Dojo Tensorflow

Jiqiong QIU
Romain Jouin
Denis Oblin

Outline

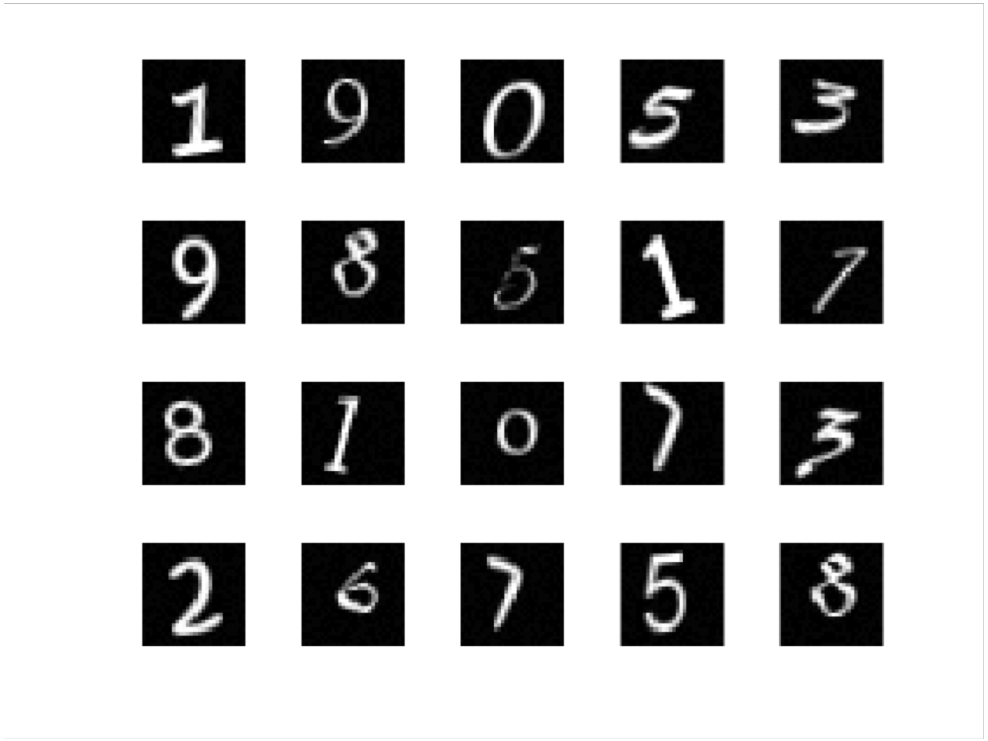
1. Overview of 1st meetup dojo
    a. What we have learned
    b. Assignment 1
2. Goal
    a. Multilayer perceptron
    b. Regularization
    c. Convolutional neural network

Overview 1st meetup

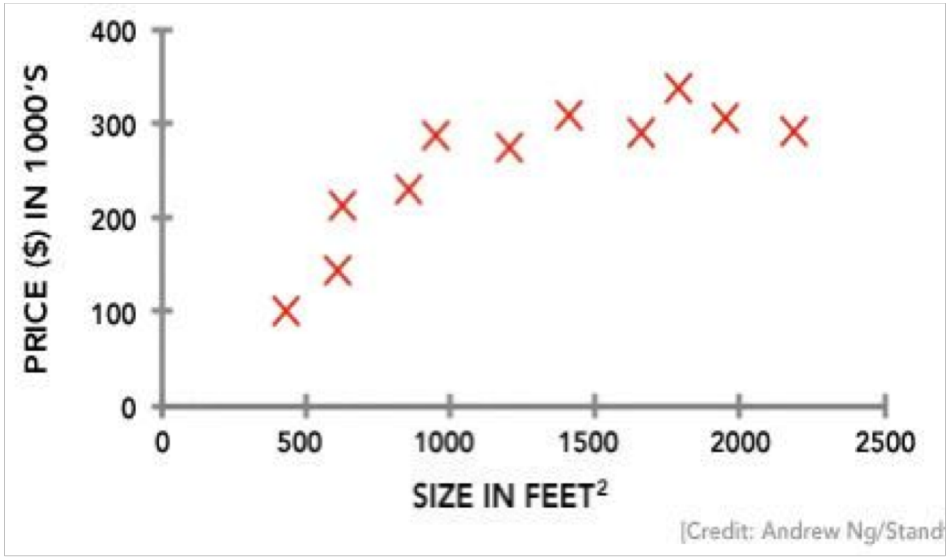
1. What we have learned
    a. Classification VS Regression
    b. Softmax
    c. One-hot encoding
    d. Cost Function: Cross-entropy

Classification VS Regression

MNIST



Housing Price Prediction



Softmax function

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

```
f = np.array([123, 456, 789]) # example with 3 classes and each having large scores
p = np.exp(f) / np.sum(np.exp(f)) # Bad: Numeric problem, potential blowup

# instead: first shift the values of f so that the highest number is 0:
f -= np.max(f) # f becomes [-666, -333, 0]
p = np.exp(f) / np.sum(np.exp(f)) # safe to do, gives the correct answer
```

One-hot encoding

Converts a categorical feature to numerical, e.g.,

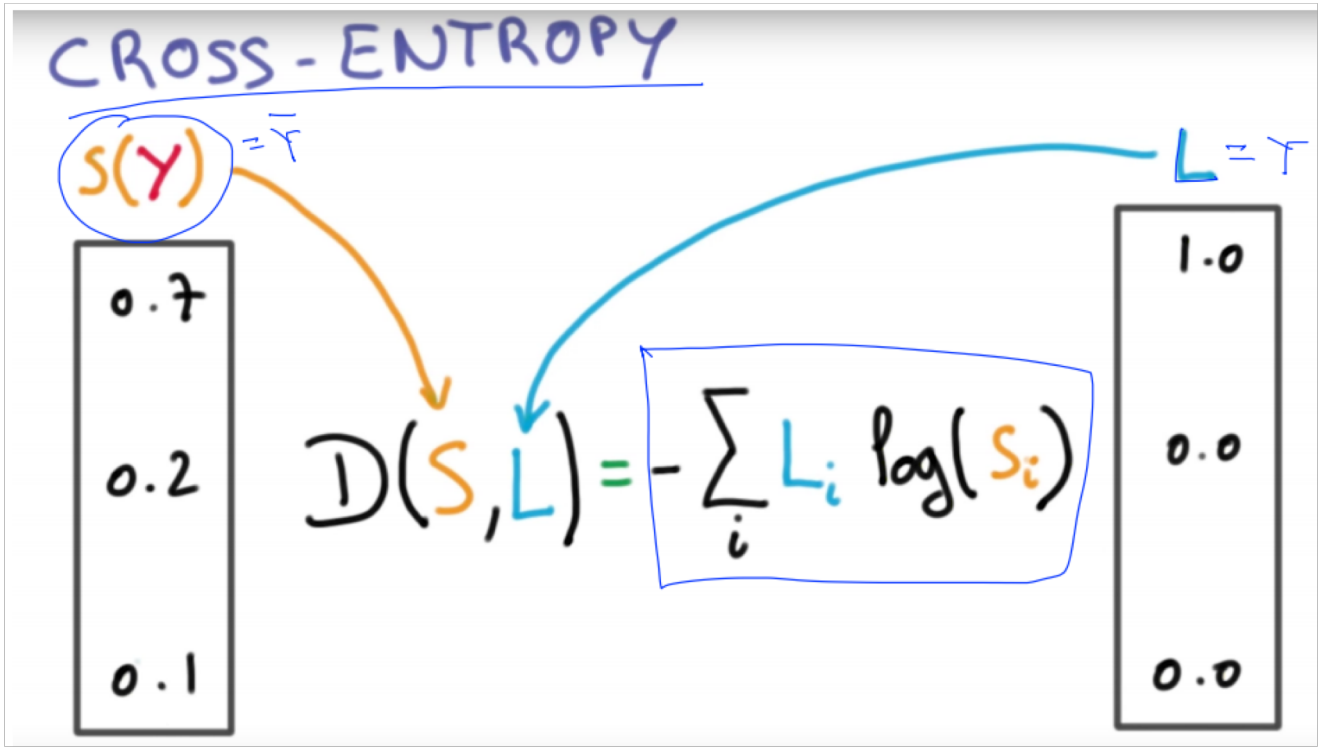
ID	Fruit	apple	banana	watermelon	pear
1	apple	1	0	0	0
2	banana	0	1	0	0
3	banana	0	1	0	0
4	watermelon	0	0	1	0
5	apple	1	0	0	0
6	pear	0	0	0	1
7	pear	0	0	0	1

Cost function: Cross-entropy

Cost function is used to evaluate the performance of learning algorithm:

Sum of squared for regression

Cross-entropy for classification

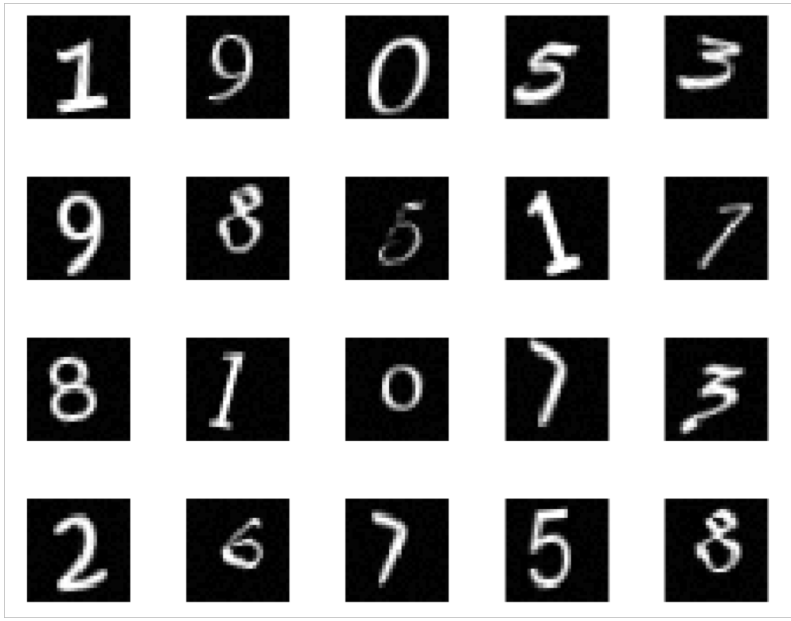


Outline

1. Overview of 1st meetup dojo
    a. What we have learned
    b. Assignment 1
2. Goal
    a. Multilayer perceptron
    b. Regularization
    c. Convolutional neural network

Assignment 1

NotMNIST dataset : 28x28 pixels, 10 class A - J



Assignment 1

What we have done:

Download the dataset

Display an image/ Study the distribution

Find the duplicated image by using hashing function (MD5)

Realize your first logistic regression model using sklearn

What we have now:

3 data sets

Training set: 200 000

Validation set: 10 000

```
print('Training:', train_dataset.shape, train_labels.shape)
print('Validation:', valid_dataset.shape, valid_labels.shape)
print('Testing:', test_dataset.shape, test_labels.shape)
```

```
Training: (200000, 28, 28) (200000,)
Validation: (10000, 28, 28) (10000,)
Testing: (10000, 28, 28) (10000,)
```

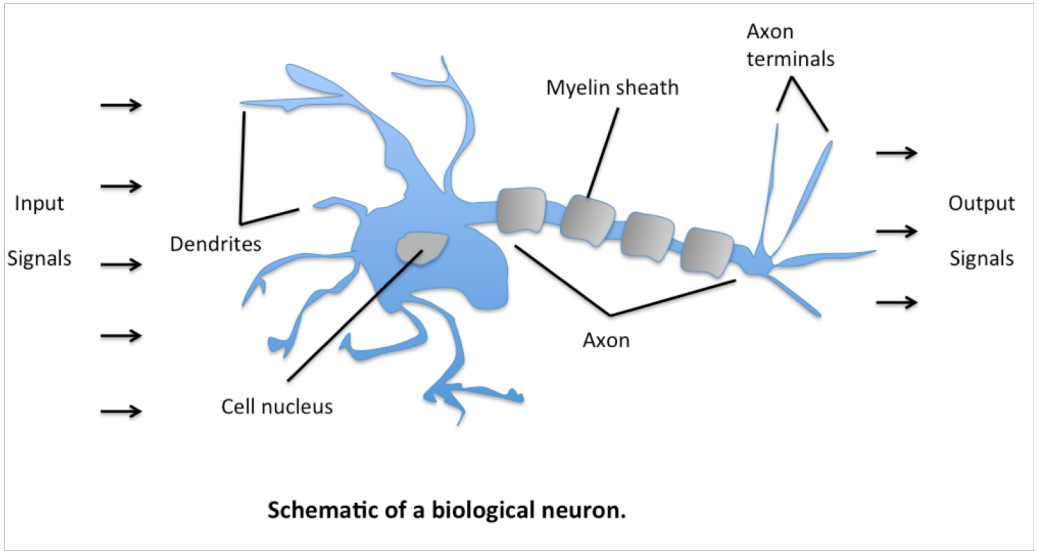
Outline

1. Overview of 1st meetup dojo
    a. What we have learned
    b. Assignment 1
2. Goal
    a. Multilayer perceptron
    b. Regularization
    c. Convolutional neural network

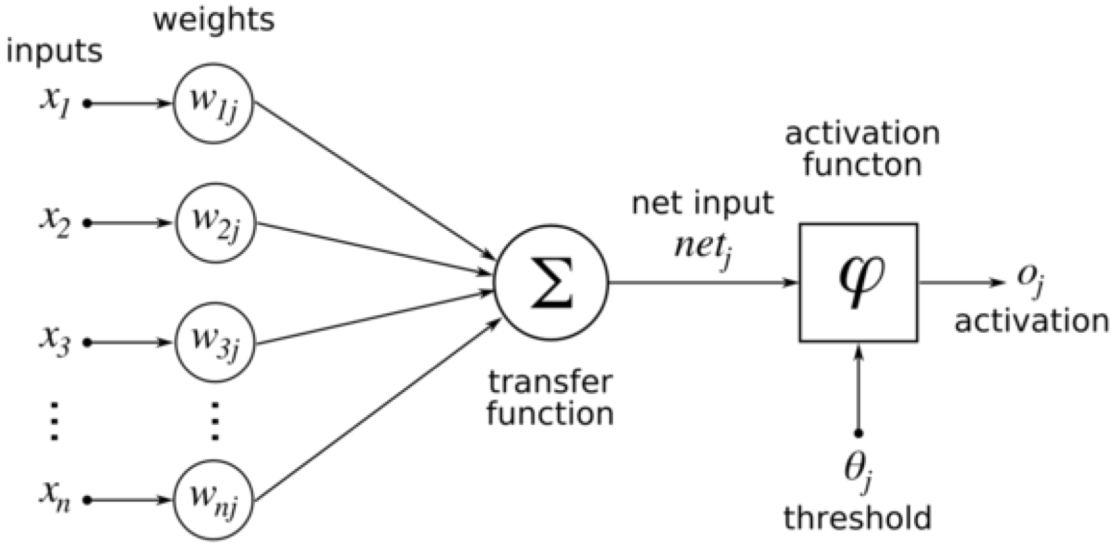
Multilayer perceptron

- Neuron (perceptron)
  - Biological neuron and perceptron
  - Activation function
- MLP (Multilayer perceptron)

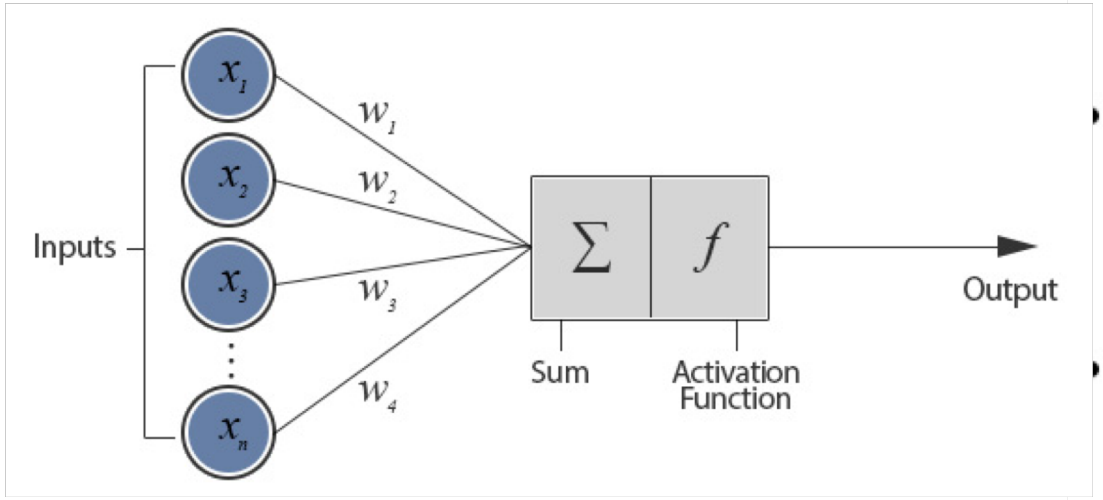
Perceptron



Schematic of a biological neuron.

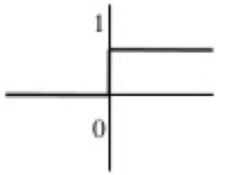


Activation function



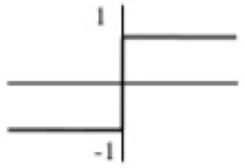
- **Step** function

$$step_t(x) = \begin{cases} 1 & x > t \\ 0 & otherwise \end{cases}$$



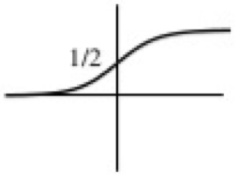
- **Sign** function

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & altrimenti \end{cases}$$

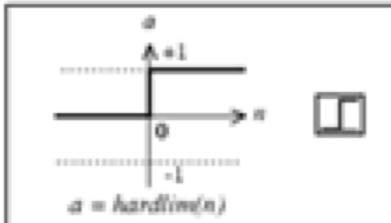
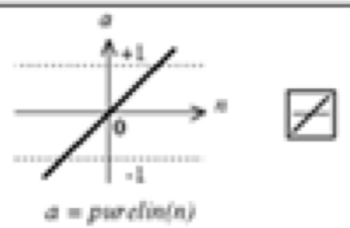
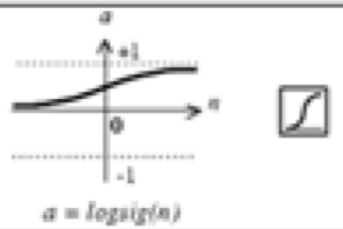
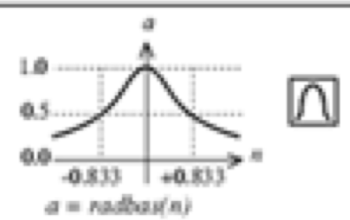
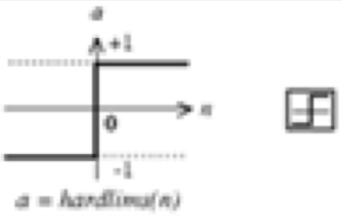
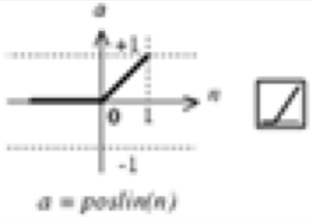
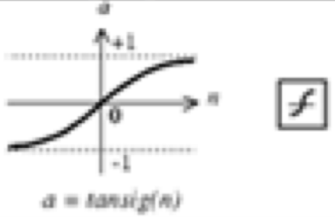
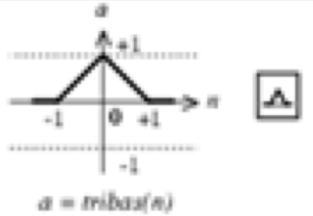
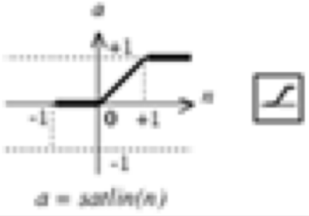
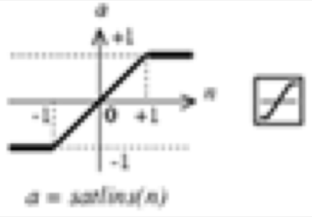
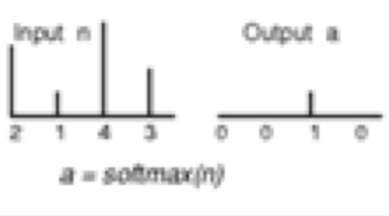
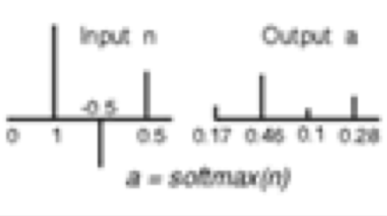


- **Sigmoid** function

$$sigmoide(x) = \frac{1}{1+e^{-x}}$$



Activation function

			
Hard-Limit TF	Linear TF	Log-Sigmoid TF	Radial Basis TF
			
Symmetric Hard-Limit TF	Positive Linear TF	Tan-Sigmoid TF	Triangular Basis TF
			
Satlin TF	Satlins TF	Compet TF	Softmax TF

Activation function (Tensorflow)

tf.nn.relu(features, name=None)

tf.nn.relu6(features, name=None)

tf.nn.elu(features, name=None)

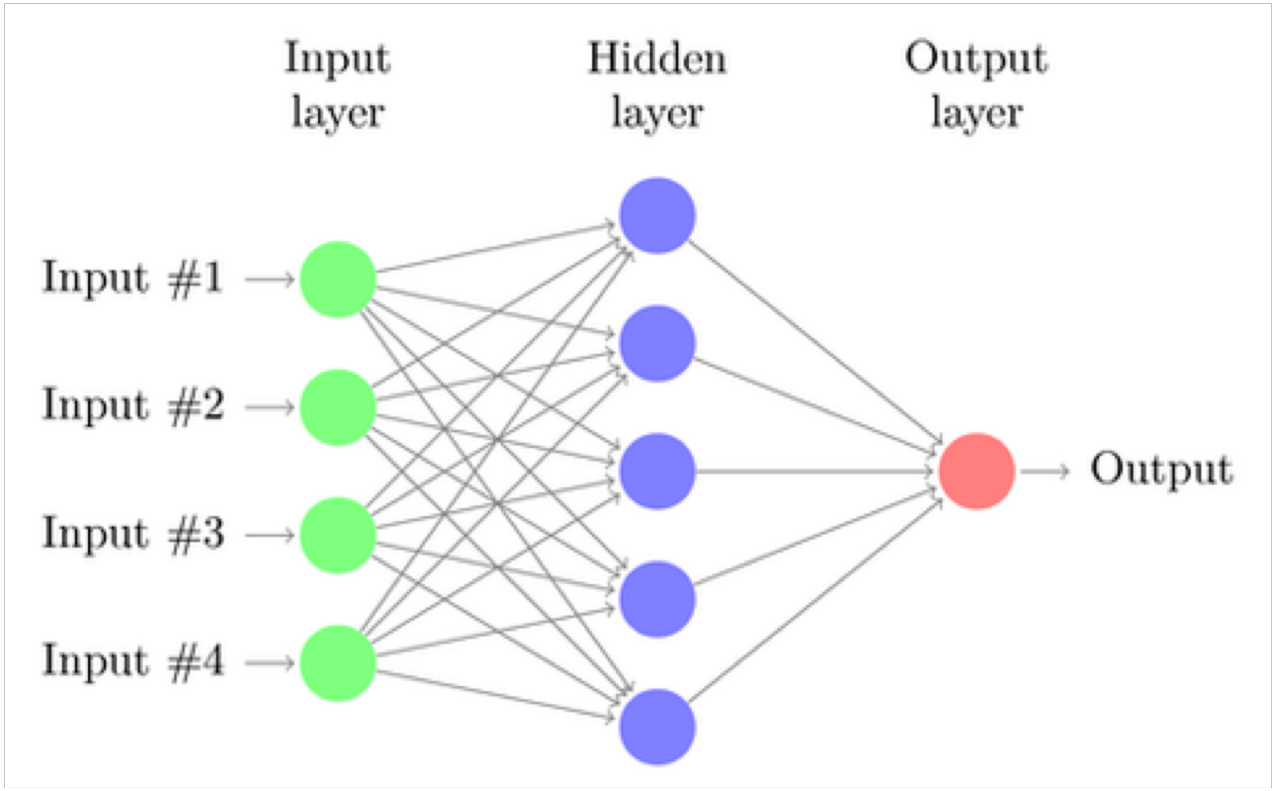
tf.nn.softplus(features, name=None)

tf.nn.softsign(features, name=None)

tf.nn.dropout(x, keep_prob, noise_shape=None, seed=None, name=None)

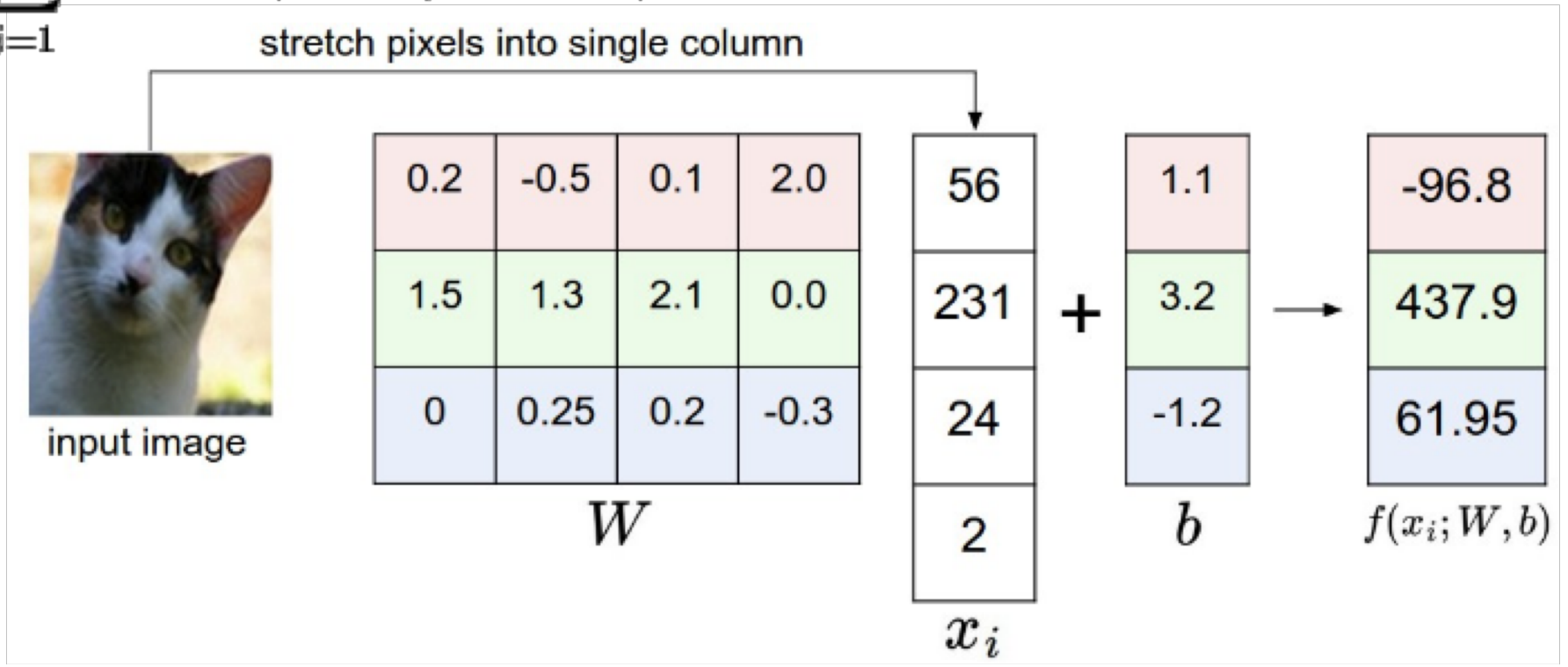
tf.nn.bias_add(value, bias, data_format=None, name=None)

MLP (Multilayer perceptron)



MLP (Multilayer perceptron)

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(\mathbf{w}^T \mathbf{x} + b)$$



MLP (Multilayer perceptron)

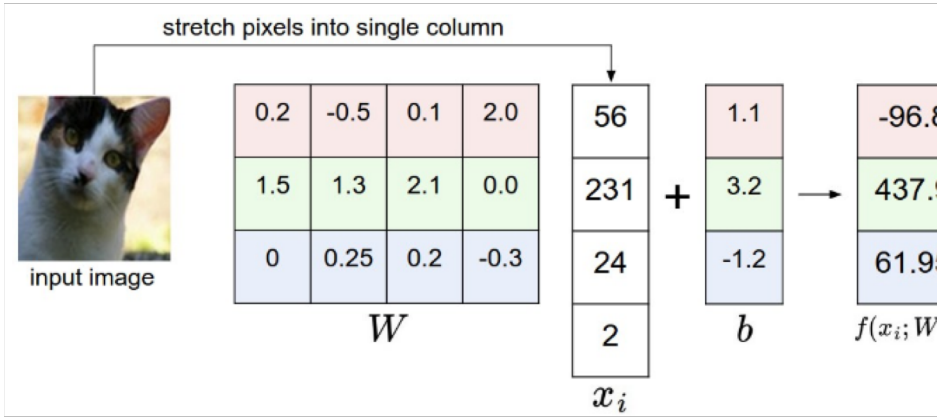
Dataset into constants

```
# Input data.
# Load the training, validation and test data into constants that are
# attached to the graph.
tf_train_dataset = tf.constant(train_dataset[:train_subset, :])
tf_train_labels = tf.constant(train_labels[:train_subset])
tf_valid_dataset = tf.constant(valid_dataset)
tf_test_dataset = tf.constant(test_dataset)
```

Weight and biases defined by variables

```
# Variables.
# These are the parameters that we are going to be training. The weight
# matrix will be initialized using random valued following a (truncated)
# normal distribution. The biases get initialized to zero.
weights = tf.Variable(
  tf.truncated_normal([image_size * image_size, num_labels]))
biases = tf.Variable(tf.zeros([num_labels]))
```

MLP (Multilayer perceptron)




```
# Training computation.
# We multiply the inputs with the weight matrix, and add biases. We compute
# the softmax and cross-entropy (it's one operation in TensorFlow, because
# it's very common, and it can be optimized). We take the average of this
# cross-entropy across all training examples: that's our loss.
logits = tf.matmul(tf_train_dataset, weights) + biases
loss = tf.reduce_mean(
  tf.nn.softmax_cross_entropy_with_logits(logits, tf_train_labels))
```

MLP (Multilayer perceptron)

```
# Predictions for the training, validation, and test data.
# These are not part of training, but merely here so that we can report
# accuracy figures as we train.
train_prediction = tf.nn.softmax(logits)
valid_prediction = tf.nn.softmax(
  tf.matmul(tf_valid_dataset, weights) + biases)
test_prediction = tf.nn.softmax(tf.matmul(tf_test_dataset, weights) + biases)
```

MLP training

1. Define cost function
2. Find the optimal weight and bias which minimise the cost function

Demo by tensorflow: http://playground.tensorflow.org

```
# Optimizer.
# We are going to find the minimum of this loss using gradient descent.
optimizer = tf.train.GradientDescentOptimizer(0.5).minimize(loss)
```

MLP training

```
num_steps = 801

def accuracy(predictions, labels):
  return (100.0 * np.sum(np.argmax(predictions, 1) == np.argmax(labels, 1))
          / predictions.shape[0])

with tf.Session(graph=graph) as session:
  # This is a one-time operation which ensures the parameters get initialized as
  # we described in the graph: random weights for the matrix, zeros for the
  # biases.
  tf.initialize_all_variables().run()
  print('Initialized')
  for step in range(num_steps):
    # Run the computations. We tell .run() that we want to run the optimizer,
    # and get the loss value and the training predictions returned as numpy
    # arrays.
    _, l, predictions = session.run([optimizer, loss, train_prediction])
    if (step % 100 == 0):
      print('Loss at step %d: %f' % (step, l))
      print('Training accuracy: %.1f%%' % accuracy(
        predictions, train_labels[:train_subset, :]))
      # Calling .eval() on valid_prediction is basically like calling run(), but
      # just to get that one numpy array. Note that it recomputes all its graph
      # dependencies.
      print('Validation accuracy: %.1f%%' % accuracy(
        valid_prediction.eval(), valid_labels))
  print('Test accuracy: %.1f%%' % accuracy(test_prediction.eval(), test_labels))
```

Assignment 2

Turn the logistic regression example with SGD into a 1-hidden layer neural network with rectified linear units (nn.relu()) and 1024 hidden nodes. This model should improve your validation / test accuracy.

Outline

1. Overview of 1st meetup dojo
    a. What we have learned
    b. Assignment 1
2. Goal
    a. Multilayer perceptron
    b. Regularization
    c. Convolutional neural network

Regularization

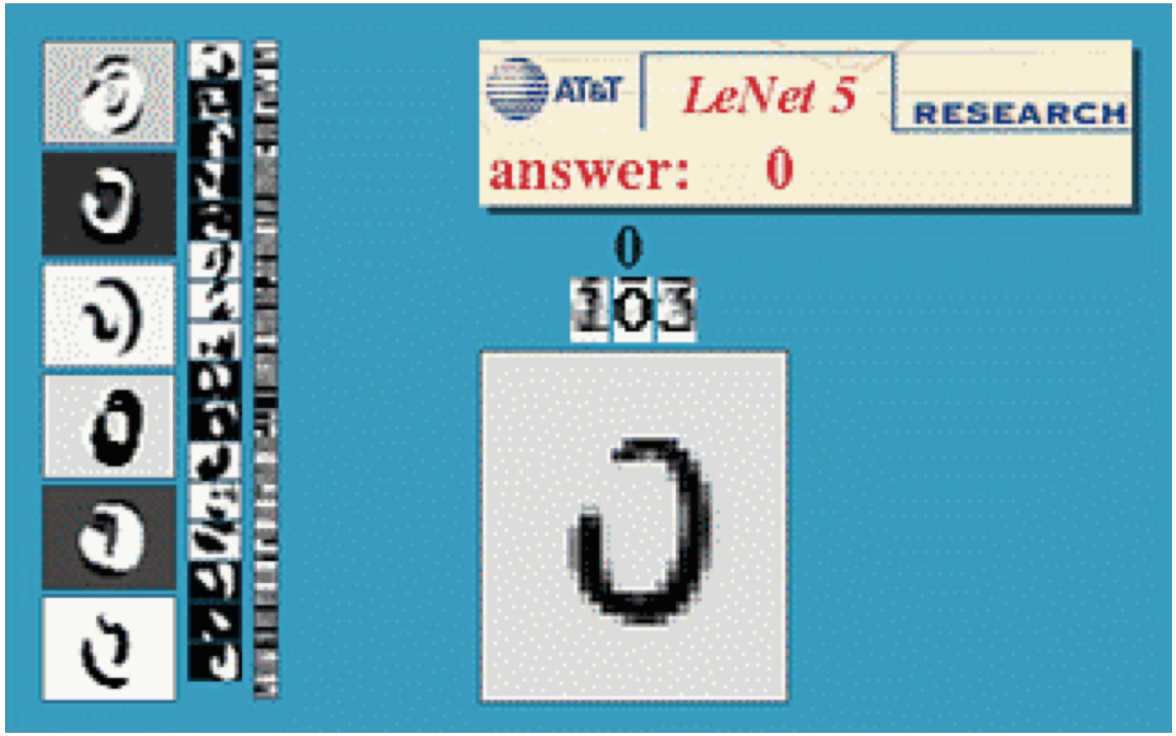
D

Outline

1. Overview of 1st meetup dojo
    a. What we have learned
    b. Assignment 1
2. Goal
    a. Multilayer perceptron
    b. Regularization
    c. Convolutional neural network

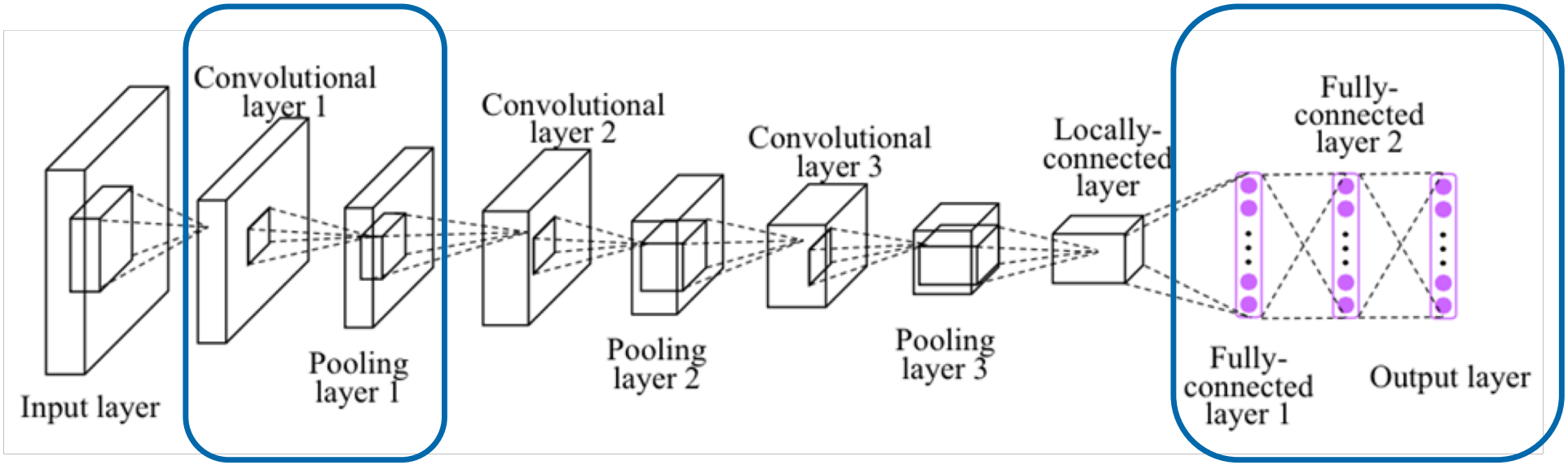
Convolutionnal Neural Network

Yann Lecun



Convolutionnal Neural Network

Convolution + Pooling



Convolutionnal Neural Network

- Convolution layer
- Pooling layer
- MLP and more
- Training and Results

Convolution Layer

Natural Images have the property of being “stationary”

Convolution Layer

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

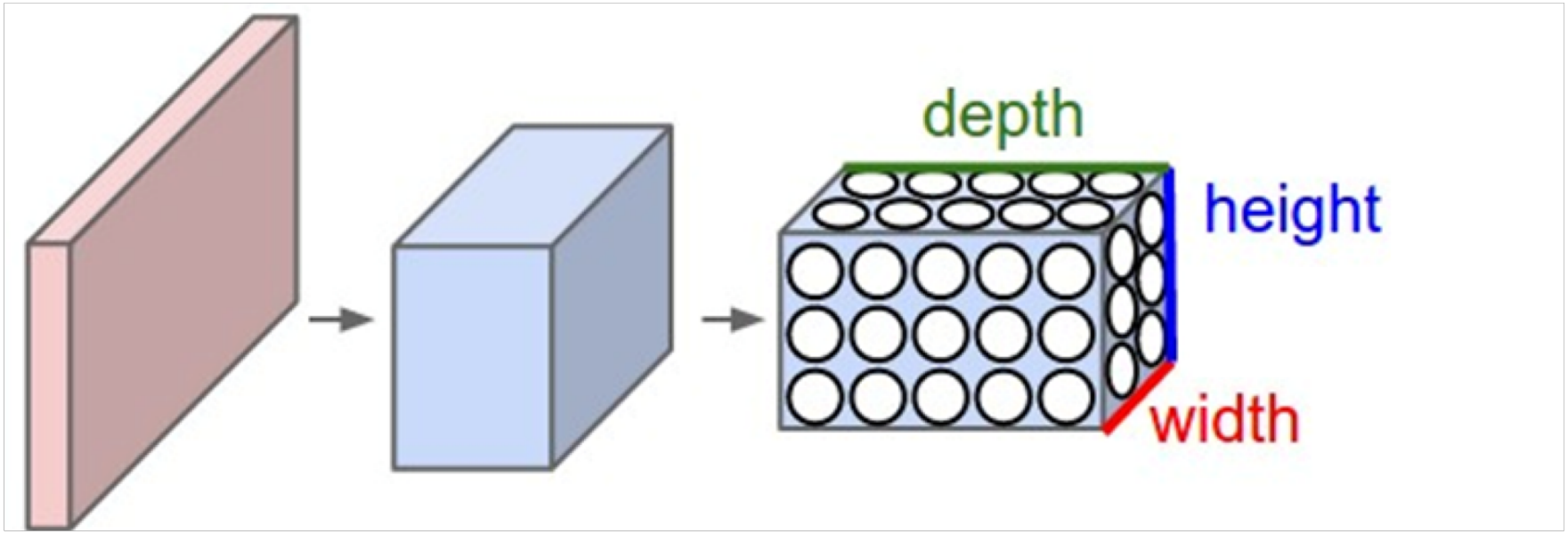
Kernel Matrix

4		

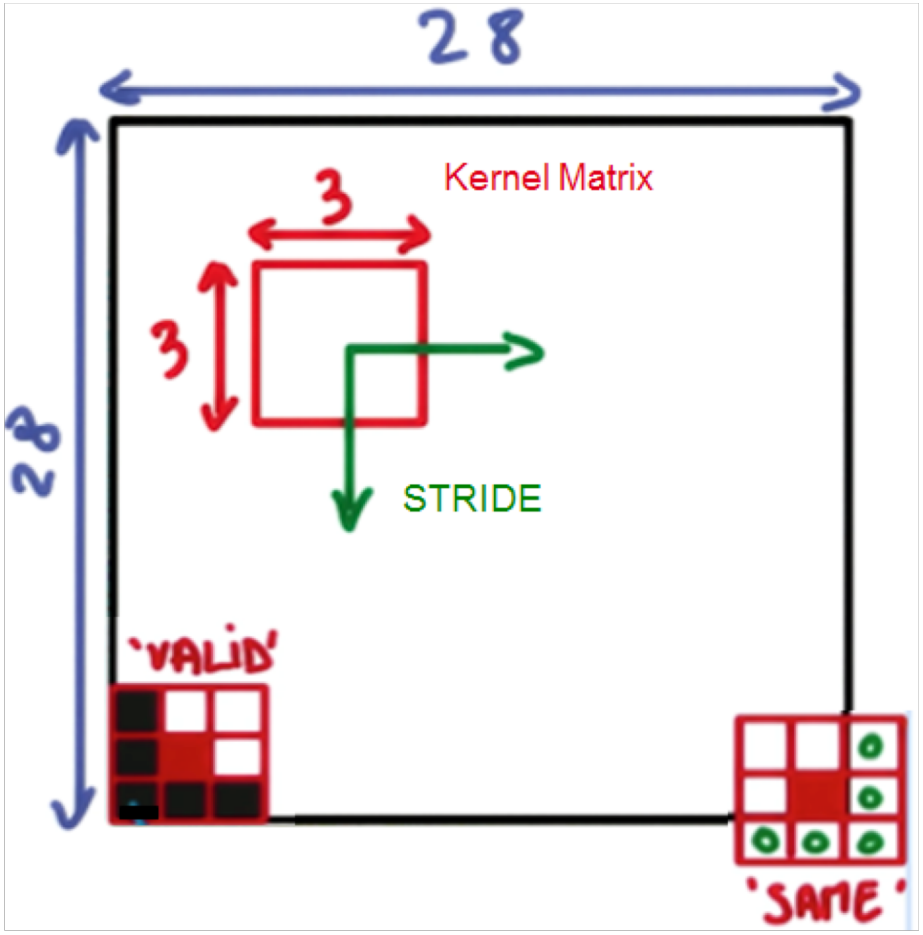
Convolved Feature

Convolution Layer

Input Depth = 3 (rgb)
Output Depth = 400



Convolution Layer



Convolution Layer

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

Usually, a relu function is used

Convolutionnal Neural Network

- Convolution layer
- Pooling layer
- MLP and more
- Training and Results

Pooling