Introduction to Deep Learning

Research Experience for Undergraduates in Ubiquitous Sensing 2018
Contents

- Introduction
- Deep Learning and Neural Networks
- Convolutional and Recurrent Neural Networks
- Transfer Learning
- Reinforcement Learning
- Hands-on tutorial
Related concepts

Artificial Intelligence

Machine Learning

Deep Learning

What is the difference between Artificial Intelligence, Machine Learning and Deep Learning? By Michael Copeland
Supervised and Unsupervised Learning

Supervised learning
- Learning a task with pre-labeled data.
- Manual task, time consuming with lots of data.
- Data augmentation.
- Problem of generalization.
- Classification and regression.

Unsupervised learning
- Learning a task with observed data not labeled.
- Clustering to understand structure and relationship in the data.
- Similarity measurement and look for clusters based on this measurement.
Deep Learning

• Resembles the way the human brain learns:
  • Connected neurons respond to external stimuli.
  • When a neuron fires, it can stimulate following neurons.
  • Learning at different levels.
  • Learn by example, extracting common patterns and adjusting parameters.
General applications of Deep Learning

- Computer vision, object detection and image recognition
- Speech recognition
- Natural language processing (NLP)
  - Natural Language Understanding
  - Natural Language Generation
- Multimedia processing and creation
- Etc.
Well-known applications of Deep Learning
Other applications of Deep Learning

- Other less known but amazing applications can be seen in:
  - 30 Amazing Applications of Deep Learning, by Yaron Hadad.
  - 8 Inspirational Applications of Deep Learning, by Jason Brownlee.
  - 27 Incredible Examples of AI and Machine Learning in Practice, by Bernard Marr
  - How AI Helps Grow 10% of U.S. Lettuce, by Allison Toh.
  - Earth to Exoplanet: Hunting for planets with Machine Learning, by Chris Shallue
A Deep Learning model resembles the way human brains are structured and the way they learn, with different learning levels.
Neural Networks: structure

- Neurons arranged in layers.
- Neurons receive signals or input data from neurons in the previous layer.
- Weighted connections.
- Based on input, each neuron computes a mathematical operation (activation function).
- Activation threshold called bias.
- Result of activation is passed to the neurons in the next layer.
Neural Networks: neuron

Weight: how important this input is to the current neuron

Activation threshold: how easy it is to activate the current neuron
Neural Networks vs. Deep Neural Networks

Neural Network:
Only 1 hidden layer

Deep Neural Network:
Multiple hidden layers
Neural Networks

- Layers stacked on top of each other.
- First layers
  - High level features
  - Simple decisions
- Higher layers
  - More complex features and decisions based on previous ones from earlier layers.
Neural Networks: how do they learn?

Trainable parameters:
- Weights $w_i$
- Bias $b_i$

- Trainable parameters start with random values.
- Training dataset contains input data and expected output.
- **Forward pass**: feed input data to the model and compute output with current parameters.
- Compare output of the model with expected output: **loss or error**.
Neural Networks: how do they learn?

- **Loss or error**: difference between current and expected output.
- Measurement of how well the model is doing.
- Perform **backpropagation** to see how much each parameter contributed to the error, and how to modify each parameter (weights and bias).
- Modify trainable parameters and start over.
Neural Networks: training and evaluation

- Repeat feed-forward and backpropagation cycle repeatedly.

- Data shuffle and split:
  - Training dataset
  - Test dataset
  - Validation dataset

- No overlapping between datasets.
Example: Hot dog or not hot dog

Input
Model
Output

Update model

NOT HOT DOG
Example: Hot dog or not hot dog

Input

Model

Output

Update model

HOT

DOG

X
Example: Hot dog or not hot dog

Input

Model

Output

HOT DOG

Update model
Example: Hot dog or not hot dog

Update model
Example: Hot dog or not hot dog

Input

Model

Output

Update model

NOT
HOT
DOG
Example: Hot dog or not hot dog

Input

Model

Output

Update model

HOT

DOG

Correct
Convolutional Neural Networks
Convolutional neural networks

- Input data are always images.
- **Regular NN** don’t work with images.
- **Convolutional NN**: use filters to extract features from images.
Convolutional neural networks

- The main layers used are:
  - **Convolutional layer**: To extract features.
  - **Pooling layer**: Decrease the dimension of the data.
  - **Fully-connected**: Connects each neuron with all the neurons in the previous layer.

*CS231n Convolutional Neural Networks for Visual Recognition*
Recurrent Neural Networks
Recurrent neural networks

- Input data is a sequence of inputs at different timesteps.
- The model “remembers” data and information in a hidden state to use it later.
Examples

Object detection and classification with Convolutional Neural Network

Predicting the next letter with Recurrent Neural Network
Transfer Learning

Apply what you learned to a new but similar problem
Transfer learning: Use the knowledge learned for some task for a new but similar one.

- Deep Neural Networks are expensive to train.
- Take a pre-trained model with high accuracy:
  - Use learned parameters as initializations.
  - Keep early layers as feature extractors.
  - Train a new classifier on top.
Reinforcement Learning
Reinforcement learning

- The model learns from its interactions with the environment.
- Learning based on changing the state of an object (model) by performing actions.
- The model receives feedback and rewards based on the state and actions performed.
- The model modifies its parameters.

Links:
- AlphaGo (2017). Documentary
- Mastering the game of Go with Deep Neural Networks and Tree Search, by D. Silver et. al. Published in Nature, 2016.
Process of training neural network

1. Data collection
2. Data pre-processing
3. Train neural network
4. Save neural network
5. Evaluate neural network
Deep Learning frameworks

- Caffe2
- TensorFlow
- PyTorch
- Chainer
- Keras
Tutorial
Tutorial: MNIST – Detect digits in images

**Purpose**: Detect and identify single digits in images.

**Steps**:
1. Data collection and processing.
2. Create and train neural network.
3. Evaluate and freeze model.
4. Integrate model in Android app.
Tutorial: MNIST – Detect digits in images

- MNIST Tutorial: Python and Keras/TensorFlow to pre-process data, create and train neural network, evaluate model and convert Keras model to TensorFlow frozen model. Code available in this Github repository.

- MNIST Tutorial: Tensorflow and Android integration, using OpenCV for image processing. Code available in this Github repository.
Tutorial: MNIST

- Download MNIST data.
- We need to split the dataset:
Tutorial: MNIST

- Helper functions:
  - `create_data_dir(dirname)`: Creates a new directory if it doesn’t exist.
  - `prepare_original_data(zip_file_path, dirname)`: Unzips the dataset in a temporal folder with the name `dirname`.
  - `move_images(old_dir, new_dir, images)`: Moves the images in the list “images” from one directory (old_dir) to a new one (new_dir).

- Process:
  1. Prepare original data: unzip data into `original_data` folder.
  2. Create `data` directory where all our data will be stored once split.
  3. Split dataset into train, test and validation dataset.
  4. Remove temporal directory `original_data`.
def split_dataset(train_portion, test_portion, original_dirname, data_dirname):
    inner_data_dir = original_dirname + '/data'
    classes = sorted(os.listdir(inner_data_dir))
    for target_class in classes:
        target_class_dir = inner_data_dir + '/' + target_class
        class_images = os.listdir(target_class_dir)
        np.random.shuffle(class_images)
        train_indx = floor(len(class_images) * train_portion)
        test_indx = floor(len(class_images) * (train_portion + test_portion))
        training_data_class = class_images[:train_indx]
        test_data_class = class_images[train_indx:test_indx]
        validation_data_class = class_images[test_indx:]
        train_dir = data_dirname + '/training/' + target_class
        test_dir = data_dirname + '/test/' + target_class
        val_dir = data_dirname + '/validation/' + target_class
        create_data_dir(train_dir)
        create_data_dir(test_dir)
        create_data_dir(val_dir)
        move_images(target_class_dir, train_dir, training_data_class)
        move_images(target_class_dir, test_dir, test_data_class)
        move_images(target_class_dir, val_dir, validation_data_class)
Tutorial: MNIST

- Once the dataset is split and data already ready. We start the training process. With Keras:
  - We create a model. In our case we’ll create a Sequential Convolutional model formed by Convolutional, Pooling, Dense and Dropout (to avoid overfitting) layers.
  - If your data are images in directories you want to load and apply data augmentation operations on the fly, you use:
    - Keras ImageDataGenerator to specify the data augmentation ops.
    - Keras ImageDataGenerator .flow_from_directory to specify the directory to read from and other configurations like batch size, target size, color mode, etc.
  - Otherwise, you have to separate input data from labels.
  - You can (and should) use Keras callbacks like Checkpoint (saves the model as it was when it returned the best evaluation metric)
  - Train the model for a given number of epochs, specifying the validation data and callbacks.
  - Save the trained model.
  - Graph metrics.
def create_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3), activation = 'relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer=keras.optimizers.Adam(lr = 0.01, decay = 0.0001), loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model
# Tutorial: MNIST data loading and augmentation

```python
training_augmentation = ImageDataGenerator(
    width_shift_range = .4,
    height_shift_range = .4,
    preprocessing_function = normalize_and_centralize)
training_generator = training_augmentation.flow_from_directory(
    'data/training/',
    batch_size = 128,
    target_size = (28,28),
    color_mode = 'grayscale',
    class_mode = 'categorical')

validation_augmentation = ImageDataGenerator(
    width_shift_range = .4,
    height_shift_range = .4,
    preprocessing_function = normalize_and_centralize)
validation_generator = validation_augmentation.flow_from_directory(
    'data/validation/',
    target_size = (28,28),
    color_mode = 'grayscale',
    class_mode = 'categorical')
```

```python
def normalize_and_centralize(image):
    signed_img = np.float32(image)
    return (signed_img - 128)/128
```

# This is a data preprocessing function that modifies the pixel values of an image so that the range is $[-1, 1]$. This helps the model learn and converge faster.
**Tutorial: MNIST train the model**

```python
model = create_model()

checkpoint_callback = ModelCheckpoint('models/model_checkpoint.h5', monitor = 'val_acc', verbose = 1, save_best_only = True, save_weights_only = False, mode = 'auto', period = 1)

# The model.fit_generator starts training the model using the training generator previously created in this script. We specify how many epochs or iterations to train for, the generator for the validation data and any callbacks we want to execute while training (in our case, only the checkpoint callback)
# This will return a training history, where we can see the accuracy of the model on the training and validation datasets at each epoch. We can use this to create a graph later.
history = model.fit_generator(
    training_generator,
    epochs = 300,
    validation_data = validation_generator,
    callbacks = [checkpoint_callback])

# After training is done, we can save the final trained model.
model.save('models/model.h5')
```
Tutorial: MNIST train

```
docker /home/raulestrada/REU

383/383 [========================] - 37s 97ms/step - loss: 1.1443 - acc: 0.7604 - val_loss: 0.7812 - val_acc: 0.7449
Epoch 00007: val_acc improved from 0.74493 to 0.74493, saving model to models/model1/checkpoint.h5
Epoch 8/300
383/383 [========================] - 32s 82ms/step - loss: 1.1276 - acc: 0.7671 - val_loss: 0.8018 - val_acc: 0.7351
Epoch 00008: val_acc did not improve from 0.74493
Epoch 9/300
383/383 [========================] - 34s 89ms/step - loss: 1.1021 - acc: 0.7716 - val_loss: 0.7650 - val_acc: 0.7568
Epoch 00009: val_acc improved from 0.74493 to 0.75678, saving model to models/model1/checkpoint.h5
Epoch 10/300
383/383 [========================] - 29s 76ms/step - loss: 1.0992 - acc: 0.7690 - val_loss: 0.7579 - val_acc: 0.7755
Epoch 00010: val_acc improved from 0.75678 to 0.77540, saving model to models/model1/checkpoint.h5
Epoch 11/300
382/383 [==========================] - 28s 84ms/step - loss: 1.0932 - acc: 0.76203 - val_loss: 0.7548 - val_acc: 0.77546
```

Model Accuracy

![Accuracy Graph](image.png)

Accuracy

- train
- test

epoch

0 50 100 150 200 250 300
Tutorial: MNIST evaluation

```
model = load_model('models/model.h5')

# Important to apply normalization and centralization since the model
# was trained with values from -1.0 to 1.0

test_data_gen = ImageDataGenerator(
    preprocessing_function = normalize_and_centralize)

# Important to apply normalization and centralization since the model
# was trained with values from -1.0 to 1.0
# an image so that the range is [-1, 1]. This helps the
# model learn and
# converge faster.

def normalize_and_centralize(image):
    signed_img = np.float32(image) - 128
    return (signed_img - 128)/128

metrics = model.evaluate_generator(test_generator, 1)
```

Found 13998 images belonging to 10 classes.
Test metrics: [0.1706324977847559, 0.9375]
raulestrada@ENB302-PC1 ~/REU
Tutorial: Keras vs Android

- TensorFlow has integration with smartphones. Keras doesn’t. Therefore, in order to use the Keras .h5 model in a smartphone application, we need to convert it first to an Android .pb frozen model.

- TensorFlow offers some utilities, but some additional configurations are needed.

```python
model = load_model('models/model.h5')

# Name of the output node
output_node_names = model.output.op.name

# We need to create a TensorFlow checkpoint file
saver = tf.train.Saver()

c_checkpoint_dir = 'models/keras_model.ckpt'
checkpoint = saver.save(K.get_session(), c_checkpoint_dir)

# Use utility to convert model to TensorFlow frozen .pb model
pred_node_names = [output_node_names]
sess = K.get_session()

c_constant_graph = graph_util.convert_variables_to_constants(sess, sess.graph.as_graph_def(), pred_node_names)

c_output = "graph.pb"

c_graph_io.write_graph(c_constant_graph, "models", c_output, as_text=False)
```
Tutorial: TensorFlow, OpenCV and Android

- TensorFlow offers some libraries to be able to run your TensorFlow models in smartphones (Android and iOS).
- TensorFlow Lite: latest libraries and framework that improves performance and model size.
- TensorFlow Mobile: for all other devices not able to use TensorFlow Lite.
- OpenCV is a library with different functions for image processing and computer vision. It also has integration with Android, although its code is C/C++.
Tutorial: Integrating TensorFlow

1. Download and install the **Android NDK**. We need it to compile TensorFlow code.
2. When creating the Android project, make sure to check "Include C++ Support".
3. In Android Studio, install CMake using the Android SDK Tools.
4. In your app, create a *libs* folder and add the **tensorflow-android library**.
5. In your app build.gradle, add `compile files('libs/libandroid_tensorflow_inference_java.jar')` inside the dependencies block. Add the *libs* folder to the source sets in that same file.
6. In your project, create an assets folder and copy the TensorFlow frozen model there.
The Android application for the tutorial takes a picture with the phone camera, crops it, pre-processes this picture and sends the data to the neural network model to identify the digit in the picture.

The full code is available in this Github repository.

Here, we’ll see the integration with the TensorFlow library.
In the class where you work with the Classifier, you need to specify some configuration options encoded in your TF frozen model.

```
private static final String TF_INFERENCE_LIBRARY_NAME = "tensorflow_inference";
// Path to frozen model
private final String MODEL_FILE = "file:///android_asset/expert-graph.pb";
// Names of nodes in the computational graph
private final String INPUT_NODE_NAME = "input"; // This is the name of your input node. Can be different!
private final String OUTPUT_NODE_NAME = "output"; // Name of your output node. Can be different!
private final String[] OUTPUT_NODES = {"output"}; // Name of your output nodes. Can be different!
private final int OUTPUT_SIZE = 10; // Output side of your model. Can be different!
// Size of input tensor
private final int INPUT_IMG_SIZE = 28*28; // Input size of your model. Can be different!

// TensorFlowInference object used to make inferences on the graph of the loaded model
private TensorFlowInferenceInterface inferenceInterface;
```
Tutorial: With TF model in the app

- First, you need to load and initialize the model:

```java
this.inferenceInterface = new TensorFlowInferenceInterface();
this.inferenceInterface.initializeTensorFlow(assetManager, MODEL_FILE);
```

- Then, you can perform the inference/classification operation.

```java
/**
 * Calls the inference on the image whose values are passed as input values, and returns the digit probabilities produced
 * by the NN model.
 * 
 * @param inputValues - The pixel values of the input image
 * @return - The digit confidence values produced by the inference of the NN model */

public float[] classifyImage(float[] inputValues) {
    this.inferenceInterface.fillNodeFloat(INPUT_NODE_NAME, new int[] {INPUT_IMG_SIZE}, inputValues);
    this.inferenceInterface.runInference(OUTPUT_NODES);
    float[] output = new float[OUTPUT_SIZE];
    this.inferenceInterface.readNodeFloat(OUTPUT_NODE_NAME, output);
    return output;
}
```
About the input and output data

- Your model will expect the **data in the format** it saw when it was being trained. That is, if you normalized your values to be between 0.0 and 1.0, then you need to do the same to evaluate your model, or to use it later (whether in a Python server or an Android application). **You must always pass the input data in the same format.**

- The results returned by the model after performing the inference are confidence/probability values for each of the classes.
Tutorial: Integrating OpenCV

- If you want to integrate the computer vision library OpenCV:
  1. Download the OpenCV-Android-SDK.
  2. Add the OpenCV Java Wrapper Module to your Android project:
     1. In Android Studio -> File -> New -> Import module
     2. Go to the OpenCV-Android-SDK folder, then SDK subfolder and select the java directory there.
  3. In your app build.gradle, add this new dependencies to the block of dependencies: compile project(':openCVLibrary341')
  4. In your smartphone, install OpenCV Manager, that contains functions your app will call while executing.
Tutorial: loading OpenCV

- Before you can use the OpenCV library functions, you need to load the library from your app. Once the library has been loaded, you can use a callback to execute some action you want.

```java
/* OpenCV loader and callback. Used for image processing purposes. */
private BaseLoaderCallback createOpenCVCallback() {
    return new BaseLoaderCallback(getApplicationContext()) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS: break;
                default: {
                    super.onManagerConnected(status);
                }
            }
        }
    };
}

/* Load the OpenCV library from the phone. In order for this to work, you need to install OpenCV manager. Once loaded, it will call the openCV callback. */
private void loadOpenCV() {
    BaseLoaderCallback callback = createOpenCVCallback();
    if (!OpenCVLoader.initDebug()) {
        Log.d("OpenCV", "Internal OpenCV library not found. Using OpenCV Manager for initialization");
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_2_0, this, callback);
    } else {
        Log.d("OpenCV", "OpenCV library found inside package. Using it!");
        callback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
    }
}
OpenCV warning

- If you use OpenCV, note when working with color images it uses BGR channels. That is, first channel is blue, then green and then red. If you trained your model with RGB images, you need to re-order the channels before passing the input data to the neural network model.
Useful and important resources!!

• Neural Networks and Deep Learning (Online book), by Michael Nielsen.
• Deep Learning with TensorFlow, by Giancarlo Zaccone et al. Packt>
• Neural Networks and Deep Learning (Coursera online course).
• Neural Networks for Machine Learning (EdX online course).
• Neural Networks tutorial using TensorFlow, on TensorFlow.
• Stanford CS231n Convolutional Neural Networks for Visual Recognition (web page).
• Stanford Convolutional Neural Networks for Visual Recognition (Sprint 2017 video lectures).
References: Deep Learning Applications

- Deep Neural Networks for Youtube Recommendations, by Covington et. Al
- Alexa Machine Learning, by Amazon Jobs
- Google Launches more realistic text-to-speech service powered by DeepMind’s AI, by James Vincent.
- Introducing DeepText: Facebook’s text understanding engine, by Abdulkader et al.
References: Neural Networks

Convolutional Neural Networks

- CS231n Convolutional Neural Networks for Visual Recognition

Recurrent Neural Networks

- Understanding LSTM networks, by Christopher Olah
- The Unreasonable Effectiveness of Recurrent Neural Networks, by Andrej Karpathy.
- Recurrent Neural Networks, on TensorFlow.
References: Learning

Transfer Learning

- Transfer Learning Using Keras – Prakash Jay on Medium. April 15, 2017
- Transfer Learning. CS231n Convolutional Neural Networks for Visual Recognition – Stanford University
- How to use transfer learning and fine-tuning in Keras and Tensorflow to build an image recognition system and classify (almost) any object – Greg Chu on Deep Learning Sandbox. May 1, 2017

Reinforcement Learning

- An Introduction to Reinforcement Learning, by Thomas Simonini
- AlphaGo (2017). Documentary
- Mastering the game of Go with Deep Neural Networks and Tree Search, by D. Silver et. al. Published in Nature, 2016.
Deploying a TF model in an Android application

- TensorFlow: Introduction to TensorFlow Mobile
- TensorFlow: Building TensorFlow on Android
- Deploying a TensorFlow Model on Android, by Yoni Tsafir on Medium