Deep Learning Nanodegree Syllabus

Build Deep Learning Models Today

Welcome to the Deep Learning Nanodegree program!

Before You Start

Educational Objectives: Become an expert in neural networks, and learn to implement them from scratch and using frameworks like PyTorch. Build convolutional networks for image recognition, recurrent networks for sequence and word generation, generative adversarial networks for image generation, and finally, learn to deploy these networks to a website.

Prerequisite Knowledge: Make sure to set aside adequate time on your calendar for focused work. In order to succeed in this program, we recommend having intermediate experience with Python or at least 40hrs of programming experience using libraries like NumPy and pandas, and basic knowledge of probability will be helpful. You'll also need to be familiar with calculus (multivariable derivatives) and linear algebra (matrix multiplication).

If you'd like to refresh your skills for this program, we suggest the AI with Python Nanodegree program.

Contact Info

While going through the program, if you have questions about anything, you can reach us at deeplearning-support@udacity.com.

Nanodegree Program Info

The Deep Learning Nanodegree program offers you a solid introduction to the world of artificial intelligence. In this program, you'll master fundamentals that will enable you to go further in the field, launch or advance a career, and join the next generation of deep learning talent that will help define a beneficial, new, AI-powered future for our world. You will study cutting-edge topics such as Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, Generative Adversarial Networks, and Network Deployment, and build projects in PyTorch and NumPy. You'll learn from authorities such Ian Goodfellow and Jun-Yan Zhu, inventors of types of generative adversarial networks, as well as AI experts, Sebastian Thrun and Andrew Trask. For anyone interested in this transformational technology, this program is an ideal point-of-entry.

The program is comprised of 5 courses and 5 projects. Each project you build will be an opportunity to prove your skills and demonstrate what you've learned in your lessons.

This is a term-based program that requires students to keep pace with their peers. The program is delivered in 1 term spread over 4 months. On average, students will need to spend about 12-15 hours per week in order to complete all required coursework, including lecture and project time.

Length of Program: 4 months
Frequency of Classes: Term-based
Number of Reviewed Projects: 5
Instructional Tools Available: Video lectures, Personalized project reviews, Interactive Jupyter notebooks, Text instructions, Quizzes, and Question-answering platforms: Knowledge, and Study Groups

Projects

Building a project is one of the best ways both to test the skills you've acquired and to demonstrate your newfound abilities to future employers. Throughout this Nanodegree program, you'll have the opportunity to prove your skills by building the following projects:

- Predicting Bike-Sharing Patterns
- Dog Breed Classifier
- Generate TV Scripts
- Generate Faces
- Deploy a Sentiment Analysis Model

In the sections below, you'll find a detailed description of each project along with the course material that presents the skills required to complete the project.

Project 1: Predicting Bike-Sharing Patterns

Learn neural networks basics, and build your first network with Python and NumPy. You'll define and train a multi-layer neural network, and use it to analyze real data. In this project, you will build and train neural networks from scratch to predict the number of bike-share users on a given day.

Supporting Lesson Content: Neural Networks

Lesson	Learning Outcomes
INTRODUCTION TO NEURAL NETWORKS	➔ In this lesson, you will learn solid foundations on deep learning and neural networks. You'll also implement gradient descent and backpropagation in Python.
IMPLEMENTING GRADIENT DESCENT	➔ Mat and Luis will introduce you to a different error function and guide you through implementing gradient descent using NumPy matrix multiplication.
TRAINING NEURAL NETWORKS	➔ Now that you know what neural networks are, in this lesson, you will learn several techniques to improve their training. Learn how to prevent overfitting of training data and best practices for minimizing the error of a network.
SENTIMENT ANALYSIS	➔ In this lesson, Andrew Trask, the author of Grokking Deep Learning, will show you how to define and train a neural networks for sentiment analysis (identifying and categorizing opinions expressed in text).
DEEP LEARNING WITH PYTORCH	➔ Learn how to use PyTorch for building and testing deep learning models.

Project 2: Dog Breed Classifier

In this project, you will define a Convolutional Neural Network that performs better than the average human when given the task: identifying dog breeds. Given an image of a dog, your algorithm will produce an estimate of the dog's breed. If supplied an image of a human, the code will *also* produce an estimate of the closest-resembling dog breed. Along with exploring state-of-the-art CNN models for classification, you will make important design decisions about the user experience for your app.

Supporting Lesson Content: Convolutional Neural Networks

Lesson Title	Learning Outcomes
CLOUD COMPUTING	➔ Take advantage of Amazon's GPUs to train your neural network faster. In this lesson, you'll setup an instance on AWS and train a neural network on a GPU.
CONVOLUTIONAL NEURAL NETWORK	➔ Alexis and Cezanne explain how Convolutional Neural Networks can be used to identify patterns in images and how they help us dramatically improve performance in image classification tasks.
CNNs IN PYTORCH	➔ In this lesson, you'll walk through an example Convolutional Neural Network (CNN) in PyTorch. You'll study the line-by-line breakdown of the code and can download the code and run it yourself.
WEIGHT INITIALIZATION	➔ In this lesson, you'll learn how to find good initial weights for a neural network. Having good initial weights often allows a neural network to arrive at an optimal solution, faster than without initialization.
AUTOENCODERS	➔ Autoencoders are neural networks used for data compression, image denoising, and dimensionality reduction. Here, you'll build autoencoders using PyTorch.
TRANSFER LEARNING IN PYTORCH	➔ Most people don't train their own networks on massive datasets. In this lesson, you'll learn how to finetune and use a pretrained network and apply it to a new task using transfer learning.
DEEP LEARNING FOR CANCER DETECTION	➔ In this lesson, Sebastian Thrun teaches us about his groundbreaking work detecting skin cancer with Convolutional Neural Networks.

Project 3: Generate TV Scripts

In this project, you will build your own Recurrent Networks and Long Short-Term Memory Networks with PyTorch. You'll perform sentiment analysis and generate new text, and use recurrent networks to generate new text that resembles a training set of TV scripts.

Supporting Lesson Content: Recurrent Neural Networks

Lesson	Learning Outcomes
RECURRENT NEURAL NETWORKS	➔ Ortal will introduce Recurrent Neural Networks (RNNs), which are machine learning models that are able to recognize and act on sequences of inputs.
LONG SHORT-TERM MEMORY NETWORK	➔ Luis explains Long Short-Term Memory Networks (LSTM), and similar architectures that form a memory about a sequence of inputs, over time.
IMPLEMENTATION OF RNN & LSTM	➔ Train recurrent neural networks to generate new characters, words, and bodies of text.
HYPERPARAMETERS	➔ In this lesson, we'll look at a number of different hyperparameters that are important for our deep learning work, such as learning rates. We'll discuss starting values and intuitions for tuning each hyperparameter.
EMBEDDINGS & WORD2VEC	➔ In this lesson, you'll learn about embeddings in neural networks by implementing a word2vec model that converts words into a representative vector of numerical values.
SENTIMENT PREDICTION RNN	➔ In this lesson, you'll learn to implement a recurrent neural network for predicting sentiment. This is intended to give you more experience building RNNs.

Project 4: Generate Faces

Learn to understand Generative Adversarial Networks with the model's inventor, Ian Goodfellow. Then, apply what you've learned in this project and implement a Deep Convolutional GAN. This DCGAN is made of a pair of multi-layer neural networks that compete against each other until one learns to generate realistic images of faces.

Supporting Lesson Content: Generative Adversarial Networks

Lesson	Learning Outcomes
GENERATIVE ADVERSARIAL NETWORK	➔ Ian Goodfellow, the inventor of GANs, introduces you to these exciting models. You'll also implement your own GAN on a simple dataset.
DEEP CONVOLUTIONAL GANs	➔ Implement a Deep Convolutional GAN to generate complex, color images of house numbers.
PIX2PIX & CYCLEGAN	➔ Jun-Yan Zhu and Cezanne lead you through a CycleGAN formulation that can learn from unlabeled sets of images.

Project 5: Deploy a Sentiment Analysis Model

In this project, you will train and deploy your own PyTorch sentiment analysis model using Amazon SageMaker on AWS. This model will be trained to do sentiment analysis on movie reviews (positive or negative reviews). You'll build the model, deploy it, and create a gateway for accessing this model from a website.

Supporting Lesson Content: Model Deployment

Lesson Title	Learning Outcomes
INTRODUCTION TO DEPLOYMENT	➔ Learn where cloud deployment is used in industry and about various methods for deployment (websites, apps, etc.). Become familiar with cloud deployment terminology.
DEPLOY A MODEL	➔ Deploy a model using Amazon SageMaker and learn to apply built-in algorithms, like XGBoost, to a variety of tasks.
CUSTOM MODELS & WEB HOSTING	➔ In this lesson, you'll train and deploy your own PyTorch model. Then, see how to define a gateway using SageMaker to allow for outside-access to your model. See how your model responds to user input.
MODEL MONITORING	➔ In this lesson, learn how to interpret log messages and monitor the behavior of your model over time. See how to implement an A/B test, in SageMaker, to evaluate the performance of two different models.
UPDATING A MODEL	➔ Developing a machine learning model is an iterative process. Learn how to look at indicators like data distribution to see if you should update a model.