Datacamp

# Lecture: Intro to DL with keras

When to use neural networks?

* Dealing with unstructured data like images
* Don’t need easily interpretable results: no need to know why the network knows it’s a cat or a dog when classifying images of cats and dogs
* Unstructured data like images
* Benefit from convolutional neural network
* Use keras (built upon tensorflow) – others?
* Sequential API model

Multi-class: classes are mutually exclusive, softmax for each class

Multi-label: classes can occur together, sigmoid for each label

Mini-batches: + networks train faster, less RAM memory required, can train on huge datasets, noise can help networks reach a lower error, escaping local minima, - more iterations need to be run, need to be adjusted, we need to find a good batch size

Batch styles reach the same value at the end, the noise between different batch styles is more or less variable

Random search (given possibles) is preferred over grid search (all possible)

Don’t use many epochs, use smaller sample of your dataset, play with batch sizes, activs, optimizers and lrs

Code can be found on:

<https://github.com/DPCscience/deep-learning/blob/master/datacamp_deeplearning.py>

remaining code (about autoencoders):

for i in range(0, 21):

# Train model for 1 epoch

h = model.fit(X\_train, y\_train, batch\_size=16, epochs=1,verbose=0)

if i%4==0:

# Get the output of the first layer

layer\_output = inp\_to\_out([X\_test])[0]

# Evaluate model accuracy for this epoch

test\_accuracy = model.evaluate(X\_test, y\_test)[1]

# Plot 1st vs 2nd neuron output

plot()

# Start with a sequential model

autoencoder = Sequential()

# Add a dense layer with the original image as input

autoencoder.add(Dense(32, input\_shape=(784 , ), activation="relu"))

# Add an output layer with as many nodes as the image

autoencoder.add(Dense(784, activation="sigmoid"))

# Compile your model

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')

# Take a look at your model structure

autoencoder.summary()

# Build your encoder

encoder = Sequential()

encoder.add(autoencoder.layers[0])

# Encode the images and show the encodings

preds = encoder.predict(X\_test\_noise)

show\_encodings(preds)

# Build your encoder

encoder = Sequential()

encoder.add(autoencoder.layers[0])

# Encode the images and show the encodings

preds = encoder.predict(X\_test\_noise)

show\_encodings(preds)

# Predict on the noisy images with your autoencoder

decoded\_imgs = autoencoder.predict(X\_test\_noise)

# Plot noisy vs decoded images

compare\_plot(X\_test\_noise, decoded\_imgs)

# Import the Conv2D and Flatten layers and instantiate model

from keras.layers import Conv2D,Flatten

model = Sequential()

# Add a convolutional layer of 32 filters of size 3x3

model.add(Conv2D(filters=32, input\_shape=(28, 28, 1), kernel\_size=3, activation='relu'))

# Add a convolutional layer of 16 filters of size 3x3

model.add(Conv2D(filters=16, input\_shape=(28, 28, 1), kernel\_size=3, activation='relu'))

# Flatten the previous layer output

model.add(Flatten())

# Add as many outputs as classes with softmax activation

model.add(Dense(10, activation='softmax'))

# Obtain a reference to the outputs of the first layer

layer\_output = model.layers[0].output

# Build a model using the model's input and the first layer output

first\_layer\_model = Model(inputs = model.input, outputs = layer\_output)

# Use this model to predict on X\_test

activations = first\_layer\_model.predict(X\_test)

# Plot the activations of first digit of X\_test for the 15th filter

axs[0].matshow(activations[0,:,:,14], cmap = 'viridis')

# Do the same but for the 18th filter now

axs[1].matshow(activations[0,:,:,17], cmap = 'viridis')

plt.show()

# Import image and preprocess\_input

from keras.preprocessing import image

from keras.applications.resnet50 import preprocess\_input

# Load the image with the right target size for your model

img = image.load\_img(img\_path, target\_size=(224, 224))

# Turn it into an array

img\_array = image.img\_to\_array(img)

# Expand the dimensions of the image

img\_expanded = np.expand\_dims(img\_array, axis = 0)

# Pre-process the img in the same way original images were

img\_ready = preprocess\_input(img\_expanded)

# Instantiate a ResNet50 model with 'imagenet' weights

model = ResNet50(weights='imagenet')

# Predict with ResNet50 on your already processed img

preds = model.predict(img\_ready)

# Decode the first 3 predictions

print('Predicted:', decode\_predictions(preds, top=3)[0])

# Split text into an array of words

words = text.split()

# Make lines of 4 words each, moving one word at a time

lines = []

for i in range(4, len(words)):

lines.append(' '.join(words[i-4:i]))

# Instantiate a Tokenizer, then fit it on the lines

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(lines)

# Turn lines into a sequence of numbers

sequences = tokenizer.texts\_to\_sequences(lines)

print("Lines: \n {} \n Sequences: \n {}".format(lines[:5],sequences[:5]))

# Import the Embedding, LSTM and Dense layer

from keras.layers import Embedding, LSTM, Dense

model = Sequential()

# Add an Embedding layer with the right parameters

model.add(Embedding(input\_dim=vocab\_size, output\_dim=8, input\_length=3))

# Add a 32 unit LSTM layer

model.add(LSTM(32))

# Add a hidden Dense layer of 32 units and an output layer of vocab\_size with softmax

model.add(Dense(32, activation='relu'))

model.add(Dense(vocab\_size, activation='softmax'))

model.summary()

def predict\_text(test\_text):

if len(test\_text.split())!=3:

print('Text input should be 3 words!')

return False

# Turn the test\_text into a sequence of numbers

test\_seq = tokenizer.texts\_to\_sequences([test\_text])

test\_seq = np.array(test\_seq)

# Get the model's next word prediction by passing in test\_seq

pred = model.predict(test\_seq).argmax(axis = 1)[0]

# Return the word associated to the predicted index

return tokenizer.index\_word[pred]









