Introduction to Deep Learning with `TensoRflow` and `KeRas` libraries with some examples in biomedical research at the Hospital Clínic of Barcelona

Roger Borràs

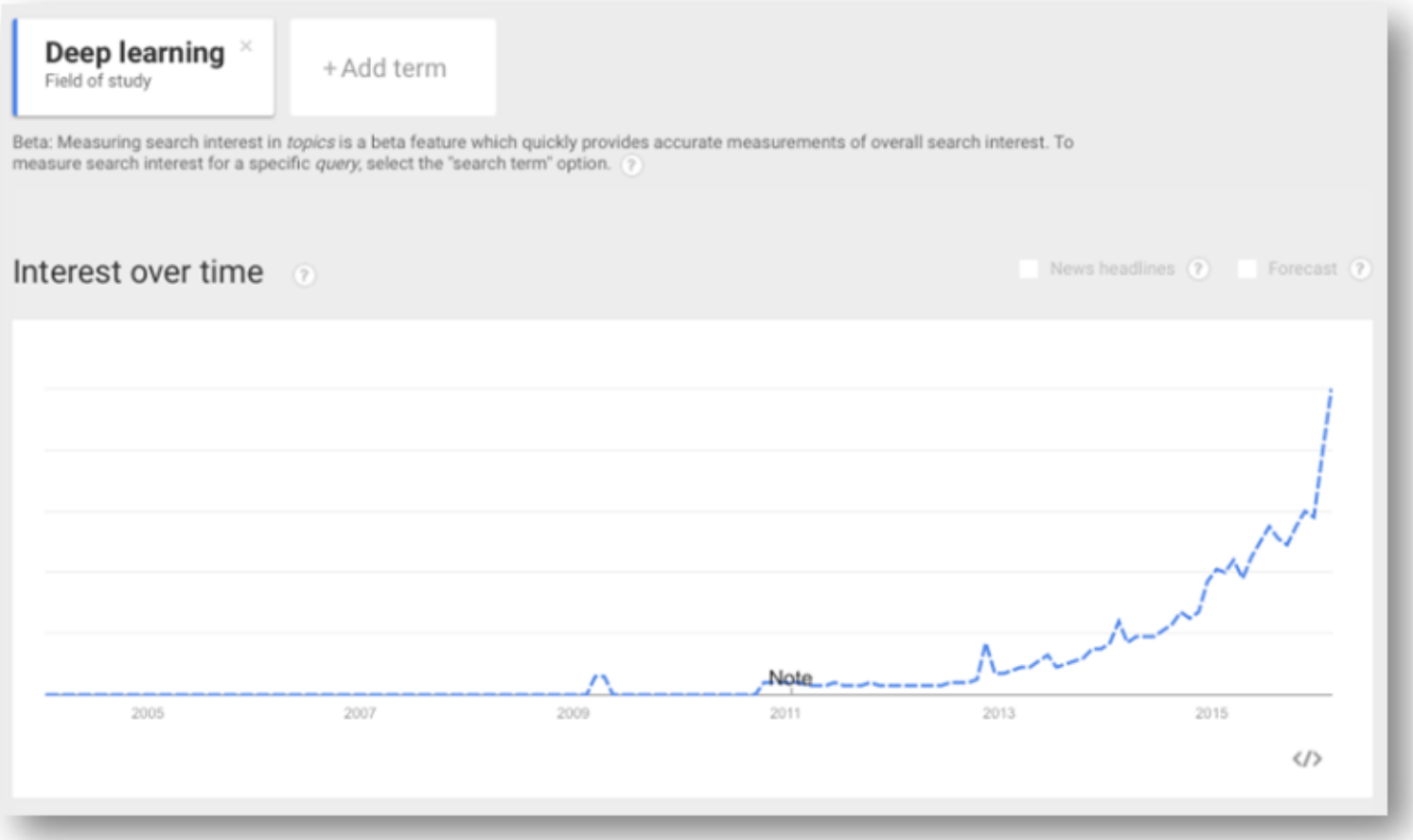
SEA

May 2018

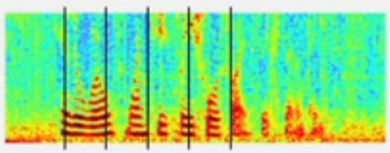
Universitat Autònoma
de Barcelona

INTRODUCTION

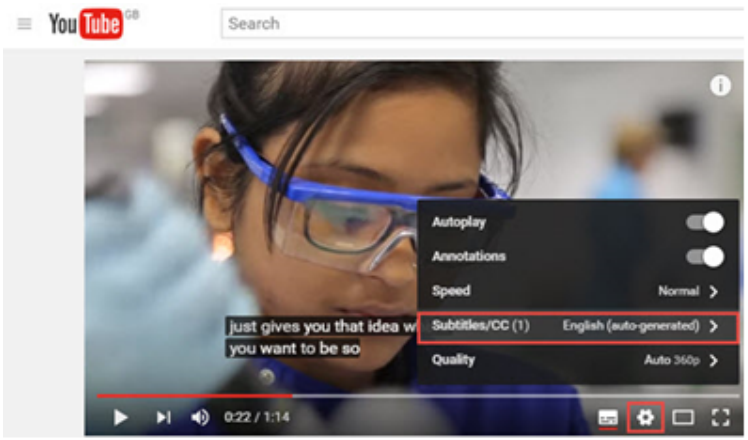
Interest

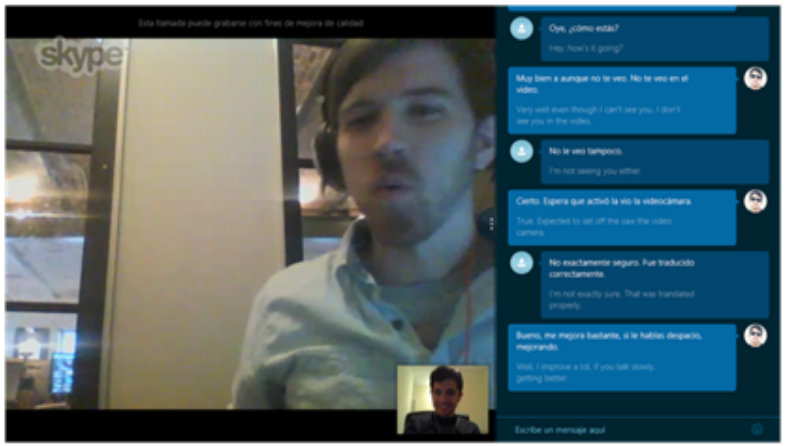

Deep learning
Field of study
+ Add term
Beta: Measuring search interest in topics is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific query, select the "search term" option.
Interest over time
News headlines
Forecast
Note
2005
2007
2009
2011
2013
2015

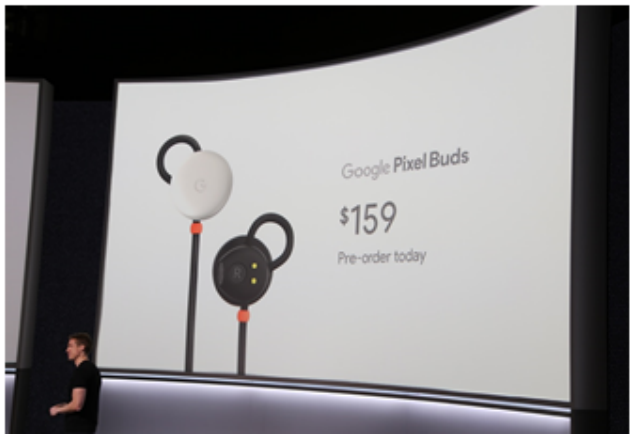
Functions a Deep Neural Network can learn

	input	output
Pixels:		"lion"
Audio:		"How cold is it outside?"
"Hello, how are you?"		"Bonjour, comment allez-vous?"
Pixels:		"A blue and yellow train travelling down the tracks"

Functions a Deep Neural Network can learn



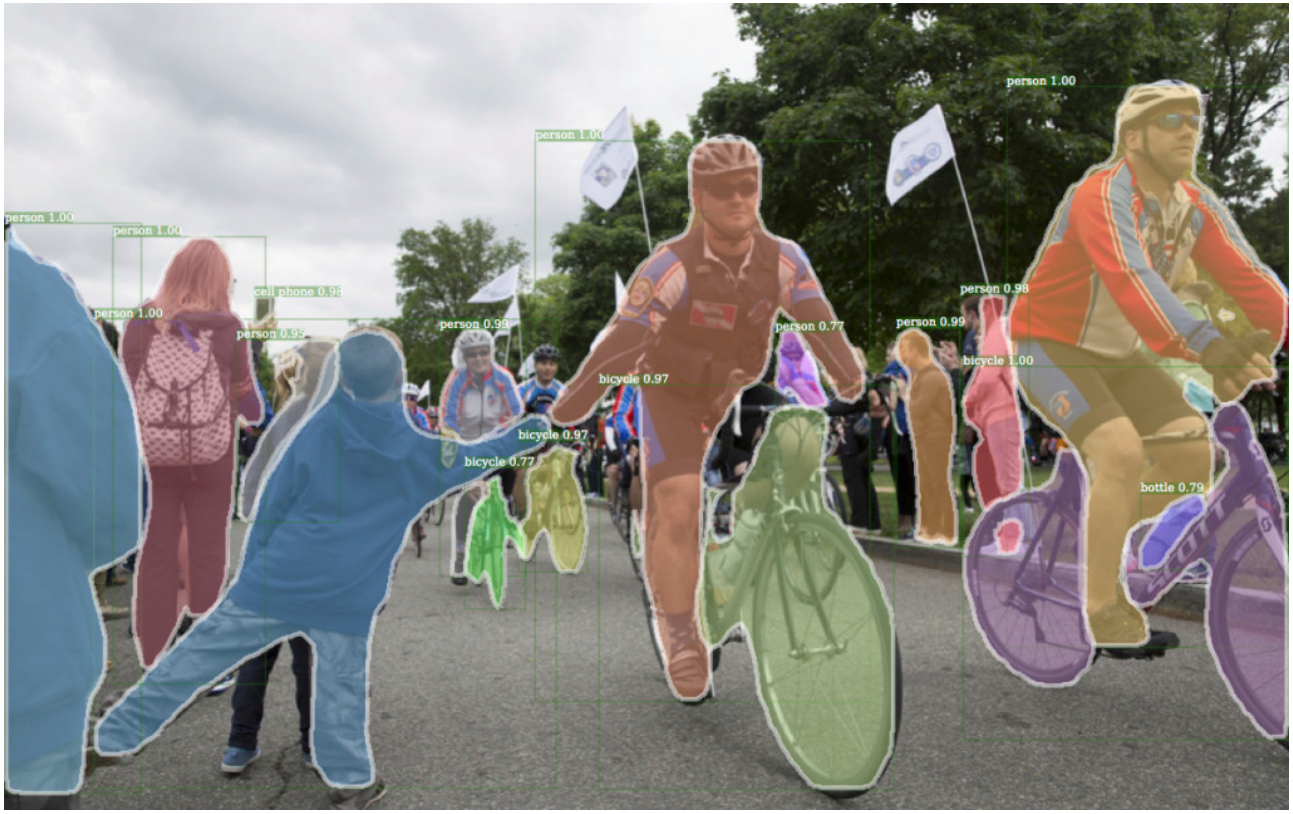




Functions a Deep Neural Network can learn

Functions a Deep Neural Network can learn

Functions a Deep Neural Network can learn



Functions a Deep Neural Network can learn

Question answering model

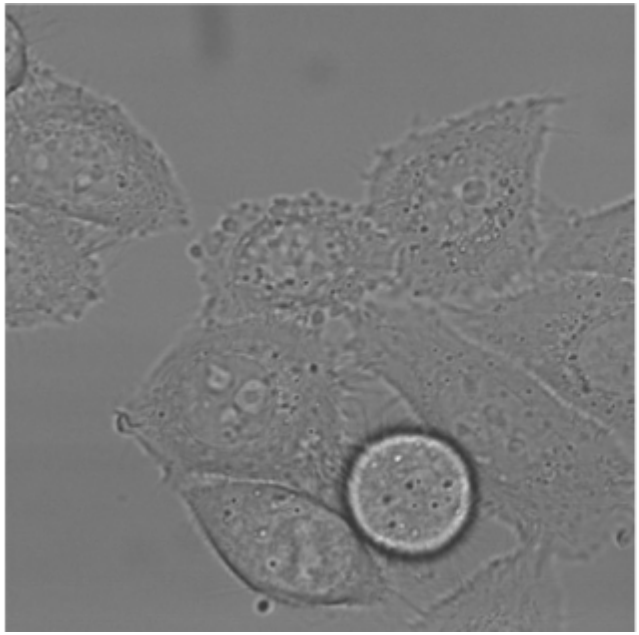
Conversation 4: Philosophical Q&A

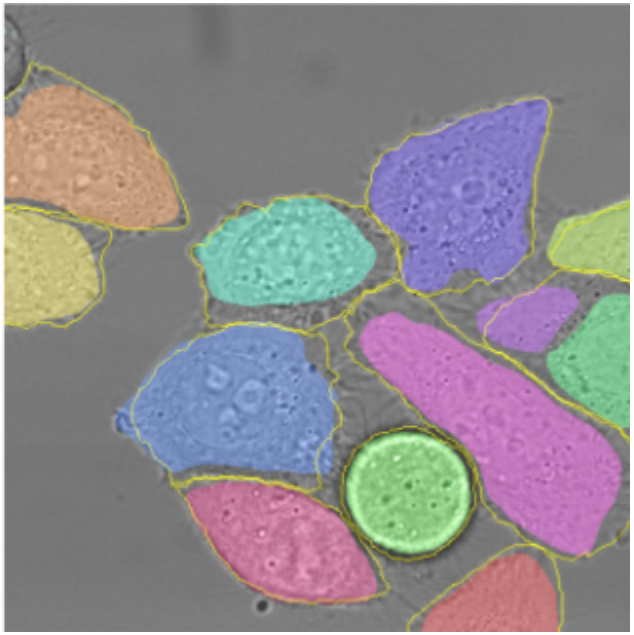
Human: *what is the purpose of life ?*
Machine: *to serve the greater good .*
Human: *what is the purpose of living ?*
Machine: *to live forever .*
Human: *what is the purpose of existence ?*
Machine: *to find out what happens when we get to the planet earth .*
Human: *where are you now ?*
Machine: *i 'm in the middle of nowhere .*
Human: *what is the purpose of dying ?*
Machine: *to have a life .*
Human: *what is the purpose of being intelligent ?*
Machine: *to find out what it is .*
Human: *what is the purpose of emotions ?*
Machine: *i don 't know .*

Deep adversarial learning

Neural Style Transfer

Healthcare Applications

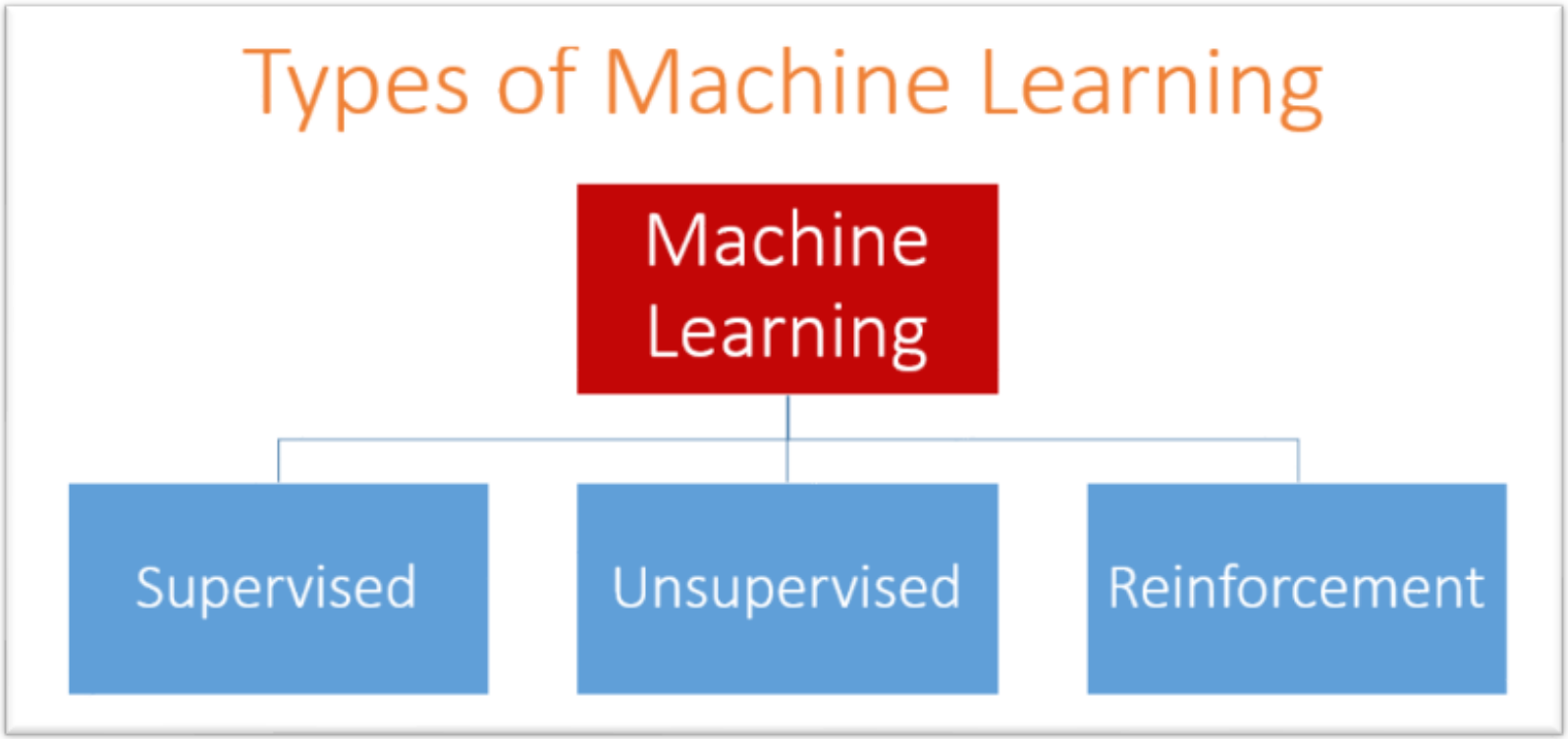




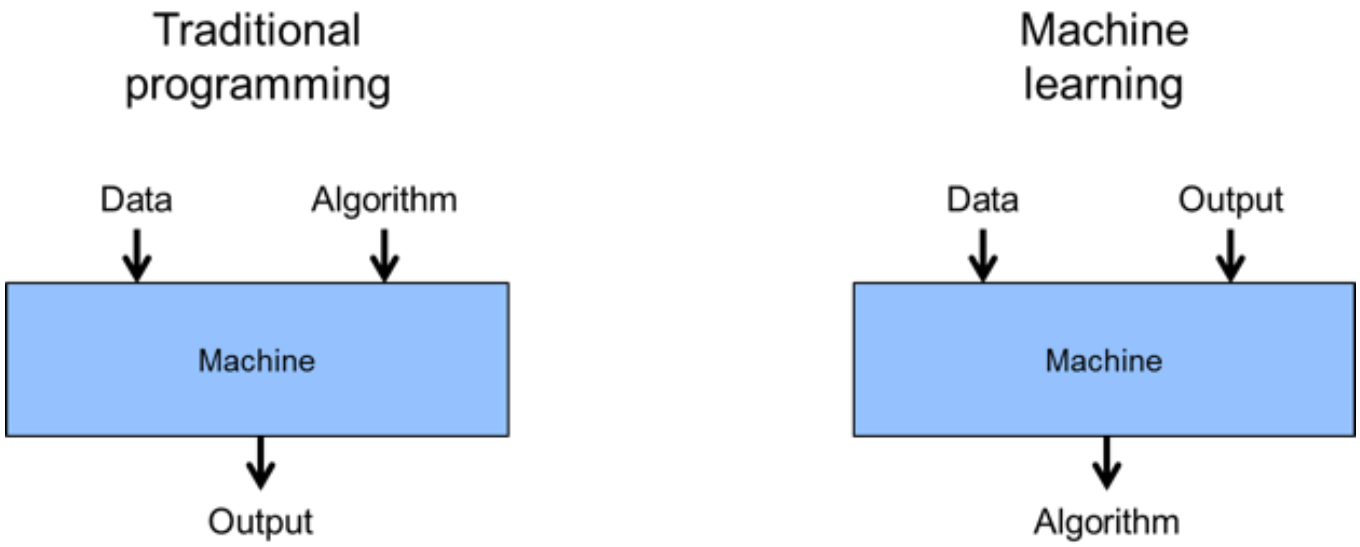
Error in automated Image Captioning

Machine Learning perspective

Machine Learning: The concept



Machine Learning perspective



Software 2.0 with Deep Neural Networks

Medium

Andrej Karpathy Follow

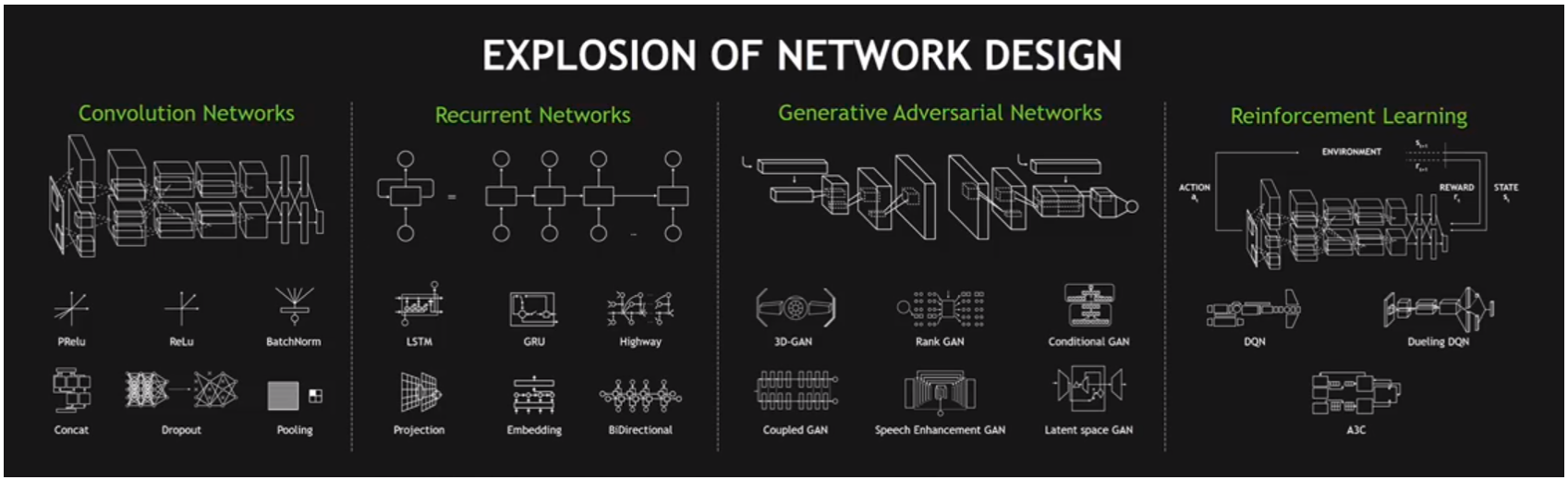
Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

Nov 12, 2017 · 8 min read

Software 2.0

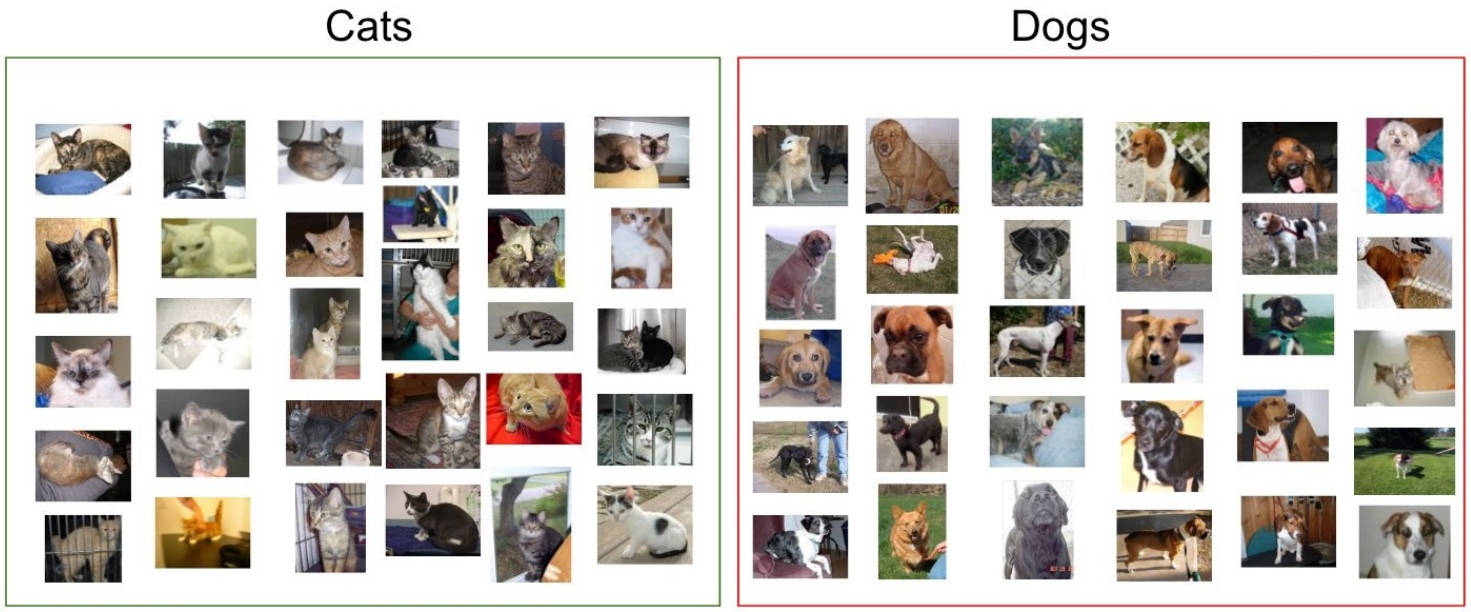
https://medium.com/@karpathy/software-2-0-a64152b37c35

Networks designs



Functions a Deep Neural Network can learn

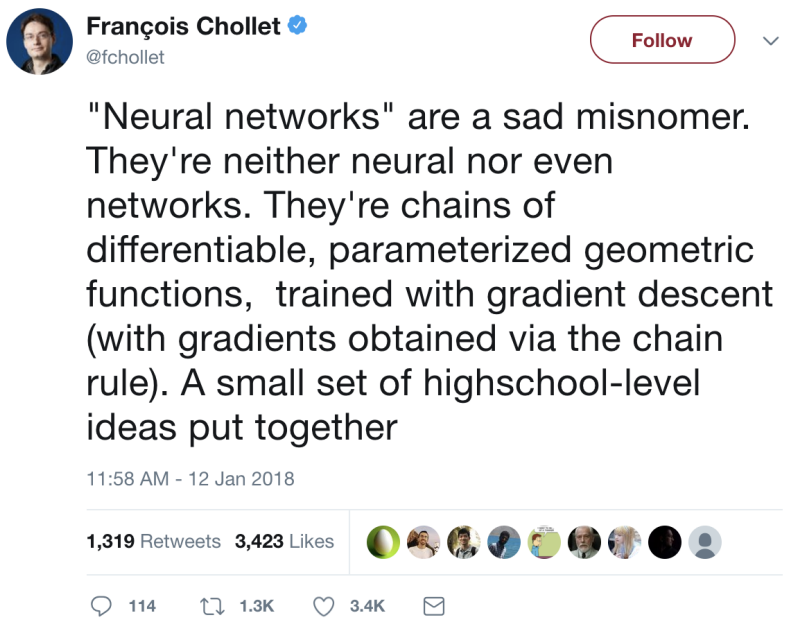
Cats vs. Dogs image Classification...



Sample of cats & dogs images from Kaggle Dataset

Magic? AI?

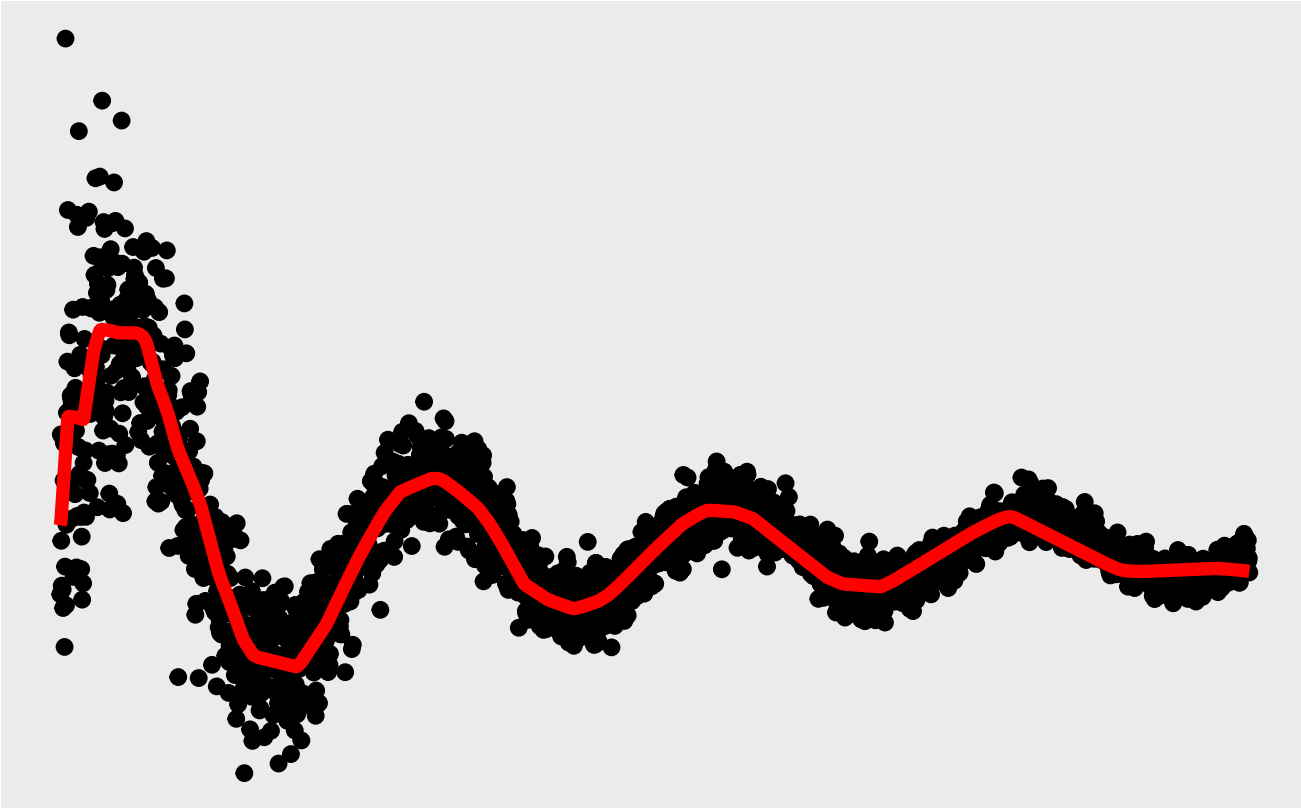
Properties



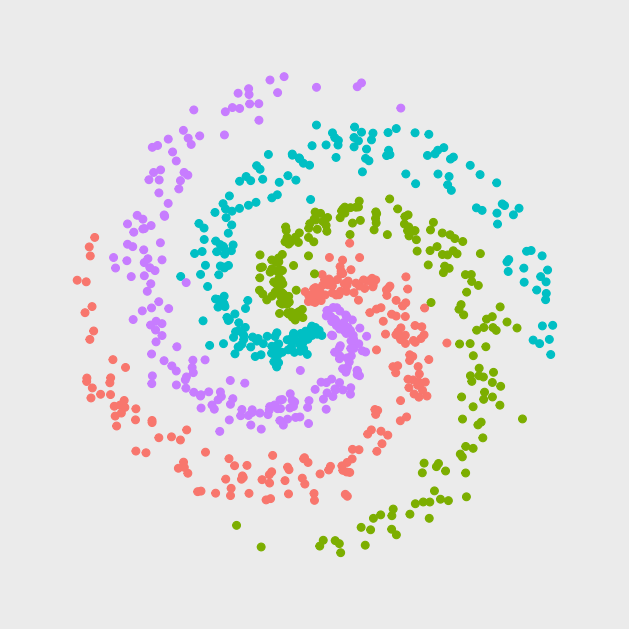
A simple mechanism that, once scaled, ends up looking like magic.

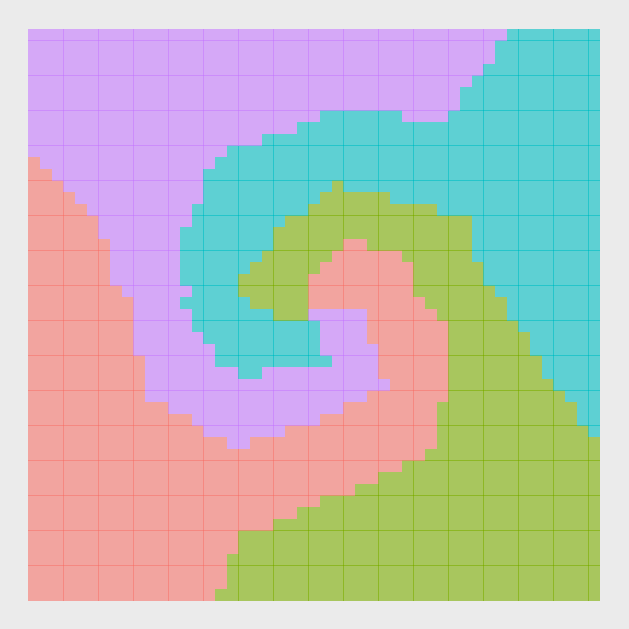
Regression problem: $(1/x)\sin(x)$

Powerful models to map inputs to outputs!

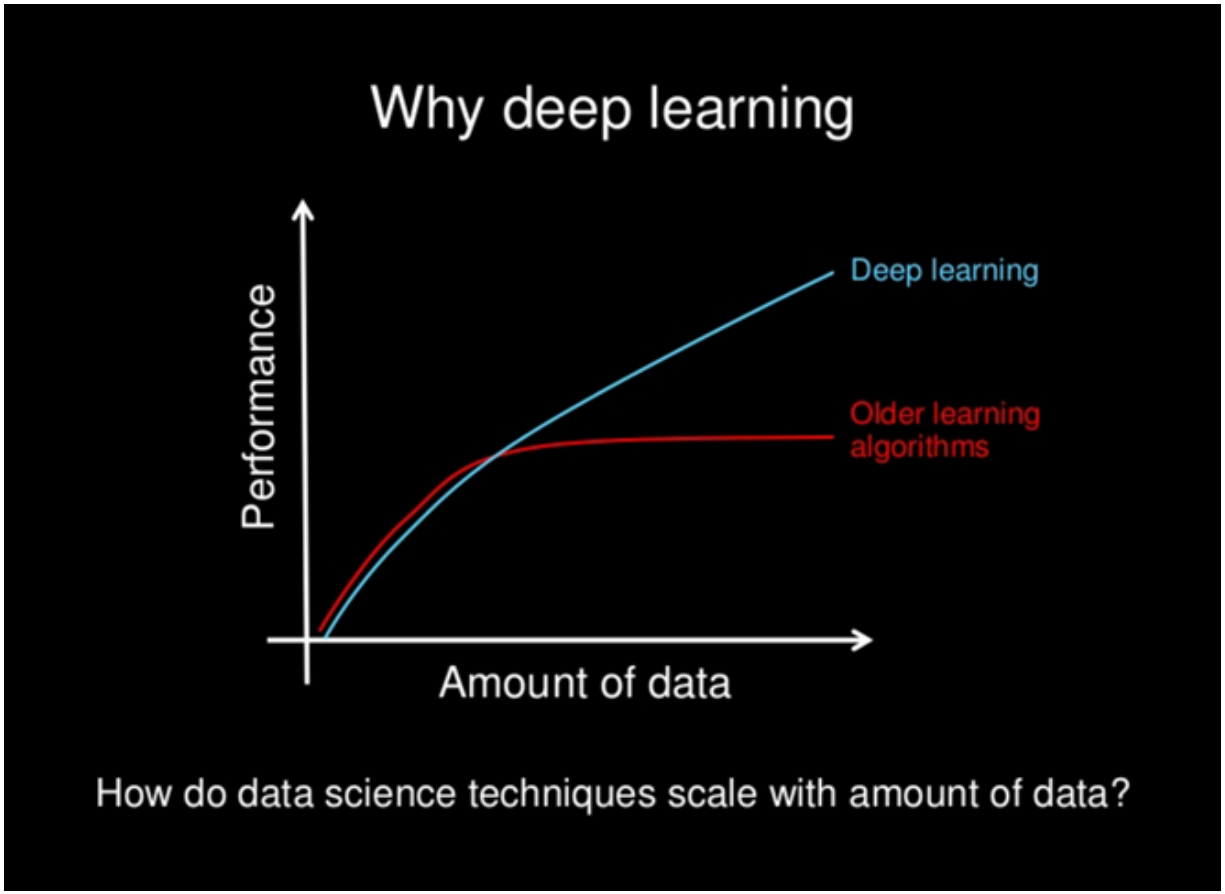


Classification problem: Spiral data

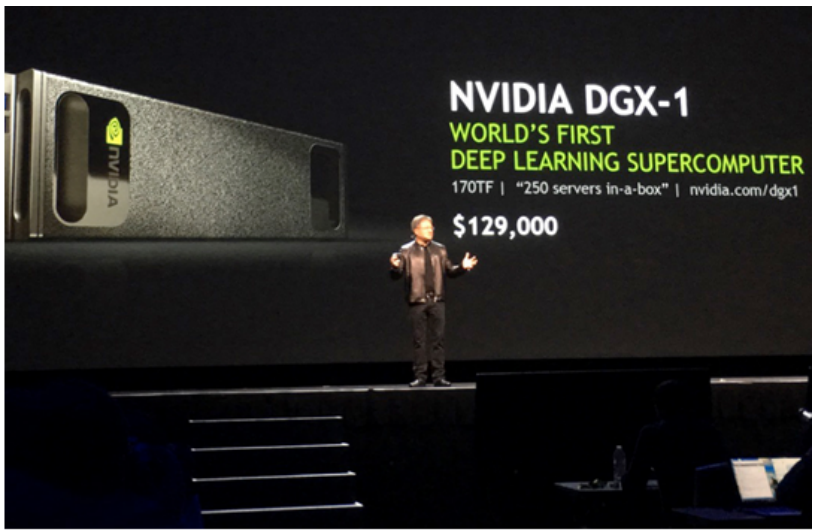




Properties


Why deep learning
Performance
Deep learning
Older learning algorithms
Amount of data
How do data science techniques scale with amount of data?

GPU computing



Deep Learning performance doubles
For data scientists and researchers

2x Faster Single GPU Training
Support for Larger Models

cuDNN

cuDNN

2x Larger Datasets
Instruction-level Profiling

CUDA

Companies and startups that use DL

NEURAL NETWORKS

General formulation

In the function estimation problem one has a system consisting of a random output variable **y** and a set of random input variables $\mathbf{x} = \{x_1, x_2, ..., x_n\}$. Given a *training* sample $\{y_i, x_i\}_1^N$, the goal is to find a function $F_\theta^* (\mathbf{x})$ that maps **x** to **y**, such that over the joint distribution of all (**y**, **x**) values, the expected value of some specified loss function $J(y, F_\theta(\mathbf{x}))$ is minimized given θ :

$$F_\theta^* (\mathbf{x}) = argmin\ J(y, F_\theta(\mathbf{x}))$$
$$\theta \in \mathbb{R}^n$$

Example: linear regression

A multiple **linear regression** allows to express the relationship between inputs and outputs as follows:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_n X_{in}$$

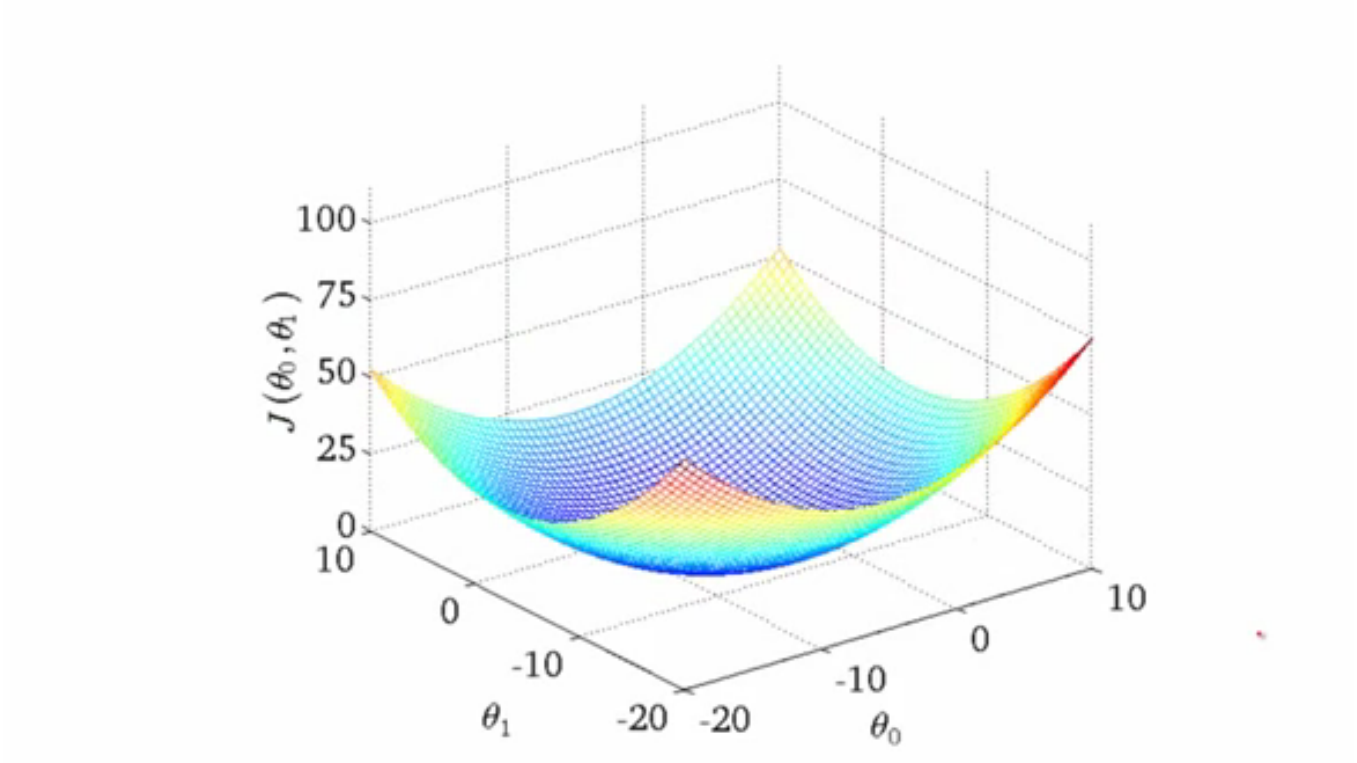
The loss function is minimizing SE:

$$J(\beta) = \sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_n X_{in}) \right)^2$$

In this particular case, we can obtein a closed formula for the optimial parameters:

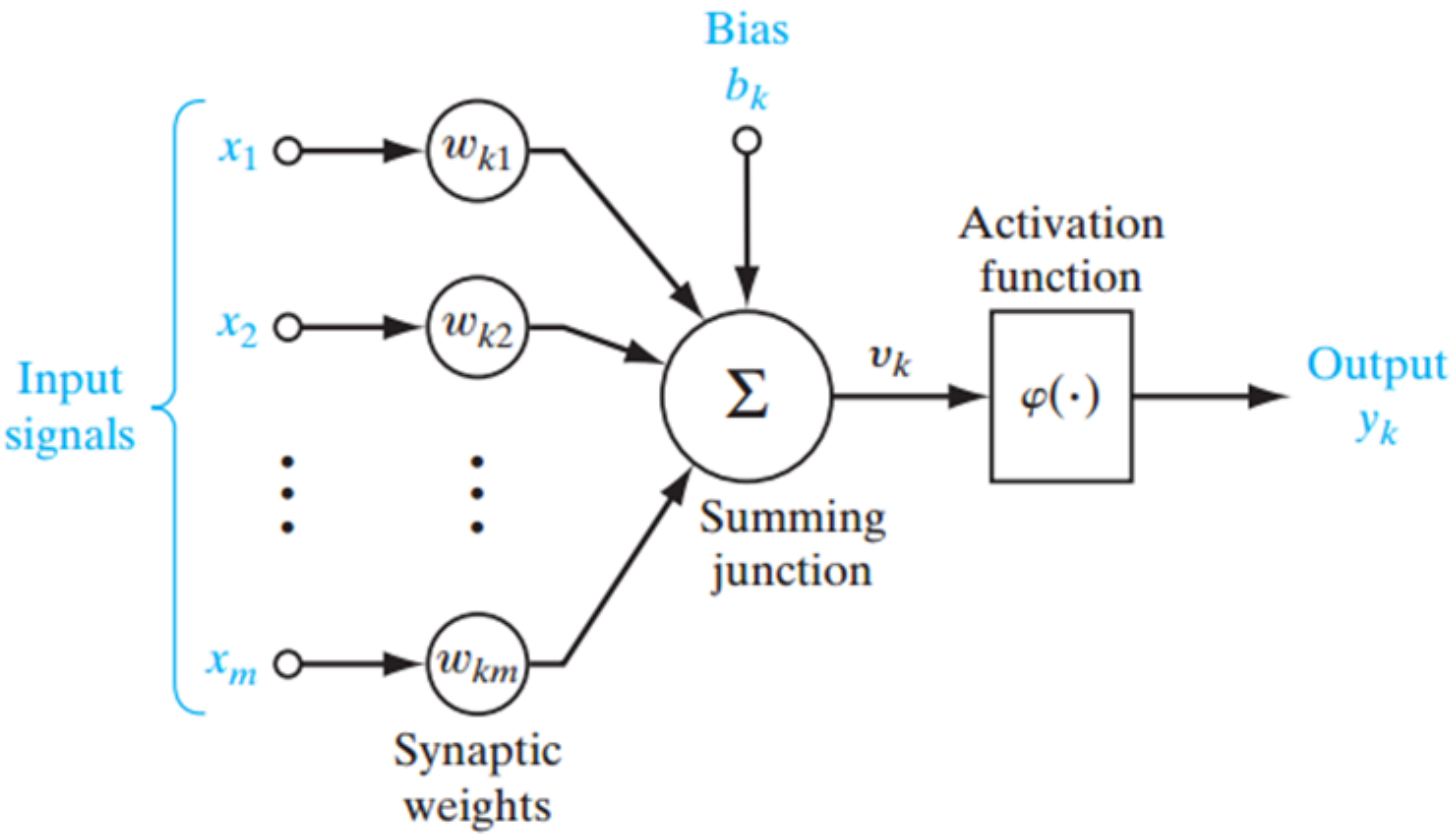
$$\widehat{\beta} = \left(X^T X \right)^{-1} \left(X^T Y \right)$$

Loss function in LR



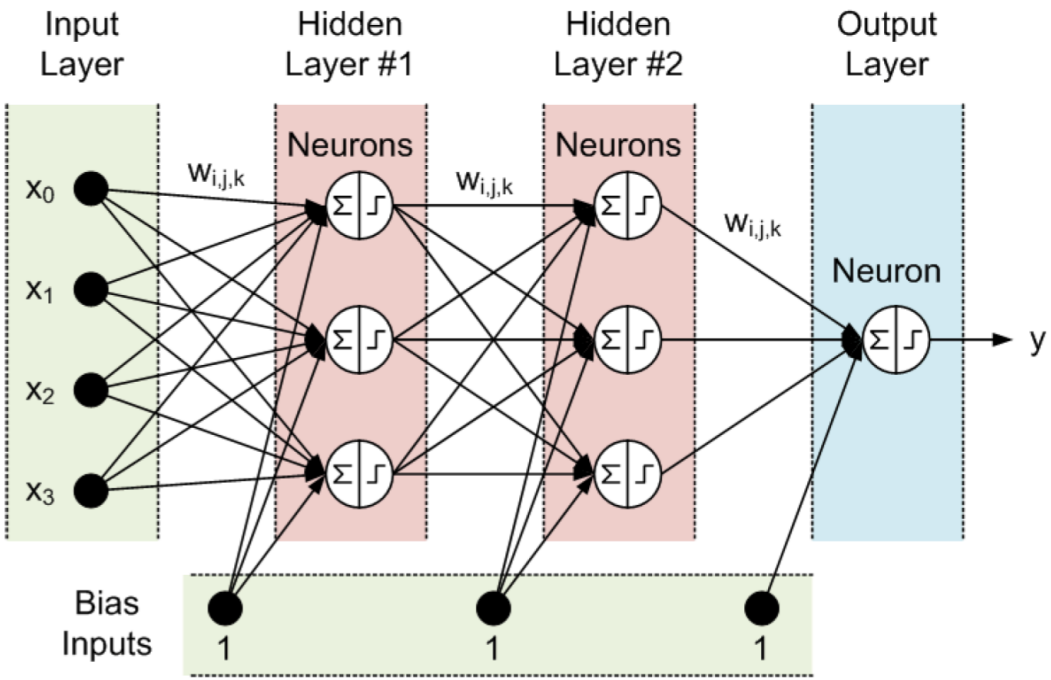
Convex optimization problem (quadratic form)

Neuron


Bias
b_k
x_1
x_2
x_m
w_{k1}
w_{k2}
w_{km}
Input signals
Activation function
Σ
v_k
$\varphi(\cdot)$
Output
y_k
Summing junction
Synaptic weights

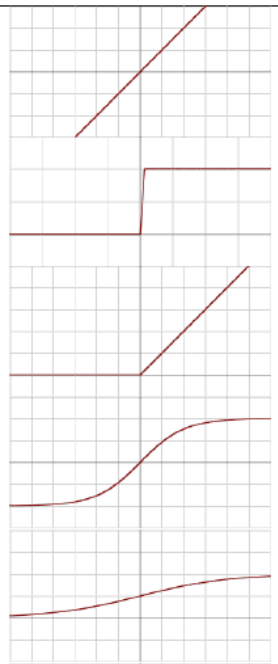
Why use hiden layers configuration?

Why use hiden layers configuration?

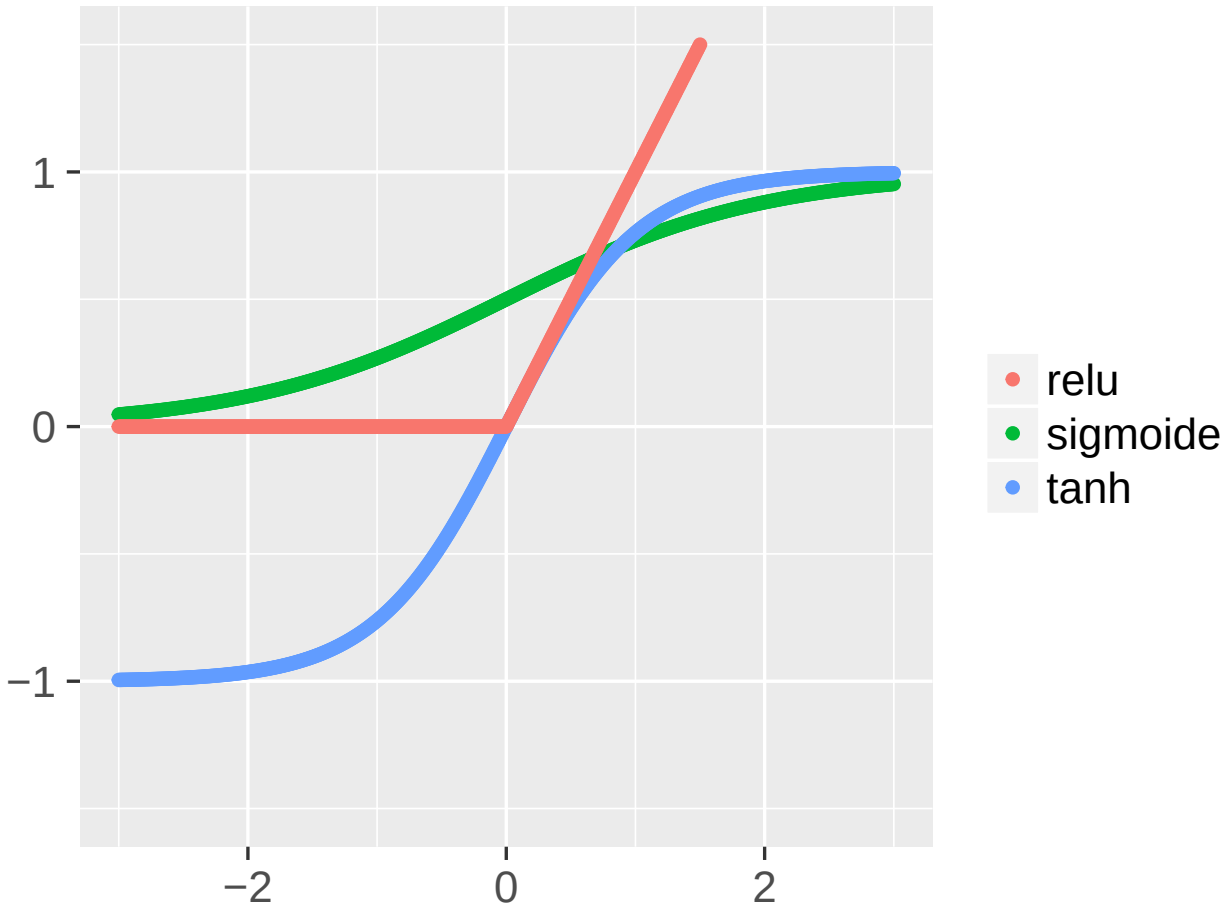


The universal approximation theorem found that a neural network with one hidden layer can approximate any continous function (George Cybenko, 1989).

Activation functions

Name	Visualization	f(x) =	Notes
Linear (= Identity)		x	Not useful for hidden layers
Heaviside Step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	Not differentiable
Rectified Linear (ReLU)		$\begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	Surprisingly useful in practice
Tanh		$\frac{2}{1+e^{-2x}} - 1$	A soft step function; ranges from -1 to 1
Logistic ('sigmoid')		$\frac{1}{1+e^{-x}}$	Another soft step function; ranges from 0 to 1

Activation functions



Loss functions

Error Metrics for Regression Problems:

- Mean Squared Error: $(\frac{1}{n}\sum(y_i - \widehat{y}_i)^2)$.
- Root Mean Squared Error.
- ...

Error Metrics for Classification Problems:

- Logistic Loss.
- Logarithmic Loss: $-\frac{1}{N}\sum\sum y_{ij}\log(p_{ij})$.
- Accuracy.
- ...

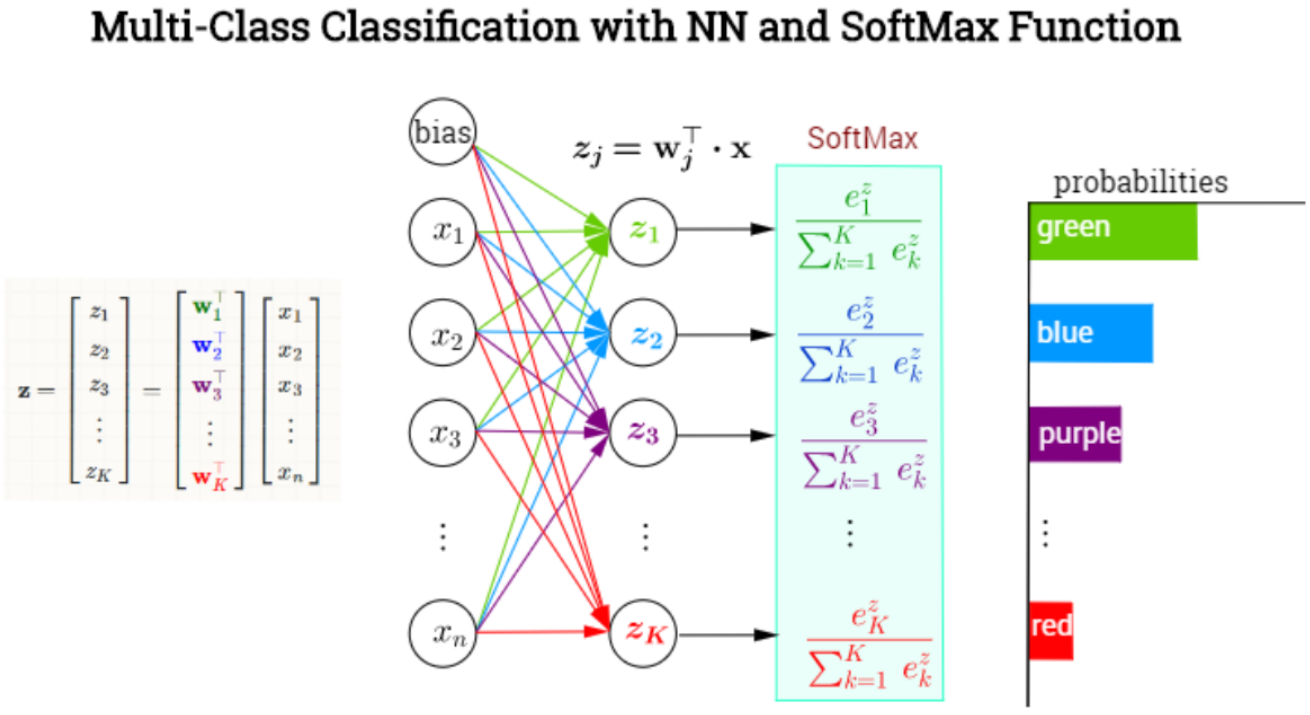
Softmax function activation

$$\sigma : \mathbb{R}^K \to [0,1]^K$$

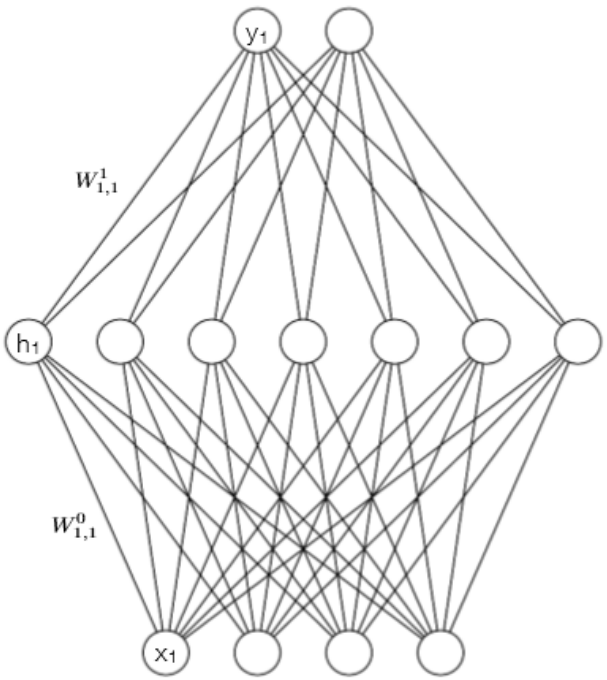
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

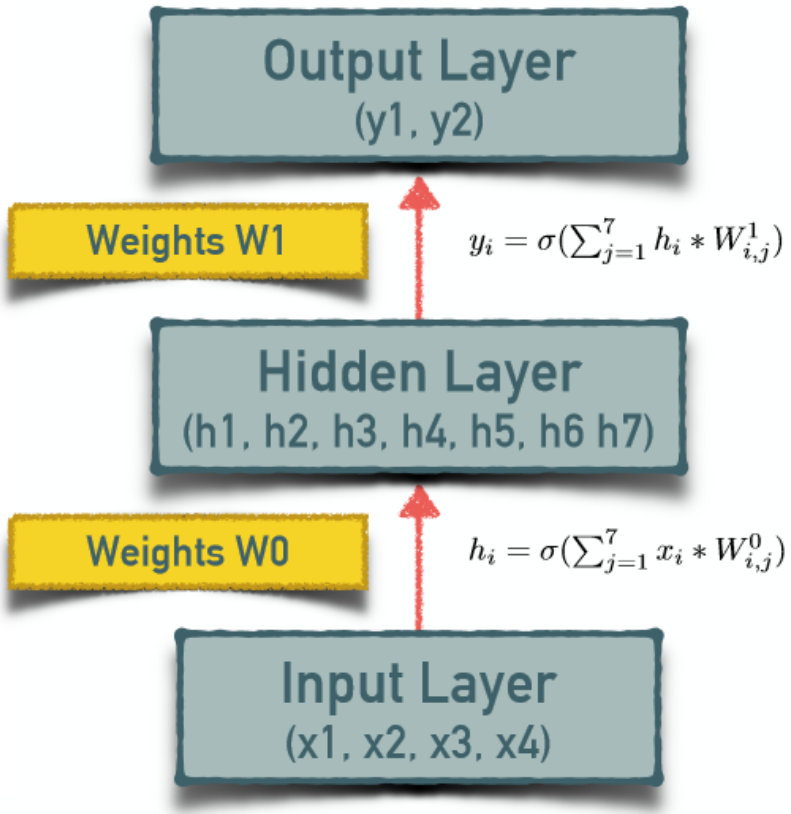
In probability theory, the output of the softmax function can be used to represent a categorical distribution - that is, a probability distribution over K different possible outcomes.

Softmax function activation



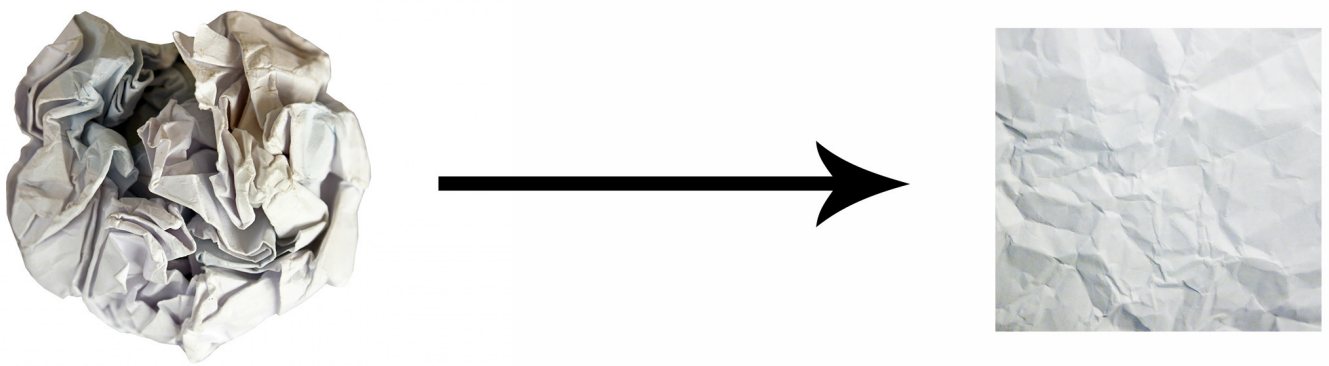
Feed Forward Neural Network with 1 hidden layer





Intuition

Complex geometric transformations broken down into simpler ones.



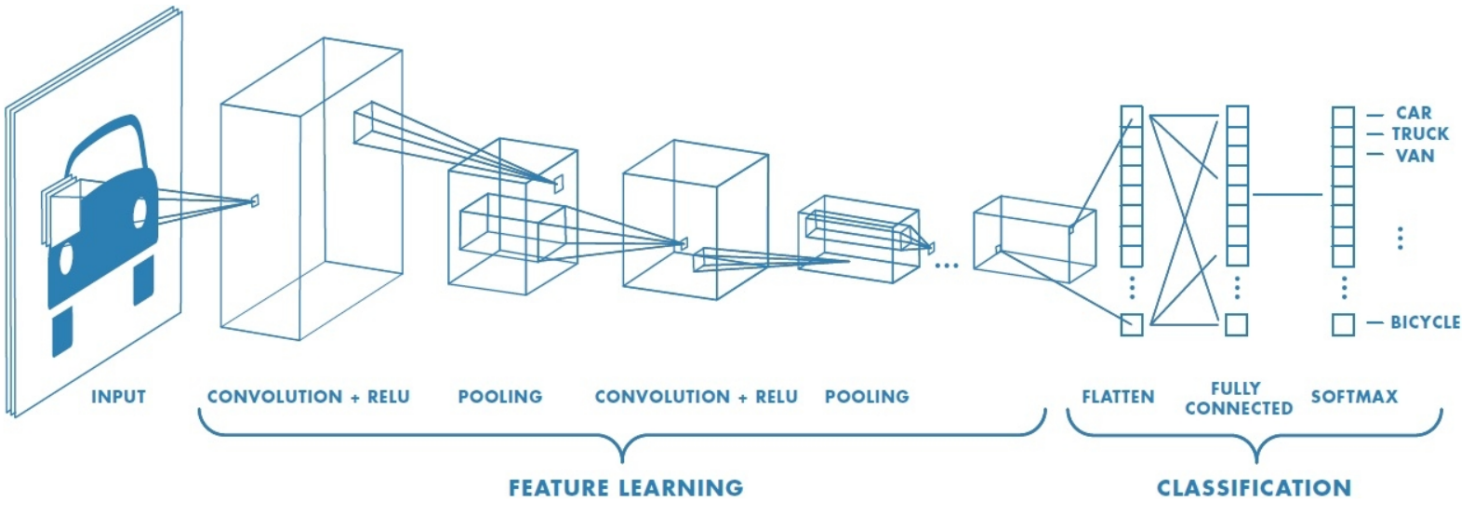
Tensorflow playground

CNN

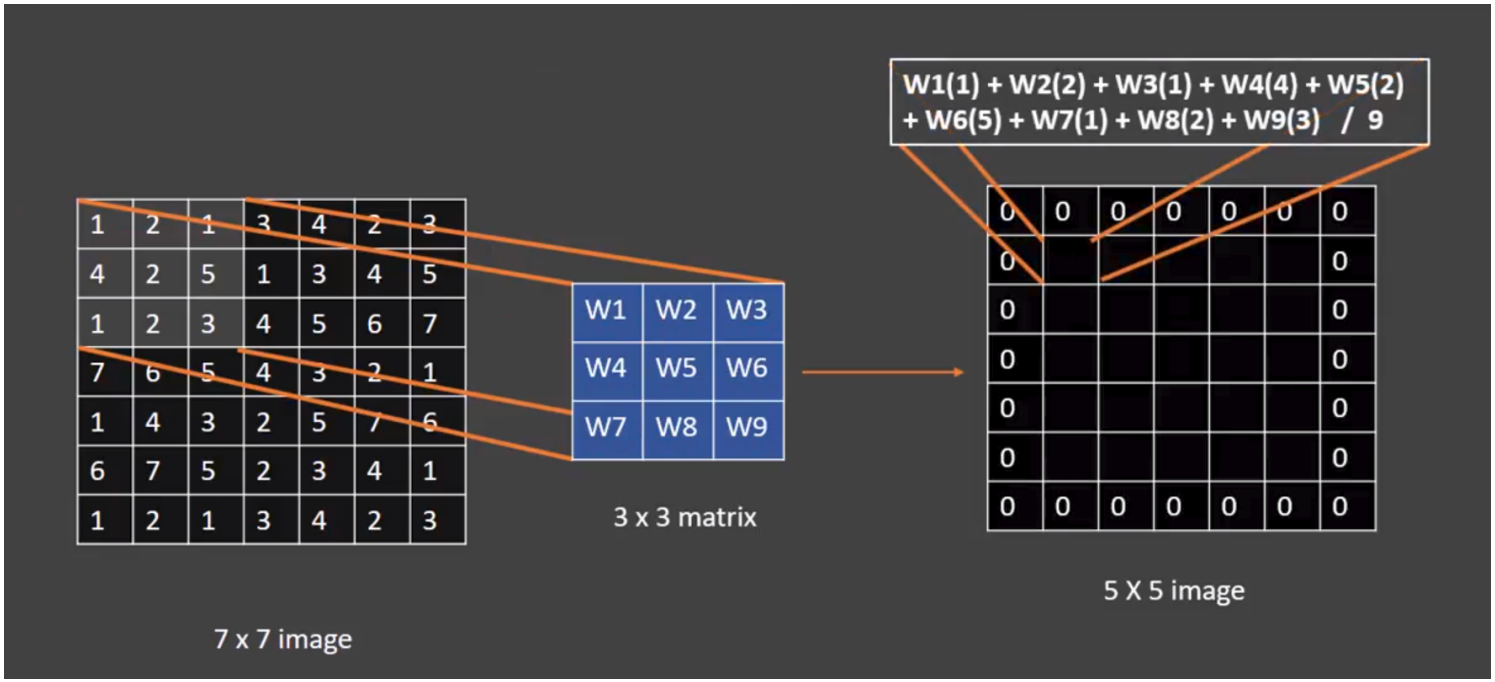
- Similar to Feed Forward Neural Networks.
- Convolutional neural networks (CNN) model can be applied to visual recognition tasks.
- The architecture of a CNN is designed to take advantage of the grid structure of data.
- Hierarchy of representations with increasing level of abstraction.
- Each stage is a kind of trainable feature transform.

Key components of CNN

- Convolutional layer.
- Pooling.

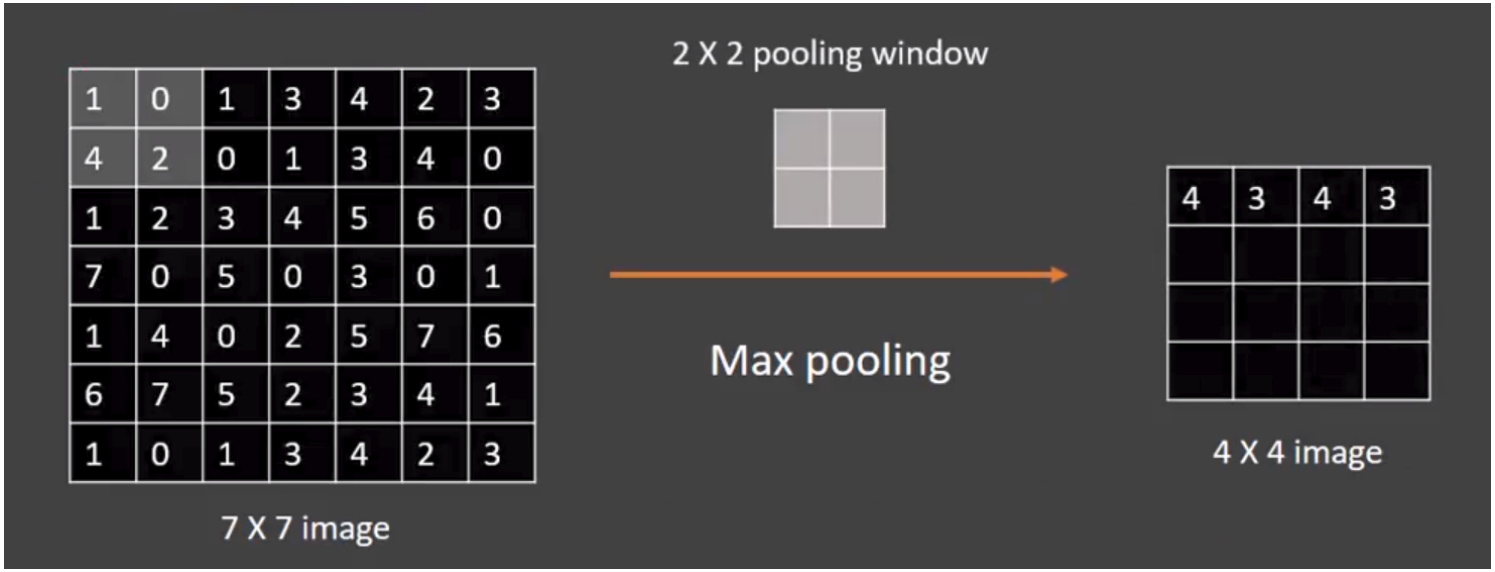


Convolutional layer



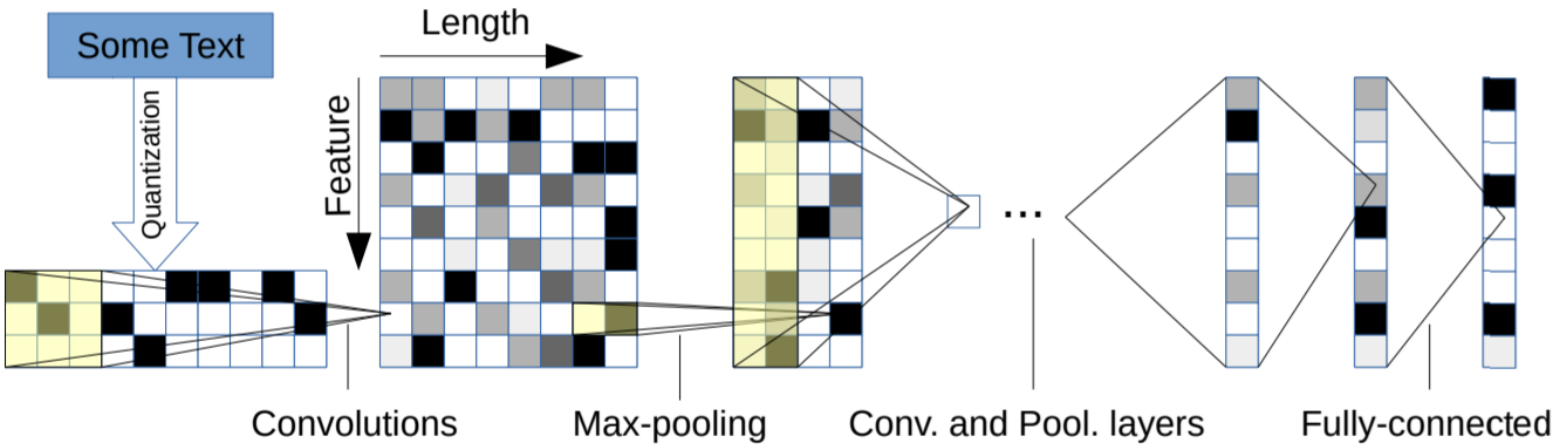
The convolution layer is the main building block of a CNN.

Pooling layer



Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently.

CNN for text


Some Text
Quantization
Length
Feature
...
Convolutions
Max-pooling
Conv. and Pool. layers
Fully-connected

RNN

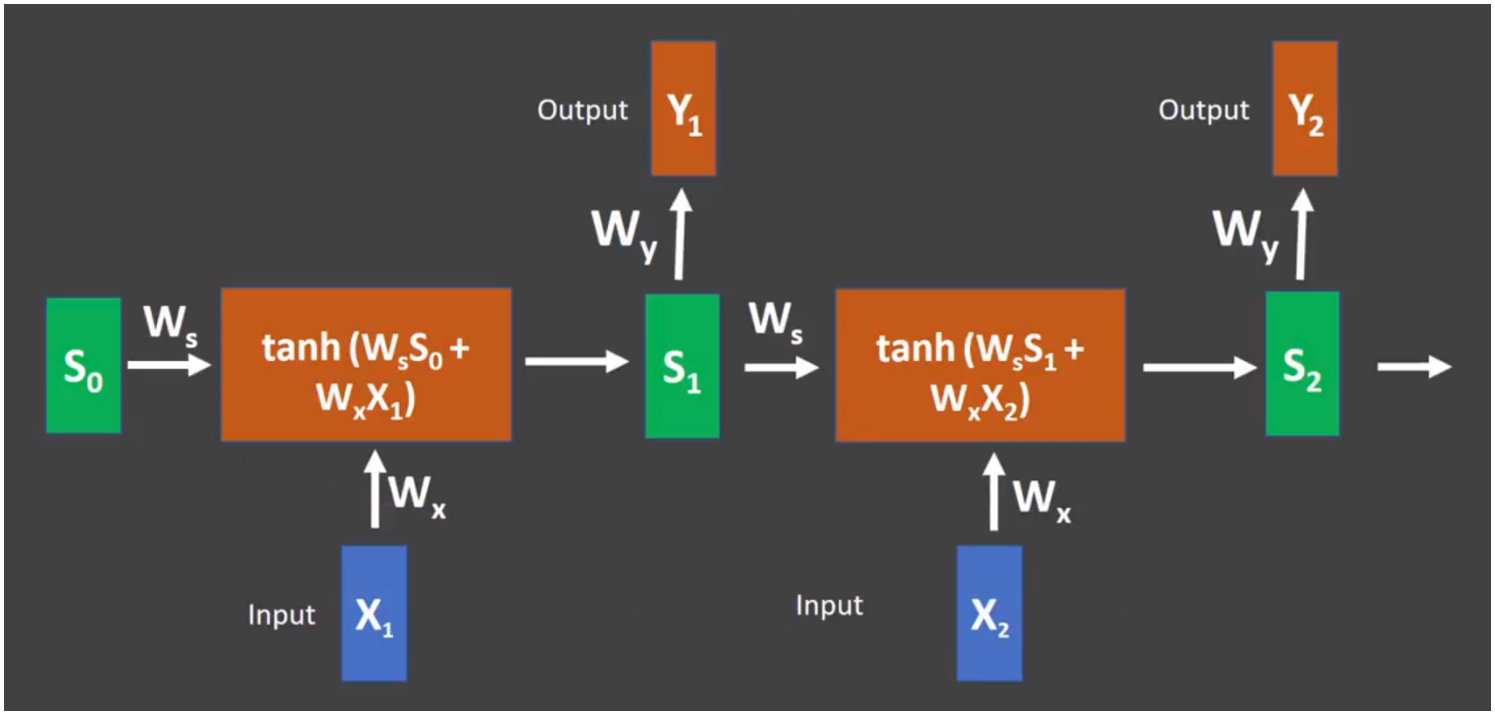
Limitation with Feed Forward Neural Networks (and CNN)...

- Not designed for sequential data.
- Fixed dimension of inputs.
- Dos not model memory.

Recurrent neural network...

- To map predictions in sequences.
- This creates an internal state of the network which allows it to exhibit dynamic temporal relations.
- Aplications: Image captioning, Sentiment analysis, Question answering, Speech recognition, Time series, Generating music, Machine translation, Self-driving cars, ...

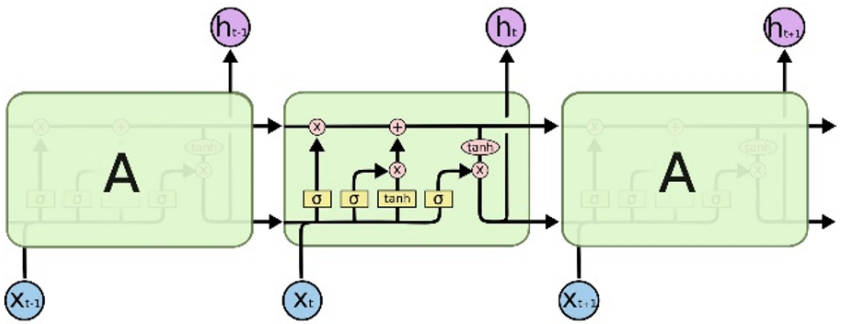
Vanilla RNN (Unrolled)

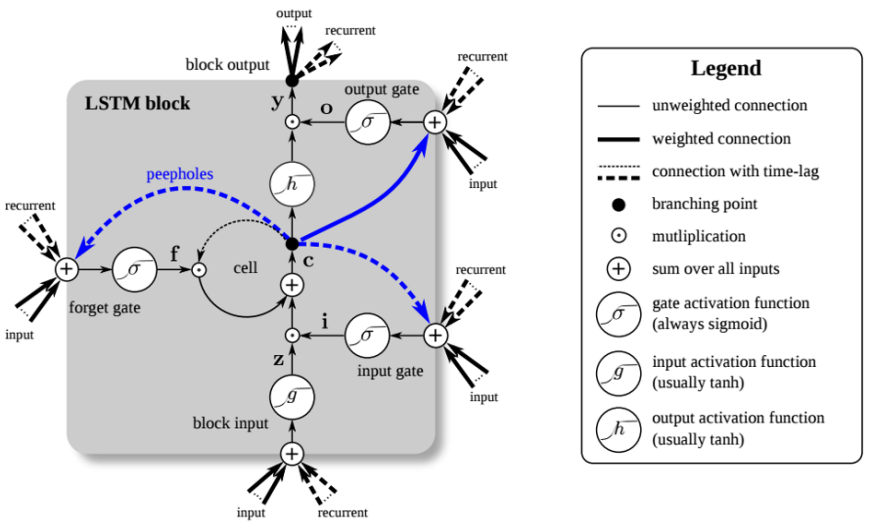


Vanilla RNN is affected by the Vanishing Gradient problem and the solution to the problem which is Long Short Term Memory networks (LSTM).

LSTM

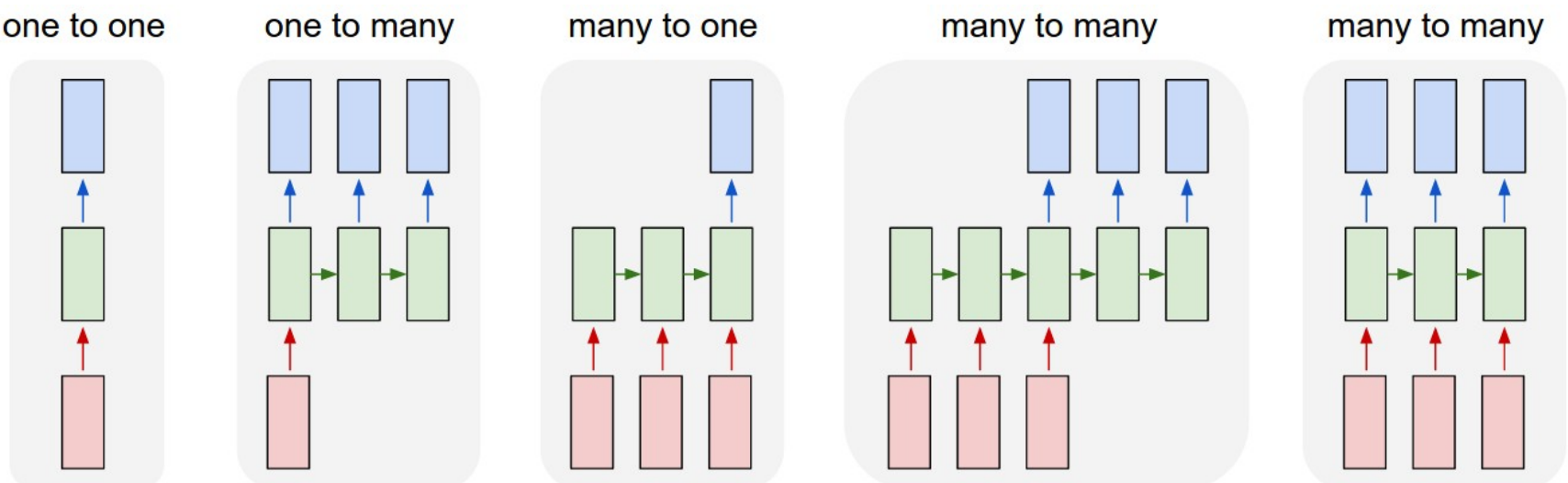
LSTMs are explicitly designed to avoid the long-term dependency problem...





"Google will soon be a big LSTM", Jüergen Schmidhuber

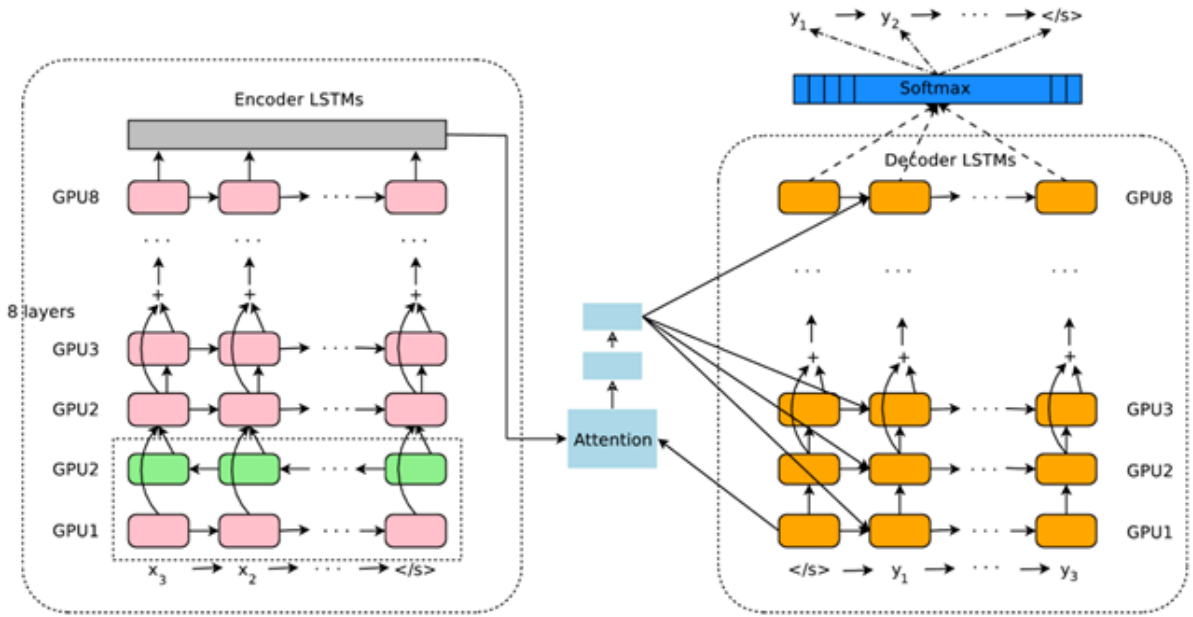
Input/output size



Operate over sequences of vectors: Sequences in the input, the output, or in the most general case both.

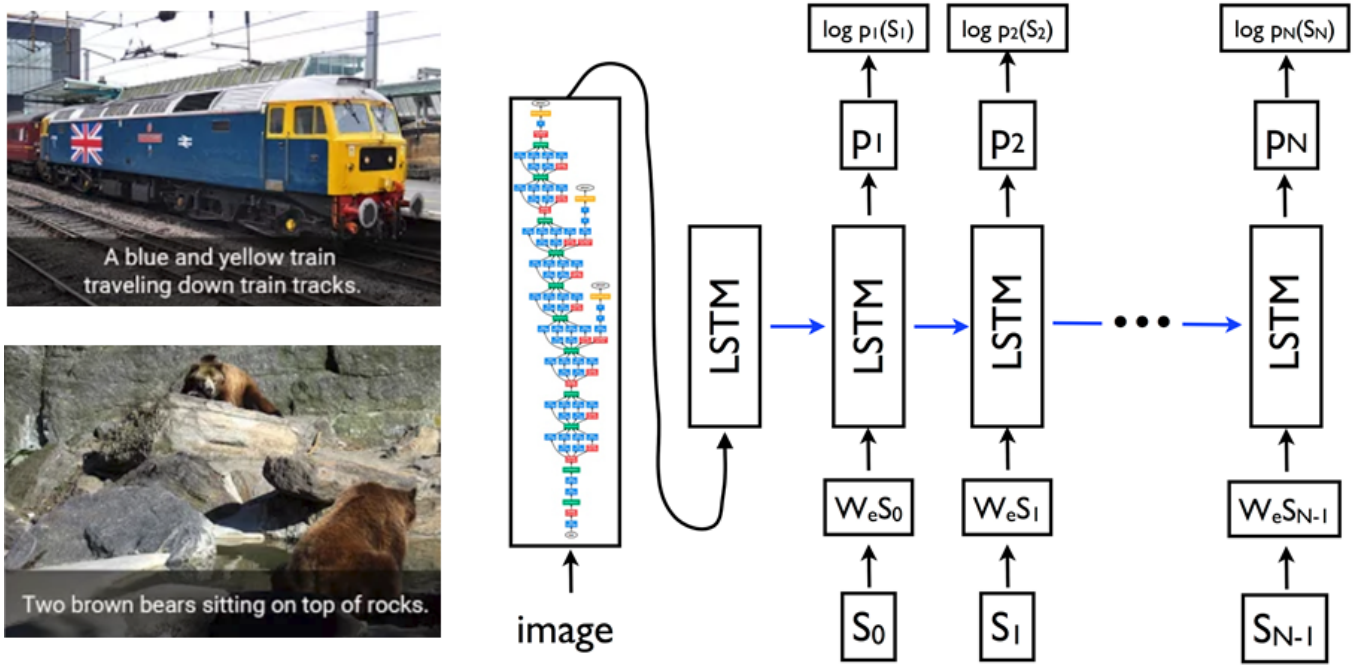
LSTM with multiple hidden layers

Seq2seq learning for NMT...



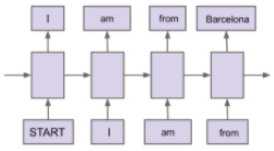
Computer requirements

Image Captioning

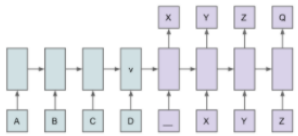


Architectures of Deep Networks

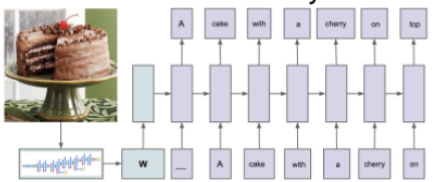
Sequence Prediction



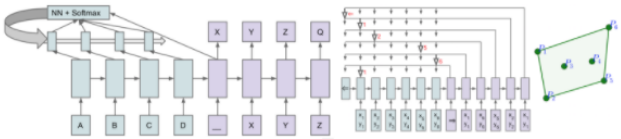
Seq2Seq



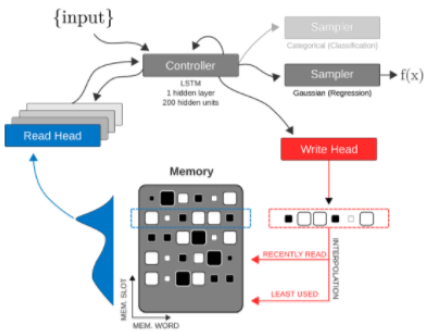
Multimodality

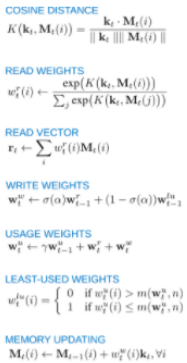


Attention/Pointers

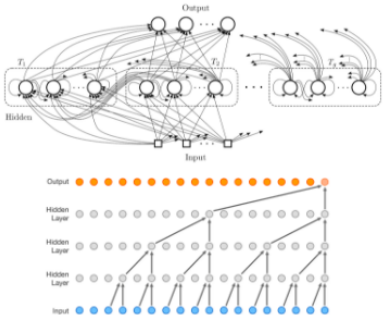


Read/Write memories

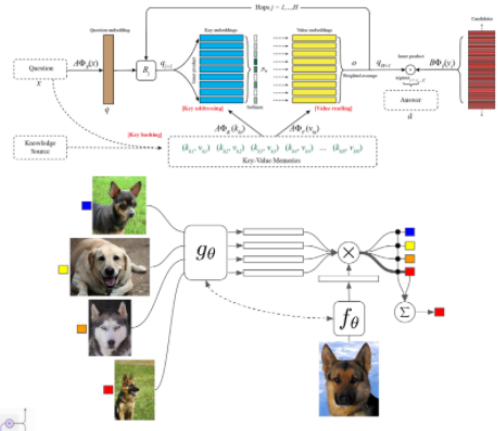




Temporal Hierarchies

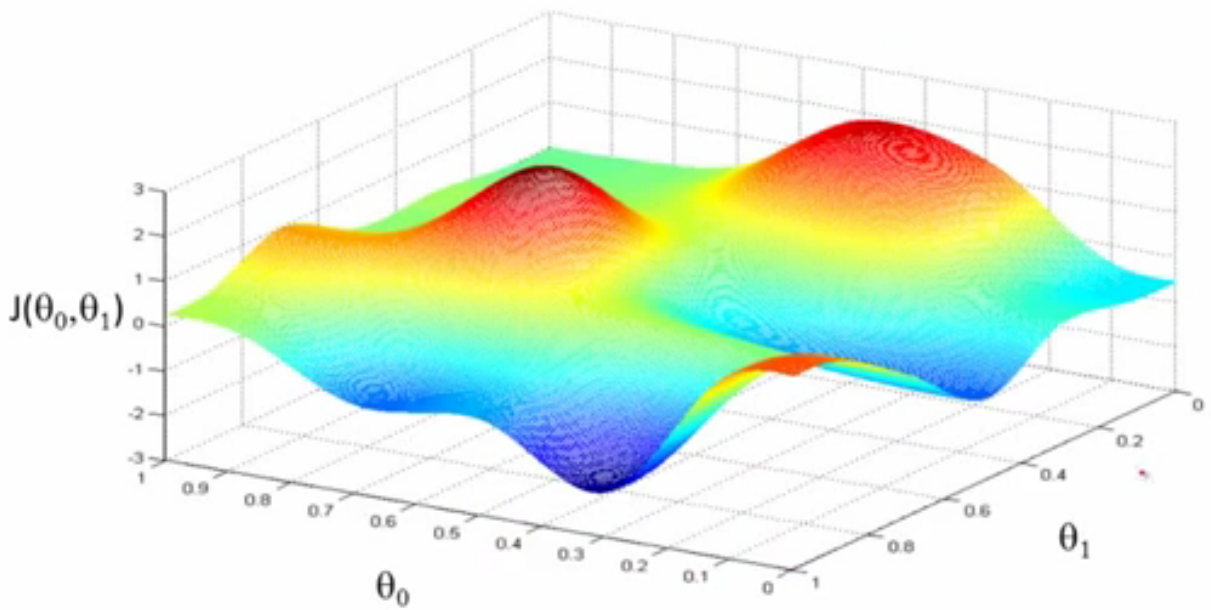


Key,Value memories



Recurrent Architectures

Loss functions in Deep Learning



Non-convex optimization problem

Gradient descent

- Gradient descent is a first-order optimization algorithm.
- Find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

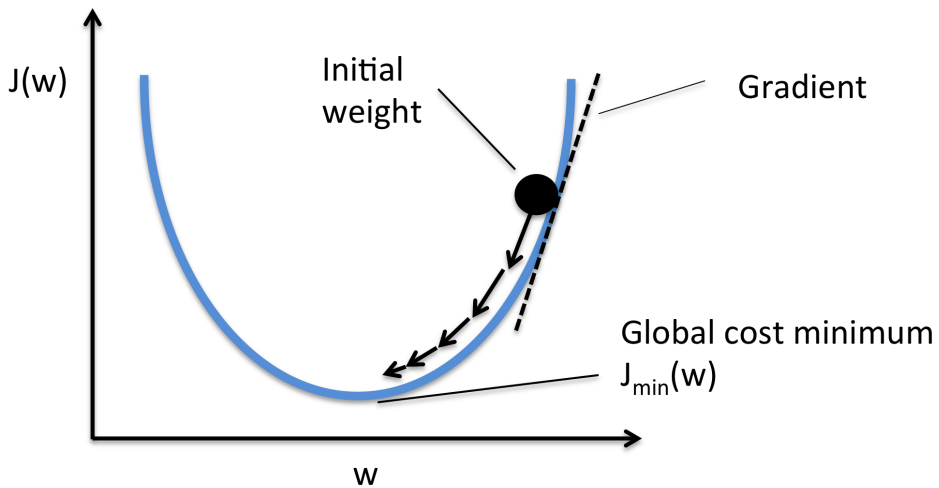


Illustration of gradient descent on a series of level sets.

Idea to train NN

- The loss function takes the predictions of the network and the true targets and computes a distance score, capturing how well the network has done.
- The fundamental trick in deep learning is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the current batch of examples (**negative gradient**).
- The gradient ($\nabla J(\theta)$) are calculated by **backpropagation**. Calculates the gradient of a loss function with respect to all the weights in the network (**chain rule**) and uses it to update the weights to minimize the loss function.

Stochastic gradient descent (SGD)

Mini-batch gradient descent performs an update for every mini-batch of n training examples:

$$\theta_{t+1} = \theta_t - \eta \nabla J \left(\theta_t / x^{(i:i+n)}; y^{(i:i+n)} \right)$$

Where η is the learning rate.

Can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient a mini-batch very efficient. Common mini-batch sizes range between 32 and 256, but can vary for different applications. **Epoch** is defined as one complete iteration. Fast with GPU computation.

Variants: RMSprop, Adam, ...

Learning rate

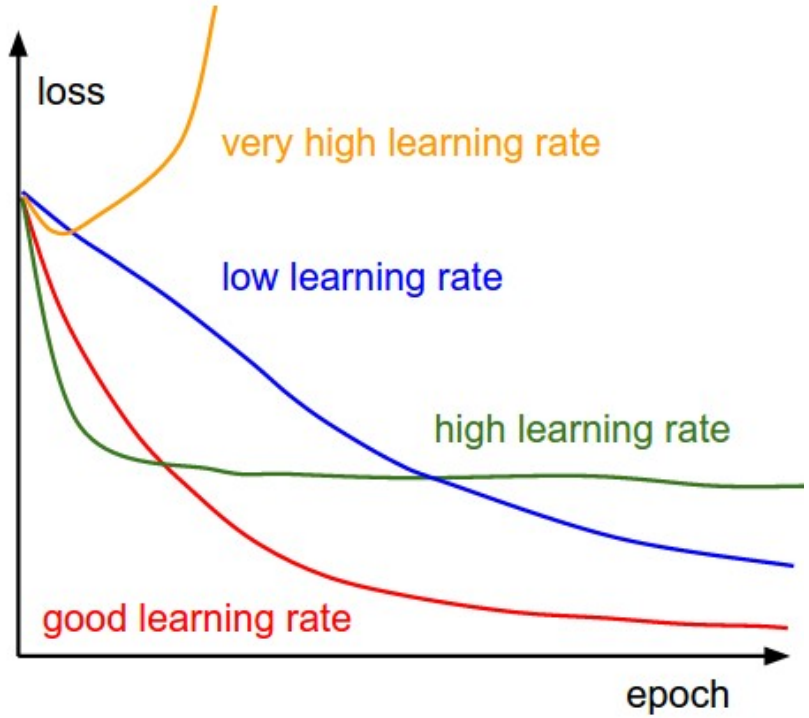


Illustration of gradient descent on a series of level sets.

Random Initialization Methods

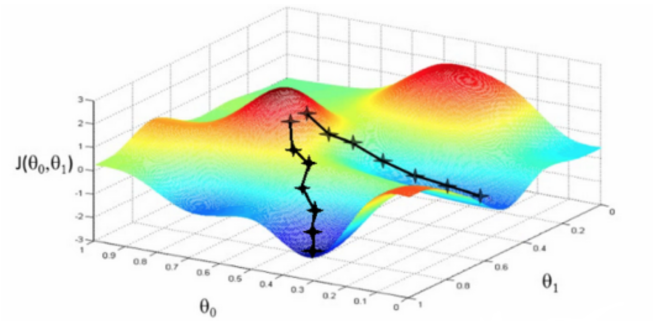
- Initialization with random variables with expectation 0 and low variance.

Xavier's initialization is based on two hypotheses:

- The variance is constant in both the forward and backward directions.
- A dense neural network with symmetric activation f will have $f^{'}(y_l) = 1$ when $y_l = 0$.

$$\text{Then, } Var[\theta_l] \simeq \frac{\sqrt{6}}{n_l + n_{l+1}}$$

The Loss Surfaces of Multilayer Networks

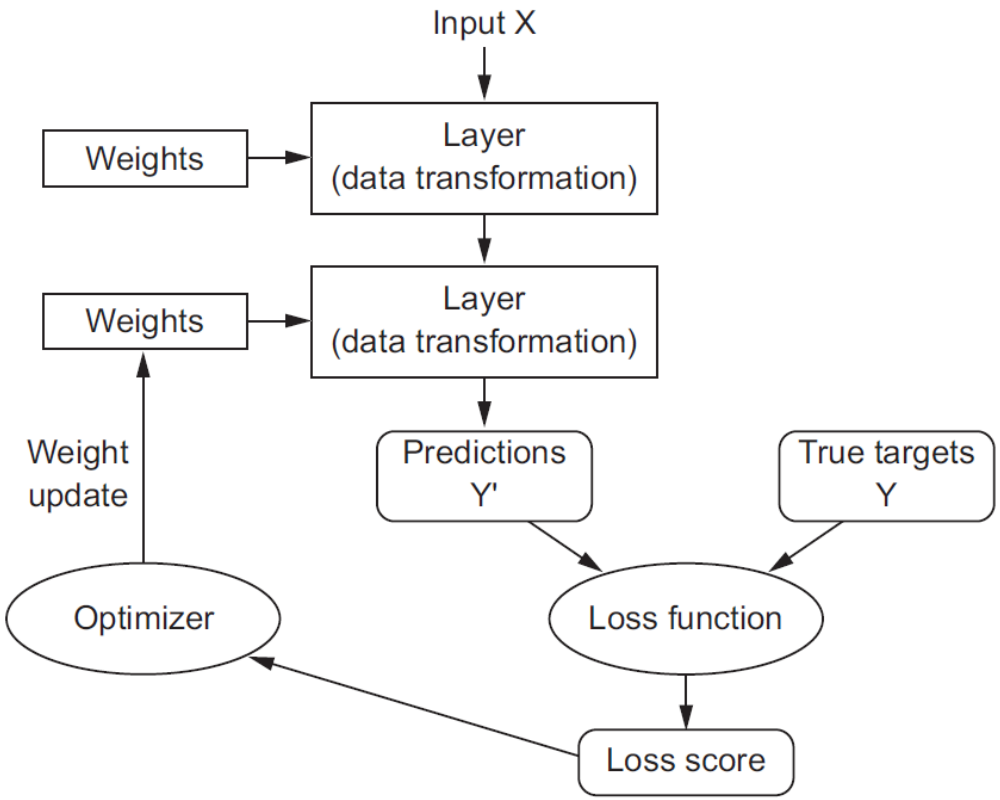


"Neural nets earned a reputation of being finicky and unreliable, which in part caused the community to focus on simpler method with convex loss functions, such as kernel machines and boosting."

"...while local minima are numerous, they are relatively easy to find, and they are all more or less equivalent in terms of performance on the test set..."

YANN LECUN

Summary


Input X
Weights
Layer
(data transformation)
Weights
Layer
(data transformation)
Weight
update
Predictions
Y'
True targets
Y
Optimizer
Loss function
Loss score

Some problems with deep learning models

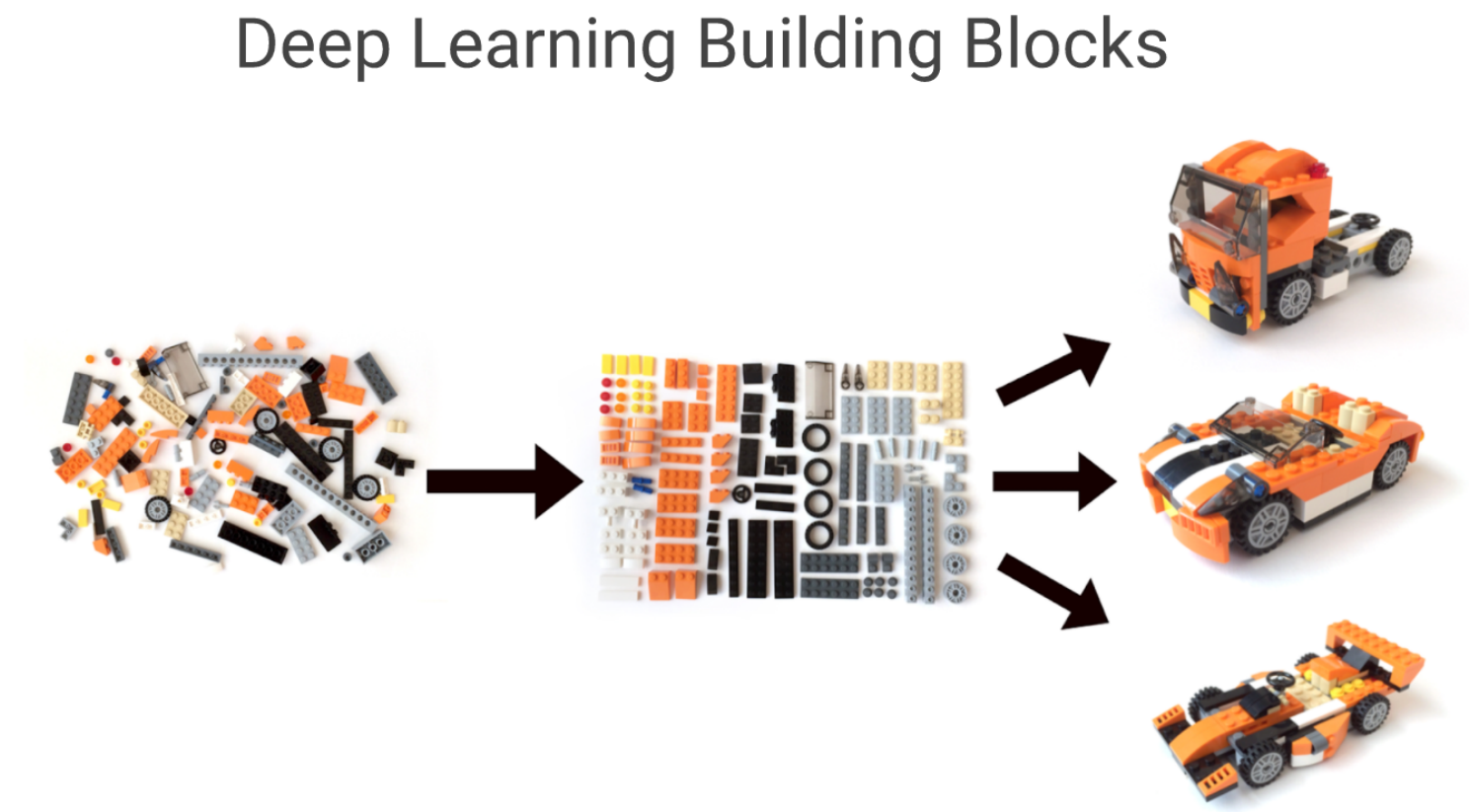
Some limitations...

- Not easily interpreted (not a black box!!).
- Can be brittle (adversarial examples).
- Typically require large amounts of data to perform well.
- Are often very computationally expensive and difficult to train.

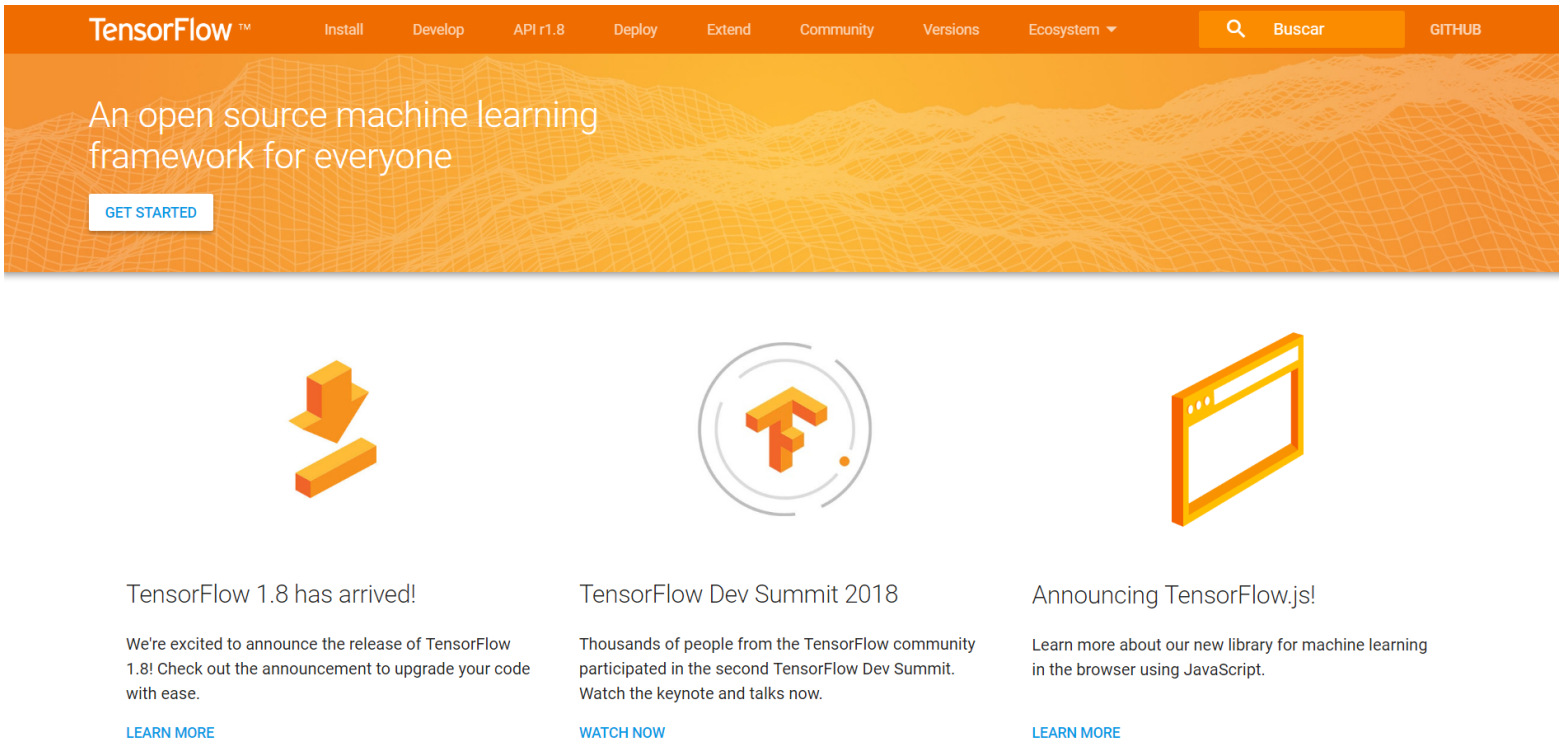
TOOLS & EXAMPLES

Backends for deep learning

Backends for deep learning

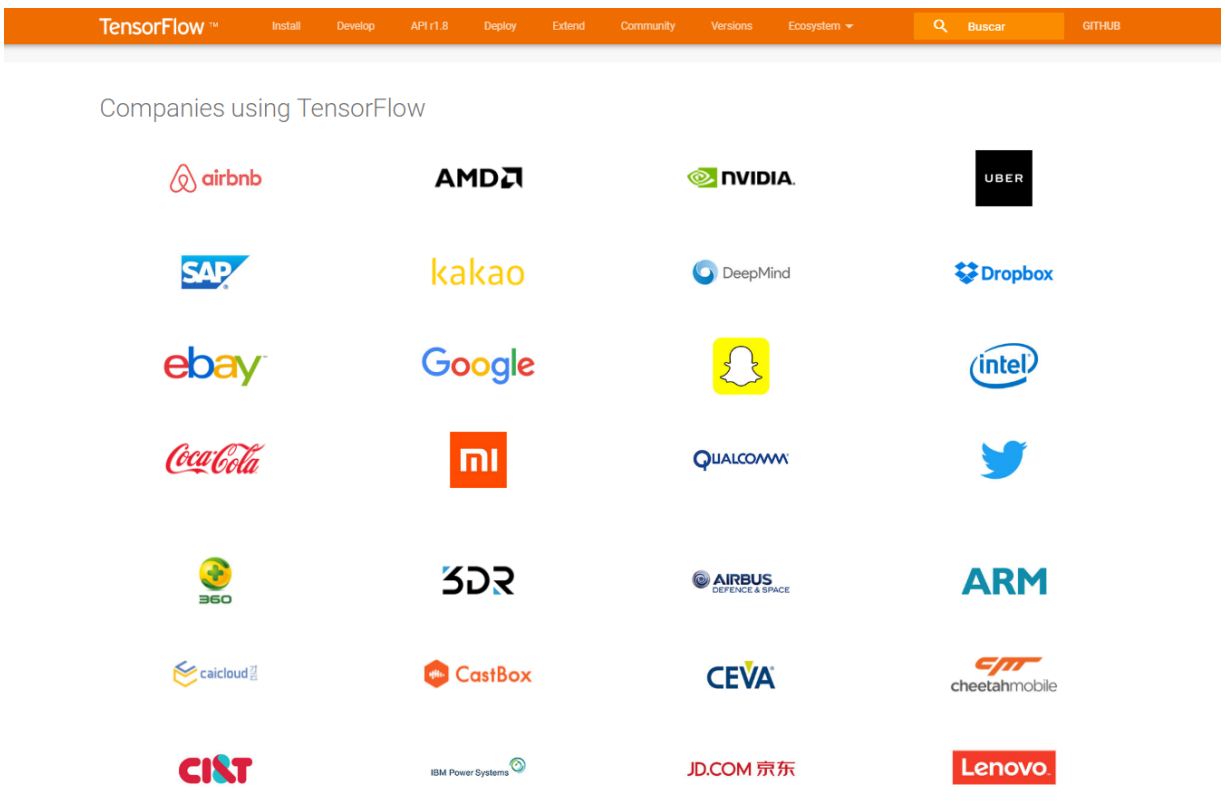


TensorFlow: An open-source machine learning framework for everyone

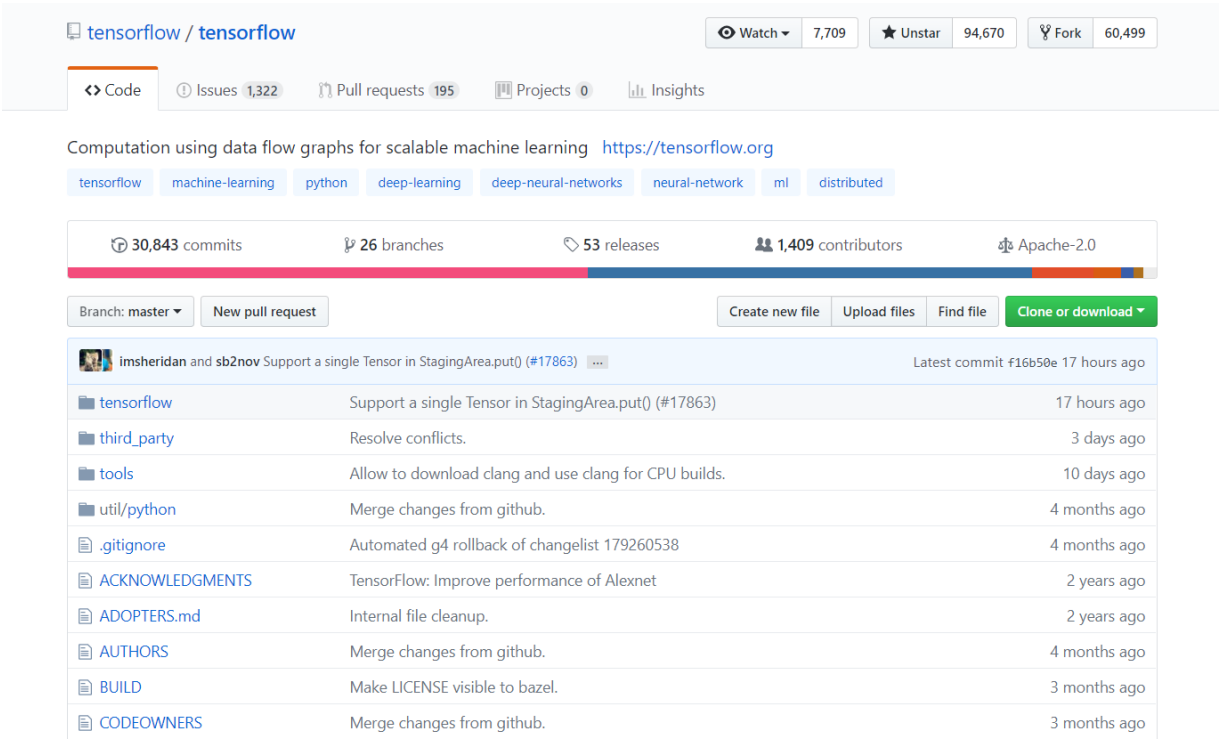


https://www.tensorflow.org/

TensorFlow: An open-source machine learning framework for everyone



TensorFlow: An open-source machine learning framework for everyone



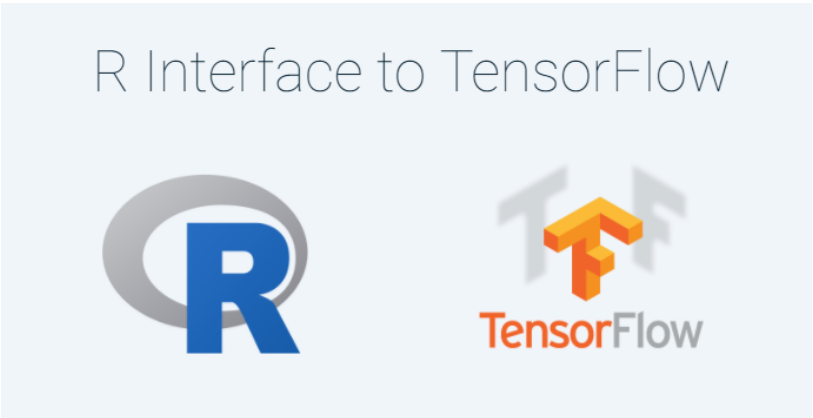
https://github.com/tensorflow/tensorflow

TensorFlow/Keras in GitHub

Top libraries by Github issues opened		
#1:	8370	tensorflow/tensorflow
#2:	5806	fchollet/keras
#3:	4558	dmlc/mxnet
#4:	3908	BVLC/caffe
#5:	2465	Theano/Theano
#6:	2462	baidu/paddle
#7:	2264	deeplearning4j/deeplearning4j
#8:	2124	Microsoft/CNTK
#9:	1601	pytorch/pytorch
#10:	1139	NVIDIA/DIGITS
#11:	1005	pfnet/chainer
#12:	738	caffe2/caffe2
#13:	709	tflearn/tflearn
#14:	664	davisking/dlib
#15:	575	torch/torch7
#16:	488	Lasagne/Lasagne
#17:	469	clab/dynet
#18:	324	NervanaSystems/neon
#19:	47	deepmind/sonnet

Top libraries by Github stars		
#1:	71627	tensorflow/tensorflow
#2:	20489	BVLC/caffe
#3:	20038	fchollet/keras
#4:	12558	Microsoft/CNTK
#5:	11369	dmlc/mxnet
#6:	7712	pytorch/pytorch
#7:	7332	torch/torch7
#8:	7297	deeplearning4j/deeplearning4j
#9:	6981	Theano/Theano
#10:	6767	tflearn/tflearn
#11:	5742	caffe2/caffe2
#12:	5544	baidu/paddle
#13:	5336	deepmind/sonnet
#14:	3242	Lasagne/Lasagne
#15:	3232	NervanaSystems/neon
#16:	2987	pfnet/chainer
#17:	2833	davisking/dlib
#18:	2525	NVIDIA/DIGITS
#19:	1775	clab/dynet

TensorFlow



https://tensorflow.rstudio.com/

Install from github

```
> library("devtools")
> devtools::install_github("rstudio/tensorflow")
> devtools::install_github("rstudio/keras")
```

Tensorflow example

tfSession()

```
> library(tensorflow)
> sess <- tf$Session()
> hello <- tf$constant('Hello, TensorFlow!')
> sess$run(hello)

b'Hello, TensorFlow!'
```

Custom linear regression example

```
> # Create 100 phony x, y data points, y = x * 0.1 + 0.3
> x_data <- runif(100, min=0, max=1)
> y_data <- x_data * 0.1 + 0.3
> # Try to find values for W and b that compute y_data = W * x_data + b
> # (We know that W should be 0.1 and b 0.3, but TensorFlow will
> # figure that out for us.)
> W <- tf$Variable(tf$random_uniform(shape(1L), -1.0, 1.0))
> b <- tf$Variable(tf$zeros(shape(1L)))
> y <- W * x_data + b
> # Minimize the mean squared errors.
> loss <- tf$reduce_mean((y - y_data) ^ 2)
> optimizer <- tf$train$GradientDescentOptimizer(0.5)
> train <- optimizer$minimize(loss)
```

Tensorflow example

tf*Session*()

```
> # Launch the graph and initialize the variables.
> sess = tf$Session()
> sess$run(tf$global_variables_initializer())
> # Fit the line (Learns best fit is W: 0.1, b: 0.3)
> for (step in 1:201) {
+   sess$run(train)
+   if (step %% 20 == 0)
+     cat(step, "-", sess$run(W), sess$run(b), "\n")
+ }

20 - -0.01607878 0.3667336
40 - 0.07722016 0.3130961
60 - 0.09552959 0.30257
80 - 0.09912271 0.3005044
100 - 0.09982784 0.300099
120 - 0.0999662 0.3000194
140 - 0.09999336 0.3000038
160 - 0.0999987 0.3000008
180 - 0.09999975 0.3000001
200 - 0.09999989 0.3000001
```

Keras: User-friendly API which makes it easy to quickly prototype deep learning models

https://keras.rstudio.com/

R interface to Keras

It contains about $\simeq$ 500 functions (v.2.1.6)

Imperative API

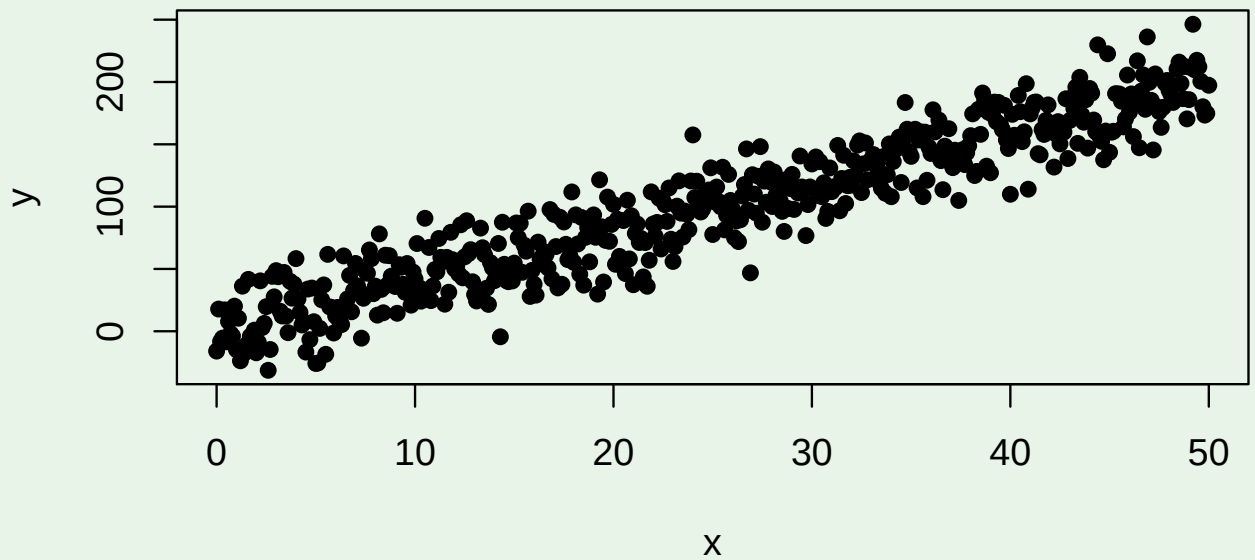
k-eval

```
> library(keras)
> x <- c(1:15)
> prod_ac <- k_cumprod(x, axis = 1)
> k_eval(prod_ac)

 [1]          1          2          6         24        120        720
 [7]       5040      40320     362880    3628800   39916800  479001600
[13] 1932053504 1278945280 2004310016

> sig <- k_sigmoid(c(-3, -2, -1, 0, 1, 2, 3))
> k_eval(sig)

[1] 0.04742587 0.11920292 0.26894143 0.50000000 0.73105860 0.88079703 0.95257413

> k_eval(k_random_normal(c(4, 6), mean = 0, stddev = 1))

          [,1]        [,2]       [,3]       [,4]        [,5]        [,6]
[1,] -1.212058  0.08014527 -1.3834615  0.9868450 -0.93238080  0.94287306
[2,]  1.536298 -0.68395638 -0.3206587 -0.2576625 -1.44378328  1.15996063
[3,]  1.337245 -1.34764791 -0.4571670 -0.6830830  0.01222763 -0.07862066
[4,] -0.198133  1.04633021 -0.6374950 -0.2465717 -0.35541692  0.33105391
```

Data

simulated data

```
> x <- seq(0, 50, by = 0.1)
> y <- 4*x + rnorm(n = length(x), 0, 20)
> plot(x, y, pch=16)
```



Model definition: Linear Regression

- In Keras, we use its own data type `keras-model-sequential()` to configure the network.
- Without hidden layers configuration.
- For the ouput, we set the activation to linear regression to get the value prediction.

keras-model-sequential

```
> linear_regression <- keras_model_sequential() %>%
+   layer_dense(units = 1, activation = 'linear', input_shape = c(1))
```

Model training

summary

```
> summary(linear_regression)

_________________________________________________________________________
Layer (type)                 Output Shape              Param #
=========================================================================
dense_1 (Dense)              (None, 1)                 2
=========================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________________
```

And the parameters learned...

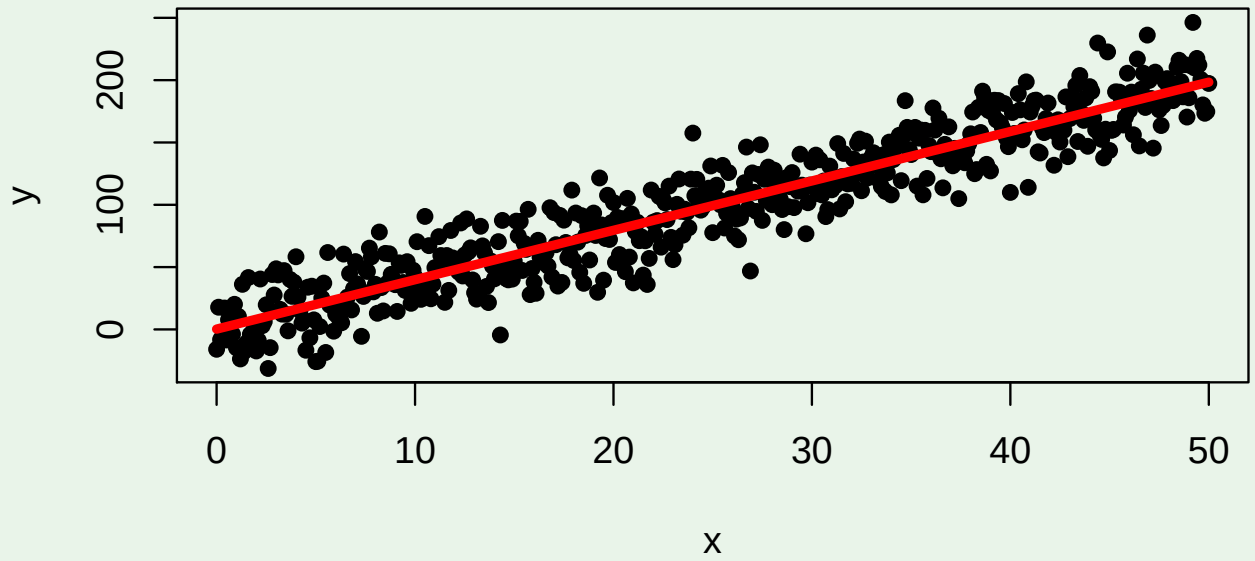
Model complie and training

```
> linear_regression %>% compile(loss = 'mean_squared_error', optimizer = optimizer_rmsprop())
> # Fit model to data
> history <- linear_regression %>% fit(x, y, batch_size = 254, epochs = 5000, verbose = 0)
> unlist(get_weights(linear_regression))

[1] 3.9628351 0.2595091
```

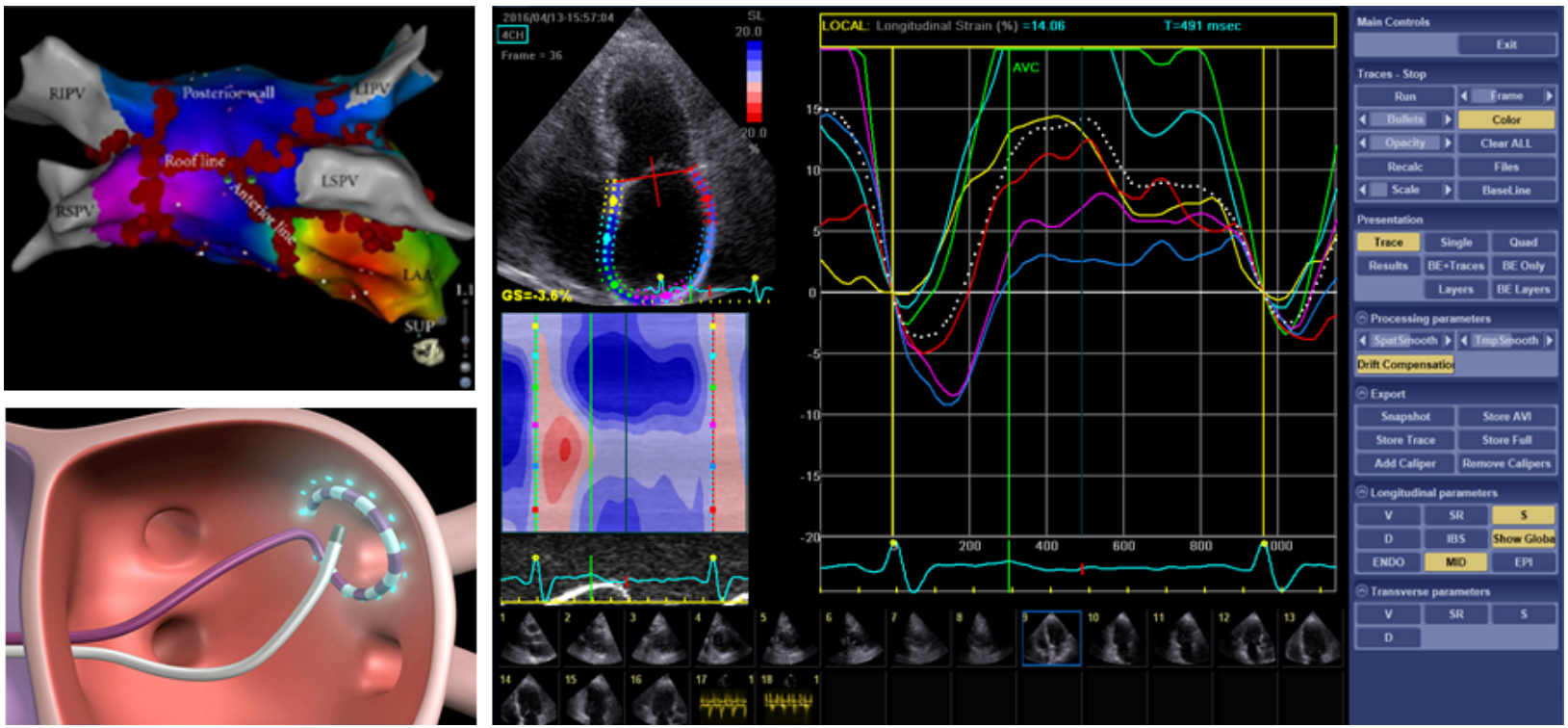
Prediction

predict

```
> #Model predictions
> prediction <- linear_regression %>% predict(x)
> plot(x, y, pch=16)
> points(x, prediction, type='l', col='red', lwd=4)
```



Paper STRAIN



Atrial Fibrillation ablation and Strain variables

Model architecture: Logistic Regression

- Without hidden layers configuration.
- For the ouput, we set the activation to softmax to get a probabilistic prediction for each category.

layer-dense

```
> logistic_regression <- keras_model_sequential() %>%
+   layer_dense(units = 1, activation = 'sigmoid', input_shape = c(21))
```

Model training

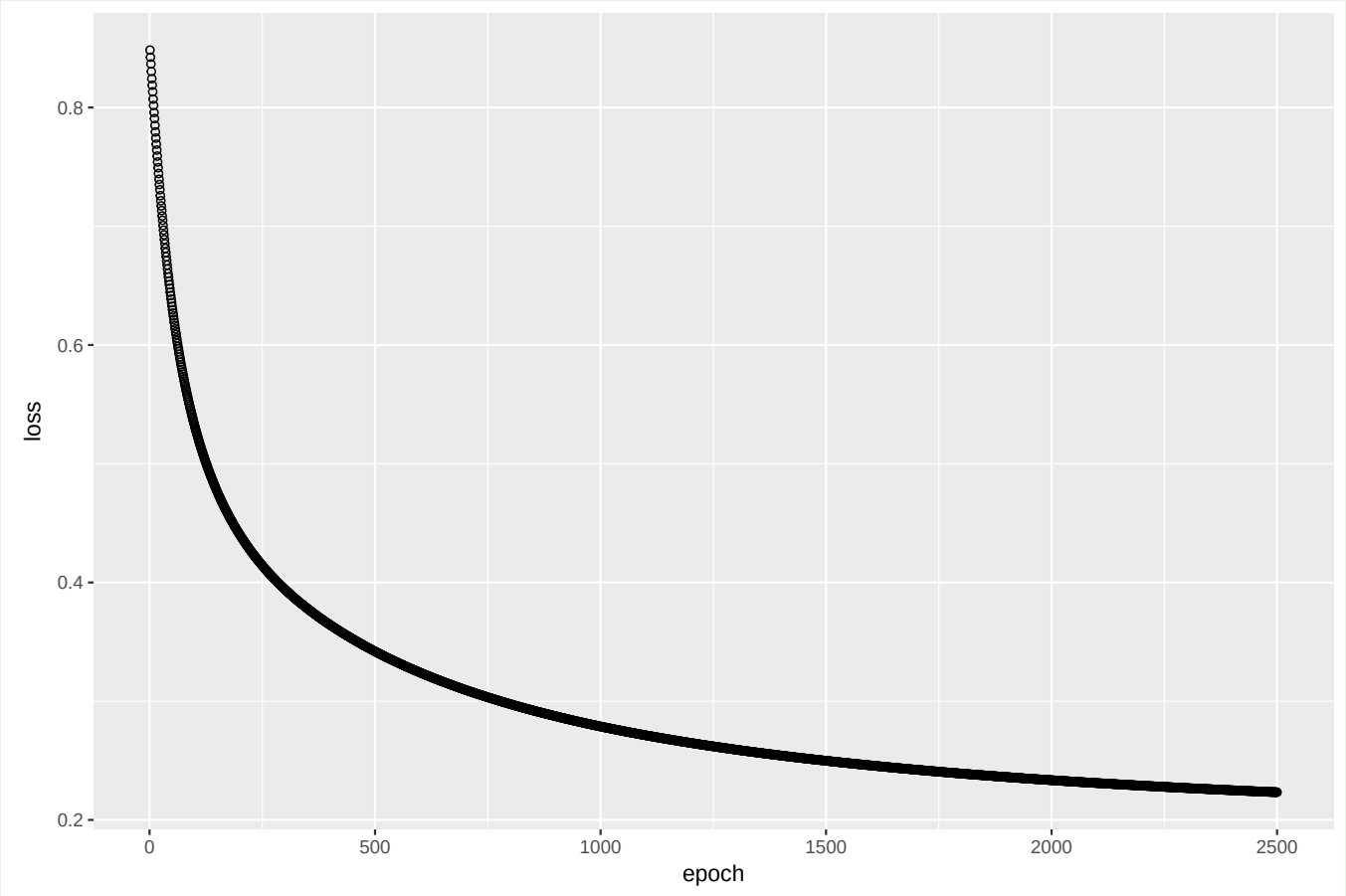
We train the model over 2500 epochs.

Model prediction

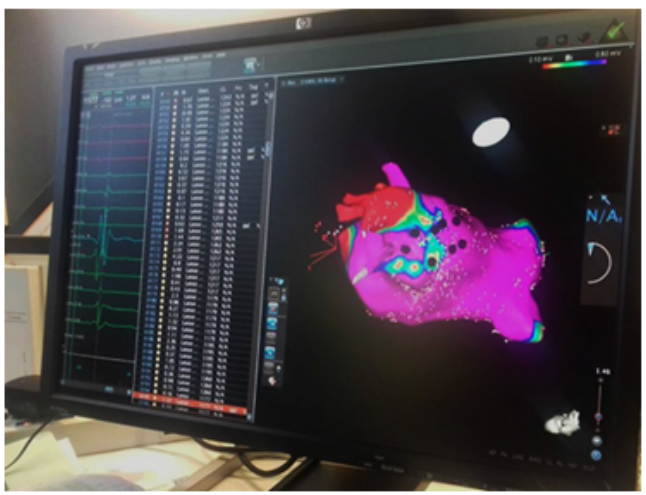
```
> logistic_regression %>% compile(loss = 'binary_crossentropy', optimizer = optimizer_adam())
> # Fit model to data
> history <- logistic_regression %>% fit(x, y, batch_size = 64, epochs = 2500, verbose = 0)
> score <- logistic_regression %>% evaluate(x, y, verbose = 0)
> score

     loss
0.2233094

> class_prediction <- logistic_regression %>% predict_classes(x)
> confusion_matrix <- table(class_prediction, y)
> sum(diag(confusion_matrix))/sum(confusion_matrix)*100

[1] 91.15044
```

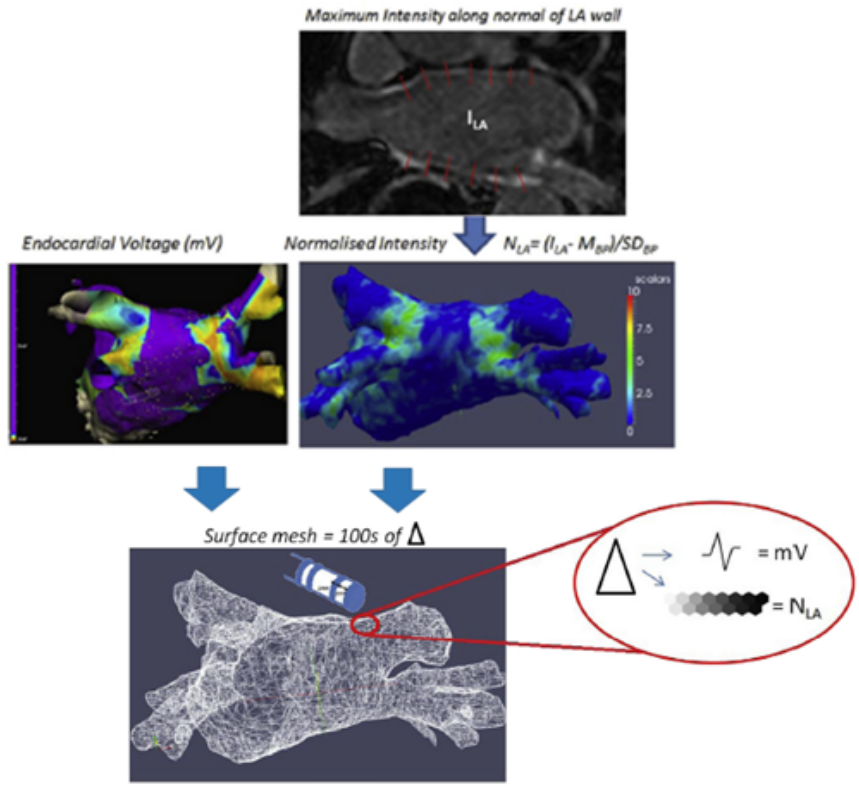
Monitoring the error

Training curve

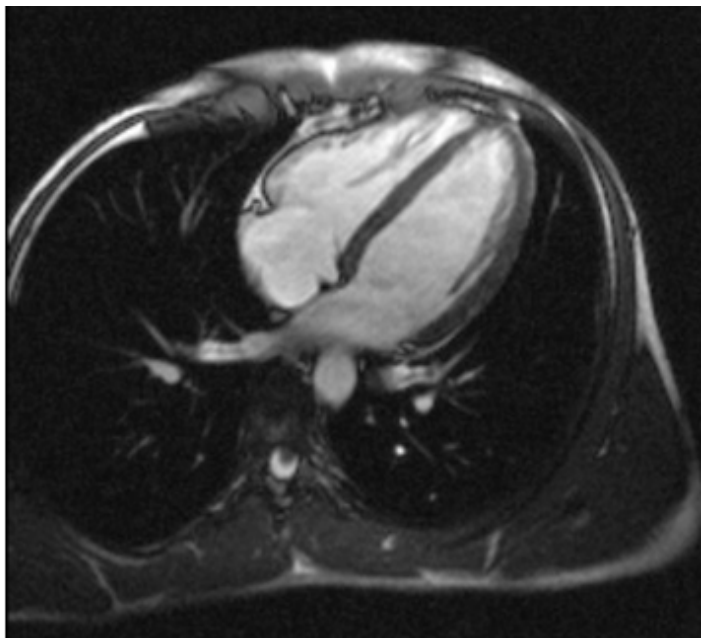


RMN to IIR prediction



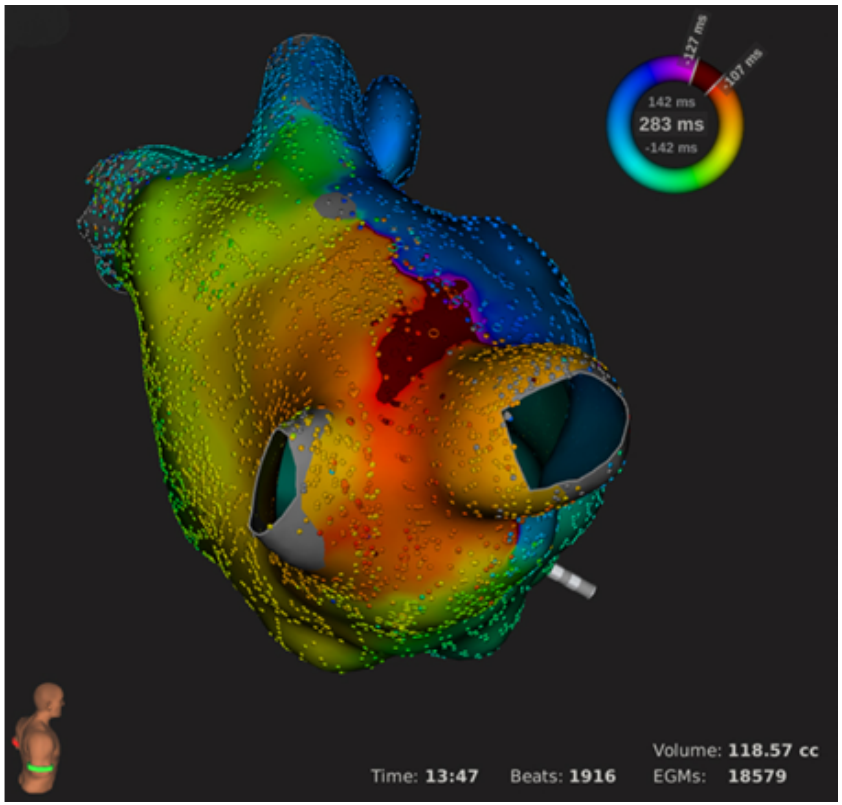


RMN to IIR prediction



Cardiac magnetic resonance image

RMN to IIR prediction



An electroanatomical mapping system is a useful tool for complex arrhythmia interventions.

RMN to IIR prediction

- Model exploration is mandatory...
- Linear models not always the best choice, voltage negative predictions for this problem!
- We need hidden layers configuration.

NNET model architecture

```
> nnet_rmn_iir <- keras_model_sequential()
> nnet_rmn_iir %>%
+   layer_dense(units = 128, input_shape = c(27)) %>%
+   layer_activation('relu') %>%
+   layer_dense(units = 64) %>%
+   layer_activation('relu') %>%
+   layer_dense(units = 1)
```

Computation graph

summary

```
> summary(nnet_rmn_iir)

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               3584
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
activation_2 (Activation)    (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 11,905
Trainable params: 11,905
Non-trainable params: 0
_________________________________________________________________
```

After 10,000 iterations with a few thousand parameters, the model gets 0.72 mv of *MAE* in the test set.

Real-world examples of data tensors

- *Vector data*—2D tensors of shape `(samples, features)`
- *Timeseries data or sequence data*—3D tensors of shape `(samples, timesteps, features)`
- *Images*—4D tensors of shape `(samples, height, width, channels)` or `(samples, channels, height, width)`
- *Video*—5D tensors of shape `(samples, frames, height, width, channels)` or `(samples, frames, channels, height, width)`

Kaggle competition

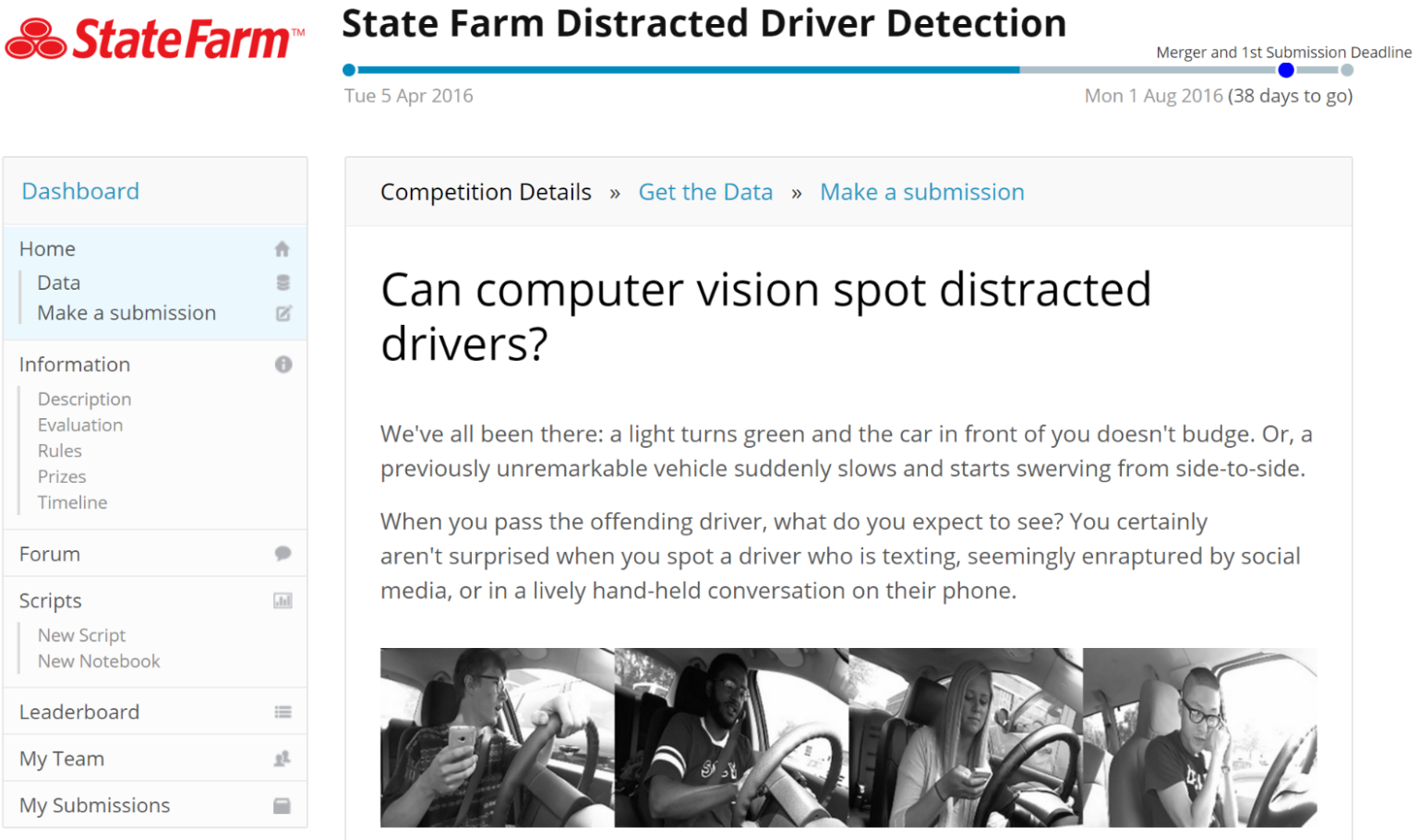


Image example

readImage

```
> library("EBImage")
> image_c0 <- readImage("c0_img_2093.jpg")
> display(image_c0, method = "raster")
```

Image example

readImage

```
> image_c2 <- readImage("c2_img_13859.jpg")
> display(image_c2, method = "raster")
```

Data

data.matrix

```
> Driver_Detection_sample[1:4, 1:7]

  target        X1        X2        X3        X4        X5        X6
1      3 0.1200282 0.1474765 0.1778427 0.1883918 0.1921196 0.2078043
2      0 0.1572761 0.1533549 0.1611973 0.1611973 0.1729608 0.1729608
3      1 0.1500741 0.1293384 0.1293384 0.1214961 0.1336643 0.1532702
4      6 0.1949529 0.1650337 0.1689125 0.1598137 0.1650761 0.1716616

> dim(Driver_Detection_sample)

[1] 5000 1025

> sample_index <- sample(1:nrow(Driver_Detection_sample), size=nrow(Driver_Detection_sample)*0.8)
> train <- data.matrix(Driver_Detection_sample[sample_index,])
> test <- data.matrix(Driver_Detection_sample[-sample_index,])
> x_train <- train[,-1]; y_train <- train[,1]
> x_test <- test[,-1]; y_test <- test[,1]
```

Data manipulation with Keras

array-reshape

```
> img_rows <- 32
> img_cols <- 32
> batch_size <- 128
> num_classes <- 10
> # Redefine  dimension of train/test inputs
> x_train <- array_reshape(x_train, c(nrow(x_train), img_rows, img_cols, 1))
> x_test <- array_reshape(x_test, c(nrow(x_test), img_rows, img_cols, 1))
> # Input image dimensions
> input_shape <- c(img_rows, img_cols, 1)
> y_train <- to_categorical(y_train, num_classes)
> y_test <- to_categorical(y_test, num_classes)
```

CNN definition

Convolutional network architecture designed for images.

CNN model definition

```
> model <- keras_model_sequential()
> model %>% layer_conv_2d(filters = 32, kernel_size = c(3,3),
+                 activation = 'relu', input_shape = input_shape) %>%
+   layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
+   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
+   layer_flatten() %>%
+   layer_dense(units = 128, activation = 'relu') %>%
+   layer_dense(units = num_classes, activation = 'softmax')
```

Compile and Train...

Fit the model

```
> model %>% compile(loss = loss_categorical_crossentropy,
+                   optimizer = optimizer_adadelta(), metrics = c('accuracy'))
> model %>% fit(x_train, y_train, batch_size = batch_size,
+               epochs = 15, verbose = 1, validation_data = list(x_test, y_test))
```

Model parameters

summary model

```
> summary(model)

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 14, 14, 64)     0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               1605760
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,625,866
Trainable params: 1,625,866
Non-trainable params: 0
_________________________________________________________________
```

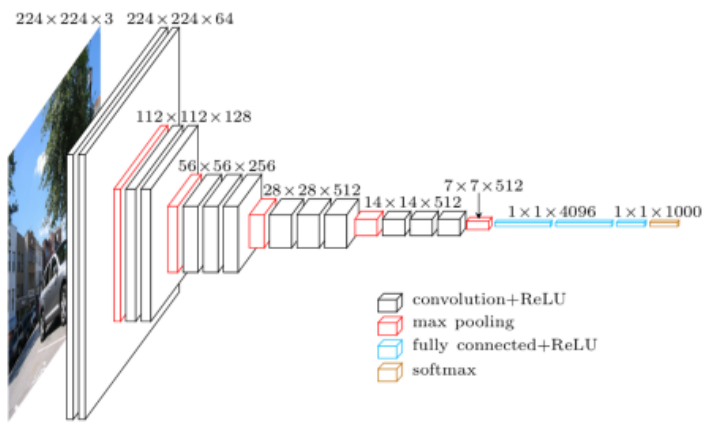
Model performance

predict

```
> scores <- model %>% evaluate(x_test, y_test, verbose = 1)
> scores

$loss
[1] 0.1115962

$acc
[1] 0.969

> predicted_class <- model %>% predict_classes(x_test)
> confusion_matrix <- table(predicted_class, test[,1])
> confusion_matrix

predicted_class   0   1   2   3   4   5   6   7   8   9
              0  99   1   0   2   0   0   0   0   1   2
              1   0  97   0   0   0   0   0   0   1   0
              2   2   0 101   1   0   1   3   0   0   0
              3   0   0   0  86   1   0   0   0   1   0
              4   1   0   0   0 109   0   0   2   0   2
              5   0   0   0   0   0  84   0   0   0   0
              6   0   1   1   0   0   1  92   1   0   0
              7   1   0   0   0   0   0   0 108   0   0
              8   0   0   0   0   0   1   0   0  90   1
              9   0   1   0   0   1   0   0   0   1 103
```

Classify images with a pretrained model

- You can use a pre-trained **VGG** network to predict the class of an image.
- **VGG** network gives the recent state-of-art prediction accuracy on **IMAGENET** dataset.
- Model with 143,667,240 parameters.



load.image

```
> library(jpeg)
> img_path <- 'D:/SESSIO_DEEP_LEARNING/BEAMER/elephant.jpg'
> ### load the image and display it in R
> jj <- readJPEG(img_path)
> plot(0:1, 0:1, type ="n", ann=FALSE, axes=FALSE)
> rasterImage(jj, 0, 0, 1, 1)
```

Classify images with a pretrained model

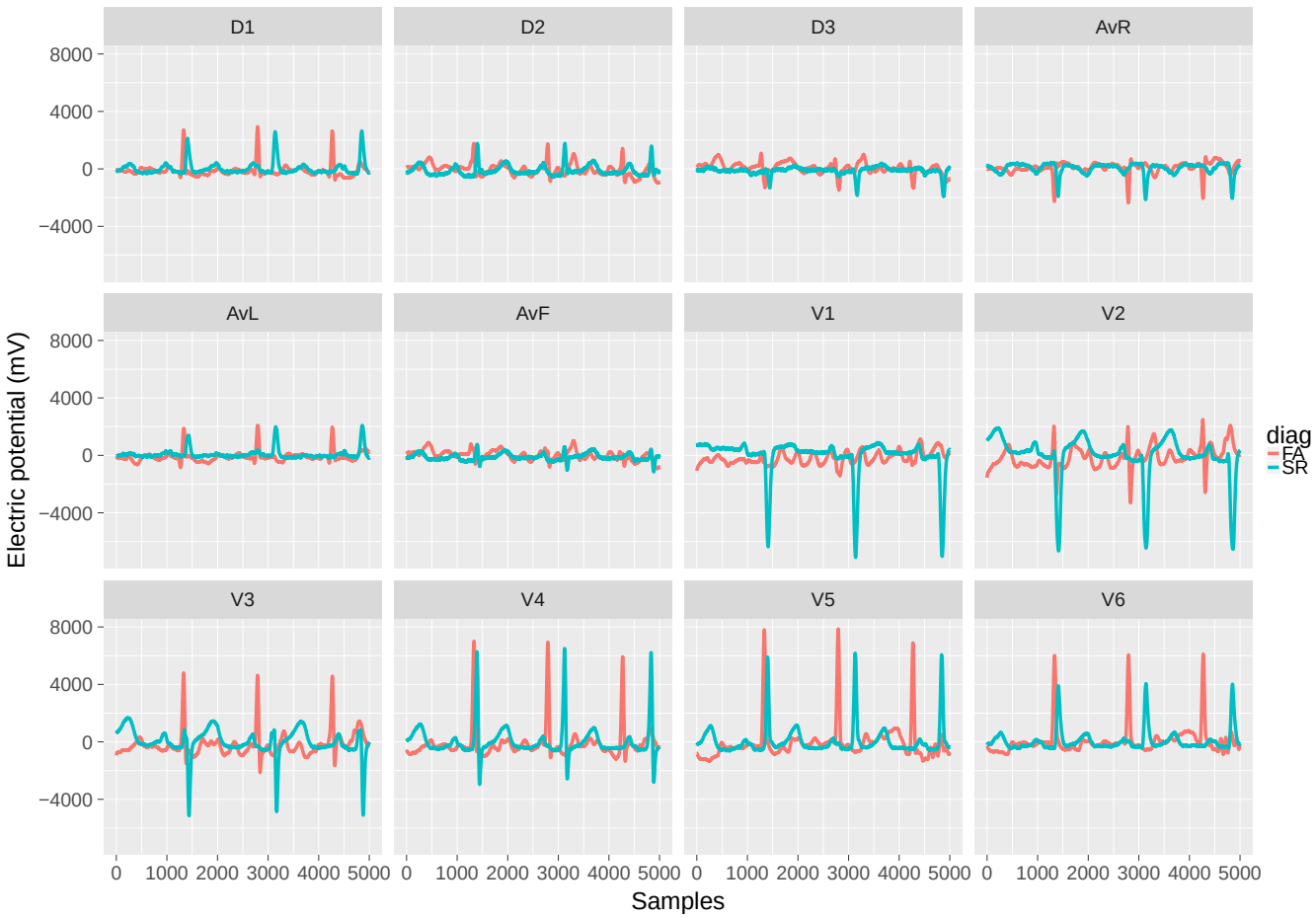
VGG prediction

```
> img <- image_load(img_path, target_size = c(224, 224))
> x <- image_to_array(img)
> ## ensure we have 4d tensor with single element in the batch dimensions,
> ## the preprocess the input for prediction
> dim(x) <- c(1, dim(x))
> x <- imagenet_preprocess_input(x)
> modelvgg19 <- application_vgg19(weights = 'imagenet')
> pred_vgg19 <- modelvgg19 %>% predict(x)
> ppvgg <- imagenet_decode_predictions(pred_vgg19, top =10)[[1]]
> ppvgg[1:5,]

  class_name class_description       score
1  n01871265            tusker 0.4945689142
2  n02504458  African_elephant 0.4688400328
3  n02504013   Indian_elephant 0.0330304392
4  n02412080               ram 0.0007902890
5  n02454379         armadillo 0.0006282972
```

It's a tusker!! :)

ECG monitoring example

ECG monitoring example



ECG monitoring example

layer-lstm

```
> model_ecg <- keras_model_sequential() %>%
+   layer_lstm(units = 32, return_sequences = TRUE, input_shape = c(1, 12)) %>%
+   layer_lstm(units = 32, return_sequences = TRUE) %>%
+   layer_lstm(units = 32) %>%
+   layer_dense(units = 1, activation = "sigmoid")
```

Accuracy 94% in the validation set.

IMDB database

Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative).

IMDB database

Data to dictionary

```
> max_features <- 2000
> max_len <- 500
> imdb <- dataset_imdb(num_words = max_features)
> imdb$train$x[[10]]

  [1]    1   14   20   47  111  439    2   19   12   15  166   12  216  125   40
 [16]    6  364  352  707 1187   39  294   11   22  396   13   28    8  202   12
 [31] 1109   23   94    2  151  111  211  469    4   20   13  258  546 1104    2
 [46]   12   16   38   78   33  211   15   12   16    2   63   93   12    6  253
 [61]  106   10   10   48  335  267   18    6  364 1242 1179   20   19    6 1009
 [76]    7 1987  189    5    6    2    7    2    2   95 1719    6    2    7    2
 [91]    2   49  369  120    5   28   49  253   10   10   13 1041   19   85  795
[106]   15    4  481    9   55   78  807    9  375    8 1167    8  794   76    7
[121]    4   58    5    4  816    9  243    7   43   50
```

Pad sequences

```
> c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb
> x_train <- pad_sequences(x_train, maxlen = max_len)
> x_test <- pad_sequences(x_test, maxlen = max_len)
```

IMDB database

Model

```
> model <- keras_model_sequential() %>%
+   layer_embedding(input_dim = max_features, output_dim = 128,
+                   input_length = max_len, name = "embed") %>%
+   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, activation = "sigmoid")
> model %>% compile(optimizer = "rmsprop", loss = "binary_crossentropy", metrics = c("acc"))
```

Accuracy 86% in the test set.

Advanced features

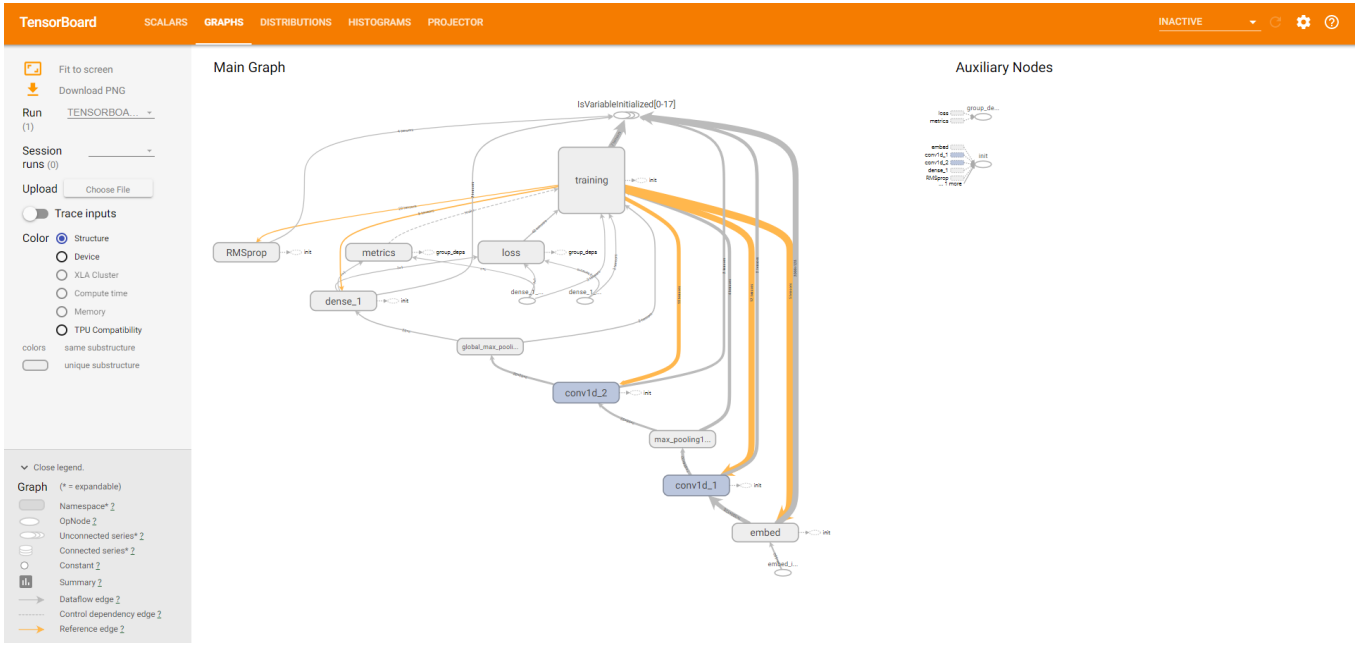
Advanced features in Keras

- Tensorboard visualization
- Deep Art
- Multitasking learning
- Model to production and google cloud integration
- Generative Adversarial Learning, Reinforcement learning (alpha Go), Transfer Learning k
- And many others to have fun!... :)

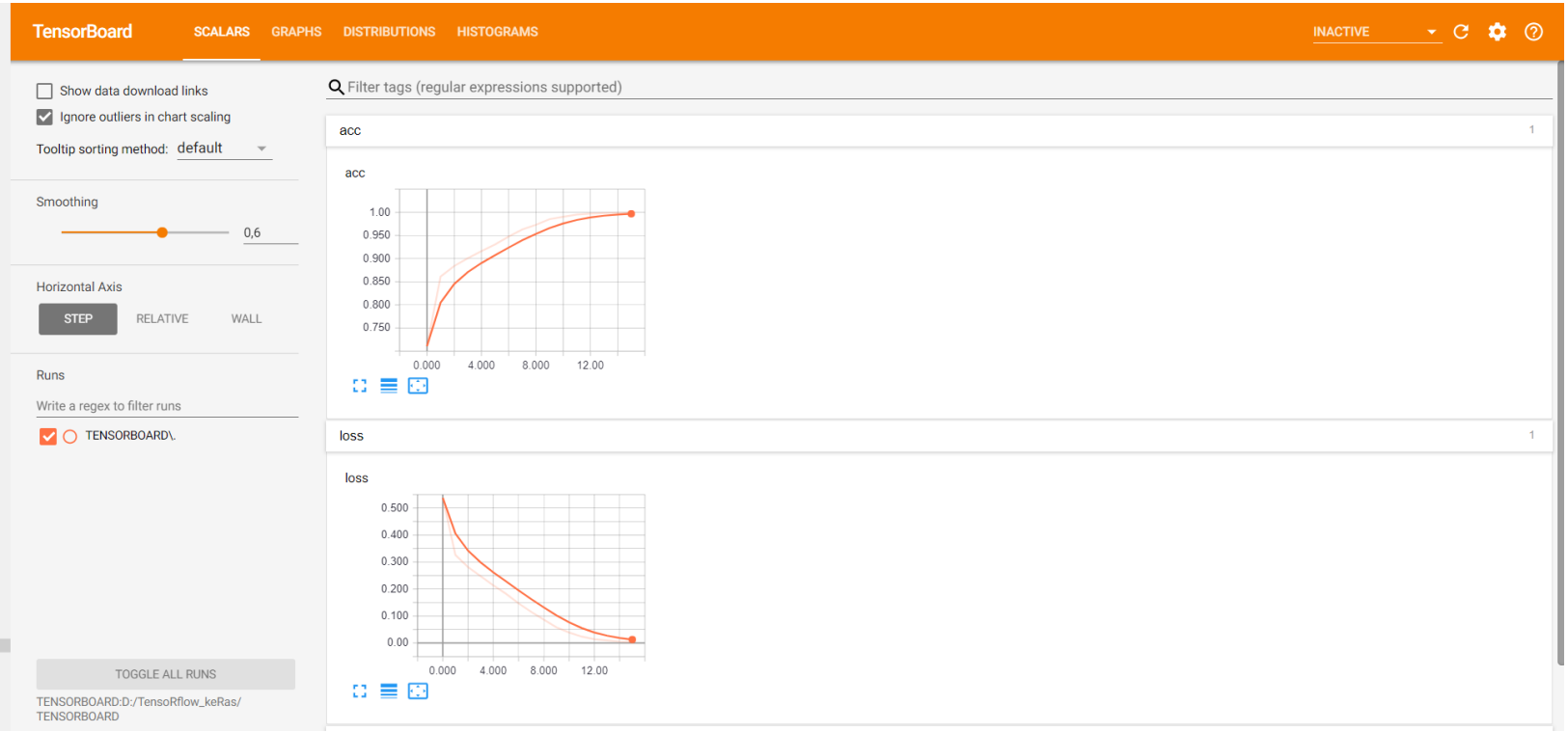
TENSORBOARD visualization

Model visualization

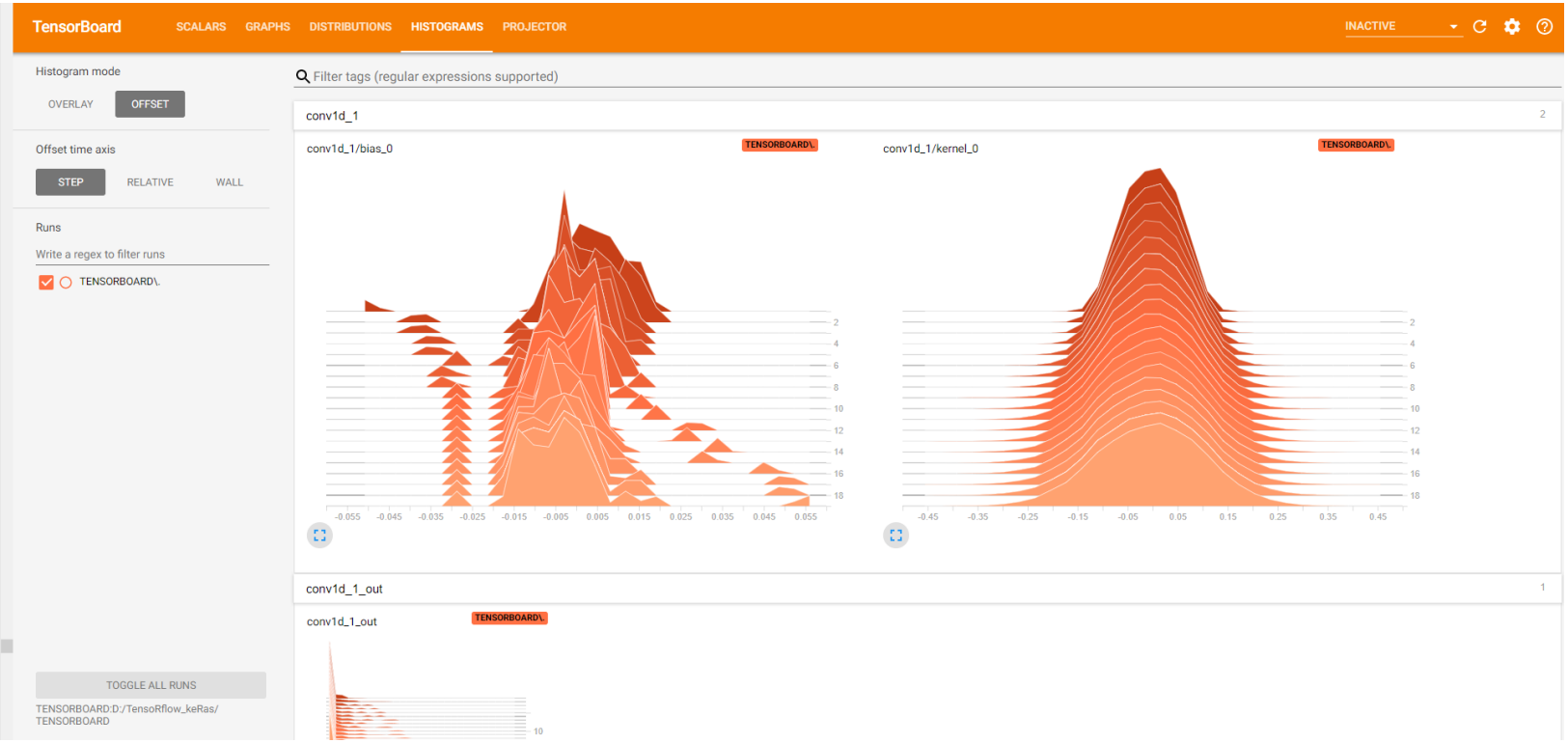
```
> callbacks <- list(callback_tensorboard(log_dir =
+                     "D:/TensoRflow_keRas/TENSORBOARD", histogram_freq = 1, embeddings_freq = 1))
> history <- model %>% fit(x_train, y_train, epochs = 10, batch_size = 128,
+                          validation_split = 0.2, callbacks = callbacks)
> tensorboard("D:/TensoRflow_keRas/TENSORBOARD")
```



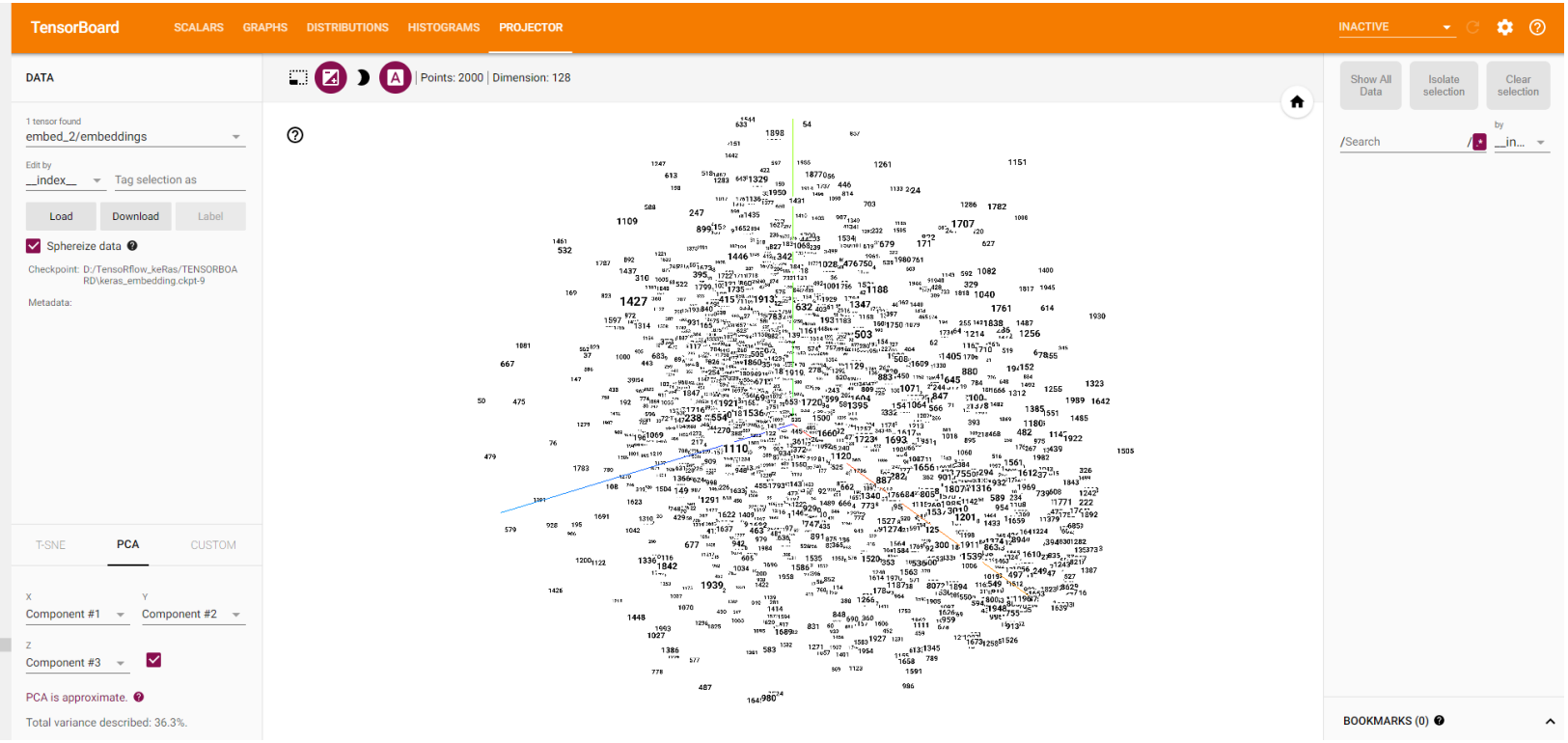
TENSORBOARD visualization



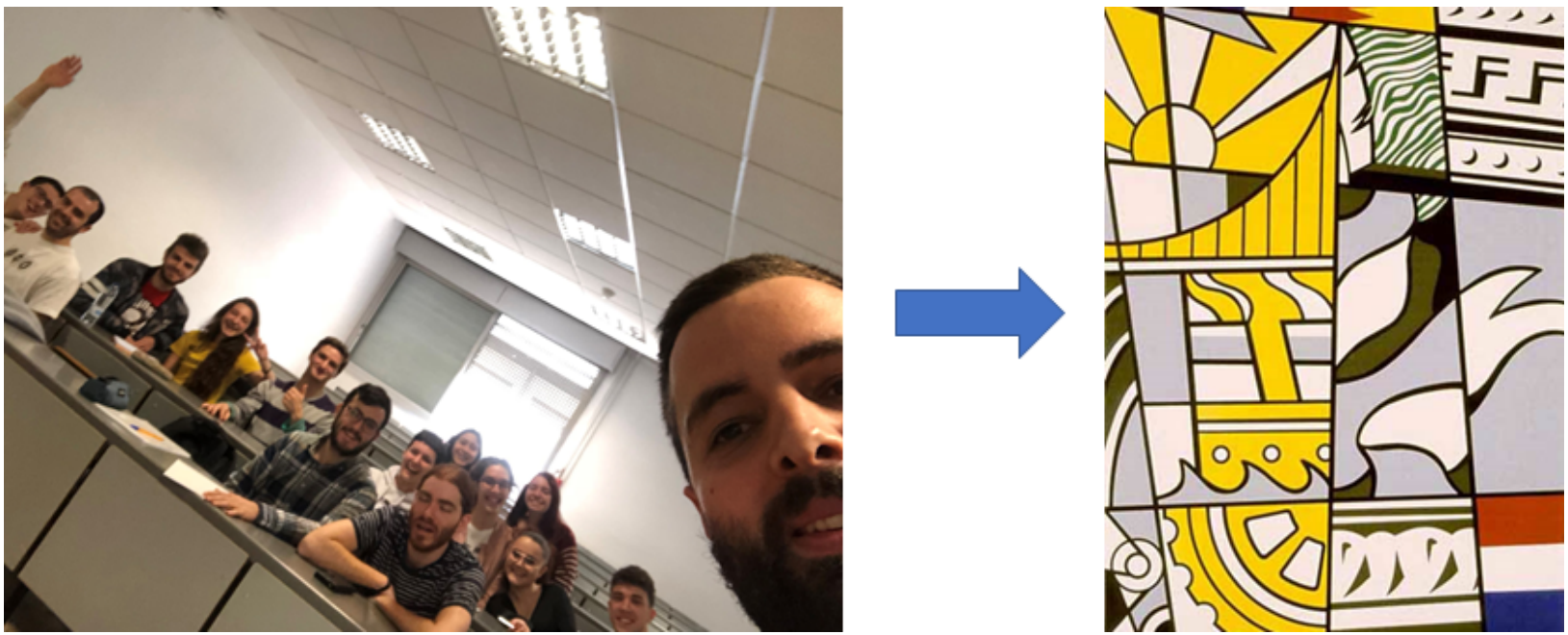
TENSORBOARD visualization



TENSORBOARD visualization



Neural Style Transfer



Neural Style Transfer in R

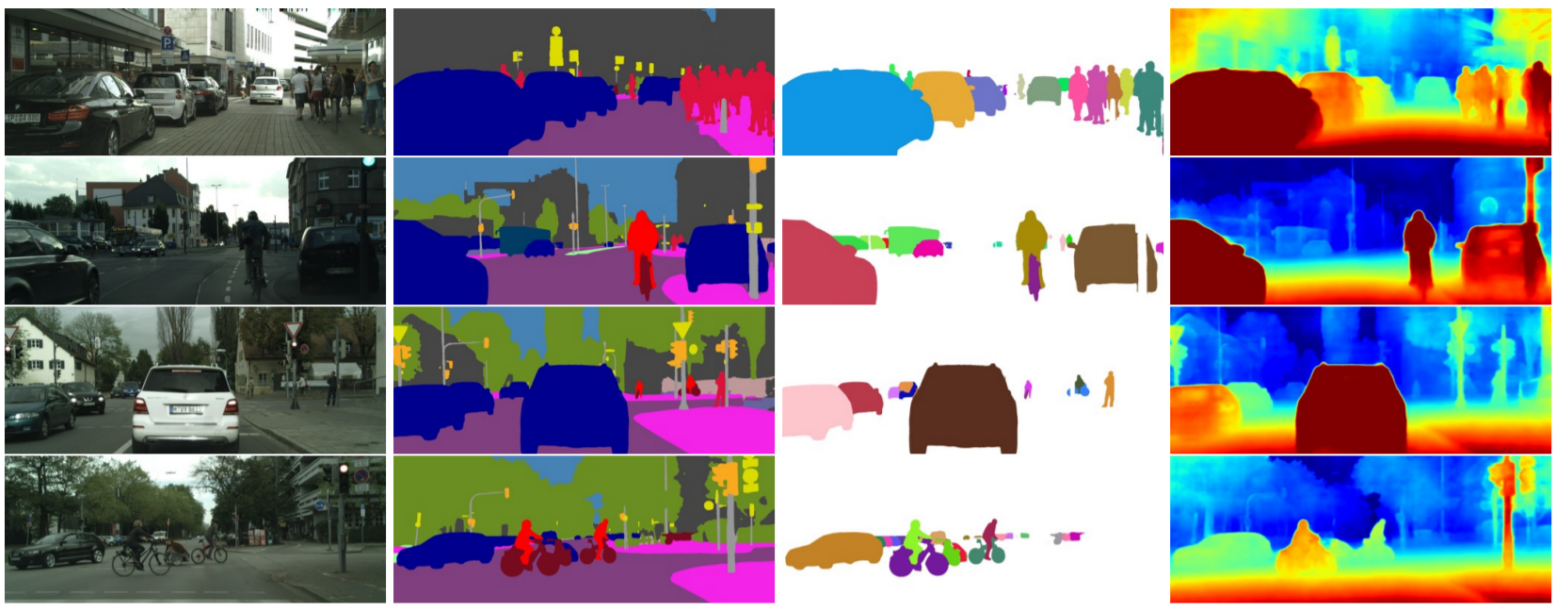
Neural Style Transfer

Model iteration

```
> for (i in 1:iterations) {
+
+   # Runs L-BFGS over the pixels of the generated image to minimize the neural style loss.
+   opt <- optim(
+     array_reshape(x, dim = length(x)),
+     fn = evaluator$loss,
+     gr = evaluator$grads,
+     method = "L-BFGS-B",
+     control = list(maxit = 15)
+   )
+
+   cat("Loss:", opt$value, "\n")
+
+   image <- x <- opt$par
+   image <- array_reshape(image, dms)
+
+   im <- deprocess_image(image)
+   plot(as.raster(im))
+ }
```

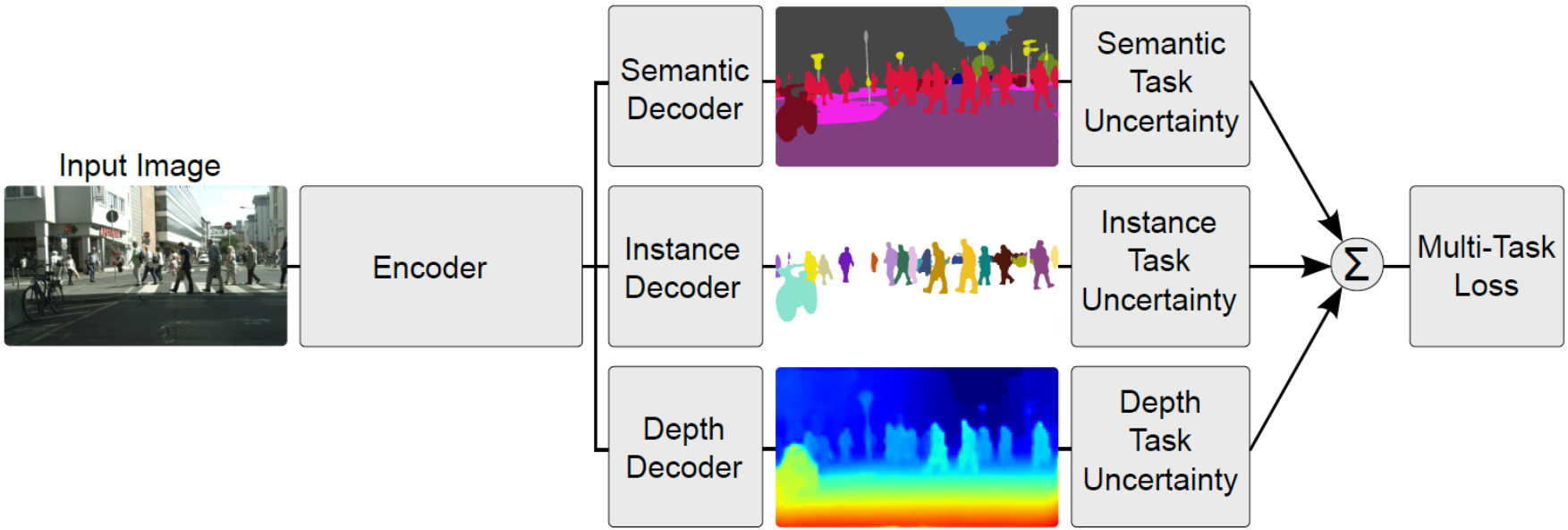
Neural Style Transfer

Multi-task learning

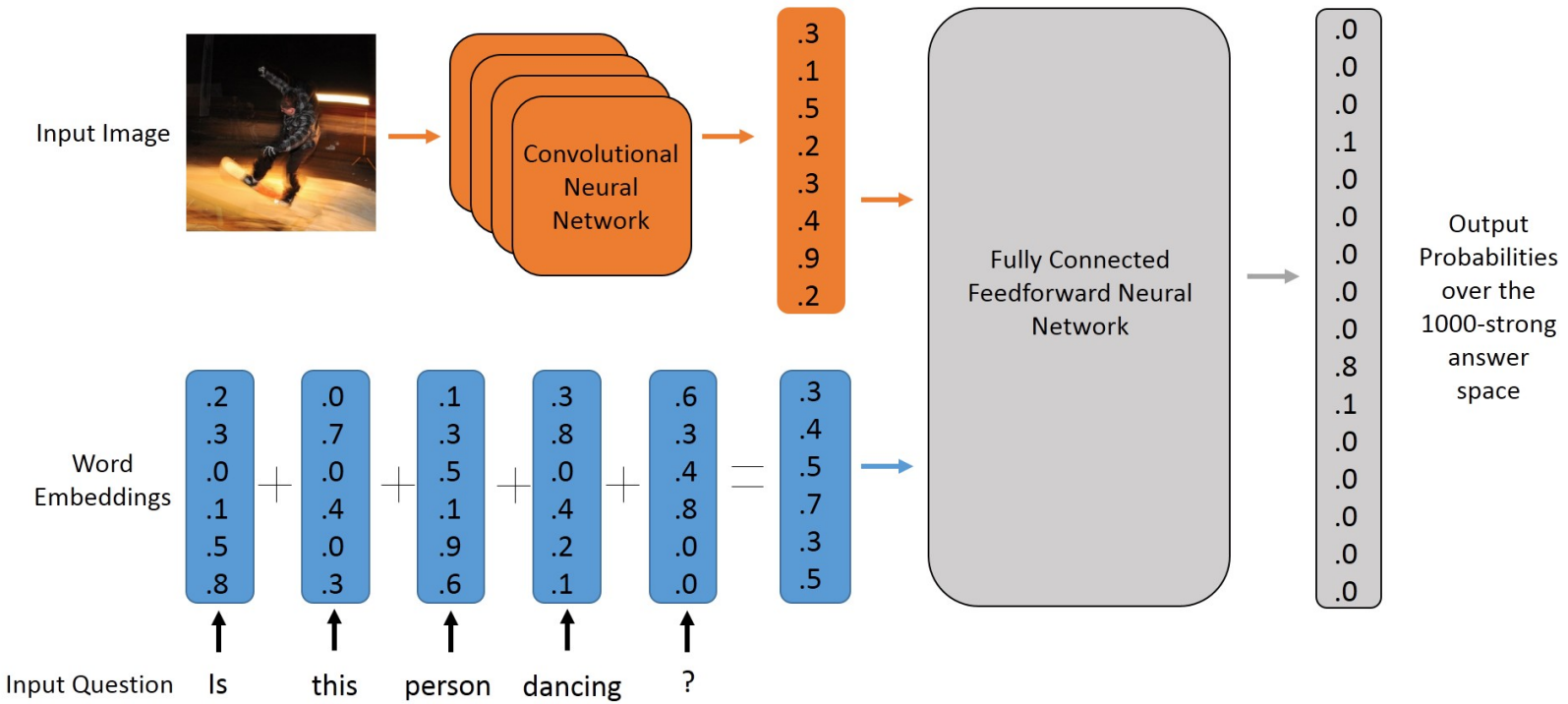


(a) Input image (b) Segmentation output (c) Instance output (d) Depth output

Multi-task learning



Multi-task learning



Multi-task learning

Functional API implementation of a two-input question-answering model

Embeds the inputs into a sequence of vectors of size 64

The text input is a variable-length sequence of integers. Note that you can optionally name the inputs.

Same process (with different layer instances) for the question

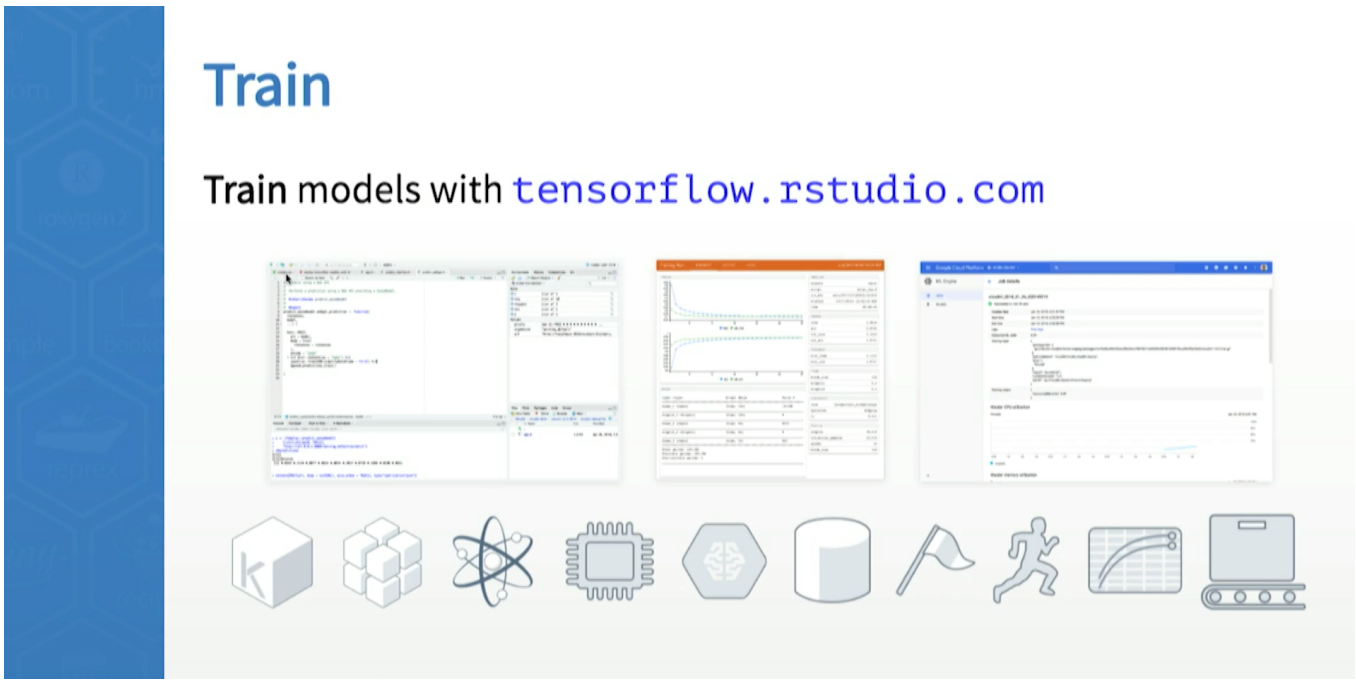
Encodes the vectors in a single vector via an LSTM

```
library(keras)

text_vocabulary_size <- 10000
ques_vocabulary_size <- 10000
answer_vocabulary_size <- 500

text_input <- layer_input(shape = list(NULL),
                          dtype = "int32", name = "text")

encoded_text <- text_input %>%
  layer_embedding(input_dim = 64, output_dim = text_vocabulary_size) %>%
  layer_lstm(units = 32)

question_input <- layer_input(shape = list(NULL),
                              dtype = "int32", name = "question")

encoded_question <- question_input %>%
  layer_embedding(input_dim = 32, output_dim = ques_vocabulary_size) %>%
  layer_lstm(units = 16)

concatenated <- layer_concatenate(list(encoded_text, encoded_question))
```

Other libraries in the ecosystem

Install from github

```
> devtools::install_github("rstudio/cloudml")
> devtools::install_github("rstudio/tfdeploy")
> devtools::install_github("rstudio/tfestimators")
```



CONCLUSIONS

Final thought

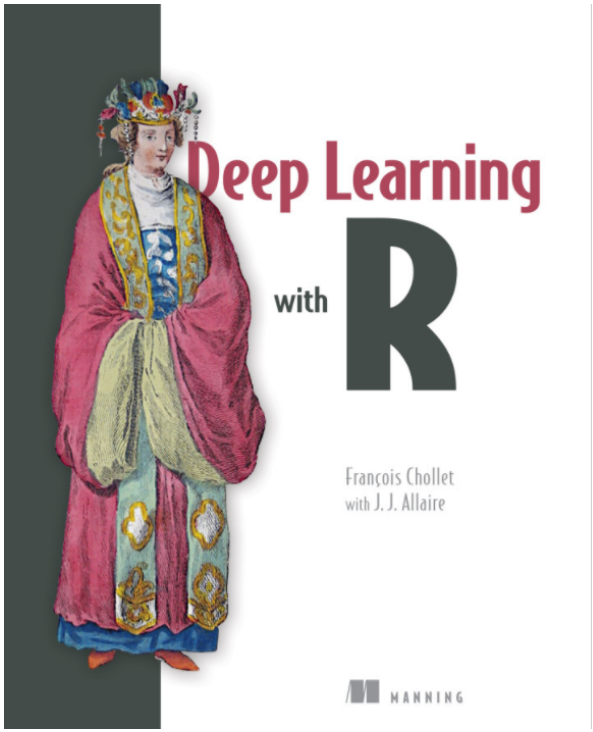
- Deep learning has made great progress and will likely increase in importance in various fields in the coming years.
- Learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations with diferent types of model architectures (FFNN, CNN, RNN).
- The fundamental trick in deep learning is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the current batch of examples with SGD.
- Is not very difficult to implement nice and useful deep learning applications with the current libraries like `TensorFlow` and `Keras`.

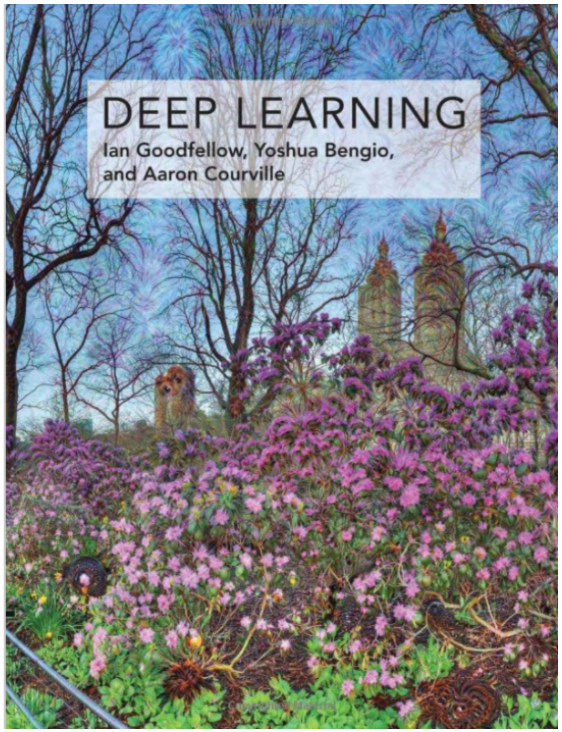
Deep Learning perspectives

"Following a scientific revolution, progress generally follows a sigmoid curve: it starts with a period of fast progress, which gradually stabilizes as researchers hit hard limitations, and then further improvements become incremental. Deep learning in 2017 seems to be in the first half of that sigmoid, with much more progress to come in the next few years."

FRANÇOIS CHOLLET

Interesting readings and references





Interesting readings and references


Andrew Ng coursera.
https://www.deeplearning.ai/


Deep Learning - The Straight Dope.
http://gluon.mxnet.io/


R Interface to TensorFlow.
https://tensorflow.rstudio.com/


Keras Examples.
https://tensorflow.rstudio.com/keras/articles/examples/

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

e-mail: rborras@clinic.cat