Deep Learning in R with Keras

Doug Ashton
Twitter: @mangothecat
Email: training@mango-solutions.com

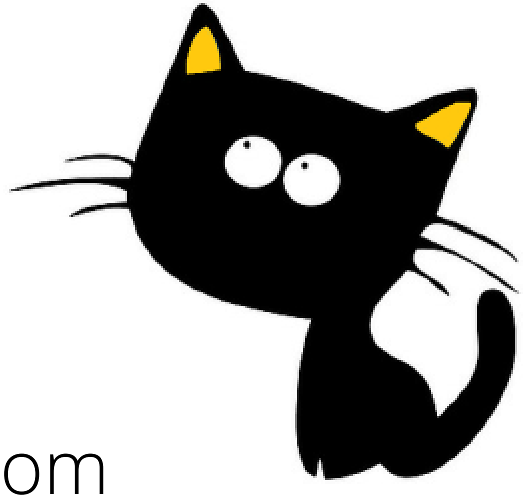
Agenda

- Introduction to Deep Learning
- First neural network with Keras
- Networks for Spatial Data (CNN)

About Us

- Your Trainer:
  - Principal Data Scientist @ Mango
  - @dougashton

- Mango
  - @mangothecat
  - mangothecat
  - training@mango-solutions.com



3 Core Teams @ Mango

Data Science

Customer-focused analytic consultants with math/stat backgrounds using technologies such as R, SAS, Python, Spark & Julia

Data Engineering

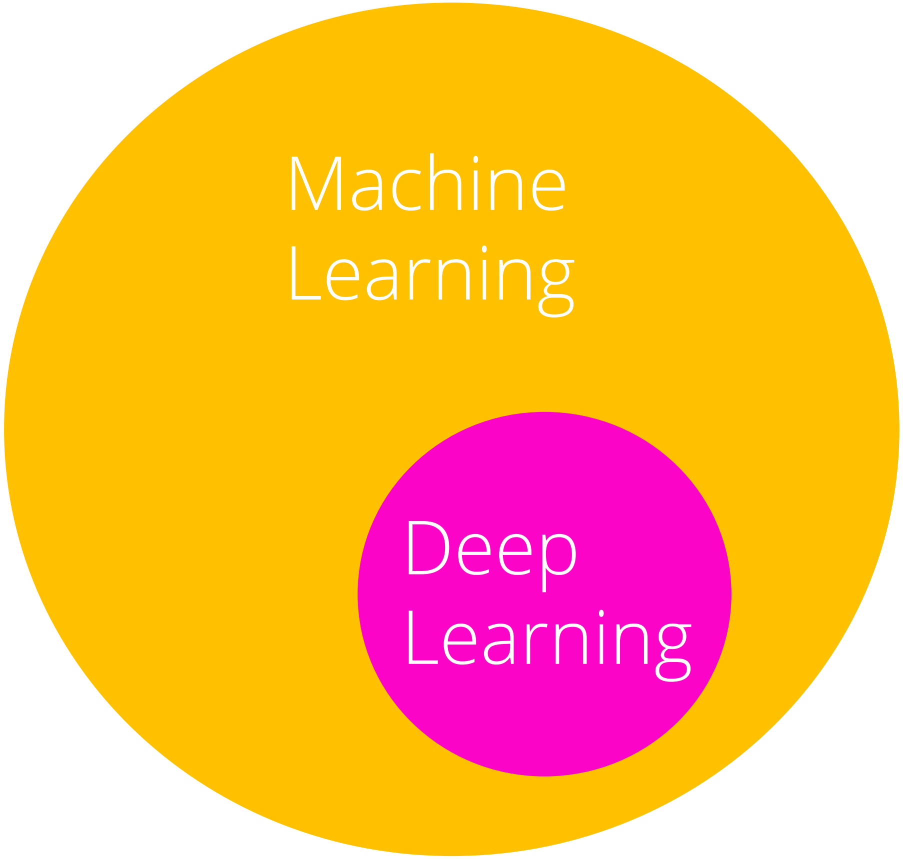
IT Consultants creating and supporting robust, performant and scalable analytic infrastructure using server, grid or cloud

Data Products

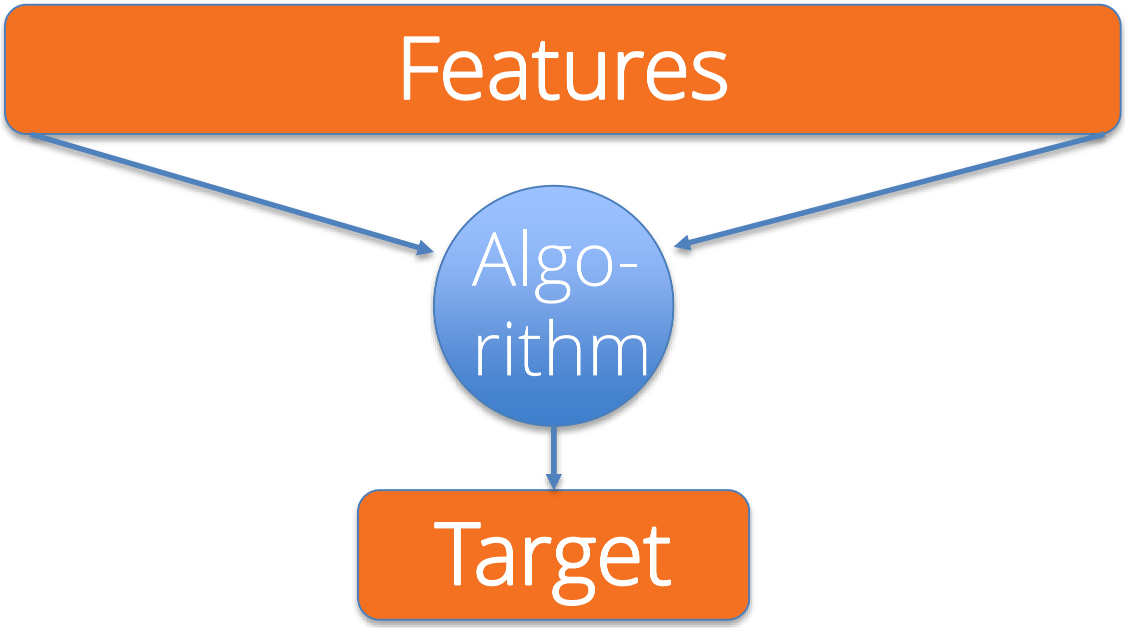
Software Developers building rich analytic web or desktop applications using technologies such as Java, .NET and JavaScript

Introduction to Deep Learning

What is Deep Learning?

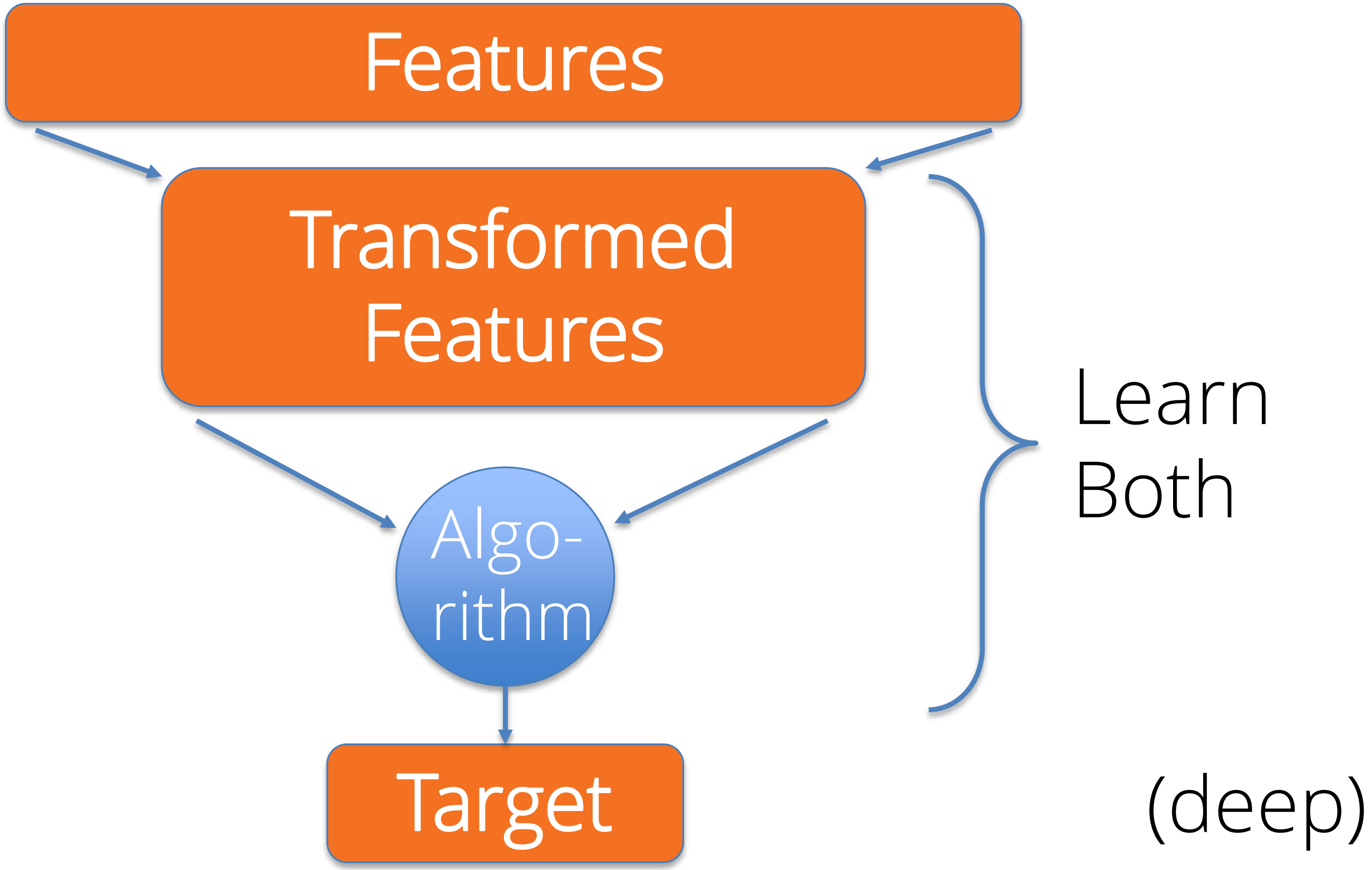


What is Deep Learning?



(shallow)

What is Deep Learning?



What Does it Solve?

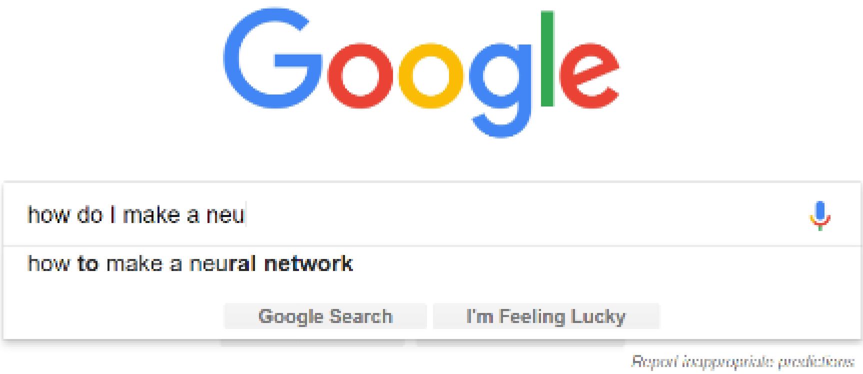
- Unstructured
  - Features are learned rather than designed
- Big
  - Generally need lots of data
- Familiar
  - Can reuse models on new problems

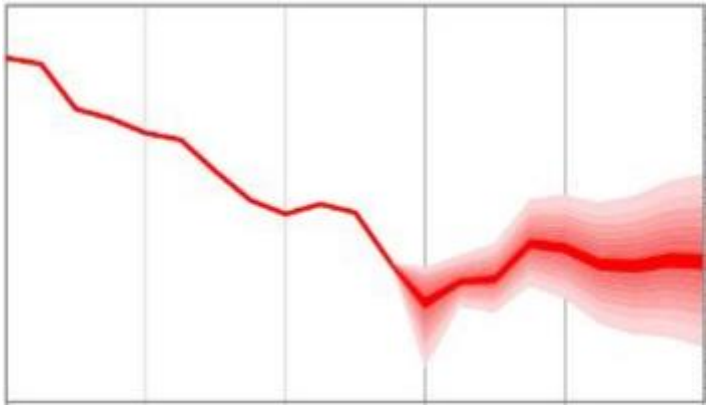
Spatial

- Computer vision
- Audio
- Time series: pattern recognition

Sequential

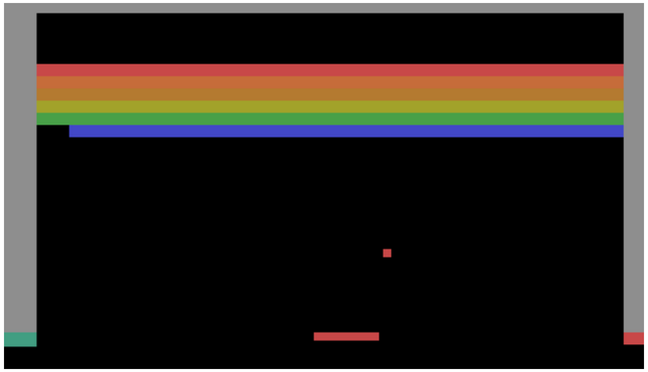
- Language
- Time series: Forecasting





Reinforcement/Adversarial

- AlphaGo
- Generative Networks



Why Now?

- Breakthrough in underlying algorithm
  - Back Propagation
- Massive increase in computer power
  - GPU / TPU
- Much larger datasets available
- Keras…

Neural Networks

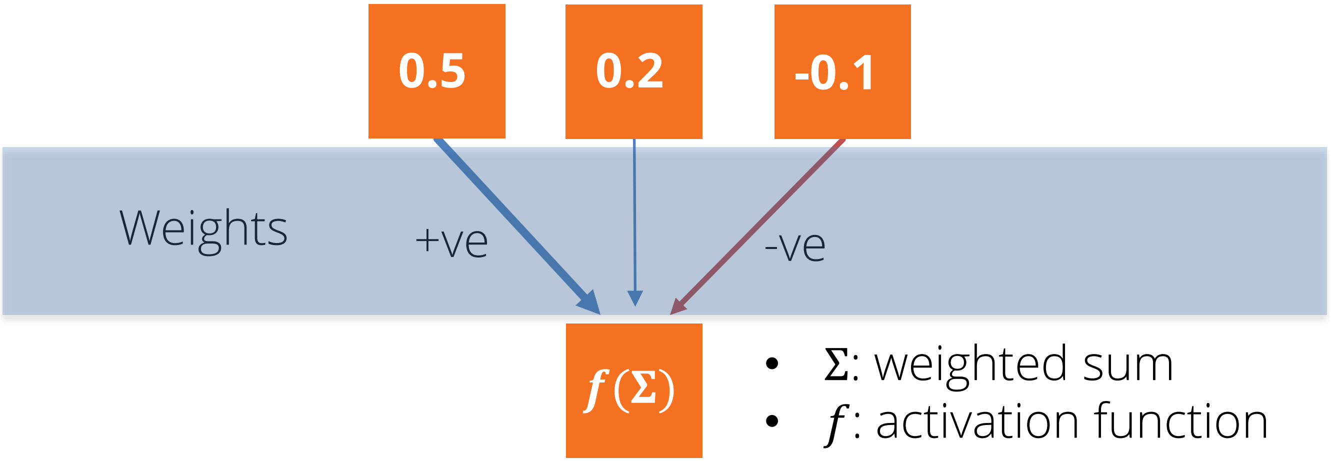
nodes

edges

A Neuron

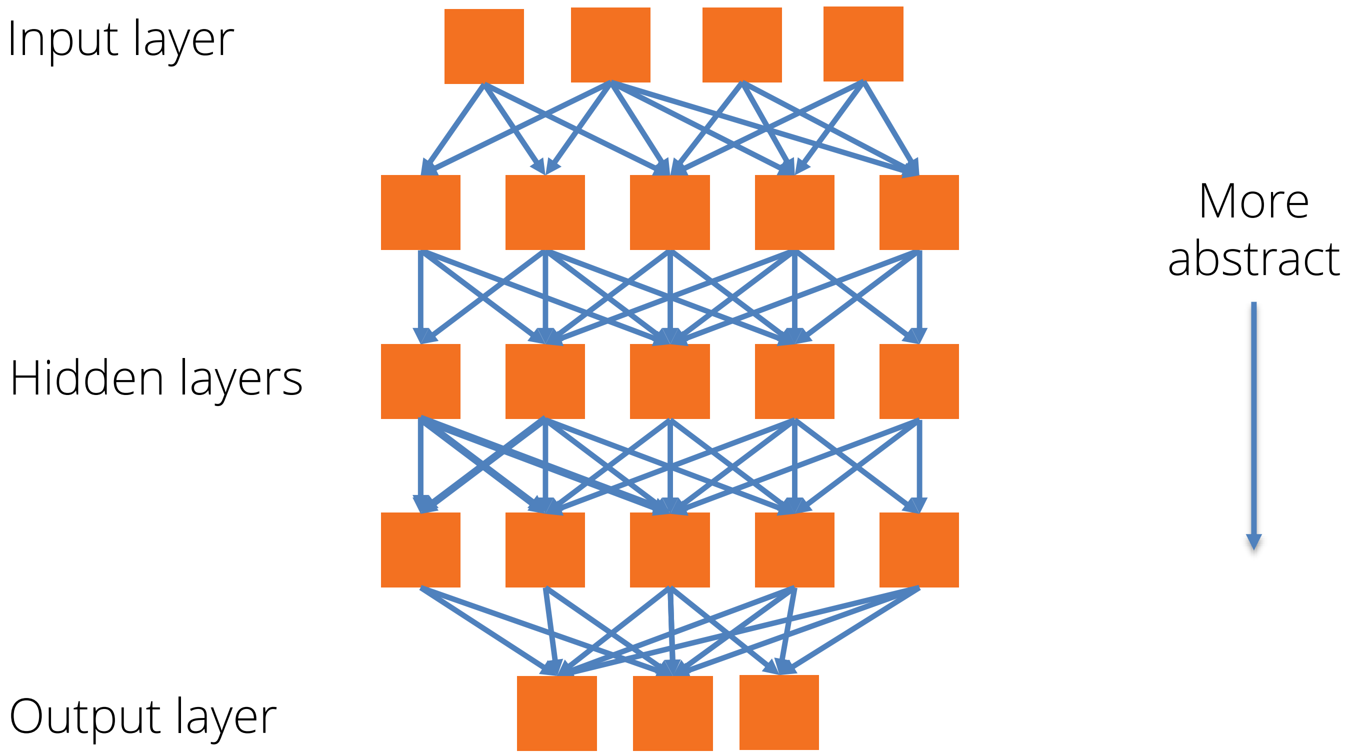
0.2

Neurons



- Σ: weighted sum
- f: activation function

Neural Network



Iris Neural Network

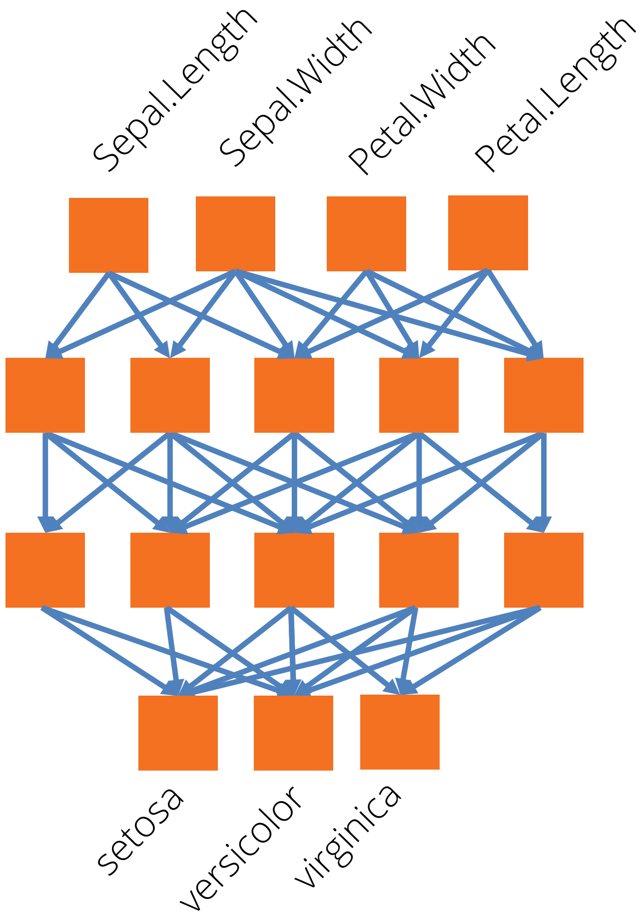
Sepal.Length Sepal.Width Petal.Width Petal.Length

`iris[1,1:4]` 5.1 3.5 1.4 0.2 Features x

`iris[1,5]` 1 0 0 Target y

setosa versicolor virginica

Iris Neural Network



TensorFlow

- Turns equations into dataflow graphs
  - https://www.tensorflow.org

- Efficient numerical solver
- Built for CPU, GPU, and TPU
- Not only for neural networks

TensorFlow and R

- RStudio built an R interface
  – https://tensorflow.rstudio.com

- Python <-> R handled by reticulate
  – https://rstudio.github.io/reticulate

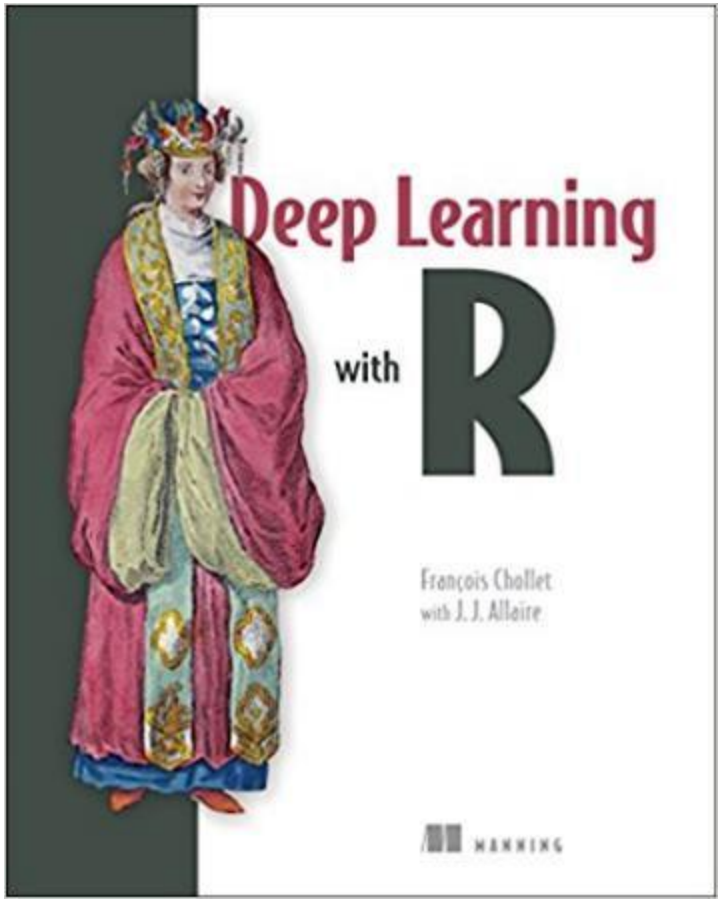
Keras

- High level interface specifically for neural networks
  - https://keras.io
  - François Chollet
- Works with multiple backends
  - TensorFlow, CNTK, Theano

Keras and R

- Rstudio built an interface to Keras
  - https://keras.rstudio.com
- Works with multiple backends
  - TensorFlow, CNTK, Theano

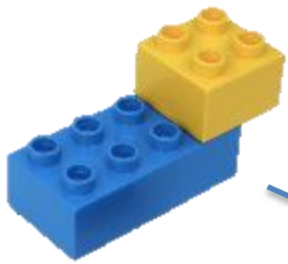
Keras and R Book



Deep Learning with R
- François Chollet
- J. J. Allaire

Manning

How it fits together



Keras

$$\sigma = \sum_i z_i$$

$$p_j = \frac{e^{x^T w_j}}{\sum_k e^{x^T w_k}}$$

TensorFlow

00100110

Alternatives for R Users

- MXNet
  - https://mxnet.incubator.apache.org/api/r/

RStudio Cloud

https://rstudio.cloud/project/489173

- Make an account
- Make a copy of the project 

github.com/mangothecat/keras-workshop

On your own machine

```
install.packages(c("tidyverse", "mlbench",
                   "recipes", "rsample",
                   "keras"))

library(keras)
install_keras() # can take a while
```

First Keras Model

First Keras Model

- Prepare Data
- Model
- Evaluate


Prepare Data

Prepare Data

- Split train and test
- Numeric Matrices/Arrays
  - Factors
  - Scaling
  - Missing values

Prepare - Split Data

```
library(rsample)

data_split <- initial_split(iris,
                            strata = "Species",
                            prop = 0.8)

fullData <- list(train = analysis(data_split),
                 test = assessment(data_split))
```

Prepare - Recipes

- Reusable pre-processing recipes
  - Define a “recipe”
  - “prep” on training data
  - “bake” on test data

```
library(recipes)
empty_recipe <- recipe(Species ~ .,
                       data = fullData$train)
empty_recipe
```

Prepare - One Hot Encode

```
library(recipes)

dummy_recipe <- empty_recipe %>%
  step_dummy(Species, one_hot = TRUE,
             role = "outcome")

dummy_recipe %>%
  prep(fullData$train) %>%
  bake(fullData$train, all_outcomes()) %>%
  head()
```

Prepare - Centre Scaling

```
scale_recipe <- empty_recipe %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

scale_recipe %>%
  prep(fullData$train) %>%
  bake(fullData$train,
       all_predictors()) %>%
  head()
```

Prepare - NAs

- Can’t have NAs
- Impute 0 (mean)
  - `map(fullData, replace_na, replace = 0)`
- Or look at recipes step_[*]impute() functions
- No NAs in `iris`

All together

```
iris_recipe <- recipe(Species ~ .,
                      data = fullData$train) %>%
  step_dummy(Species, one_hot = TRUE,
             role = "outcome") %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  prep(training = fullData$train)
```

Prepare - Matrices

```
## Create x and y matrix

xIris <- map(fullData,~bake(object = iris_recipe,
                            newdata = .x,
                            all_predictors(),
                            composition = "matrix"))

yIris <- map(fullData,~bake(object = iris_recipe,
                            newdata = .x,
                            all_outcomes(),
                            composition = "matrix"))
```

Exercise

- Load the Breast Cancer Data
- Create a train/test split (80/20)
- Remove the ID column
- Prepare the data for Keras
  - Create dummy variables
  - Scale the data
  - Replace missing values

Model

Model

- Networks can have complex shapes
- Sequential models are linear stack

```
model <- keras_model_sequential()
```

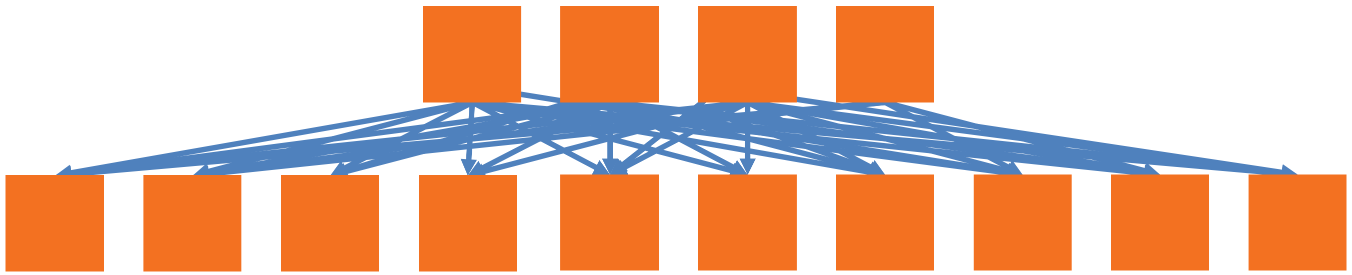
- Model objects *change in place*

Model - Layers

```
model %>%
    layer_dense(units = 10,
                input_shape = 4)
```

- Only need `input_shape` once
- Shape doesn't include observations

Model - Dense Layers



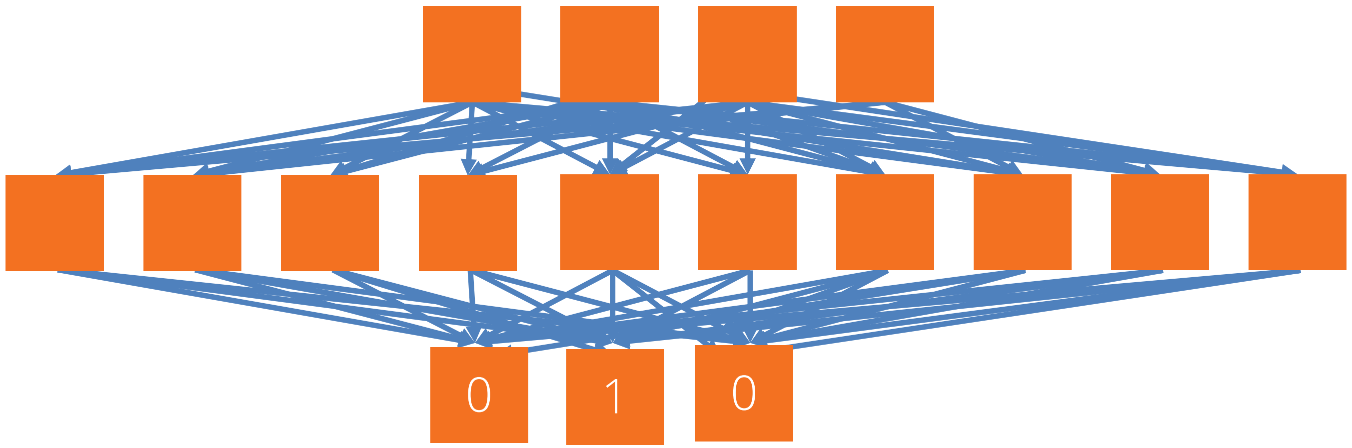
```
model %>%
    layer_dense(units = 10,
                input_shape = 4)
```

Model - Softmax Layer

```
model %>%
    layer_dense(units = 3,
                activation = 'softmax')
```

- Usually on the output
- Use for categorical output

Model - Softmax Layer



Model - Summary

```
> model
Model
________________________________________________________________________
Layer (type)                         Output Shape                  Param #
========================================================================
dense_1 (Dense)                      (None, 10)                    50
________________________________________________________________________
dense_2 (Dense)                      (None, 3)                     33
========================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
________________________________________________________________________
```

Compile

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

- Optimizer: Mostly rmsprop
- Metrics: Mostly accuracy
- Loss: 3 main choices

Compile - Loss

Output	Loss Function
Binary Classification	binary_crossentropy
Multi-class Classification (single label)	categorical_crossentropy
Multi-class Classification (multiple labels)	binary_crossentropy
Regression	mse

Fit

```
history <-
  model %>%
    fit(xIris$train,
        yIris$train,
        epochs = 100,
        validation_data =
             list(xIris$test,
                  yIris$test))
```

Exercise

Using the pre-cleaned Breast Cancer Data:

- Create a model with:
  - A dense layer with 5 hidden units
  - A dense, output layer using the "sigmoid" activation function
- Compile the model using "binary_crossentropy" as the loss function
- Fit the model over 20 epochs

Evaluate


5.13
ROLEX
2.26
SETS
GAMES
POINTS
PREVIOUS SETS
J. HENIN-HARDENNE
V
Miss E. DANIILIDOU
1
5
0
1
6
15

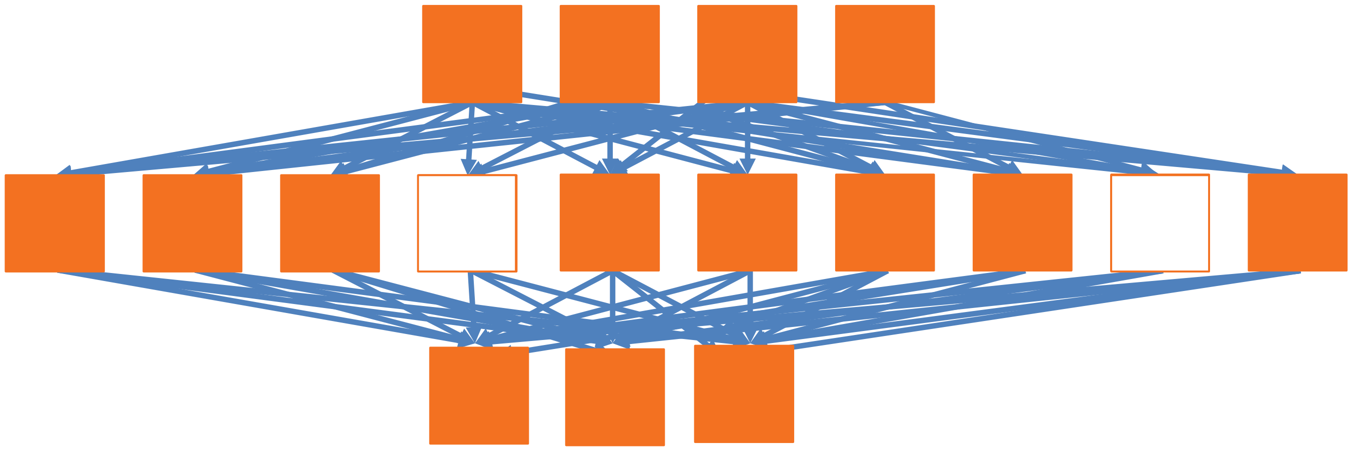
Exercise

- Using the model that you built in the last exercise and the pre-cleaned test breast cancer data evaluate the performance of your model
- Predict the classes for the test data

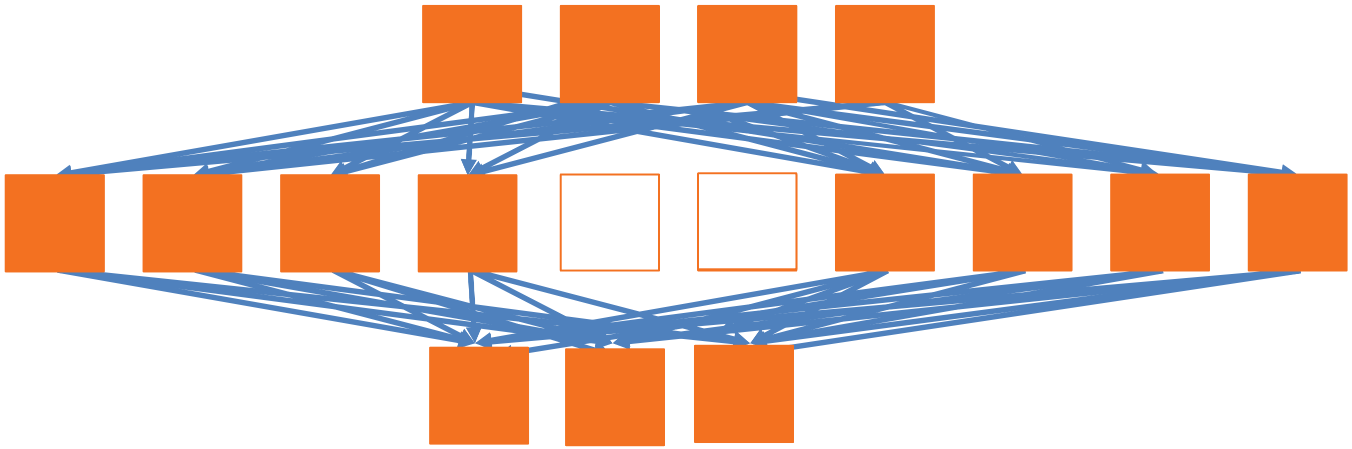
Improving the Model

- Change number of hidden units
- Add more layers
- Add dropout
  - Helps prevent overfitting
- Mostly trial and error

Dropout

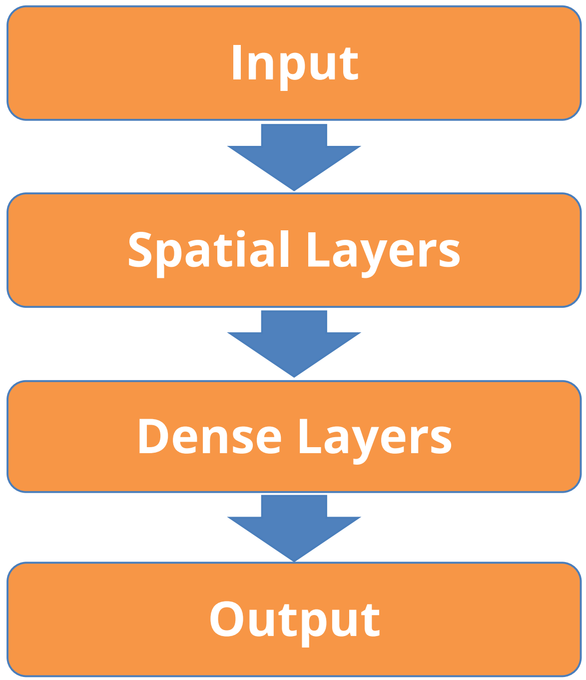


Dropout



Networks for Spatial Data

Convolutional Neural Networks



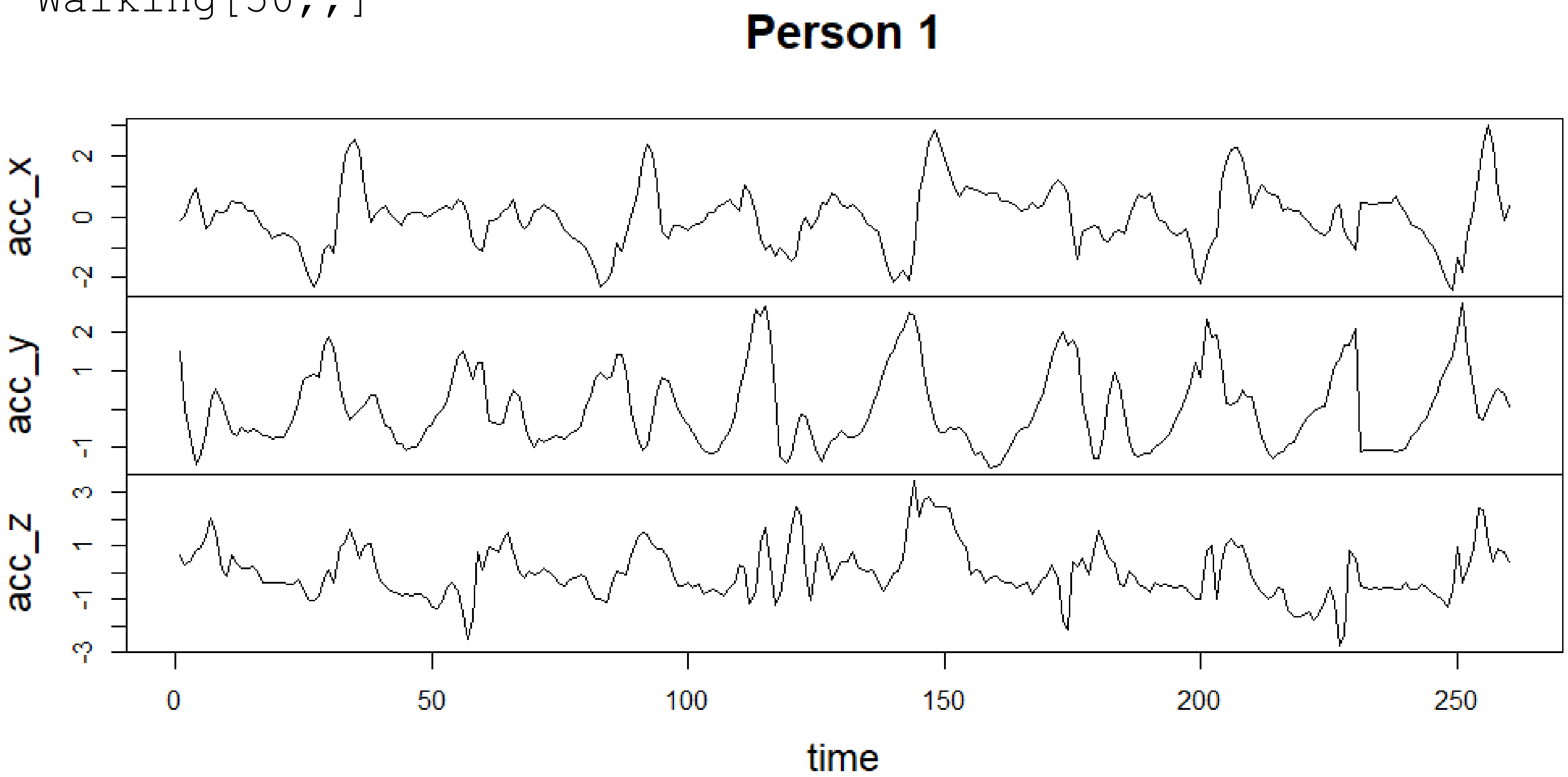
Walking Data

- Accelerometer data from the UCI
- Filtered to walking activity
- 15 Different people
- Can we recognise someone by their gait?
- Chopped into 5 second chunks

https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer

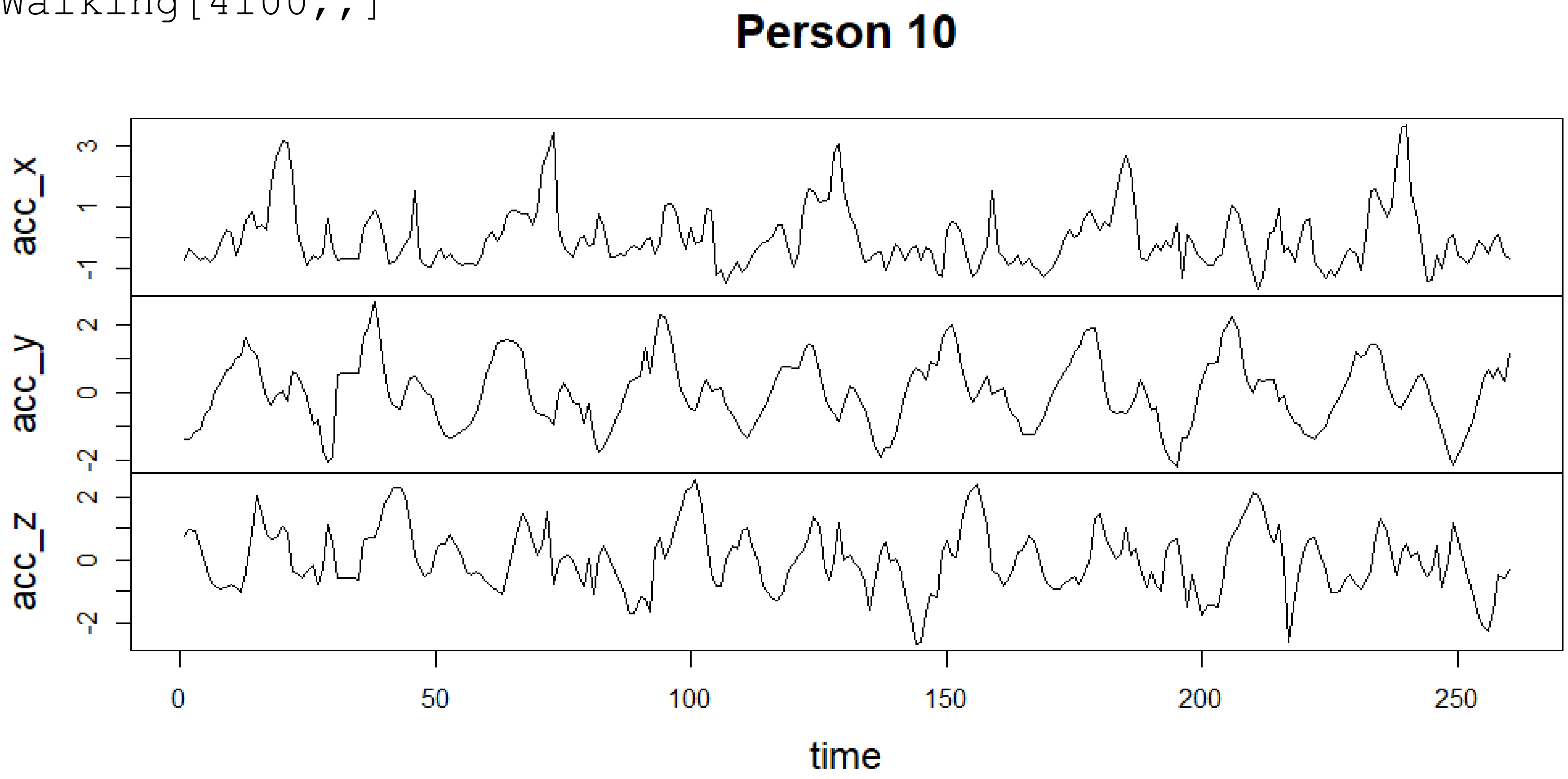
Walking Data

```
Walking[50,,]
```

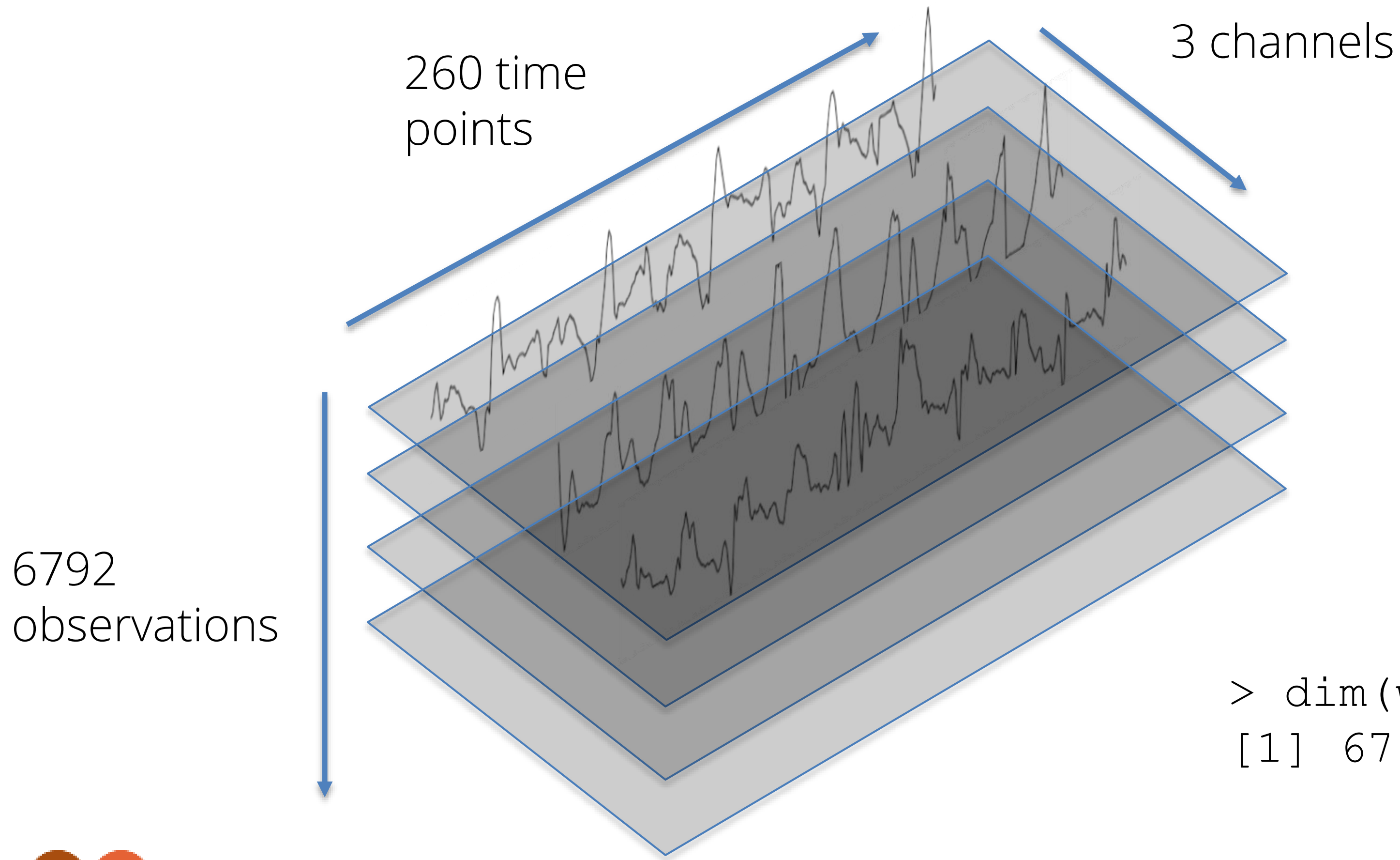


Walking Data

```
Walking[4100,,]
```



Walking Data




```
> dim(walking$x)
[1] 6792  260    3
```

Exercise

- Load the walking data.

```
walking <- readRDS("/data/walking.rds")
```

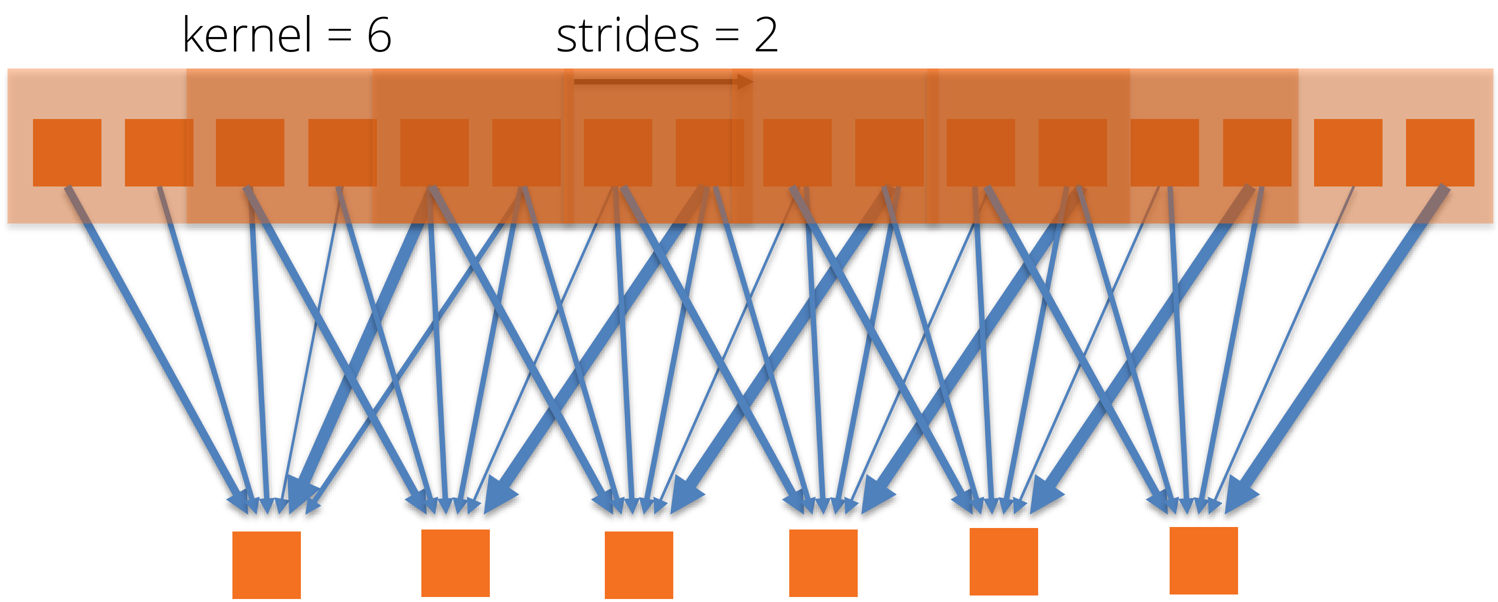
- Create two lists, xWalk and yWalk, each with an 80:20 split of train and test sets for x and y data respectively.
- (hint) nr <- nrow(walking$y)
- ids <- sample(nr, size = nr*0.8)

Walking Data

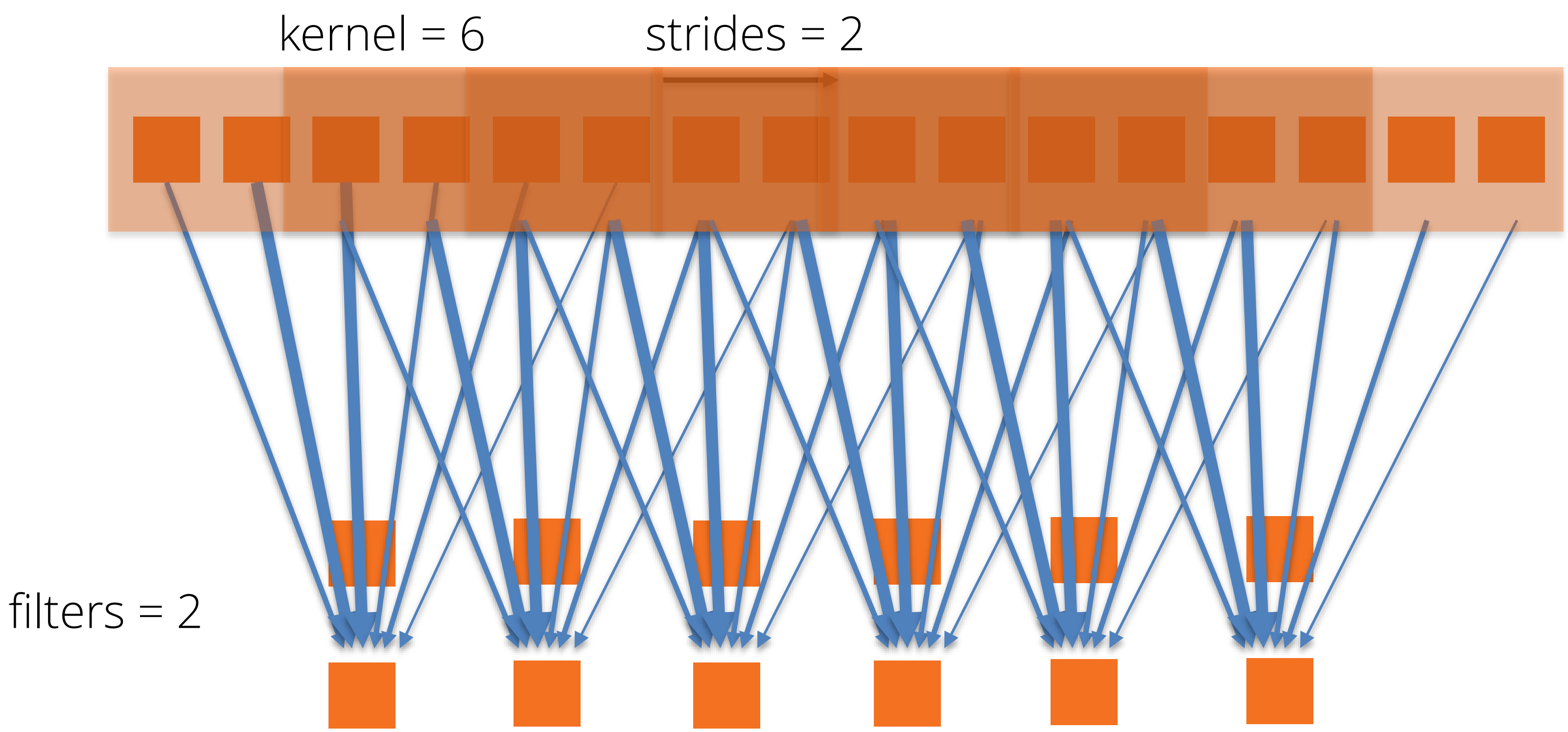
```
xWalk <- readRDS("/data/xWalk.rds")

yWalk <- readRDS("/data/yWalk.rds")
```

Convolution Layer

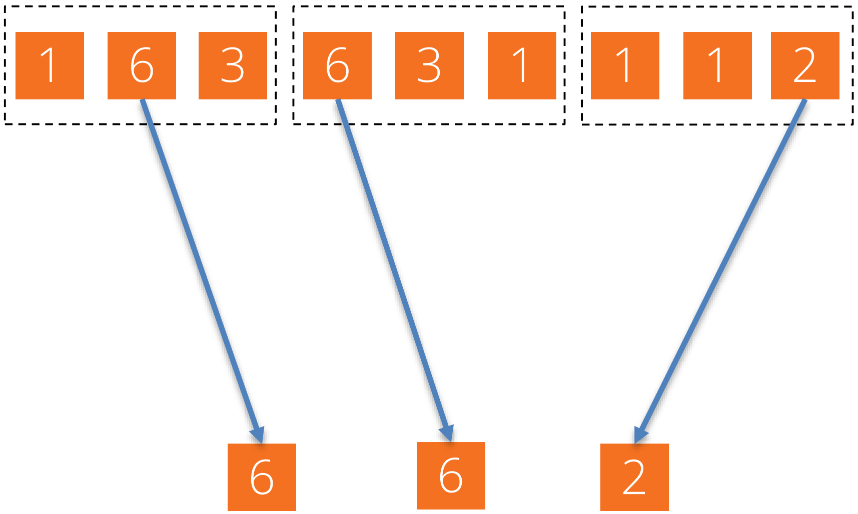


Convolution Layer - Filters

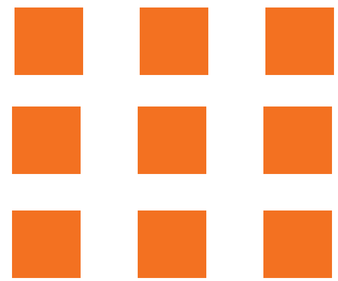


Max Pooling

pool_size = 3



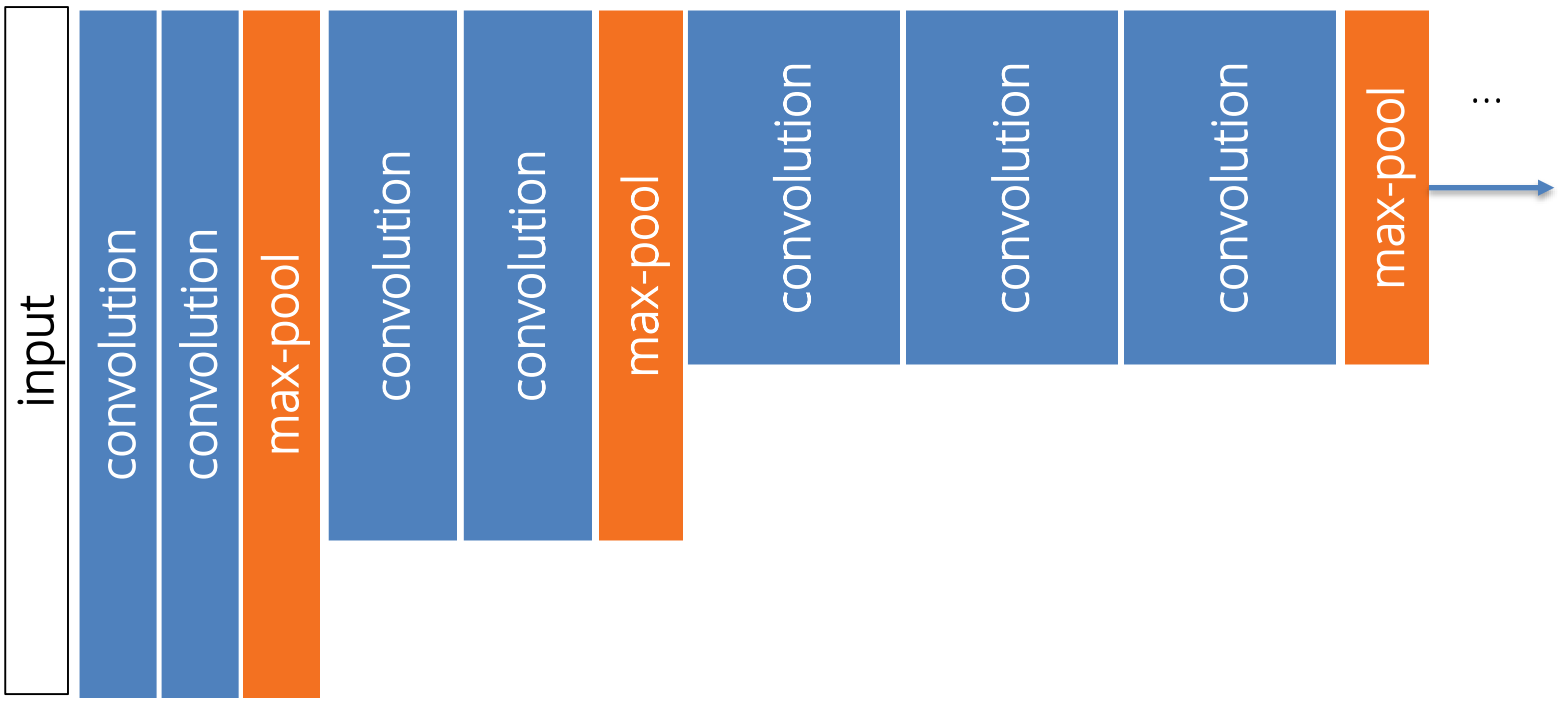
Flattening



Exercise

- Reproduce the above model and compile it.
- Train the model with fit and assess performance on the validation set over 15 epochs.
- How does this compare to only using dense layers (you'll still need to flatten)?

CNN Architectures - VGG



Exercise

- How do further epochs affect performance?
- Try changing the kernel size and number of filters. How does this affect your results?
- Try adding more dense layers. How does this affect training time and model performance?
- Try adding a dropout layer.

What Next?

- Pre-trained Networks
- CloudML