data representation in Keras

- in Keras, data are stored as tensors
- scalars - 0D tensors
- vectors - 1D tensors
- matrices - 2D tensors
- every tensor has attributes
  - rank, number of axes
  - shape, dimensions in each axis
  - type, data type, e.g., integer
- 100×15 numeric matrix has rank 2, shape (100, 5), and type 'float'
- in Keras, all data must usually be floating point numeric data

4D color image data tensor



Step 1. Load and prep data in Keras

- Keras as some built-in data, including MNIST zipcode data
- first dimension is the 'sample axis'
- B&W image data have three axes: (samples, height, width)
- color image data typically have extra 'channel' axis

```
mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y

> str(train_images)
int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
> str(train_labels)
int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...
```

Step 1. Load and prep data in Keras

- Keras as some built-in data, including MNIST zipcode data
- first dimension is the ‘sample axis’
- B&W image data have three axes: (samples, height, width)
- color image data typically have extra ‘channel’ axis

```
train_images <- array_reshape(train_images, c(60000, 28 * 28))
train_images <- train_images / 255
test_images <- array_reshape(test_images, c(10000, 28 * 28))
test_images <- test_images / 255

train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)
```

Step 2. Defining Keras model

- the `keras_model_sequential()` function creates feed-forward NNs
- this type is made of "linear stack" of layers

```
model <- keras_model_sequential() %>%
  layer_dense(
    units = 32,
    activation="relu",
    input_shape = c(28*28)) %>%
  layer_dense(
    units = 10,
    activation = "softmax")
```

Choosing loss and last layer activation

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Step 3. Compile Keras model

- the `compile()` to specify optimizer, loss function, and metrics to monitor
- `compile()` modifies the model "in place"

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

Step 4. Fit Keras model

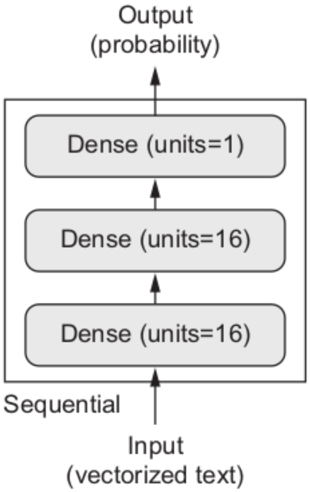
- use the `fit()` to pass the input and target data to the model for optimization, and to select the batch size and number of training epochs

```
model %>% fit(
  train_images,
  train_labels,
  batch_size = 128,
  epochs = 10
)
```

Keras example 1: Classifying moview reviews

- IMDB data set has 50k polarized reviews
- written reviews are a sequence of words preprocessed into a sequence of integers that index a dictionary
- human has read each review and classified each as positive or negative
- about 50% positive review, 50% negative
- create NN to classify reviews

IMDB Classification Network

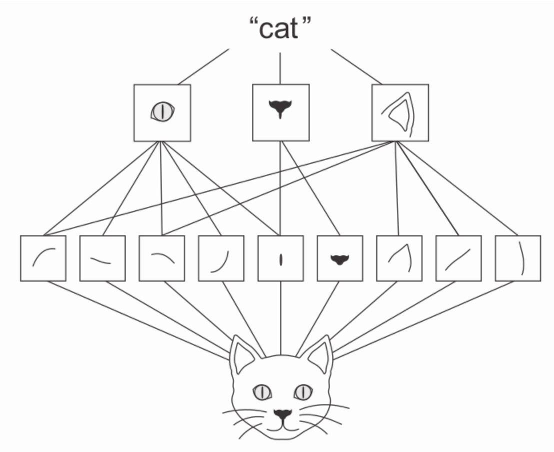


Classifying movie reviews: `keras-imdb-reviews.R`

Convnets

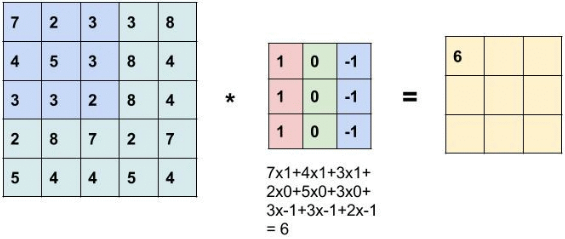
- convnets (convolutional NNs)
- convolution involves weight sharing and local connectivity
- hidden layers are grouped into filters
- each filter is a "shape detector"
- multiple layers can learn more complex shapes

Layers of shape detectors



Convolution

A single filter does this:



Convolution

For each convolutional layer:

- must specify size of patches from input
- must specify the number of filters
- must specify padding: “valid” or “same”
- must specify stride

Padding: valid

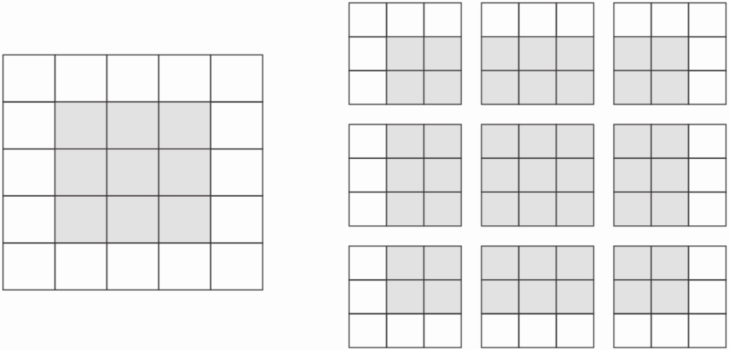


Figure 5.5 Valid locations of 3 × 3 patches in a 5 × 5 input feature map

Padding: same

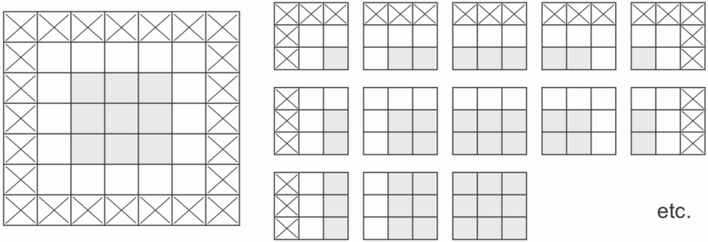


Figure 5.6 Padding a 5 × 5 input in order to extract 25 3 × 3 patches

Stride: 2×2

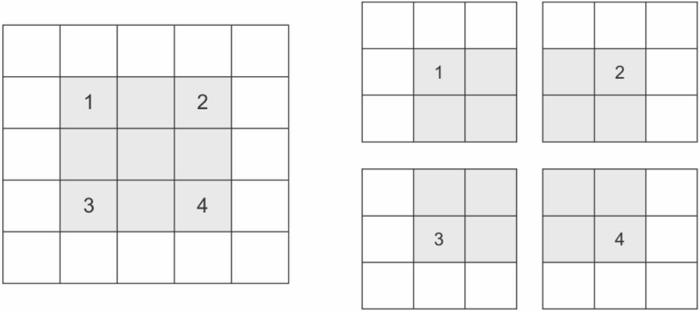


Figure 5.7 3 × 3 convolution patches with 2 × 2 strides

Classifying zipcodes:

`keras-mnist-zipcodes.R`

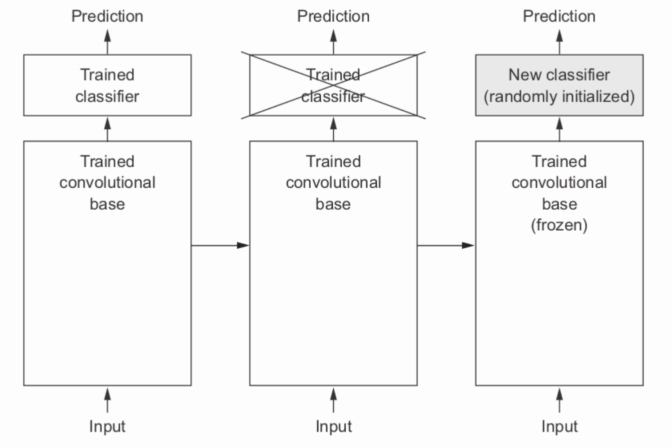
Starting from scratch vs. pretrained convnet

- it's possible to train deep NN on few hundred images
- alternately, can fine-tuning a pretrained network
- or use pretained network for feature extraction
- pretrained network with large general dataset can serve as a generic model for visual classification that we can use in other problems; this is a feature of deep learning that isn't available in shallow methods

Using pretrained convnet: feature extraction/fine-tuning

- use pretrained network to extract features
- from “convolutional base” of pretrained network
- use derived features to train new convnet

Using pretrained convnet: feature extraction/fine-tuning



Using pretrained convnet: feature extraction/fine-tuning

- Get convolutional base from VGG16 NN trained on ImageNet

```
conv_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(150, 150, 3)
)
```

Using pretrained convnet: feature extraction/fine-tuning

```
> conv_base
Model
Model: "vgg16"
________________________________________________________________________________
Layer (type)                    Output Shape                  Param #
================================================================================
input_1 (InputLayer)            [(None, 150, 150, 3)]         0
________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 150, 150, 64)          1792
________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 150, 150, 64)          36928
________________________________________________________________________________
block1_pool (MaxPooling2D)      (None, 75, 75, 64)            0
________________________________________________________________________________
block2_conv1 (Conv2D)           (None, 75, 75, 128)           73856
________________________________________________________________________________
block2_conv2 (Conv2D)           (None, 75, 75, 128)           147584
________________________________________________________________________________
block2_pool (MaxPooling2D)      (None, 37, 37, 128)           0
________________________________________________________________________________
block3_conv1 (Conv2D)           (None, 37, 37, 256)           295168
________________________________________________________________________________
block3_conv2 (Conv2D)           (None, 37, 37, 256)           590080
________________________________________________________________________________
block3_conv3 (Conv2D)           (None, 37, 37, 256)           590080
________________________________________________________________________________
block3_pool (MaxPooling2D)      (None, 18, 18, 256)           0
________________________________________________________________________________
block4_conv1 (Conv2D)           (None, 18, 18, 512)           1180160
________________________________________________________________________________
block4_conv2 (Conv2D)           (None, 18, 18, 512)           2359808
________________________________________________________________________________
block4_conv3 (Conv2D)           (None, 18, 18, 512)           2359808
________________________________________________________________________________
block4_pool (MaxPooling2D)      (None, 9, 9, 512)             0
________________________________________________________________________________
block5_conv1 (Conv2D)           (None, 9, 9, 512)             2359808
________________________________________________________________________________
block5_conv2 (Conv2D)           (None, 9, 9, 512)             2359808
________________________________________________________________________________
block5_conv3 (Conv2D)           (None, 9, 9, 512)             2359808
________________________________________________________________________________
block5_pool (MaxPooling2D)      (None, 4, 4, 512)             0
================================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Using pretrained convnet: feature extraction/fine-tuning

- Add the last couple of layers onto pretrained convolutional base

```
model <- keras_model_sequential() %>%
  conv_base %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

freeze_weights(conv_base)
```

Using pretrained convnet: feature extraction/fine-tuning

```
> model
Model
Model: "sequential_3"
________________________________________________________________________________
Layer (type)                      Output Shape                    Param #
================================================================================
vgg16 (Model)                     (None, 4, 4, 512)               14714688
________________________________________________________________________________
flatten_1 (Flatten)               (None, 8192)                    0
________________________________________________________________________________
dense_8 (Dense)                   (None, 256)                     2097408
________________________________________________________________________________
dense_9 (Dense)                   (None, 1)                       257
================================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
________________________________________________________________________________
```

Feature extraction examples

- https://github.com/jjallaire/deep-learning-with-r-notebooks/tree/master/notebooks
- 5.2 and 5.3