Lecture 30:
Intro to Deep Learning (Cont.)

Big Data and Machine Learning for Applied Economics
Econ 4676

Ignacio Sarmiento-Barbieri

Universidad de los Andes

November 25, 2021

Announcements

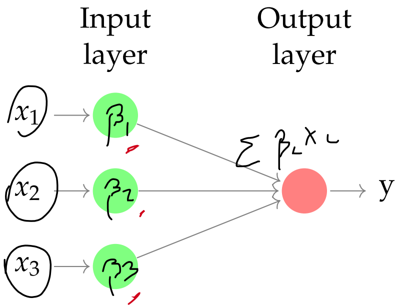
- PS4 grades are up (great job!!)
- Friday PS5 presentations
- Today you need to submit a .csv by 8:00 pm.
  - It should be in the `stores` folder
  - Name it as `predictions-problem_set_5_sarmiento-cano.csv`
  - If I can't grab you predictions file from your repo with `grep` you won't get credit.
- Friday upload PS 6: *"To finish your journey, formulate a theoretical question based on the topics learned so far. It can be anything you want as long as it relates to the course. Your grade will depend on the "ingenuity" of the question and the "correctness" of the answer."*

Agenda

Deep Learning: Recap

- Let's start with a familiar and simple model, the linear model

$$
\begin{aligned}
y &= f(X) + u \\
y &= \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + u
\end{aligned}
\tag{1}
$$



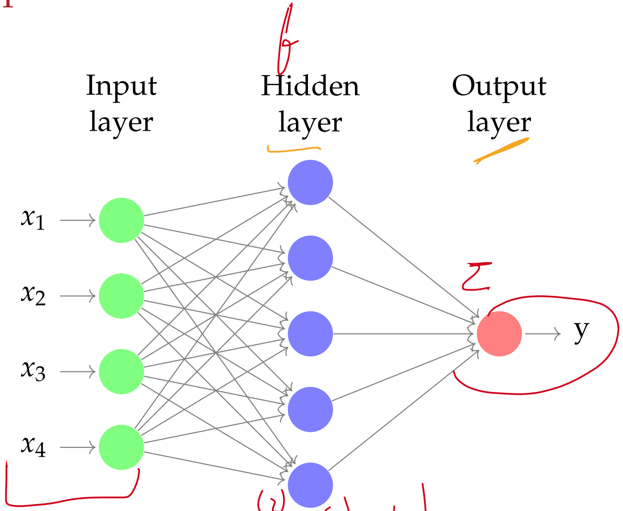
Multilayer Perceptrons

- Linear Models may be to simple, and miss the nonlinearities that best approximate $f^*(x)$
- We can overcome these limitations of linear models and handle a more general class of functions by incorporating one or more hidden layers.

$$f(x) = f^{(2)}(f^{(1)}(x)) \tag{2}$$

- These chain structures are the most commonly used structures of neural networks.

Multilayer Perceptrons



$$y = f^{(2)}\left(f^{(1)}(x)\right)$$

Multilayer Perceptrons

- The overall length of the chain gives the depth of the model. The name "deep learning" arose from this terminology.
- The final layer of a feedforward network is called the output layer
- During neural network training, we try to train $f(x)$ to match $f^*(x)$
- In the training data we observe the first layer, inputs (x), and the last layer, output (y)
- We do not observe the intermediate layers,they are then called hidden layers.
- Finally, these networks are called neural because they are loosely inspired by neuroscience.

Worked Example: The ”Exclusive OR (XOR)” Function

- The exclusive disjunction of a pair of propositions, (p, q), is supposed to mean that p is true or q is true, but not both
- It’s truth table is:

q	p	q v p
0	0	0
0	1	1
1	0	1
1	1	0

- When exactly one of these binary values is equal to 1, the XOR function returns 1. Otherwise, it returns 0

Worked Example: The "Exclusive OR (XOR)" Function

- Let's use a linear model

$$y = X\beta + \iota\alpha \tag{3}$$

$$y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \iota = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{4}$$

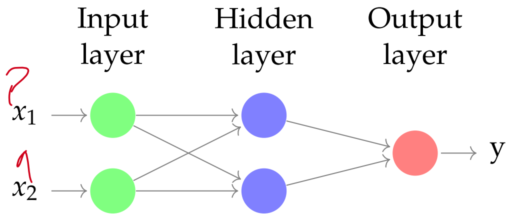
- Solution $\alpha = \frac{1}{2}$ $\beta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

- Prediction $\hat{y} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$

$$\hat{y} = \underbrace{X\hat{\beta}}_{?} + \frac{1}{2}$$

Worked Example: The "Exclusive OR (XOR)" Function

- Let's use multilayer perceptrons (feedforward network) with one hidden layer containing two hidden units



- This network has a vector of hidden units h that are computed by a function $f^{(1)}(x; W, c)$.
- The second layer is the output layer of the network. $f(x; W, c, w, b) = f^{(2)}(f^{(1)}(x))$

Worked Example: The "Exclusive OR (XOR)" Function

- Which $f^{(1)}$ should we specify?
  - Clearly **not** linear, otherwise it would defeat the entire purpose
  - We are going to use the rectified linear unit or ReLU (it is usually the default recommendation, there are many others (more on this later))
  - ReLU is defined as $g(z) = max\{0, z\}$
- For $f^{(2)}$? For this example, a linear model will suffice

$$f^{(2)} = f^{(1)} w + b \tag{5}$$

- The final model is then

$$f(x, W, C, w, b) = max\{0, XW + c\} w + b \tag{6}$$

Worked Example: The ”Exclusive OR (XOR)” Function

- Suppose this is the solution to the XOR problem

$$f(x) = max\{0, XW + c\}\, w + b \quad ✓$$

$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$c = \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}$$

$$w = \begin{pmatrix} 1 & -2 \end{pmatrix}$$

$$b = 0$$

Worked Example: The "Exclusive OR (XOR)" Function

- Lets work out the example step by step

$$f(x) = max\{0, XW + c\}\, w + b \tag{7}$$

$$XW = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}$$

$$XW + c = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

$$max\{0, XW + c\} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

Worked Example: The ”Exclusive OR (XOR)” Function

$$\hat{y} = max\{0, XW + c\}\, w + b = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

- The neural network has obtained the correct answer for every data point

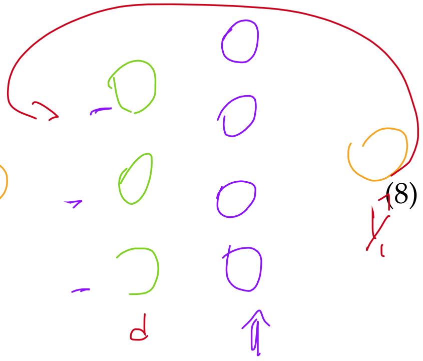
Worked Example: The ”Exclusive OR (XOR)” Function

- In this example, we simply specified the solution, then showed that it obtained zero error.
- In a real situation, obviously we can’t guess the solution
- What we do is gradient based optimization
- Remember that the convergence point of gradient descent depends on the initial values of the parameters and step size.
- In practice, gradient descent would usually not find clean, easily understood, integer-valued solutions like we did here.

Multilayer Perceptrons: Theory

- Why not a linear model?

$$\begin{aligned} \mathbf{H} &= \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)} \\ \mathbf{Y} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \end{aligned} \tag{8}$$



- where $\mathbf{X} \in \mathbb{R}^{n \times d}$, n obs. and d inputs (features).
- H is a hidden layer with h hidden units, $\mathbf{H} \in \mathbb{R}^{n \times h}$
- Because the hidden and output layers are both fully connected, we have
  - hidden-layer weights $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ and biases $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$
  - output-layer weights $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ and biases $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$

h

$y - X\hat{\beta}$

ε

Multilayer Perceptrons: Theory

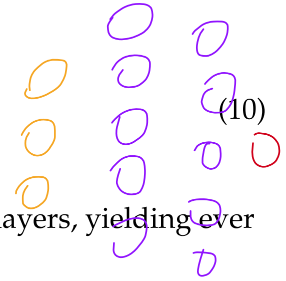
- Why not a linear model?
- Note that after adding the linear hidden layer , we gain nothing for our troubles!

$$\begin{aligned}\mathbf{Y} &= (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\ &= \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \underbrace{\mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}} \\ &= \boxed{\mathbf{X}\mathbf{W} + \mathbf{b}.}\end{aligned} \tag{9}$$

Multilayer Perceptrons: Theory

- The gain comes from using nonlinear activation function f
- Note that, with activation functions in place, it is no longer possible to collapse our MLP into a linear model:

$$\begin{aligned}\mathbf{H} &= f(\overset{\varepsilon}{\downarrow}\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),\\ \mathbf{Y} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.\end{aligned} \tag{10}$$



- Note that we can build more general MLPs, by stacking hidden layers, yielding ever more expressive models.

$$\begin{aligned}\mathbf{H}^{(1)} &= f_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\\ \mathbf{H}^{(2)} &= f_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})\\ \mathbf{Y} &= \mathbf{H}^2\mathbf{W}^{(3)} + \mathbf{b}^{(3)}.\end{aligned} \tag{11}$$

Activation Functions

ReLU Function

- Activation functions are fundamental to deep learning, let us briefly survey some common activation functions.
- ReLU Function
  - The most popular choice, due to both simplicity of implementation and its good performance on a variety of predictive tasks, is the rectified linear unit (ReLU).
  - ReLU provides a very simple nonlinear transformation. Given an element x, the function is defined as the maximum of that element and 0:

$$\mathrm{ReLU}(x) = \max\{x, 0\}.$$

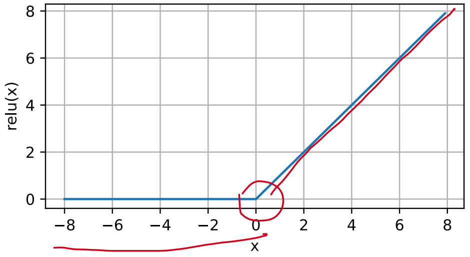
$$\frac{\partial \, \mathrm{ReLU}}{\partial w}$$

$$\underline{\max}\{X\hat{w}_h + b, 0\}$$

$$w \qquad b$$

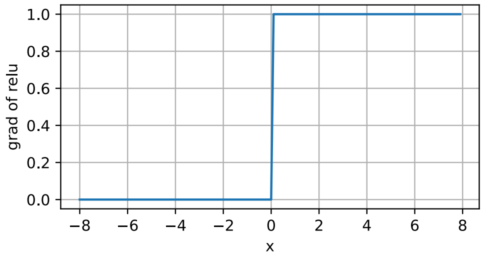
Activation Functions

- ReLU function retains only positive elements and discards all negative elements by setting them to 0.
- It is piecewise linear.



Activation Functions

- Part of the appeal of ReLU has to do with it's well behaved derivative
  - Note that the ReLU function is not differentiable when the input takes value precisely equal to 0. In these cases, we default to the left-hand-side derivative and say that the derivative is 0 when the input is 0. (we may even get away with this because the input may never actually be zero!)
  - This makes optimization better behaved and it mitigates the problem of vanishing gradients



Activation Functions

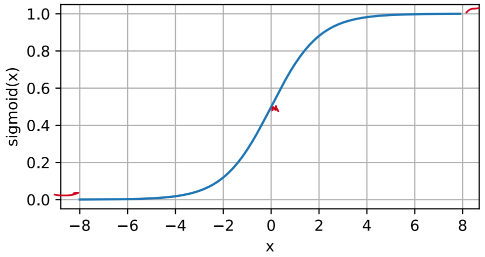
Sigmoid Function (Logit)

- The sigmoid function transforms its inputs, for which values lie in the domain $\mathbb{R}$, to outputs that lie on the interval (0, 1).
- For that reason, the sigmoid is often called a squashing function: it squashes any input in the range (-inf, inf) to some value in the range (0, 1):

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

Activation Functions

Sigmoid Function (Logit)



- In the earliest neural networks, scientists were interested in modeling biological neurons which either fire or do not fire. Thus the pioneers of this field, going all the way back to McCulloch and Pitts, the inventors of the artificial neuron, focused on thresholding units.
- A thresholding activation takes value 0 when its input is below some threshold and value 1 when the input exceeds the threshold.
- When attention shifted to gradient based learning, the sigmoid function was a natural choice because it is a smooth, differentiable approximation to a thresholding unit.

Activation Functions

Tanh Function

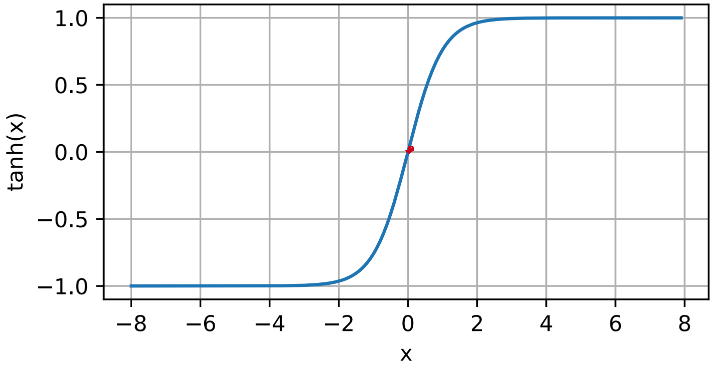
- Like the sigmoid function, the tanh (hyperbolic tangent) function also squashes its inputs, transforming them into elements on the interval between -1 and 1:

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

- Although the shape of the function is similar to that of the sigmoid function, the *tanh* function exhibits point symmetry about the origin of the coordinate system.

Activation Functions

Tanh Function



Activation Functions

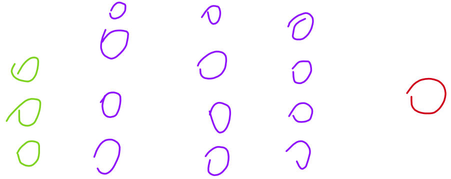
- Other Activation functions
  - $h = cos(Wx + b)$ Goodfellow et all (2016) claim that on the MNIST dataset they obtained an error rate of less than 1 percent
  - Radial basis function (RBF): $exp\left(\frac{1}{\sigma^2)}||W - x||^2\right)$
  - Softplus: $log(1 + e^x)$
  - Hard tanh: $max(-1, min(1, x))$
- Hidden unit design remains an active area of research, and many useful hidden unit types remain to be discovered

Output Functions

- The choice of cost function is tightly coupled with the choice of output unit.
- Most of the time, we simply use the distance between the data distribution and the model distribution.
  - Linear $y = W'h + b \rightarrow \mathbb{R}$
  - Sigmoid (Logistic) $\frac{1}{1+\exp(-x)} \rightarrow$ classification $\{0, 1\}$
  - Softmax $\frac{exp(x)}{\sum exp(x))} \rightarrow$ classification multiple categories

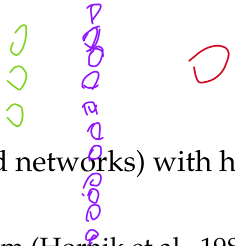
(multinomial logit)

Architecture Design



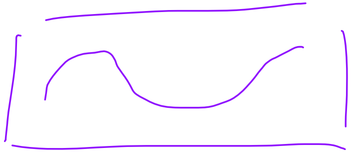
- Another key design consideration for neural networks is determining the architecture.
- The word architecture refers to the overall structure of the network: how many units it should have and how these units should be connected to each other.
- In these chain-based architectures, the main architectural considerations are choosing the depth of the network and the width of each layer.

Architecture Design



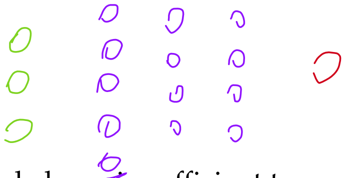
- A multilayer perceptron (feedforward networks) with hidden layers provide a universal approximation framework.
  - The universal approximation theorem (Hornik et al., 1989; Cybenko, 1989) states that a feedforward network with a linear output layer and at least one hidden layer with any “squashing” activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden unit.
  - The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well (Hornik et al., 1990).

Architecture Design



- The universal approximation theorem means that regardless of what function we are trying to learn, we know that a large MLP will be able to represent this function.
- We are not guaranteed, however, that the training algorithm will be able to learn that function. Even if the MLP is able to represent the function, learning can fail for two different reasons.
  1. The optimization algorithm used for training may not be able to find the value of the parameters that corresponds to the desired function.
  2. The training algorithm might choose the wrong function as a result of overfitting

Architecture Design



$H = X\underline{W} + \beta$

- A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasible large and may fail to learn and generalize correctly.
- In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error.
- The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error

$lm(y \sim .\hat{}\,\underline{3})$

Numerical Stability and Initialization

- Vanishing and exploding gradients are common issues in deep networks. Great care in parameter initialization is required to ensure that gradients and parameters remain well controlled.
- Initialization heuristics are needed to ensure that the initial gradients are neither too large nor too small.
- ReLU activation functions mitigate the vanishing gradient problem. This can accelerate convergence.
- Random initialization is key to ensure that symmetry is broken before optimization

- random state (Python)
- set seed (R)
- seed (~~S~~tata)

Deep Learning: Demo

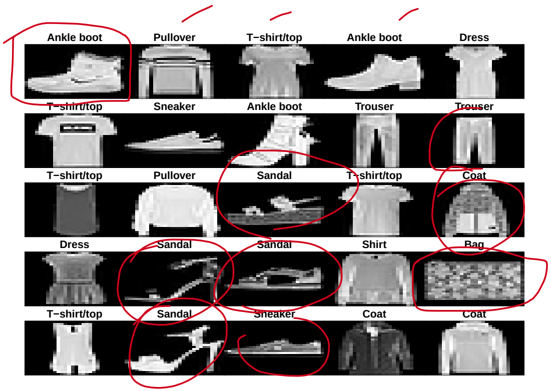
```
library(keras)
fashion_mnist <- dataset_fashion_mnist()
```

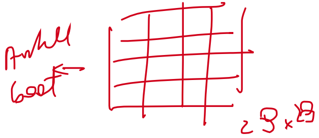
Deep Learning: Demo

```r
c(train_images, train_labels) %<-% fashion_mnist$train
c(test_images, test_labels) %<-% fashion_mnist$test

class_names = c('T-shirt/top',
                'Trouser',
                'Pullover',
                'Dress',
                'Coat',
                'Sandal',
                'Shirt',
                'Sneaker',
                'Bag',
                'Ankle boot')
```

Deep Learning: Demo

```r
train_images <- train_images / 255
test_images <- test_images / 255
```

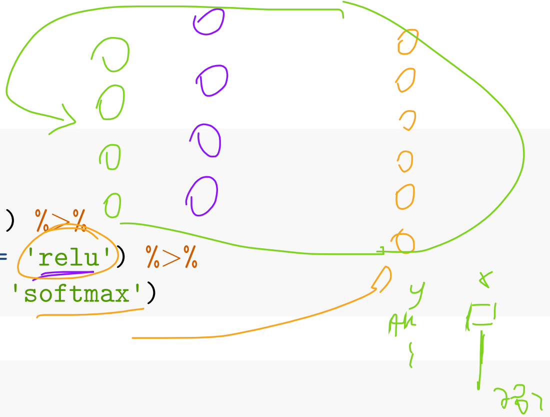




Deep Learning: Demo

```r
model <- keras_model_sequential()
model %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

```r
model %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
model %>% fit(train_images, train_labels, epochs = 5, verbose = 2)
```



Deep Learning: Demo

w^*

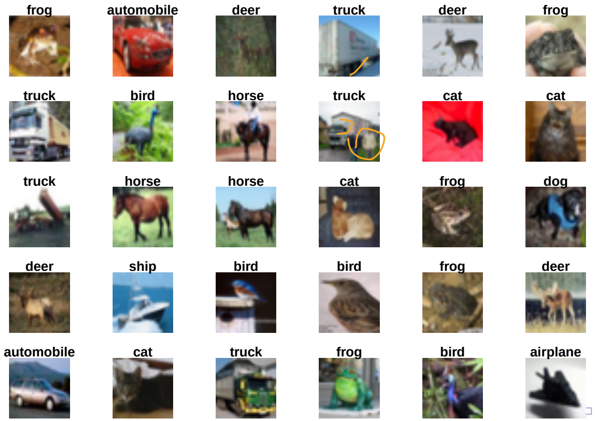
```
## Epoch 1/5
## 1875/1875 - 2s - loss: 0.5003 - accuracy: 0.8238
## Epoch 2/5
## 1875/1875 - 2s - loss: 0.3782 - accuracy: 0.8643
## Epoch 3/5
## 1875/1875 - 2s - loss: 0.3362 - accuracy: 0.8784
## Epoch 4/5
## 1875/1875 - 2s - loss: 0.3141 - accuracy: 0.8844
## Epoch 5/5
## 1875/1875 - 2s - loss: 0.2934 - accuracy: 0.8922
```

```r
score <- model %>% evaluate(test_images, test_labels, verbose = 0)

cat('Test loss:', score[1], "\n")
```

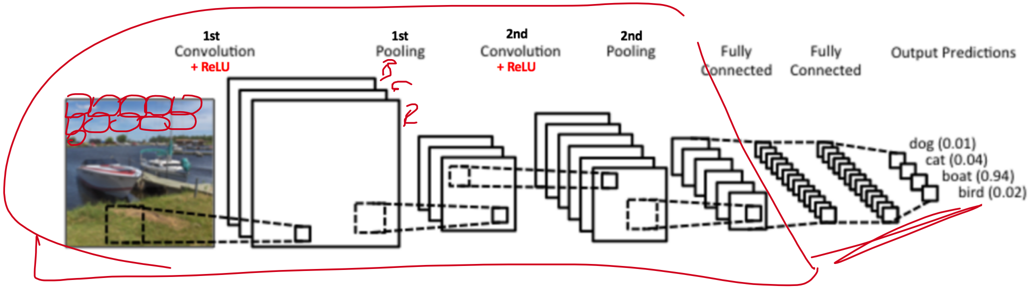
```
## Test loss: 0.3377942

## Test accuracy: 0.8792
```

Deep Learning: CNN demo and teaser

```
#packages
library(tensorflow)
library(keras)
#load the data
cifar <- dataset_cifar10()
```



Deep Learning: CNN demo and teaser



```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(32,32,3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu")

summary(model)
```

Deep Learning: CNN demo and teaser

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                    Param #
## ================================================================================
## conv2d_2 (Conv2D)                  (None, 30, 30, 32)              896
## ________________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D)     (None, 15, 15, 32)              0
## ________________________________________________________________________________
## conv2d_1 (Conv2D)                  (None, 13, 13, 64)              18496
## ________________________________________________________________________________
## max_pooling2d (MaxPooling2D)       (None, 6, 6, 64)                0
## ________________________________________________________________________________
## conv2d (Conv2D)                    (None, 4, 4, 64)                36928
## ================================================================================
## Total params: 56,320
## Trainable params: 56,320
## Non-trainable params: 0
## ________________________________________________________________________________
```

Deep Learning: CNN demo and teaser

More layers

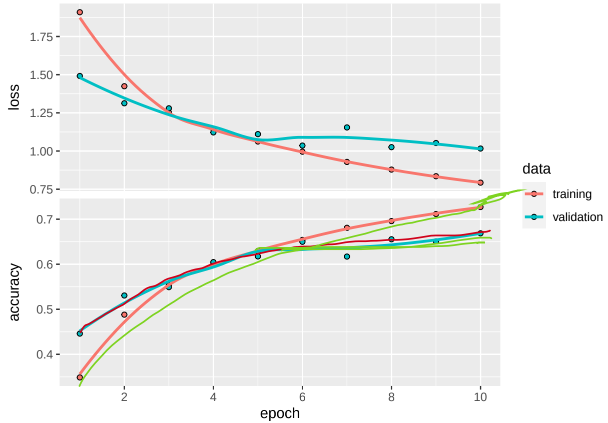
```
model %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
summary(model)
```

```
## Model: "sequential"
## ____________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ====================================================================================
## conv2d_2 (Conv2D)                   (None, 30, 30, 32)              896
## ____________________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D)      (None, 15, 15, 32)              0
## ____________________________________________________________________________________
## conv2d_1 (Conv2D)                   (None, 13, 13, 64)              18496
## ____________________________________________________________________________________
## max_pooling2d (MaxPooling2D)        (None, 6, 6, 64)                0
## ____________________________________________________________________________________
## conv2d (Conv2D)                     (None, 4, 4, 64)                36928
## ____________________________________________________________________________________
## flatten (Flatten)                   (None, 1024)                    0
## ____________________________________________________________________________________
## dense_1 (Dense)                     (None, 64)                      65600
## ____________________________________________________________________________________
## dense (Dense)                       (None, 10)                      650
## ====================================================================================
## Total params: 122,570
```

Deep Learning: CNN demo and teaser

```r
model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history <- model %>%
  fit(
    x = cifar$train$x, y = cifar$train$y,
    epochs = 10,
    validation_data = unname(cifar$test),
    verbose = 2
  )
```

Deep Learning: CNN demo and teaser



```r
evaluate(model, cifar$test$x, cifar$test$y, verbose = 0)
```

```
##     loss accuracy
## 1.016327 0.668600
```

Review & Next Steps

- Deep Learning: Intro
- Please fill the perception survey `https://encuestacursosuniandes.com/`
- Next class: PS5 presentations and turn in PS6
- Questions? Questions about software?

Further Readings

- Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola (2020) Dive into Deep Learning. Release 0.15.1. http://d2l.ai/index.html
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press. http://www.deeplearningbook.org
- Rstudio (2020). Tutorial TensorFlow https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_classification/
- Taddy, M. (2019). Business data science: Combining machine learning and economics to optimize, automate, and accelerate business decisions. McGraw Hill Professional.