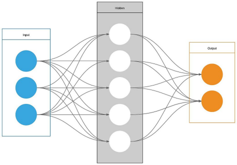
Data Analysis, Neural Networks and the use of Keras

Cecilia Jarne

cecilia.jarne@unq.edu.ar

Organization For Computational Neurosciences

WTPC Workshop en técnicas de programación científica



Twitter: @ceciliajarne

CONICET

Universidad Nacional de Quilmes

What is this talk about?

- What is Machine Learning, Neural Networks, and Deep Learning?
- Neural Networks concepts and topologies.
- A brief review of the use of python ML libraries.

We are interested in:

- Data Visualization.
- Data Analysis (Identify features from the data).
- Data Classification.
- Implementation of different algorithms as intelligent as we can get.

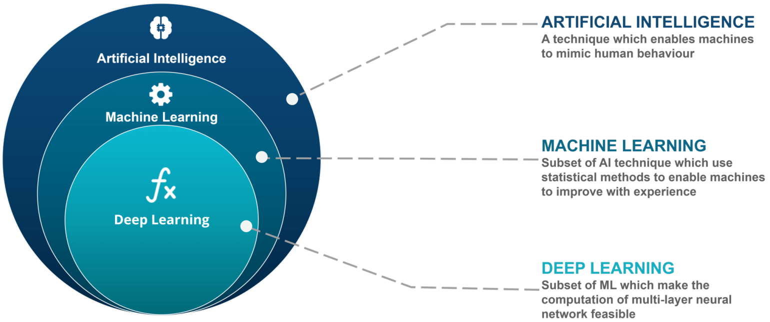
Which are the available algorithms?

TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	**X** Sometimes too simple to capture complex relationships between variables. **X** Tendency for the model to "overfit".
Linear	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	**X** Sometimes too simple to capture complex relationships between variables. **X** Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	**X** Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
Tree-based	Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	**X** Can be slow to output predictions relative to other algorithms. **X** Not easy to understand predictions.
Tree-based	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	**X** A small change in the feature set or training set can create radical changes in the model. **X** Not easy to understand predictions.
Neural networks	Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several **layers of neural networks put one after the other.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	**X** Very, very slow to train, because they have so many layers. Require a lot of power. **X** Almost impossible to understand predictions.

Which are the available algorithms?

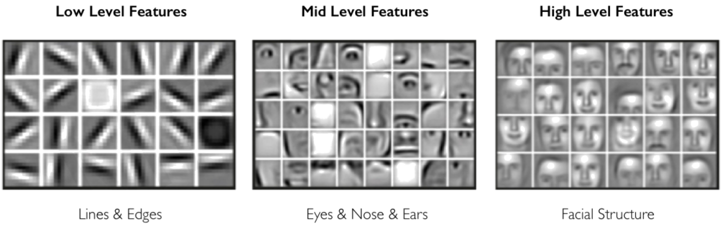
TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Linear	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	X Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
Tree-based	Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	X Can be slow to output predictions relative to other algorithms. X Not easy to understand predictions.
Tree-based	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	X A small change in the feature set or training set can create radical changes in the model. X Not easy to understand predictions.
Neural networks	Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several **layers of neural networks put one after the other.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	X Very, very slow to train, because they have so many layers. Require a lot of power X Almost impossible to understand predictions.

What about machine learning?



Why Deep Learning?

- Data driven approach.
- Can we learn the underlying features directly from data?



Neural Networks date back decades, so why the resurgence?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

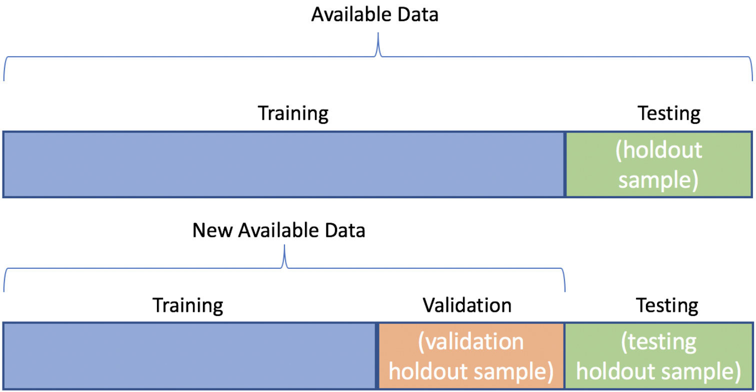
WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

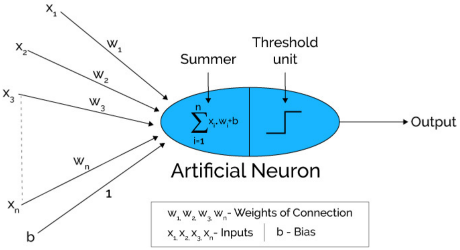
How we split our data to train a model?

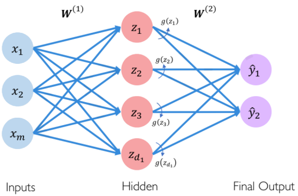


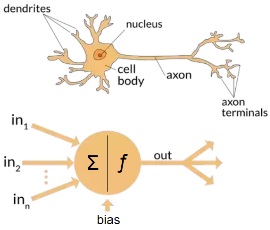
How we use (split) our data to train a model?

- Training set: The data where the model is trained on. We train our model, by pairing the input with the expected output.
- Validation set: Data the model has not been trained on and is used to tune hyperparameters. Here we estimate how well your model has been trained.
- Test set: Same like the validation set.. just used at the final end. This is an Application phase: we apply our developed model to the real-world data and get the results. This fase is split into two parts:
    - First you look at your models and select the best performing approach using the validation data (=validation).
    - Then you estimate the accuracy of the selected approach (=test).

Neural Networks:



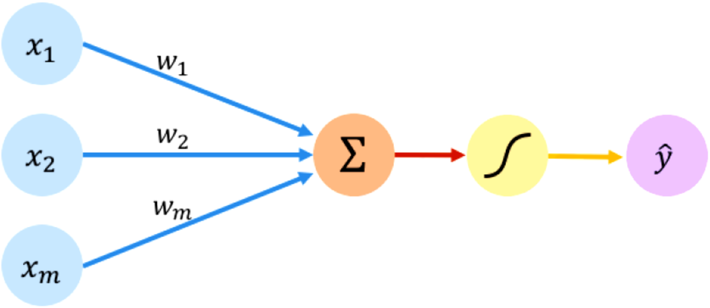




The Perceptron: Forward Propagation

The structural building block of Deep Learning

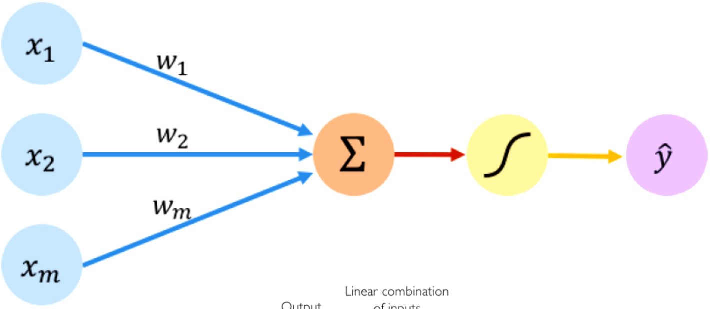
Input $\rightarrow$ Weights $\rightarrow$ Sum $\rightarrow$ Non linearity $\rightarrow$ Output



The Perceptron: Forward Propagation

The structural building block of Deep Learning:

Input → Weights → Sum → Non linearity → Output



Output

Linear combination of inputs

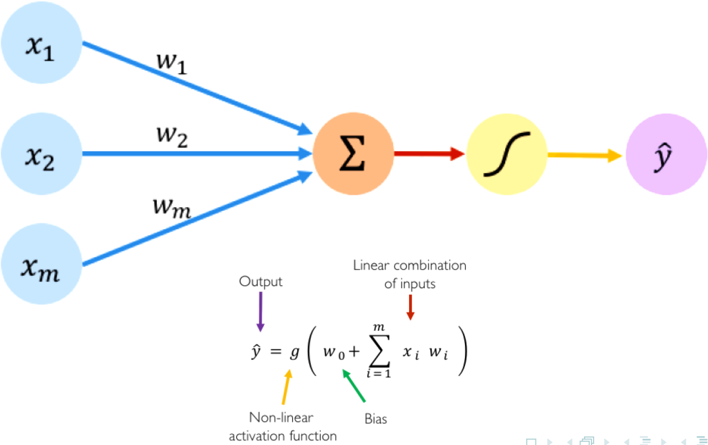
$$\hat{y} = g\left(\sum_{i=1}^{m} x_i \, w_i\right)$$

Non-linear activation function

The Perceptron: Forward Propagation

The structural building block of Deep Learning:

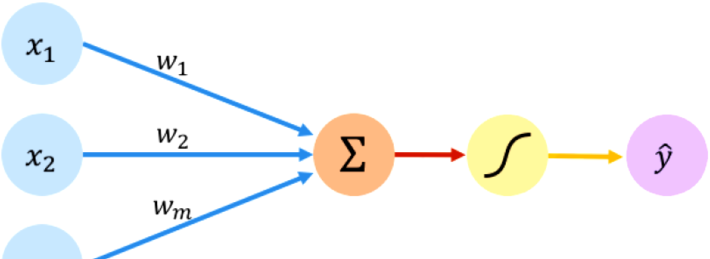
Input $\rightarrow$ Weights $\rightarrow$ Sum $\rightarrow$ Non linearity $\rightarrow$ Output



The Perceptron: Forward Propagation

The structural building block of Deep Learning:

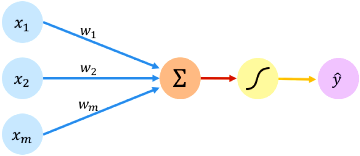
Input $\rightarrow$ Weights $\rightarrow$ Sum $\rightarrow$ Non linearity $\rightarrow$ Output



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

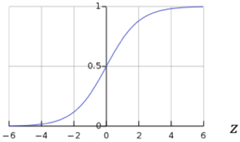


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

- Example: sigmoid function

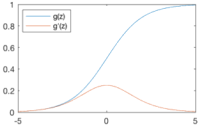
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Neural Networks: Activation Functions

The purpose of Activation functions is to introduce non linearities into the network

Sigmoid Function

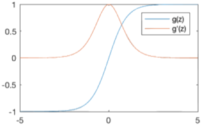


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

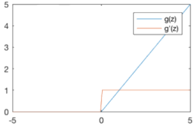


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)

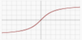
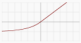
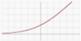


$$g(z) = \max(0, z)$$

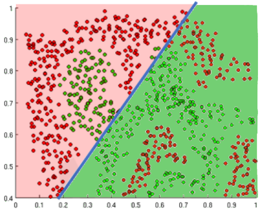
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

```
tf.nn.relu(z)
```

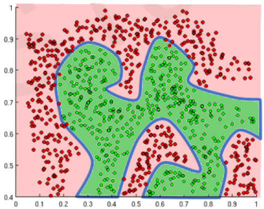
Neural Networks: Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Neural Networks: Non linear decision

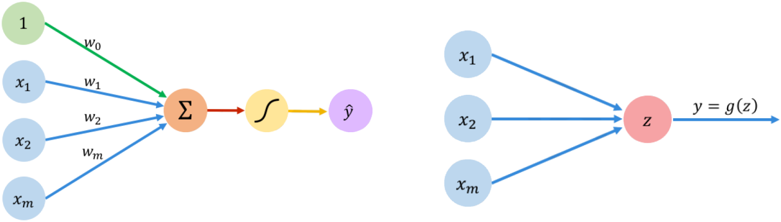


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

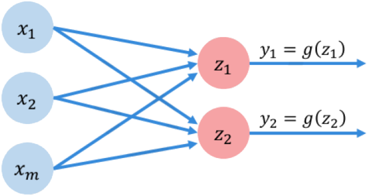
A simplified perceptron:



$$z = w_0 + \sum_{j=1}^{m} x_j\, w_j$$

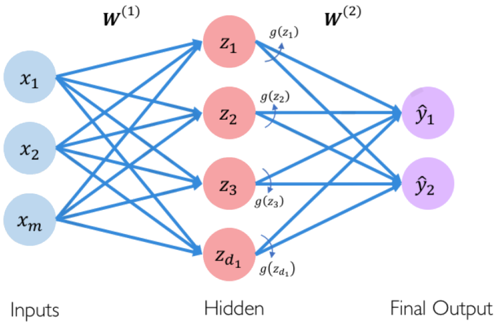
Building Neural Networks: Multi Outputs

Because all inputs are connected to all outputs these are called Dense layers:



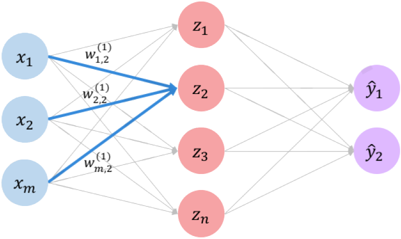
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Single Layer Neural Network



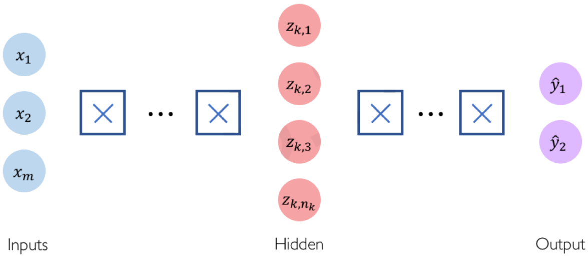
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)}\right)$$

Single Layer Neural Network



$$
\begin{aligned}
z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)} \\
&= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}
\end{aligned}
$$

A deep neural network structure



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

(... or Cost Function or Objective Function)

Depends on the kind of problem.

- Regression $\rightarrow$ Mean square error
- Classification $\rightarrow$ Cross entropy, binary cross entropy.

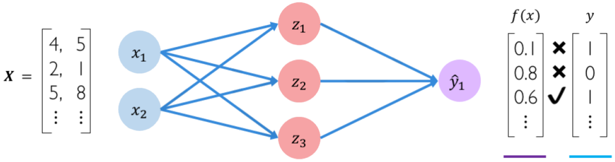
Neural Networks: Loss Functions

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\lVert \mathbf{y} - \mathbf{o} \rVert_1$
$\mathcal{L}_2$	L_2 loss	$\lVert \mathbf{y} - \mathbf{o} \rVert_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\lVert \mathbf{y} - \sigma(\mathbf{o}) \rVert_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\lVert \mathbf{y} - \sigma(\mathbf{o}) \rVert_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert \sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)} \rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)} \mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)} \mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)} \mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)} \mathbf{y}^{(j)}}{\lVert \sigma(\mathbf{o}) \rVert_2^2 + \lVert \mathbf{y} \rVert_2^2 - \sum_j \sigma(\mathbf{o})^{(j)} \mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)} \mathbf{y}^{(j)}}{\lVert \sigma(\mathbf{o}) \rVert_2 \lVert \mathbf{y} \rVert_2}$

See: `https://arxiv.org/pdf/1702.05659.pdfs`

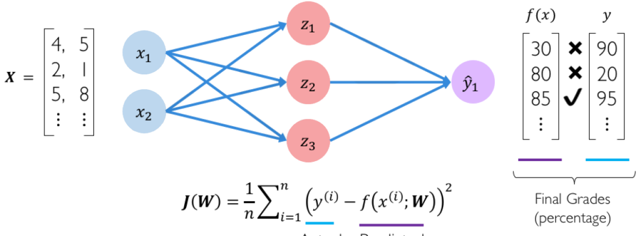
Neural Networks: Crossentropy

***Cross entropy loss** can be used with models that output a probability between 0 and 1*



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + \underbrace{(1 - y^{(i)})}_{\text{Actual}} \log\Big(1 - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

Mean squared error loss *can be used with regression models that output continuous real numbers*



$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)}; W\big)}_{\text{Predicted}}\Big)^2$$

Training Neural Networks: Loss Optimization

We want to find the network weights that achieve the lowest loss

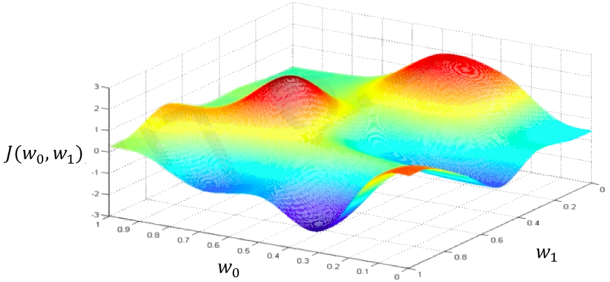
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Training Neural Networks: Loss Optimization

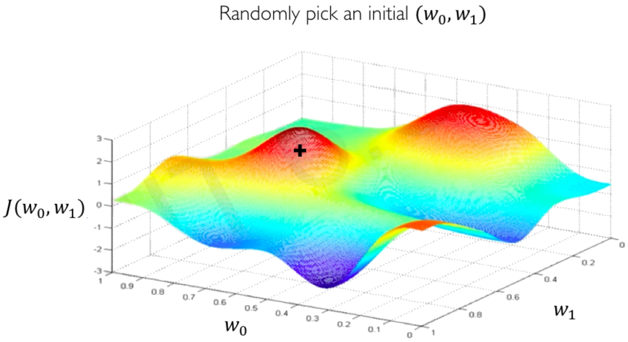
We want to find the network weights that achieve the lowest loss

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$



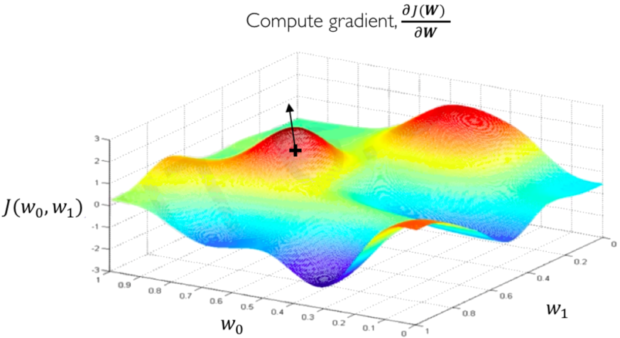
Training Neural Networks: Loss Optimization

We want to find the network weights that achieve the lowest loss

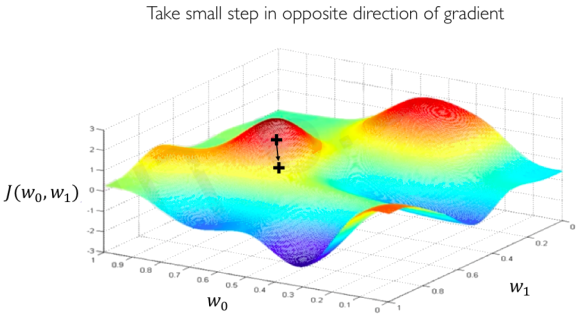


Training Neural Networks: Loss Optimization

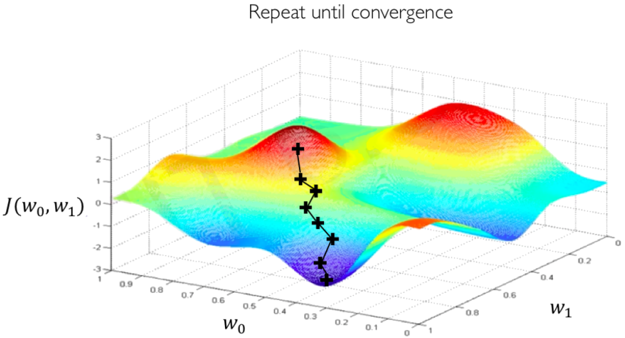
We want to find the network weights that achieve the lowest loss



We want to find the network weights that achieve the lowest loss



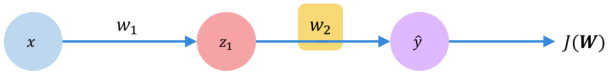
We want to find the network weights that achieve the lowest loss



Algorithm

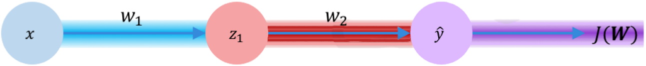
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

How a small change in one weight affect the loss



Training Neural Networks:

We apply chain rule



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

*Repeat this for **every weight in the network** using gradients from later layers*

Training Neural Networks:Update gradient

Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta}\,\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

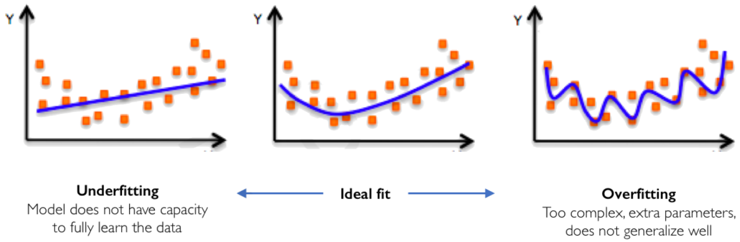
How can we set the
learning rate?

Training Neural Networks:Algorithms

Gradient Descent Algorithms

Algorithm	TF Implementation	Reference
SGD	`tf.keras.optimizers.SGD`	Kiefer & Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." 1952.
Adam	`tf.keras.optimizers.Adam`	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
Adadelta	`tf.keras.optimizers.Adadelta`	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
Adagrad	`tf.keras.optimizers.Adagrad`	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
RMSProp	`tf.keras.optimizers.RMSProp`	

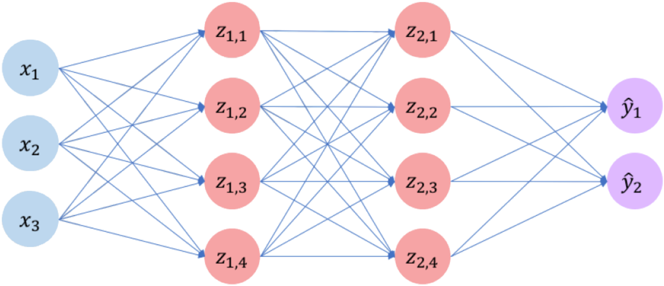
The problem of Overfitting



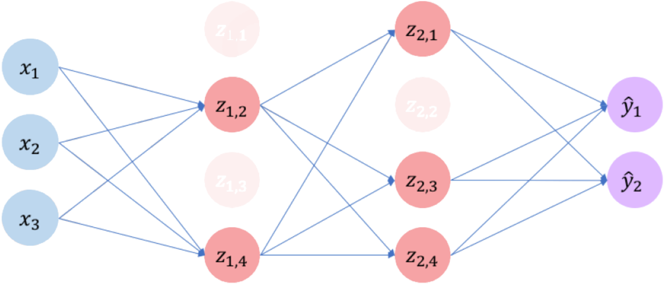
Regularization

- A technique that constraint the optimization problem to avoid complex models.
- We use it to improve the generalization on our model to unseen data.
- There are different kind of methods.

During training randomly set some activations to 0.



During training randomly set some activations to 0.



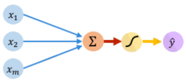
Regularization II: early stooping

To stop before having the opportunity to overfit by monitoring testing and training data.

Summary

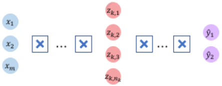
The Perceptron

- Structural building blocks
- Nonlinear activation functions



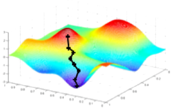
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



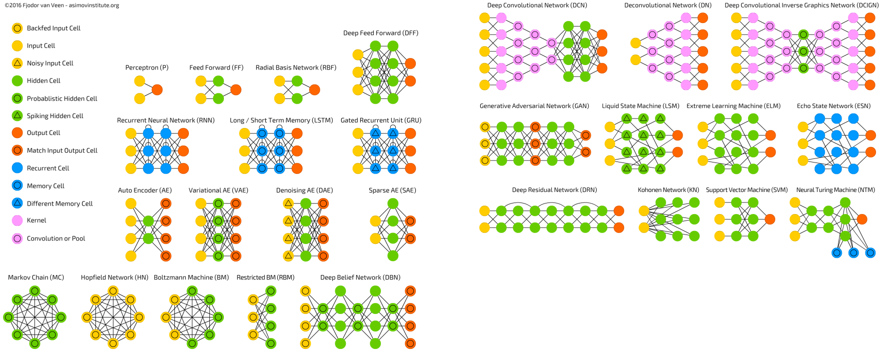
Training in Practice

- Adaptive learning
- Batching
- Regularization



Neural Networks: The Zoo of topologies

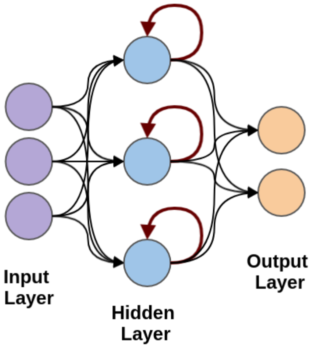
Neural Networks: The Zoo of topologies



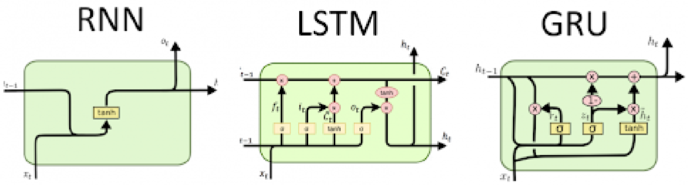
Topologies

- Feed Foward networks (alredy seen).
- Recurrent neural networks — simple, LSTM, and GRU.
- Convolutional neural networks

Topologies: Recurrent neural networks



Topologies: Recurrent neural networks



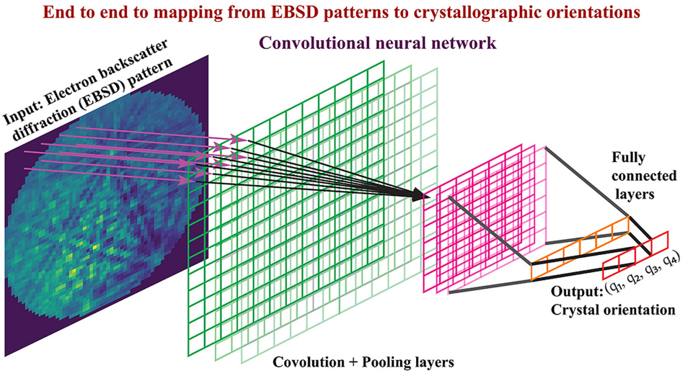
RNN — simple, LSTM, and GRU

- Recurrent neural networks are a class of neural networks that exploit the sequential nature of their input.
- Inputs could be: a text, a speech, time series, and anything else where the occurrence of an element in the sequence is dependent on the elements that appeared before it.

Convolutional and pooling layers

- ConvNets are a class of neural networks using convolutional and pooling operations for progressively learning rather sophisticated models based on progressive levels of abstraction.
- This learning via progressive abstraction resembles vision models that have evolved over millions of years inside the human brain.
- People called it deep with 3-5 layers a few years ago, and now it has gone up to 100-200.

Convolutional Neural Networks.



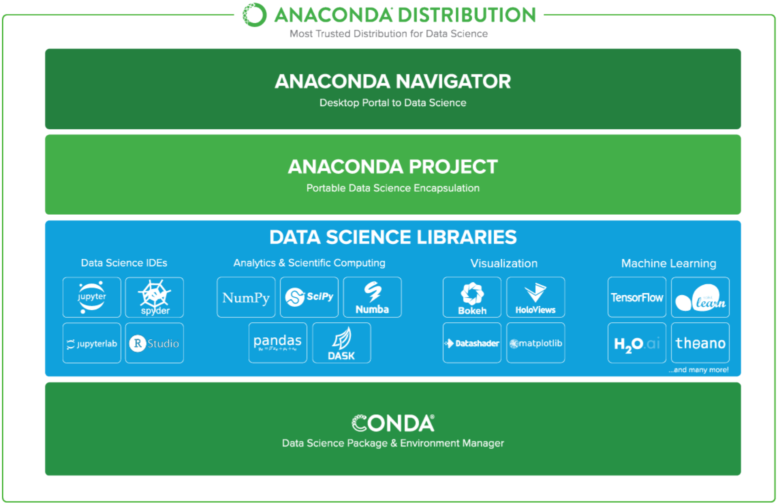
About environment and installation

python

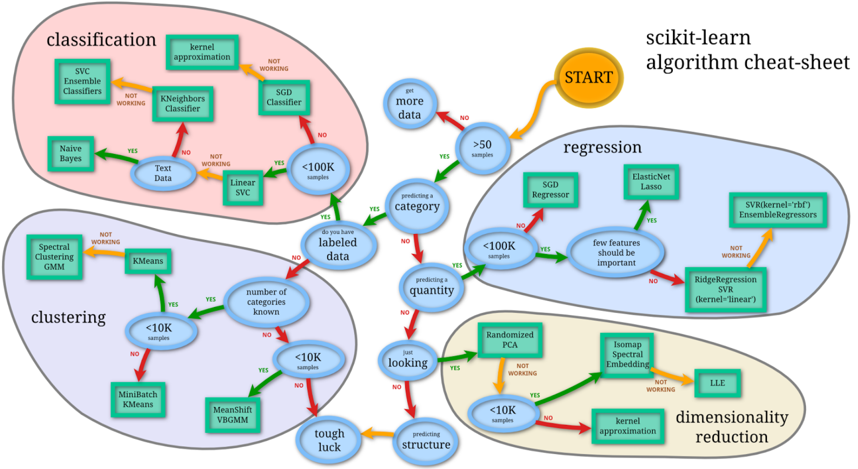
Where can we work?

- Locally in Virtual env:
  Main purpose of Python virtual env is to create an isolated environment for Python projects. Each project can have its own dependencies, regardless of what dependencies every other project has.
  https://realpython.com/python-virtual-environments-a-primer/
- Google Cloud ML.
  https://cloud.google.com/ai-platform/docs/getting-started-keras
- other Services

About installation Local: What is Anaconda for?



We can use Scikit learn also...



How do we implement this algorithms?

- From zero with math libraries and python.
- Using dedicated open source frameworks:
  - Tensorflow.
  - Keras.

Tensorflow:

Tensorflow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Keras Basics:

Keras: A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras is now embedded in Tensorfow

https://www.tensorflow.org/guide/keras/functional

Keras Basics:

- Modularity: A model is either a sequence or a graph of standalone modules that can be combined together like LEGO blocks for building neural networks
- The libraries predefines a large number of modules implementing different types of neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes.
- Minimalism: The library is implemented in Python and each module is kept short and self-describing.
- Easy extensibility: The library can be extended with new functionalities.

Keras Basics:

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science **Interactively** at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

DataCamp
Learn Python for Data Science **Interactively**

Keras Basics: code example

You can create a Sequential model by passing a list of layer instances to the constructor:

```

from keras.models import Sequential
from keras.layers import Dense,
     Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])

```

You can also simply add layers via the *.add*() method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

More:

https://github.com/katejarne/Keras_tensorflow_course

Classes and material will be at:

http://ceciliajarne.web.unq.edu.ar/cns-2020-tutorial/

References:

- Deep Learning (The MIT Press Essential Knowledge series)
- http://introtodeeplearning.com/ MIT course.
- https://www.tensorflow.org/
- https://keras.io/ Francois Chollet et al. Keras. 2015.
- Martín Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.