Notebooks headers

In this document, i present the whole notebook assignments headers of deeplearning.ai. It may help someone know the code contents of the course or to fast check the applications the courses has dealt with.

Table of contents

Neural Networks and Deep Learning

Python Basics with Numpy (optional assignment)

Welcome to your first assignment. This exercise gives you a brief introduction to Python. Even if you've used Python before, this will help familiarize you with functions we'll need.

Instructions:

- You will be using Python 3.
- Avoid using for-loops and while-loops, unless you are explicitly told to do so.
- Do not modify the (# GRADED FUNCTION [function name]) comment in some cells. Your work would not be graded if you change this. Each cell containing that comment should only contain one function.
- After coding your function, run the cell right below it to check if your result is correct.

After this assignment you will:

- Be able to use iPython Notebooks
- Be able to use numpy functions and numpy matrix/vector operations
- Understand the concept of "broadcasting"
- Be able to vectorize code

Let's get started!

Logistic Regression with a Neural Network mindset

Welcome to your first (required) programming assignment! You will build a logistic regression classifier to recognize cats. This assignment will step you through how to do this with a Neural Network mindset, and so will also hone your intuitions about deep learning.

Instructions:

- Do not use loops (for/while) in your code, unless the instructions explicitly ask you to do so.

You will learn to:

- Build the general architecture of a learning algorithm, including:
  - Initializing parameters
  - Calculating the cost function and its gradient
  - Using an optimization algorithm (gradient descent)
- Gather all three functions above into a main model function, in the right order.

Planar data classification with one hidden layer

Welcome to your week 3 programming assignment. It's time to build your first neural network, which will have a hidden layer. You will see a big difference between this model and the one you implemented using logistic regression.

You will learn how to:

- Implement a 2-class classification neural network with a single hidden layer
- Use units with a non-linear activation function, such as tanh
- Compute the cross entropy loss
- Implement forward and backward propagation

Building your Deep Neural Network: Step by Step

Welcome to your week 4 assignment (part 1 of 2)! You have previously trained a 2-layer Neural Network (with a single hidden layer). This week, you will build a deep neural network, with as many layers as you want!

- In this notebook, you will implement all the functions required to build a deep neural network.
- In the next assignment, you will use these functions to build a deep neural network for image classification.

After this assignment you will be able to:

- Use non-linear units like ReLU to improve your model
- Build a deeper neural network (with more than 1 hidden layer)
- Implement an easy-to-use neural network class

Notation:

- Superscript $[l]$ denotes a quantity associated with the `l.th` layer.
  - Example: $a^{[L]}$ is the `L.th` layer activation. $W^{[L]}$ and $b^{[L]}$ are the `L.th` (last) layer parameters.
- Superscript (i) denotes a quantity associated with the `i.th` example.
  - Example: $x^{(i)}$ is the `i.th` training example.
- Lowerscript $_i$ denotes the `i.th` entry of a vector.
  - Example: $a^{[l]}_i$ denotes the `i.th` entry of the `l.th` layer's activations).

Let's get started!

Deep Neural Network for Image Classification: Application

When you finish this, you will have finished the last programming assignment of Week 4, and also the last programming assignment of this course!

You will use use the functions you'd implemented in the previous assignment to build a deep network, and apply it to cat vs non-cat classification. Hopefully, you will see an improvement in accuracy relative to your previous logistic regression implementation.

After this assignment you will be able to:

- Build and apply a deep neural network to supervised learning.

Let's get started!

Improving Deep Neural Networks

Initialization

Welcome to the first assignment of "Improving Deep Neural Networks".

Training your neural network requires specifying an initial value of the weights. A well chosen initialization method will help learning.

If you completed the previous course of this specialization, you probably followed our instructions for weight initialization, and it has worked out so far. But how do you choose the initialization for a new neural network? In this notebook, you will see how different initializations lead to different results.

A well chosen initialization can:

- Speed up the convergence of gradient descent
- Increase the odds of gradient descent converging to a lower training (and generalization) error

To get started, run the following cell to load the packages and the planar dataset you will try to classify.

Regularization

Welcome to the second assignment of this week. Deep Learning models have so much flexibility and capacity that **overfitting can be a serious problem**, if the training dataset is not big enough. Sure it does well on the training set, but the learned network **doesn't generalize to new examples** that it has never seen!

You will learn to: Use regularization in your deep learning models.

Let's first import the packages you are going to use.

TensorFlow Tutorial

Welcome to this week's programming assignment. Until now, you've always used numpy to build neural networks. Now we will step you through a deep learning framework that will allow you to build neural networks more easily. Machine learning frameworks like TensorFlow, PaddlePaddle, Torch, Caffe, Keras, and many others can speed up your machine learning development significantly. All of these frameworks also have a lot of documentation, which you should feel free to read. In this assignment, you will learn to do the following in TensorFlow:

- Initialize variables
- Start your own session
- Train algorithms
- Implement a Neural Network

Programing frameworks can not only shorten your coding time, but sometimes also perform optimizations that speed up your code.

Optimization Methods

Until now, you've always used Gradient Descent to update the parameters and minimize the cost. In this notebook, you will learn more advanced optimization methods that can speed up learning and perhaps even get you to a better final value for the cost function. Having a good optimization algorithm can be the difference between waiting days vs. just a few hours to get a good result.

Gradient descent goes "downhill" on a cost function `J`. Think of it as trying to do this: At each step of the training, you update your parameters following a certain direction to try to get to the lowest possible point.

Notations: As usual, ∂J / ∂a = `da` for any variable `a`.

To get started, run the following code to import the libraries you will need.

Gradient Checking

Welcome to the final assignment for this week! In this assignment you will learn to implement and use gradient checking.

You are part of a team working to make mobile payments available globally, and are asked to build a deep learning model to detect fraud--whenever someone makes a payment, you want to see if the payment might be fraudulent, such as if the user's account has been taken over by a hacker.

But backpropagation is quite challenging to implement, and sometimes has bugs. Because this is a mission-critical application, your company's CEO wants to be really certain that your implementation of backpropagation is correct. Your CEO says, "Give me a proof that your backpropagation is actually working!" To give this reassurance, you are going to use "gradient checking".

Let's do it!

Structuring Machine Learning Projects

There were no code assignments in this course.

Convolutional Neural Networks

Convolutional Neural Networks: Step by Step

Welcome to Course 4's first assignment! In this assignment, you will implement convolutional (CONV) and pooling (POOL) layers in numpy, including both forward propagation and (optionally) backward propagation.

Notation:

- Superscript $^{[l]}$ denotes an object of the `l.th` layer.
    - Example: $a^{[4]}$ is the `4.th` layer activation. $W^{[5]}$ and $b^{[5]}$ are the `5.<sup>th</sup>` layer parameters.
- Superscript $^{(i)}$ denotes an object from the `i.th` example.
    - Example: $x^{(i)}$ is the `i.th` training example input.

- Lowerscript $_i$ denotes the `i.th` entry of a vector.
    - Example: $a^{[l]}_i$ denotes the `i.th` entry of the activations in layer `l`, assuming this is a fully connected (FC) layer.
- `n_H`, `n_W` and `n_C` denote respectively the height, width and number of channels of a given layer. If you want to reference a specific layer `l`, you can also write n_H$^{[l]}$, n_W$^{[l]}$, n_C$^{[l]}$.
- `n_H_prev`, `n_W_prev` and `n_C_prev` denote respectively the height, width and number of channels of the previous layer. If referencing a specific layer `l`, this could also be denoted n_H$^{[l-1]}$, n_W$^{[l-1]}$, n_C$^{[l-1]}$.

We assume that you are already familiar with `numpy` and/or have completed the previous courses of the specialization. Let's get started!

Convolutional Neural Networks: Application

Welcome to Course 4's second assignment! In this notebook, you will:

- Implement helper functions that you will use when implementing a TensorFlow model
- Implement a fully functioning ConvNet using TensorFlow

After this assignment you will be able to:

- Build and train a ConvNet in TensorFlow for a classification problem

We assume here that you are already familiar with TensorFlow. If you are not, please refer the *TensorFlow Tutorial* of the third week of Course 2 ("*Improving deep neural networks*").

Keras tutorial - the Happy House

Welcome to the first assignment of week 2. In this assignment, you will:

1. Learn to use Keras, a high-level neural networks API (programming framework), written in Python and capable of running on top of several lower-level frameworks including TensorFlow and CNTK.
2. See how you can in a couple of hours build a deep learning algorithm.

Why are we using Keras? Keras was developed to enable deep learning engineers to build and experiment with different models very quickly. Just as TensorFlow is a higher-level framework than Python, Keras is an even higher-level framework and provides additional abstractions. Being able to go from idea to result with the least possible delay is key to finding good models. However, Keras is more restrictive than the lower-level frameworks, so there are some very complex models that you can implement in TensorFlow but not (without more difficulty) in Keras. That being said, Keras will work fine for many common models.

In this exercise, you'll work on the "Happy House" problem, which we'll explain below. Let's load the required packages and solve the problem of the Happy House!

Residual Networks

Welcome to the second assignment of this week! You will learn how to build very deep convolutional networks, using Residual Networks (ResNets). In theory, very deep networks can represent very complex functions; but in practice, they are hard to train. Residual Networks, introduced by He et al., allow you to train much deeper networks than were previously practically feasible.

In this assignment, you will:

- Implement the basic building blocks of ResNets.
- Put together these building blocks to implement and train a state-of-the-art neural network for image classification.

This assignment will be done in Keras.

Before jumping into the problem, let's run the cell below to load the required packages.

Deep Learning & Art: Neural Style Transfer

Welcome to the second assignment of this week. In this assignment, you will learn about Neural Style Transfer. This algorithm was created by Gatys et al. (2015) (https://arxiv.org/abs/1508.06576).

In this assignment, you will:

- Implement the neural style transfer algorithm
- Generate novel artistic images using your algorithm

Most of the algorithms you've studied optimize a cost function to get a set of parameter values. In Neural Style Transfer, you'll optimize a cost function to get pixel values!

Autonomous driving - Car detection

Welcome to your week 3 programming assignment. You will learn about object detection using the very powerful YOLO model. Many of the ideas in this notebook are described in the two YOLO papers: Redmon et al., 2016 (https://arxiv.org/abs/1506.02640) and Redmon and Farhadi, 2016 (https://arxiv.org/abs/1612.08242).

You will learn to:

- Use object detection on a car detection dataset
- Deal with bounding boxes

Run the following cell to load the packages and dependencies that are going to be useful for your journey!

Sequence Models

Building a recurrent neural network - step by step

Welcome to Course 5's first assignment! In this assignment, you will implement your first Recurrent Neural Network in numpy.

Recurrent Neural Networks (RNN) are very effective for Natural Language Processing and other sequence tasks because they have "memory". They can read inputs $x^{<t>}$ (such as words) one at a time, and remember some information/context through the hidden layer activations that get passed from one time-step to the next. This allows a uni-directional RNN to take information from the past to process later inputs. A bidirection RNN can take context from both the past and the future.

Notation:

- Superscript $[l]$ denotes an object associated with the `l.th` layer.
  - Example: $a^{[4]}$ is the `4.th` layer activation. $W^{[5]}$ and $b^{[5]}$ are the `5.th` layer parameters.
- Superscript (i) denotes an object associated with the `i.th` example.
  - Example: $x^{(i)}$ is the `i.th` training example input.
- Superscript `<t>` denotes an object at the `t.th` time-step.
  - Example: $x^{<t>}$ is the input x at the `t.th` time-step. $x^{(i)<t>}$ is the input at the `t.th` timestep of example `i`.
- Lowerscript $_i$ denotes the `i.th` entry of a vector.
  - Example: $a^{[l]}_i$ denotes the `i.th` entry of the activations in layer `l`.

We assume that you are already familiar with `numpy` and/or have completed the previous courses of the specialization. Let's get started!

Dinosaur Island -- Character-level language model

Welcome to Dinosaurus Island! 65 million years ago, dinosaurs existed, and in this assignment they are back. You are in charge of a special task. Leading biology researchers are creating new breeds of dinosaurs and bringing them to life on earth, and your job is to give names to these dinosaurs. If a dinosaur does not like its name, it might go beserk, so choose wisely!
Luckily you have learned some deep learning and you will use it to save the day. Your assistant has collected a list of all the dinosaur names they could find, and compiled them into this dataset. (Feel free to take a look by clicking the previous link.)
To create new dinosaur names, you will build a character level language model to generate new names. Your algorithm will learn the different name patterns, and randomly generate new names. Hopefully this algorithm will keep you and your team safe from the dinosaurs' wrath!

By completing this assignment you will learn:

- How to store text data for processing using an RNN
- How to synthesize data, by sampling predictions at each time step and passing it to the next RNN-cell unit
- How to build a character-level text generation recurrent neural network
- Why clipping the gradients is important

We will begin by loading in some functions that we have provided for you in `rnn_utils`. Specifically, you have access to functions such as `rnn_forward` and `rnn_backward` which are equivalent to those you've implemented in the previous assignment.

Jazz improvisation with LSTM

Welcome to your final programming assignment of this week! In this notebook, you will implement a model that uses an LSTM to generate music. You will even be able to listen to your own music at the end of the assignment.

You will learn to:

- Apply an LSTM to music generation.
- Generate your own jazz music with deep learning.

Please run the following cell to load all the packages required in this assignment. This may take a few minutes.

Emojify

Welcome to the second assignment of Week 2. You are going to use word vector representations to build an Emojifier.

Have you ever wanted to make your text messages more expressive? Your emojifier app will help you do that. So rather than writing "Congratulations on the promotion! Lets get coffee and talk. Love you!" the emojifier can automatically turn this into "Congratulations on the promotion! 👍 Lets get coffee and talk. ☕ Love you! ❤️"

You will implement a model which inputs a sentence (such as "Let's go see the baseball game tonight!") and finds the most appropriate emoji to be used with this sentence (⚾). In many emoji interfaces, you need to remember that ❤️ is the "heart" symbol rather than the "love" symbol. But using word vectors, you'll see that even if your training set explicitly relates only a few words to a particular emoji, your algorithm will be able to generalize and associate words in the test set to the same emoji even if those words don't even appear in the training set. This allows you to build an accurate classifier mapping from sentences to emojis, even using a small training set.

In this exercise, you'll start with a baseline model (Emojifier-V1) using word embeddings, then build a more sophisticated model (Emojifier-V2) that further incorporates an LSTM.

Lets get started! Run the following cell to load the package you are going to use.

Word Vector Representation

Welcome to your first assignment of this week!

Because word embeddings are very computionally expensive to train, most ML practitioners will load a pre-trained set of embeddings.

After this assignment you will be able to:

- Modify word embeddings to reduce their gender bias

Let's get started! Run the following cell to load the packages you will need.

Machine Translation (Neural Machine Translation)

Welcome to your first programming assignment for this week!

You will build a Neural Machine Translation (NMT) model to translate human readable dates ("25th of June, 2009") into machine readable dates ("2009-06-25"). You will do this using an attention model, one of the most sophisticated sequence to sequence models.

This notebook was produced together with NVIDIA's Deep Learning Institute.

Let's load all the packages you will need for this assignment.

Trigger word detection

Welcome to the final programming assignment of this specialization!

In this week's videos, you learned about applying deep learning to speech recognition. In this assignment, you will construct a speech dataset and implement an algorithm for trigger word detection (sometimes also called keyword detection, or wakeword detection). Trigger word detection is the technology that allows devices like Amazon Alexa, Google Home, Apple Siri, and Baidu DuerOS to wake up upon hearing a certain word.

For this exercise, our trigger word will be "Activate." Every time it hears you say "activate," it will make a "chiming" sound. By the end of this assignment, you will be able to record a clip of yourself talking, and have the algorithm trigger a chime when it detects you saying "activate."

After completing this assignment, perhaps you can also extend it to run on your laptop so that every time you say "activate" it starts up your favorite app, or turns on a network connected lamp in your house, or triggers some other event?

In this assignment you will learn to:

- Structure a speech recognition project
- Synthesize and process audio recordings to create train/dev datasets
- Train a trigger word detection model and make predictions

Lets get started! Run the following cell to load the package you are going to use.