CSCI S-89C Deep Reinforcement Learning

Harvard Summer School

Dmitry Kurochkin

Summer 2020
Lecture 11

Contents

Contents

Quiz 10

Question 1

4 / 4 pts

Suppose we want to build a Neural Network for classification. If there are 16 categories to classify, which of the following output layers should be used in Keras?

Please select:

Correct!

- (●) model.add(layers.Dense(16, activation='softmax'))
- ( ) model.add(layers.Dense(15, activation='softmax'))
- ( ) model.add(layers.Dropout(0.3))
- ( ) model.add(layers.Dense(16, activation='relu'))

Quiz 10

Question 2

4 / 4 pts

Suppose we have built the following Neural Network in Keras:

n = 700

```
model = models.Sequential()
model.add(layers.Dense(8, activation='relu', input_shape=(n,)))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(1, activation='relu'))
```

How many parameters does it have?

Correct!

5,689

Quiz 10

Question 3

4 / 4 pts

All of the following activation functions are used in Deep Neural Networks:

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

$$\sigma^{(2)}(x) = \tanh x$$

$$\sigma^{(3)}(x) = \max\{0, x\}$$

$$\sigma^{(4)}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0.01x, & \text{otherwise.} \end{cases}$$

Correct!

- (•) True
- ( ) False

Quiz 10

Question 4

4 / 4 pts

Suppose we have the following Neural Network with one input x, one hidden layer with one neuron u, and scalar output y:

$$u = f\left(w_0^{(1)} + w_1^{(1)} x\right), \hat{y} = f\left(w_0^{(2)} + w_1^{(2)} u\right),$$

where

$$f(x) = \begin{cases} x, \text{ if } x \geq 0, \\ 0, \text{ if } x < 0. \end{cases}$$

The *errors* are defined as follows:

$\varepsilon^{(1)} \doteq \frac{\partial L}{\partial u}$ and $\varepsilon^{(2)} \doteq \frac{\partial L}{\partial \hat{y}}$, where $L \doteq \frac{1}{2}(\hat{y} - y)^2$.

If $(w_0^{(1)}, w_1^{(1)}, w_0^{(2)}, w_1^{(2)})^T = (-1.9, 0.7, 0.3, 1.4)^T$ and we observe $(x, y) = (1.2, 0.5)$, please find $\varepsilon^{(2)}$.

Correct!

-0.2

Quiz 10

Question 5

4 / 4 pts

Suppose we have the following Neural Network with one input x, one hidden layer with one neuron u, and scalar output y:

$$u = f\left(w_0^{(1)} + w_1^{(1)} x\right), \hat{y} = f\left(w_0^{(2)} + w_1^{(2)} u\right),$$

where

$$f(x) = \begin{cases} x, \text{ if } x \geq 0, \\ 0, \text{ if } x < 0. \end{cases}$$

The *errors* are defined as follows:

$\varepsilon^{(1)} \doteq \frac{\partial L}{\partial u}$ and $\varepsilon^{(2)} \doteq \frac{\partial L}{\partial \hat{y}}$, where $L \doteq \frac{1}{2}(\hat{y} - y)^2$.

If $(w_0^{(1)}, w_1^{(1)}, w_0^{(2)}, w_1^{(2)})^T = (-1.9, 0.7, 0.3, 1.4)^T$ and we observe $(x, y) = (1.2, 0.5)$, please find $\varepsilon^{(1)}$.

Correct!

-0.28

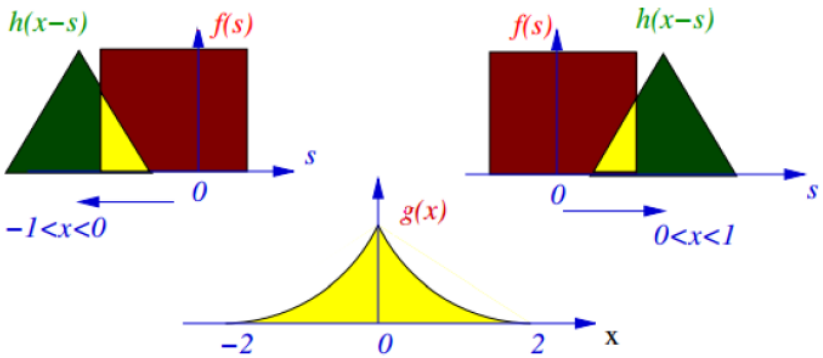
Contents

Convolution

Convolution of two functions $f(s)$ and $h(s)$ is defined as folows:

$$(f * h)(x) \doteq \int_{-\infty}^{+\infty} f(s)h(x-s)ds$$

Example:



Convolution

Convolution can be used for edge detection. Example:
Original image:

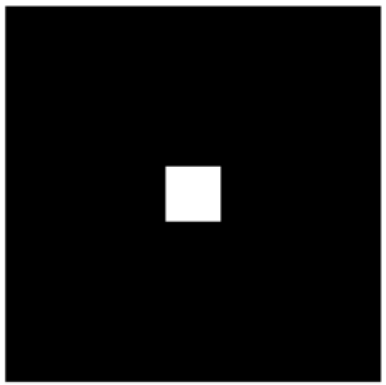
```
plot_image(images[0, :, :, 0])
save_fig("husky-original", tight_layout=False)
plt.show()
```

```
Saving figure husky-original
```

Convolution

Convolution can be used for edge detection. Example:
Filter:

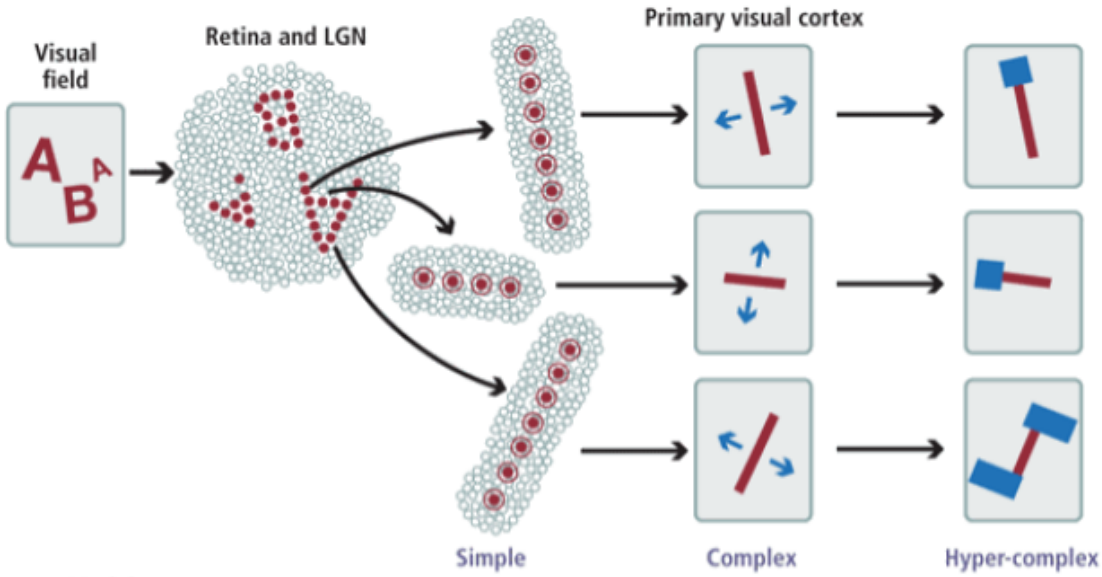
```
import numpy as np
import matplotlib.pyplot as plt


def plot_image(image):
    plt.imshow(image, cmap="gray", interpolation="nearest")
    plt.axis("off")

fmap = np.zeros(shape=(7, 7, 1, 3), dtype=np.float32)
fmap[:, 3, 0, 0] = 1
fmap[3, :, 0, 1] = 1
fmap[:, :, 0, 2] = -1
fmap[3, 3, 0, 2] = 48

plot_image(fmap[:, :, 0, 2])
plt.show()
```



Convolution

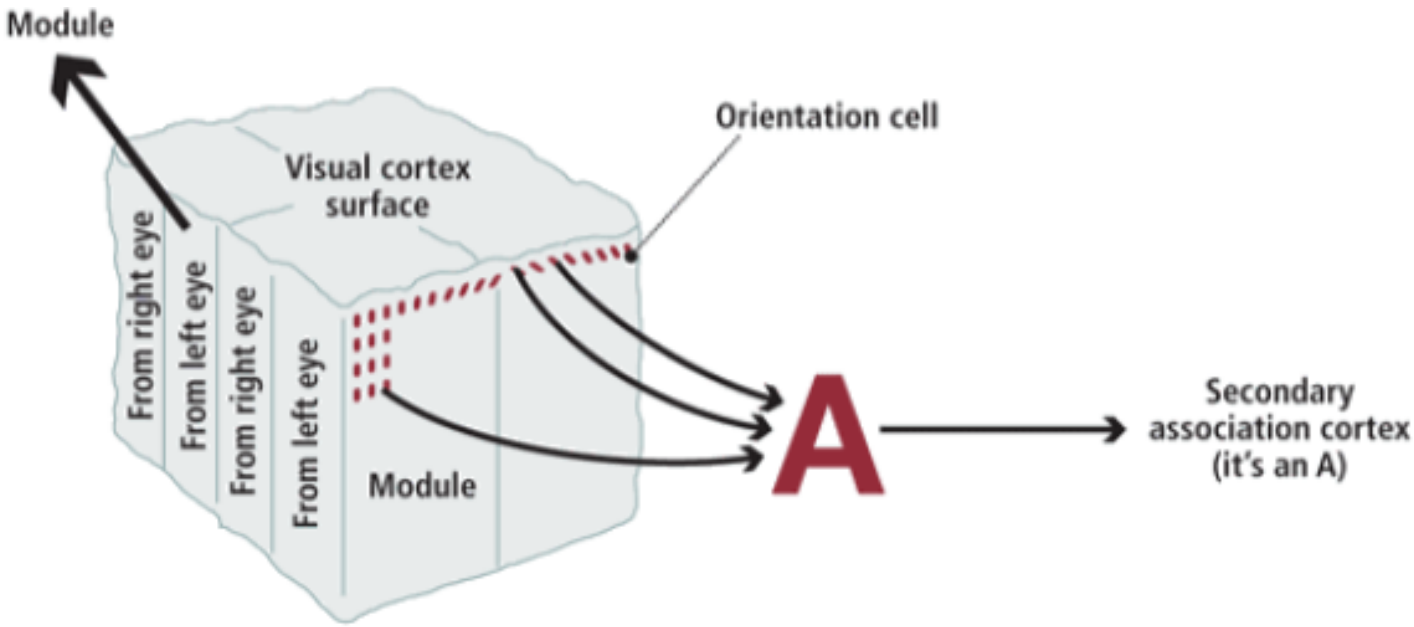
Convolution can be used for edge detection. Example:
Convolution:

Visual Cortex

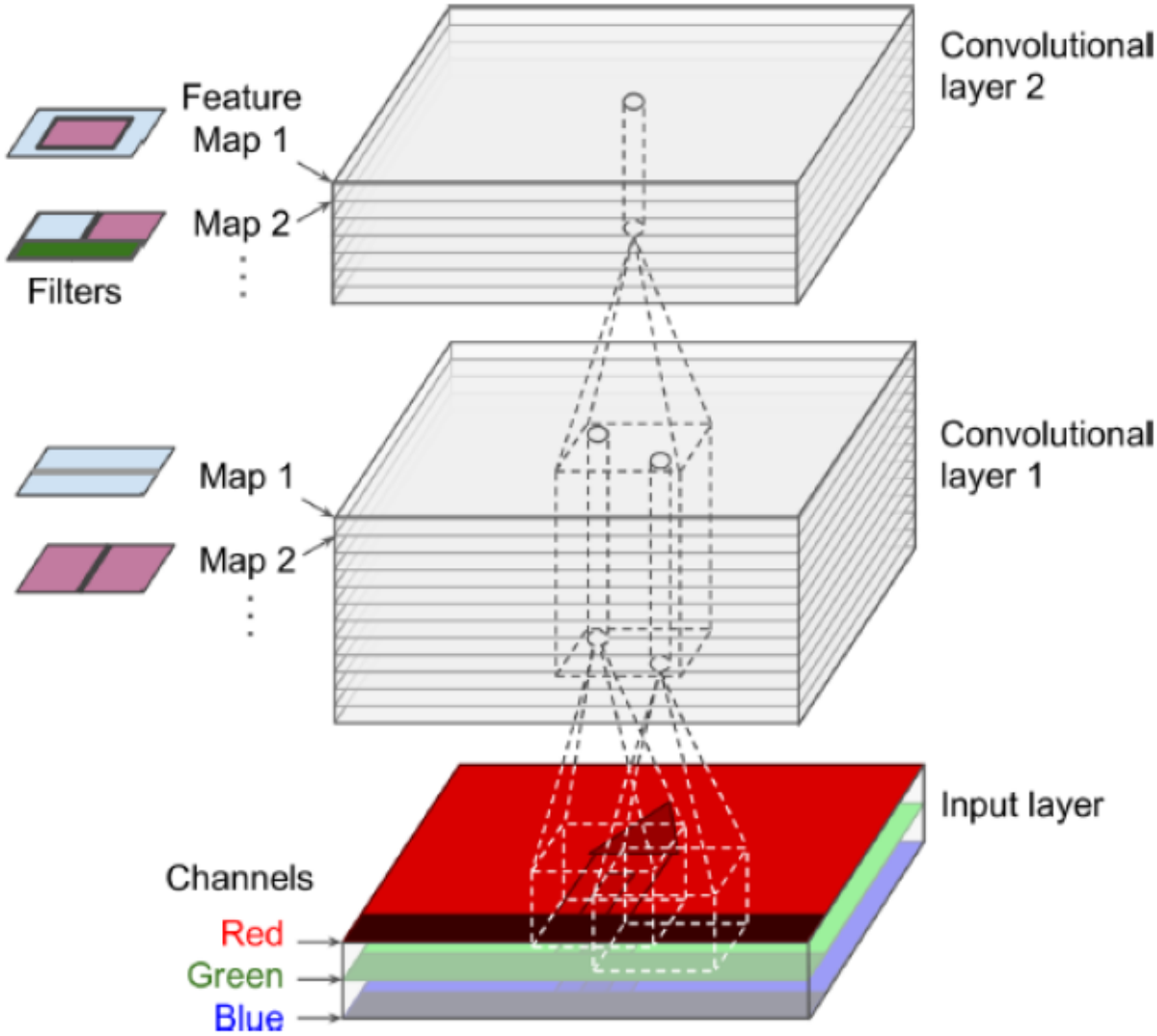


- In the Visual Cortex, simple cells become active when they are subjected to stimuli such as edges.
- Complex cells then combine the information of several simple cells and detect the position and orientation of a structure.
- Hyper complex cells then detect endpoints and crossing lines from this position and orientation information, which is then used in the brain's secondary cortex for information association.

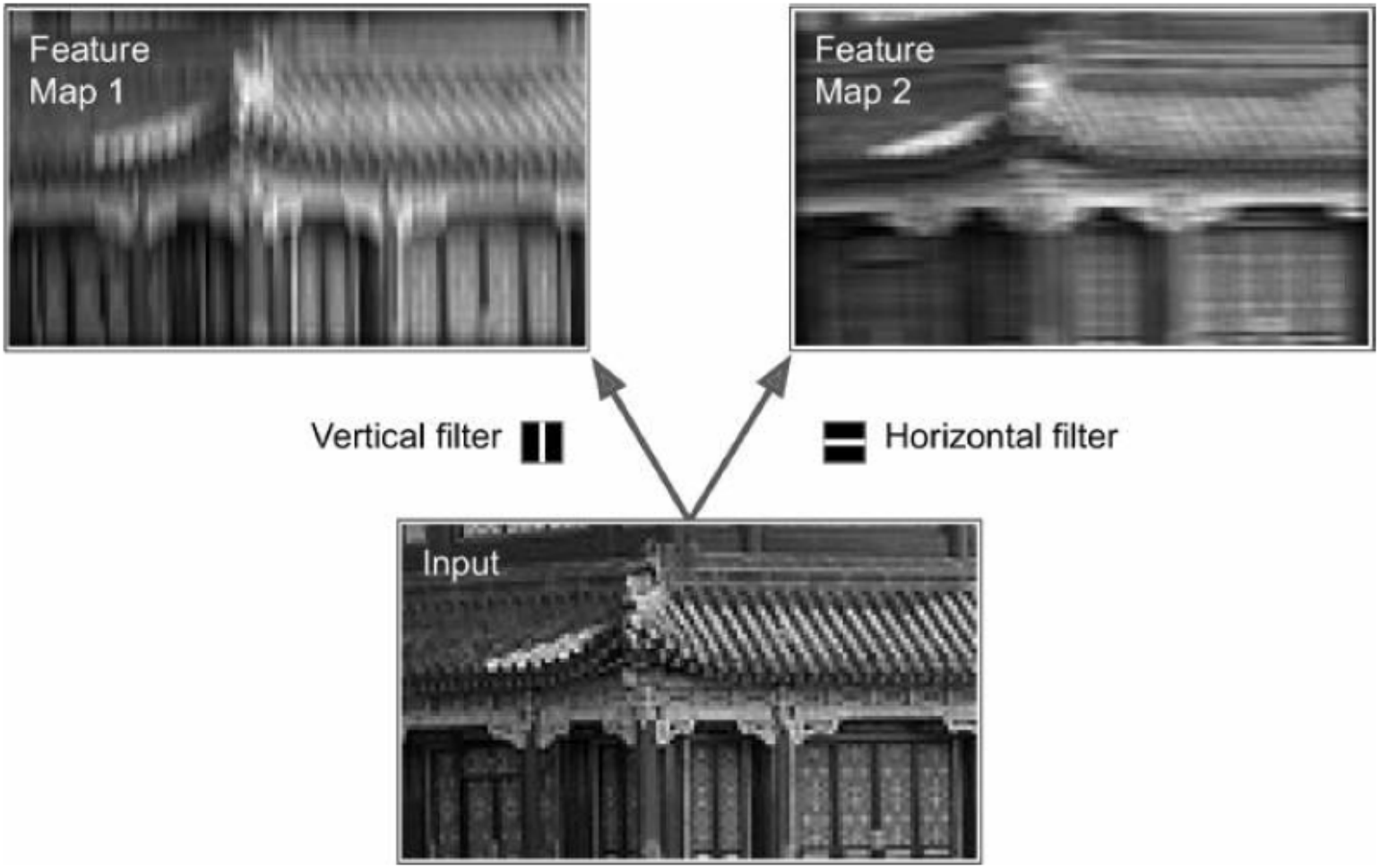
Visual Cortex



Convolutional Layers

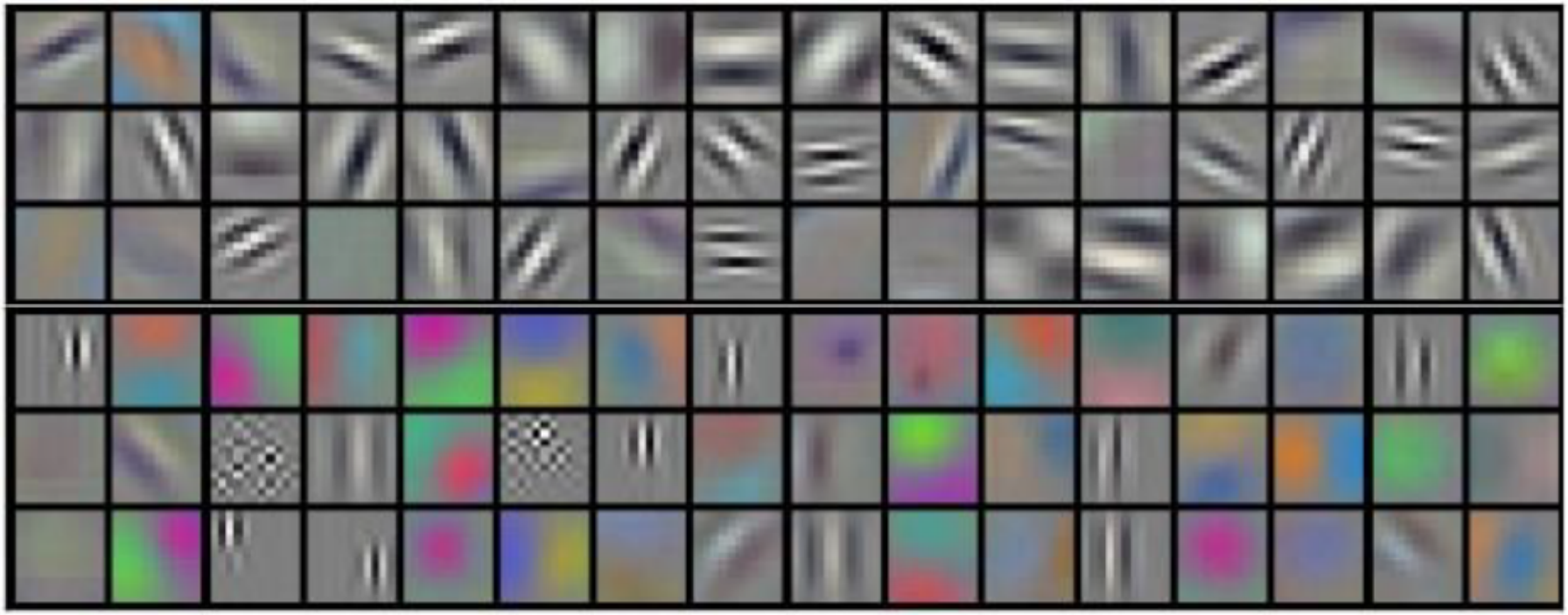


Convolutional Layers: Filters



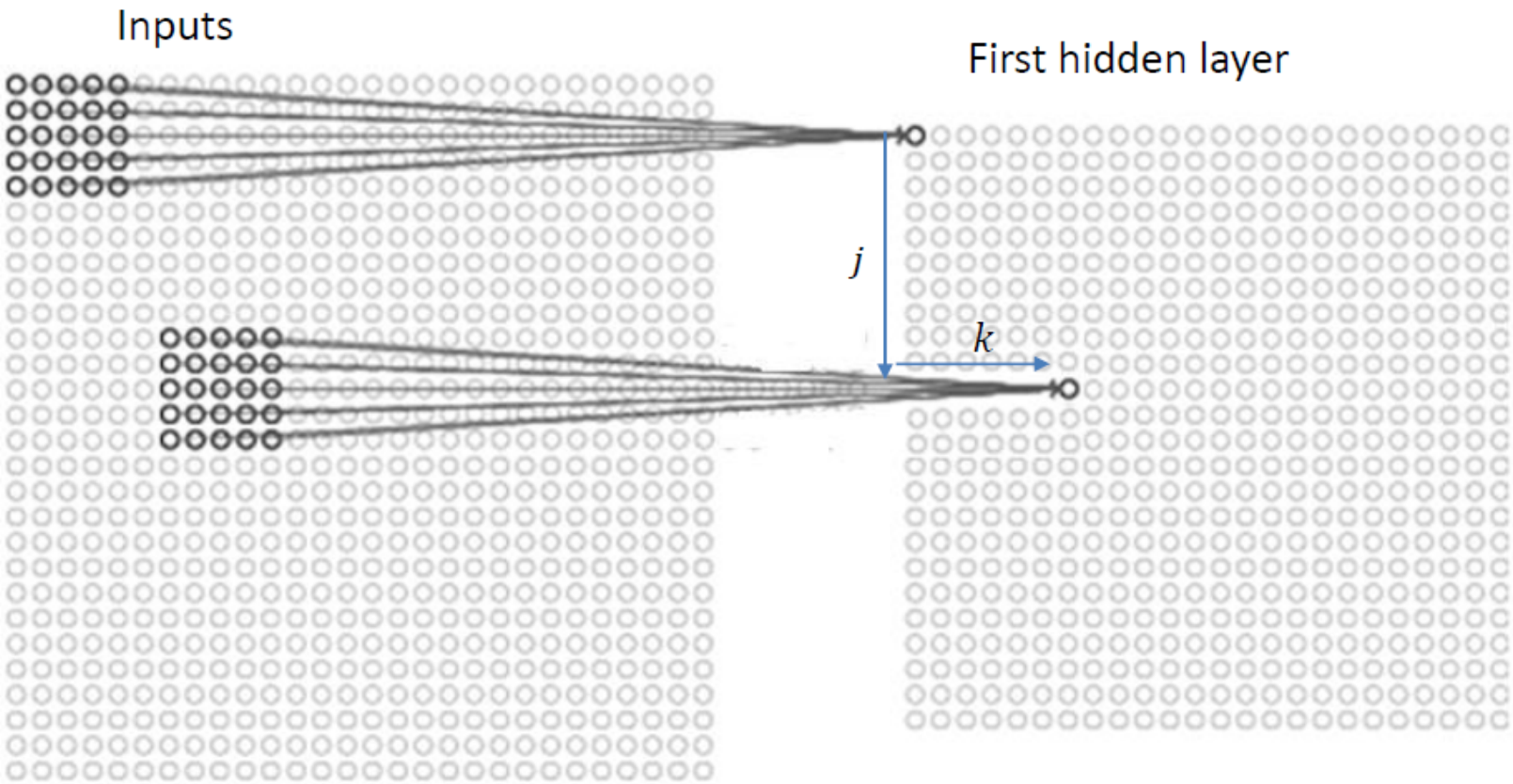
Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Convolutional Layers: Filters

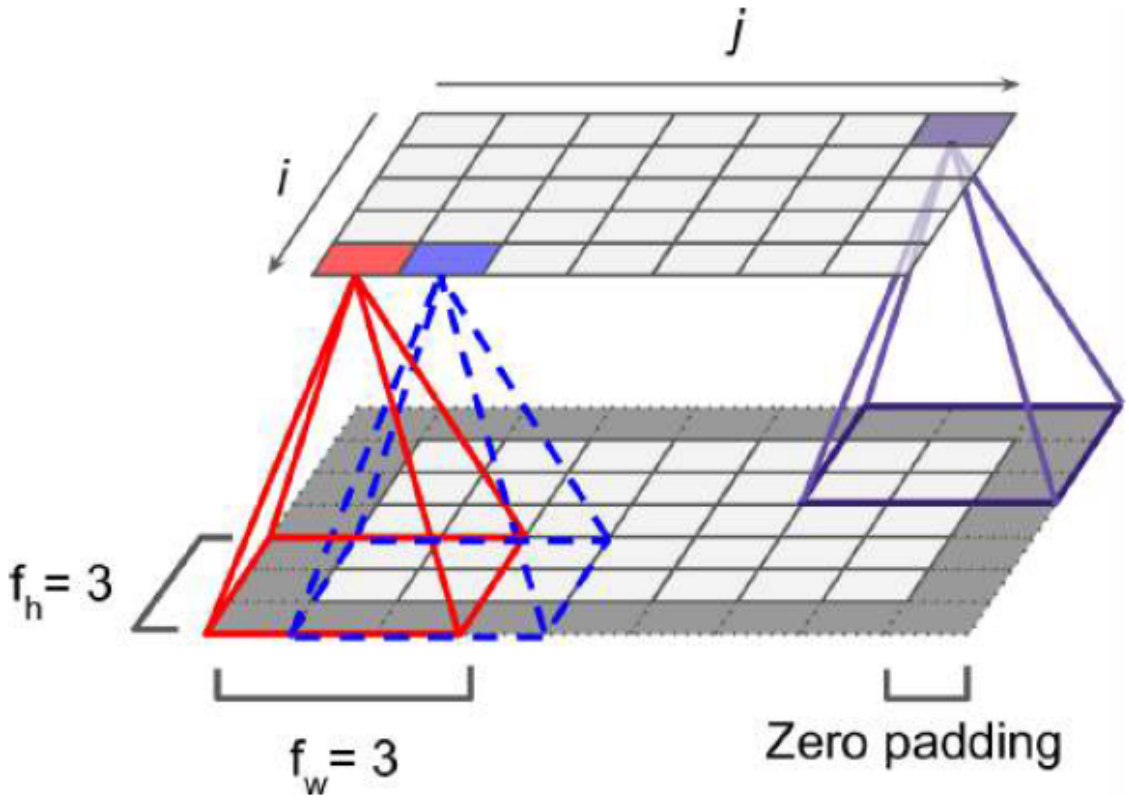


- In early work on CNNs by Krizhevsky et al., they considered 96 filters of size 11×11. Each filter detects certain feature: horizontal edge, vertical edge, slanted edge, etc.
- Eventually, it became apparent that it is not necessary to preselect the filters, but that we could let neural network training process determine the shape of those filters.

Convolutional Layers: Padding



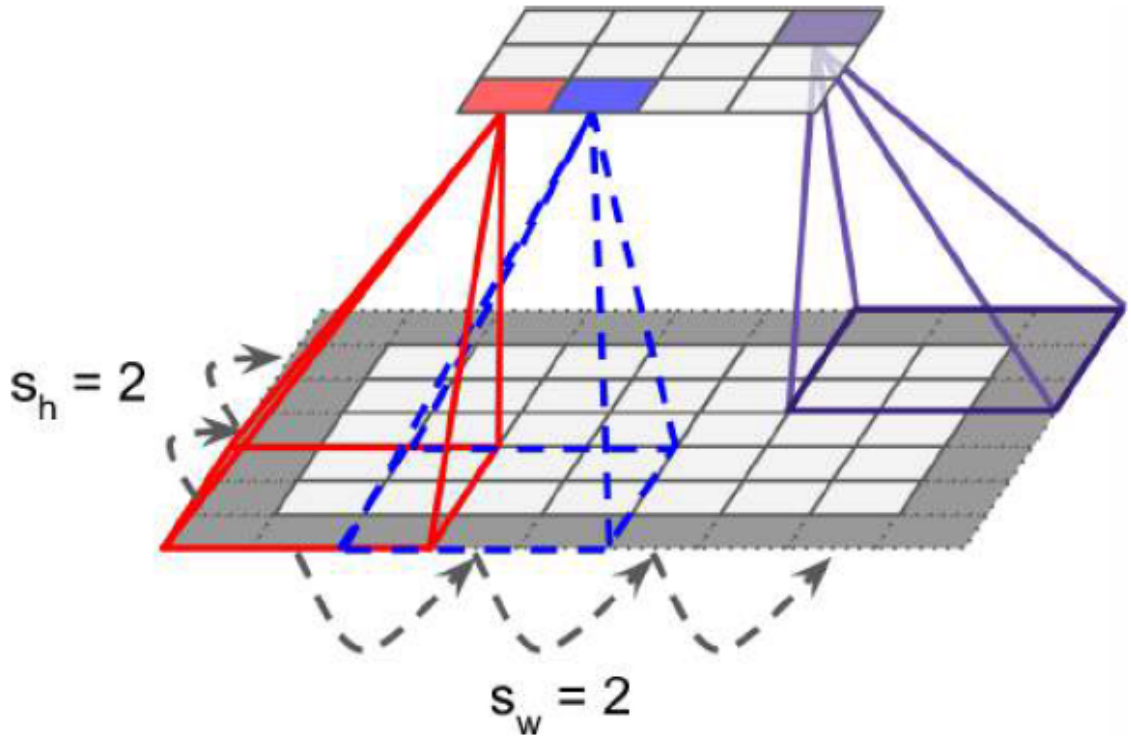
Convolutional Layers: Padding



Keras: padding can be set to either 'SAME' or 'VALID'

Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Convolutional Layers: Strides



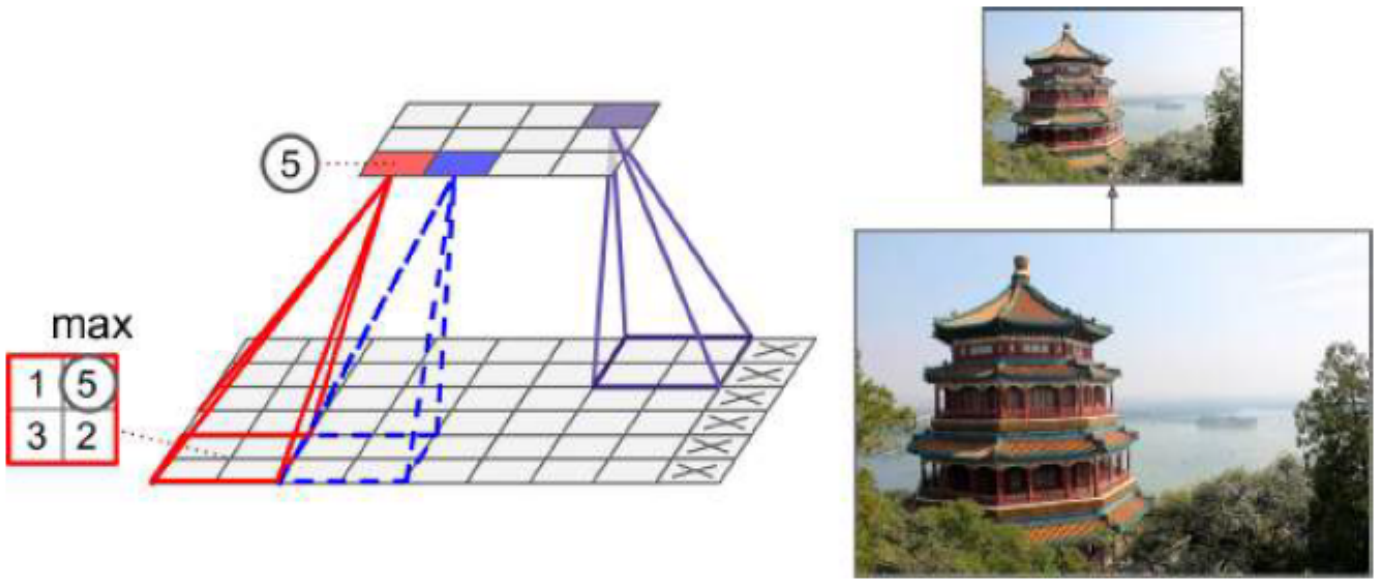
Keras: strides can be set to (w_w, w_h)

Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

MaxPooling Layer

Example:

Max pooling layer with 2×2 filter, strides$= 2$, and padding='VALID'



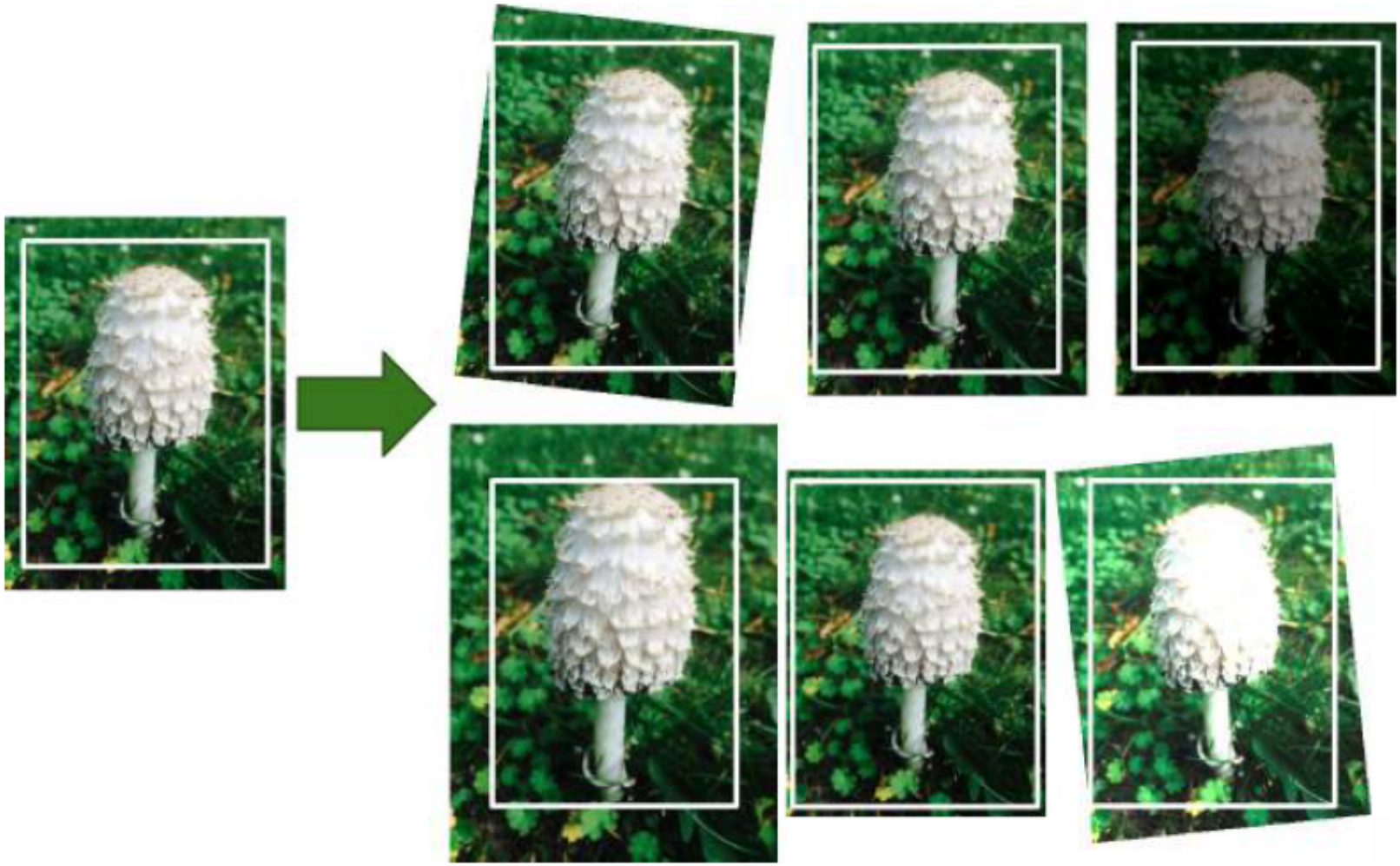
Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

CNN Example: Keras

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

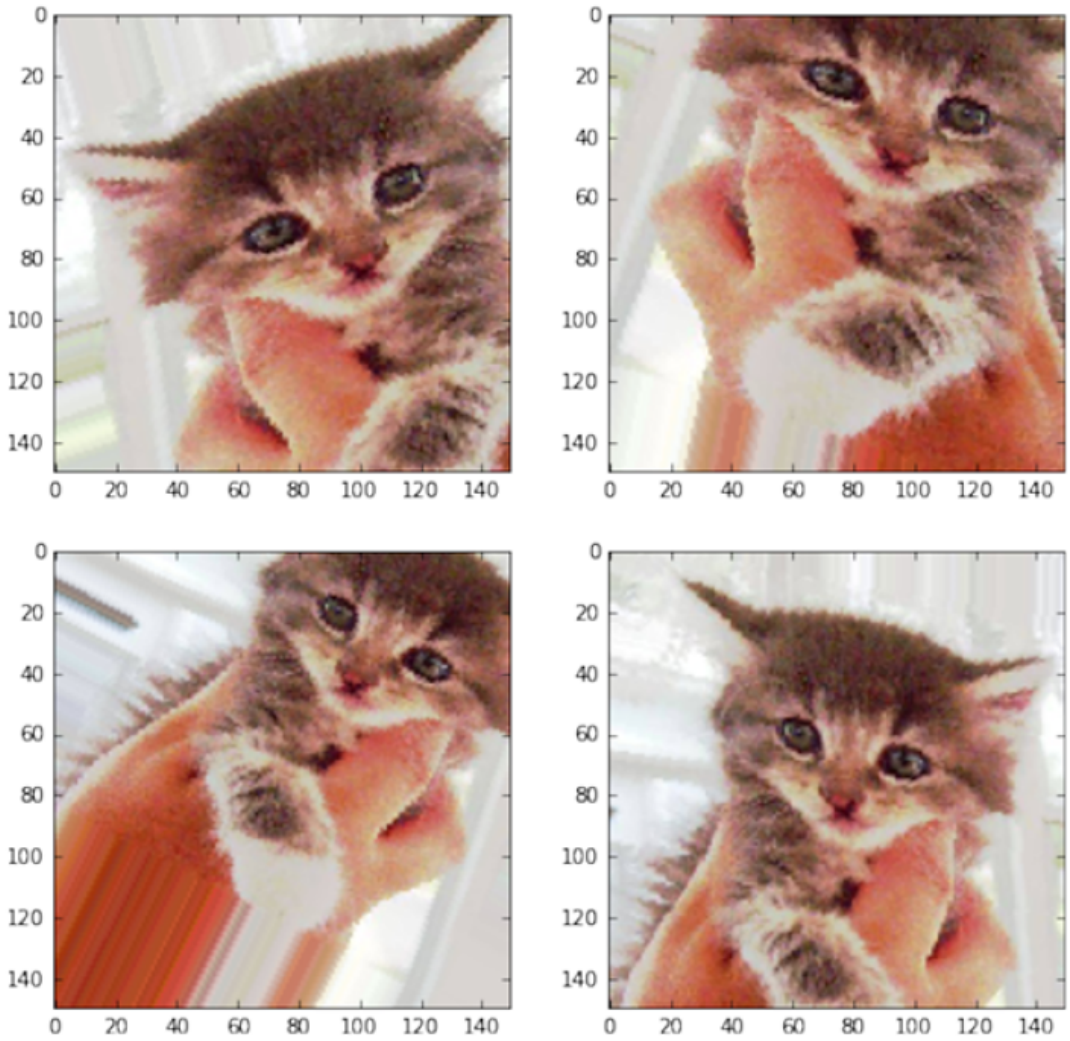
Contents

Image Data Augmentation



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Image Data Augmentation

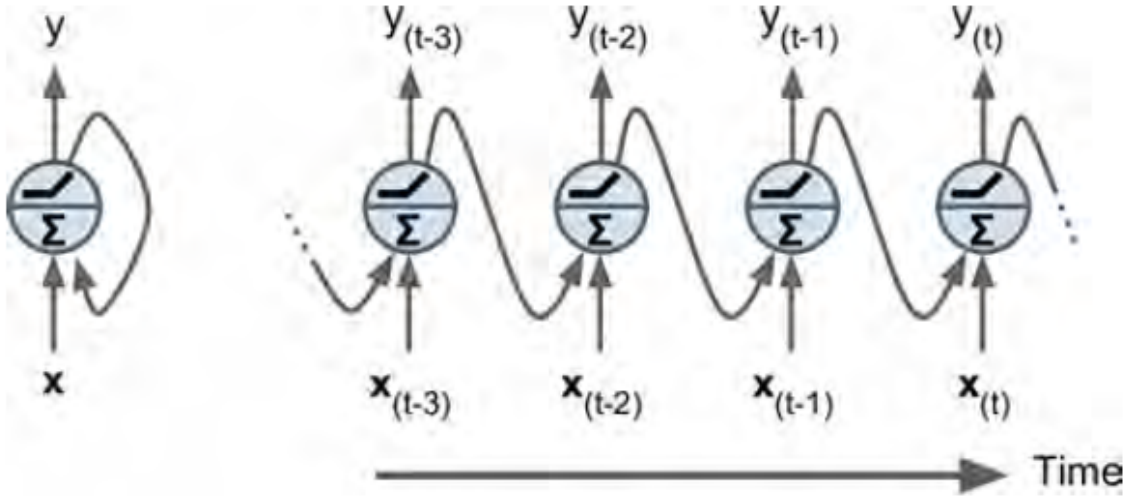


Source: *Deep Learning* by F. Chollet

Contents

RNN: Recurrent Neuron

Recurrent Neuron:

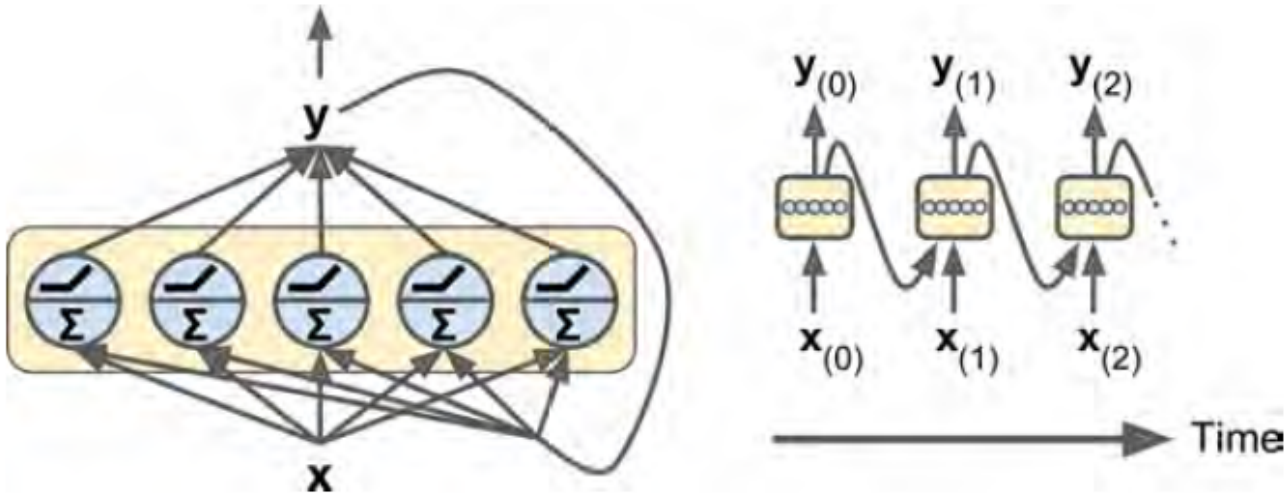


Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Contents

RNN: Layer of Recurrent Neurons

Layer of Recurrent Neurons:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Layer of Recurrent Neurons: Keras

Simple RNN in Keras:

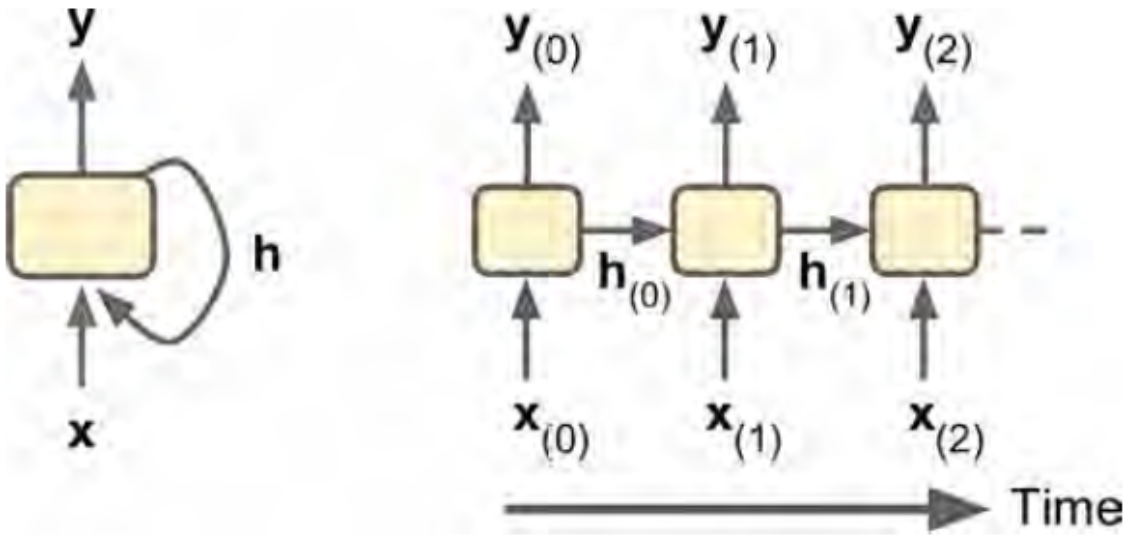
```
n_features = 2
n_timesteps = 200

model = models.Sequential()
model.add(layers.SimpleRNN(3, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(layers.Dense(1, activation='linear'))

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_6 (SimpleRNN)     (None, 3)                 18
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 4
=================================================================
Total params: 22
Trainable params: 22
Non-trainable params: 0
```

Contents

RNN: Memory Cells

Layer of Recurrent Neurons:

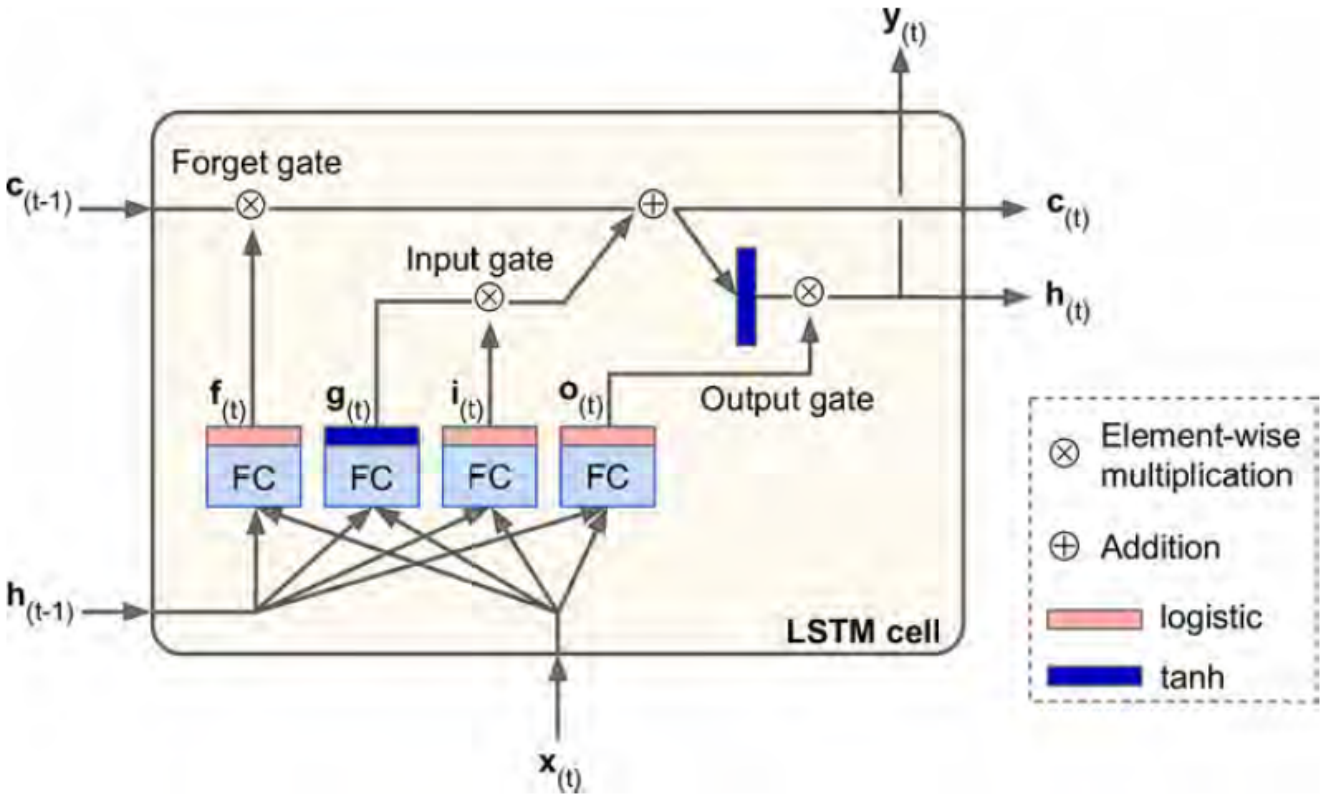


Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Contents

Long Short-Term Memory (LSTM) Cell

LSTM Cell:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Long Short-Term Memory (LSTM) Cell

LSTM Cell Model:

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i\right) \\
\mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f\right) \\
\mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g\right) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)
\end{aligned}
$$

- $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_{(t)}$.
- $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, and $\mathbf{W}_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{(t-1)}$.
- $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, and $\mathbf{b}_g$ are the bias terms for each of the four layers. Note that TensorFlow initializes $\mathbf{b}_f$ to a vector full of 1s instead of 0s. This prevents forgetting everything at the beginning of training.

Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Applications of LSTM Cells

- greatly improved speech recognition on over 4 billion Android phones (since mid 2015)
- greatly improved machine translation through Google Translate (since Nov 2016)
- greatly improved machine translation through Facebook (over 4 billion LSTMbased translations per day as of 2017)
- Siri and Quicktype on almost 2 billion iPhones (since 2016)
- generating answers by Amazon's Alexa and numerous other similar applications.

Long Short-Term Memory (LSTM) Cell: Keras

LSTM in Keras:

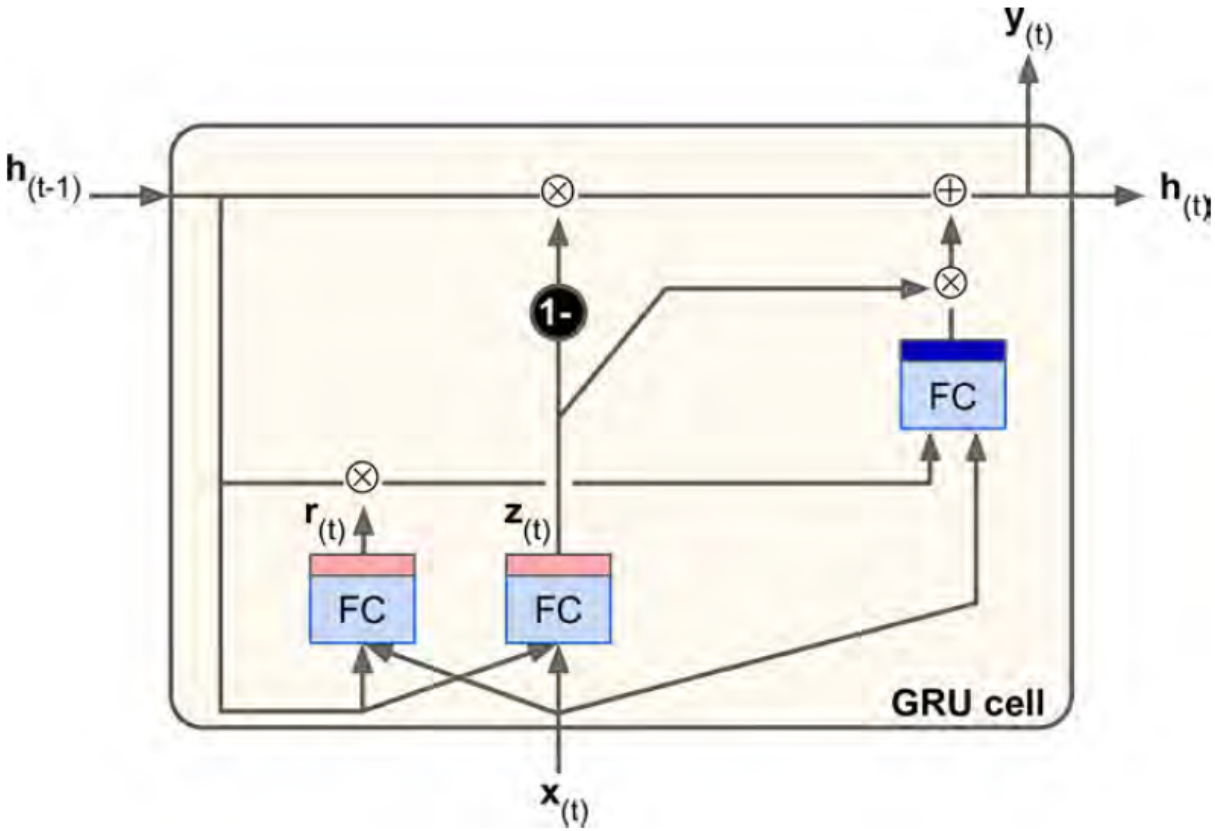
```
n_features = 2
n_timesteps = 200

model = models.Sequential()
model.add(LSTM(16, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(layers.Dense(1, activation='linear'))

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 16)                1216
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,233
Trainable params: 1,233
Non-trainable params: 0
```

Contents

Gated Recurrent Unit (GRU) Cell

GRU Cell:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Gated Recurrent Unit (GRU) Cell

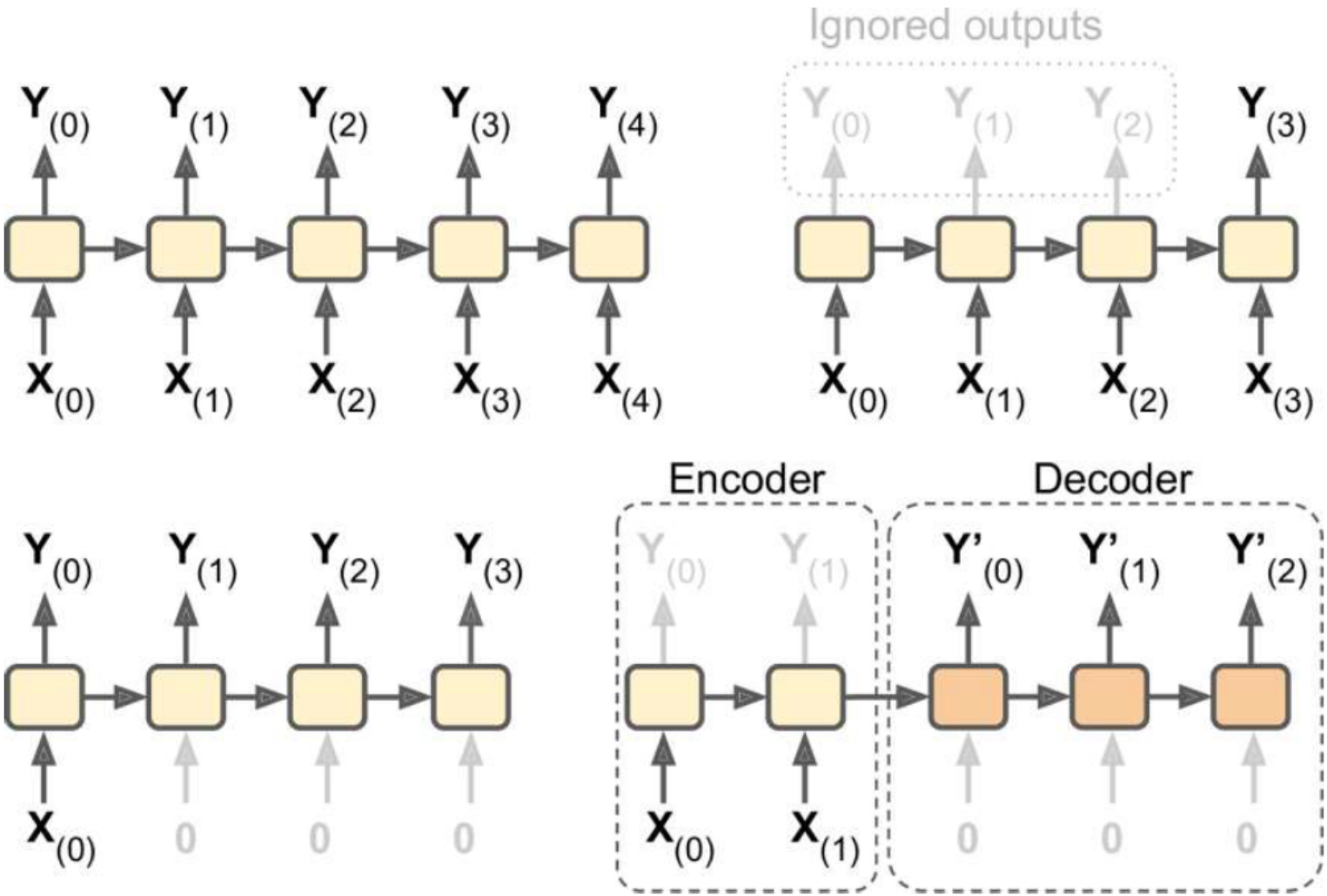
GRU Cell Model:

$$
\begin{aligned}
\mathbf{z}_{(t)} &= \sigma\left({\mathbf{W}_{xz}}^{T} \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hz}}^{T} \cdot \mathbf{h}_{(t-1)}\right) \\
\mathbf{r}_{(t)} &= \sigma\left({\mathbf{W}_{xr}}^{T} \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hr}}^{T} \cdot \mathbf{h}_{(t-1)}\right) \\
\mathbf{g}_{(t)} &= \tanh\left({\mathbf{W}_{xg}}^{T} \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hg}}^{T} \cdot \left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right)\right) \\
\mathbf{h}_{(t)} &= \left(1 - \mathbf{z}_{(t)}\right) \otimes \tanh\left({\mathbf{W}_{xg}}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{z}_{(t)} \otimes \mathbf{g}_{t}\right)
\end{aligned}
$$

Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Contents

Input and Output Sequences of RNNs



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron