Machine Learning in Production

Andrew Ng (DeepLearning.AI)

Table of Contents

[1. Week 1 1](#_Toc186469735)

[1.1 Deployment Example 1](#_Toc186469736)

[1.1.1 Requirements surrounding ML Infrastructure 3](#_Toc186469737)

[1.2 ML Project Lifecycle 3](#_Toc186469738)

[1.3 Case Study: Speech recognition Deployment stage 4](#_Toc186469739)

[1.3.1 Scoping 4](#_Toc186469740)

[1.3.2 Data 4](#_Toc186469741)

[1.3.3 Modelling 5](#_Toc186469742)

[1.3.4 Deployment 6](#_Toc186469743)

# Week 1

## 1.1 Deployment Example

A diagram of a cloud computing process

Description automatically generated

Let's say you're using computer vision to inspect phones coming off a manufacturing line to see if there are defects on them. This phone shown on the left doesn't have any scratches on it, but if there was a scratch or crack or something, a computer vision algorithm would hopefully be able to find this type of scratch or defect and maybe put a bounding box around it as part of quality control. If you get a dataset of scratched phones, you can train a CV algorithm, maybe a neural network to detect these types of defects. But what do you now need to do to put this into production deployment? This would be an example of how you could deploy a system like this. You might have an edge device. By edge device, I mean, a device that is living inside the factory that is manufacturing these smart phones. That edge device would have a piece of inspection software, whose job it is to take a picture of the phone, see if there's a scratch, and then decide on whether this phone is acceptable or not. This is commonly done in factories. This is called automated visual defect inspection. What the inspection software does is, it will control a camera that will take a picture of the smartphone as it rolls off the manufacturing line, and it then has to make an API call to pass this picture to a prediction server, and the job of the prediction server is to accept these API calls, you receive an image, make a decision as to whether or not this phone is effective and return this prediction, and then the inspection software can make the appropriate control decision, whether to let this phone move on in the manufacturing line or whether to shove it to the side because it was defective and not acceptable. After you have trained a learning algorithm, maybe you trained a neural network to take as input x, pictures of phones and map them to y, predictions about whether the phone is defective or not, you still have to take this machine learning model, put it in a prediction server, set up API interfaces, and really write all of the rest of the software in order to deploy this learning algorithm into production. This prediction server is sometimes in the Cloud, and sometimes the prediction server is at the edge as well. In fact, in manufacturing, we use edge deployments a lot because you can't have your factory go down every time your Internet access goes down. But Cloud deployments with prediction server is a server in the Cloud is also used for many applications. Let's say you write all the software. What could possibly go wrong? It turns out that just because you've trained a learning algorithm that does well on your test set, which is to be celebrated. It's great when you do well when you hold a test set. Unfortunately, reaching that milestone doesn't mean you're done. There can still be quite a lot of work and challenges ahead to get a valuable production deployment running. For example, let's say your training set has images that look like this. There's a good phone on the left, the one in the middle has a big scratch across it, and you've trained your learning algorithm to recognize that phones like this on the left are okay, meaning there are no defects, and maybe draw bounding boxes around scratches or other defects it finds in phones. When you deploy it in the factory, you may find that the low or high production deployments gives you back images like this, much darker ones, because the lighting conditions in the factory have changed for some reason compared to the time when the training set was collected. This problem is sometimes called concept drift or data drift. Ee want to make sure that we don't just do well on the holdout test set, but that our systems create value in a practical production deployment environment.

### 1.1.1 Requirements surrounding ML Infrastructure

A diagram of a software code

Description automatically generated with medium confidence

## 1.2 ML Project Lifecycle

A diagram of a lifecycle

Description automatically generated

These are the major steps of a Machine Learning project. First is scoping, in which you must define the project or decide what to work on. What exactly do you want to apply Machine Learning to, and what is X and what is Y. After having chosen the project, you then must collect data or acquire the data you need for your algorithm. This includes defining the data and establishing a baseline, and then also labeling and organizing the data. There are some best practices for this that are non-intuitive that you learn more about later in this week. After you have your data, you then must train the model. During the model phase, you must select and train the model and perform error analysis. You might know that Machine Learning is often a highly iterative task. During the process of error analysis, you may go back and update the model, or you may also go back to the earlier phase and decide you need to collect more data as well. As part of error analysis before taking a system to deployments, I'll often also carry out a final check, maybe a final audit, to make sure that the system's performance is good enough and that it's sufficiently reliable for the application. Sometimes, an engineer thinks that when you deploy a system, you're done. I now tell most people, when you deploy a system for the first time, you are maybe about halfway to the finish line, because it's often only after you turn on live traffic that you then learn the second half of the important lessons needed to get the system to perform well. To carry out the deployment step, you must deploy it in production, write the software needed to put into production, then also monitor the system, track the data that continues to come in, and maintain the system. For example, if the data distribution changes, you may need to update the model. After the initial deployment, maintenance will often mean going back to perform more error analysis and maybe retrain the model, or it might mean taking the data you get back. Now that the system is deployed and is running on live data and feeding that back into your dataset to then potentially update your data, retrain the model, and so on until you can put an updated model into deployment.

## 1.3 Case Study: Speech recognition Deployment stage

A diagram of a software development process

Description automatically generated

Let's use the machine learning project life cycle to set through a speech recognition example so you can understand all the steps needed to build and deploy such a system.

### 1.3.1 Scoping

The first step of that was scoping must first define the project and just decide to work on speech recognition, say for voice search as part of defining the project. That also encourage you to try to estimate the key metrics. This will be very problem dependent. Almost every application will have its own unique set of goals and metrics. In the case of speech recognition, some things we care about are: How accurate is the speech system? What's the latency? How long does the system take to transcribe speech? and what is the throughput? How many queries per second can we handle? And then, if possible, you might also try to estimate the resources needed. So how much time, how much compute how much budget, as well as the timeline. How long will it take to carry out this project?

### 1.3.2 Data

The next step is the data stage where you must define the data and establish a baseline and label and organize the data. One of the challenges of practical speech recognition systems is the data label consistency. A single clip can have 3 valid transcriptions:

* ‘Um, today’s weather’
* ‘Um… today’s weather’
* ‘Today’s weather’ - Don't want to transcribe noise

 It turns out that any of these three ways of transcribing the audio is just fine. I would probably prefer either the first or the second, not the third. But what would hurt your learning algorithm's performance is if one third of the transcriptionists used the first, one third, the second and one third, the third way of transcribing. Because then your data is inconsistent and confusing for the learning algorithm. So spotting and correcting inconsistencies like that, maybe just asking everyone to standardize on this first convention, that can have a significant impact on your learning algorithm’s performance.

Other examples of data definition questions for an audio clip like ‘today's weather’ is how much silence you want before and after each clip after a speaker has stopped speaking. Do you want to include another 100 milliseconds of silence after that? Or 300 milliseconds or 500 milliseconds?

Or how do you perform volume normalization? Some speakers speak loudly, some are less loud and then there's a tricky case of if you have a single audio clip with some loud volume and some soft volume, all within the same audio clip. So how do you perform volume normalization. Questions like all of these are data definition questions.

A lot of progress in machine learning, that is a lot of machine learning research, was driven by researchers working to improve performance on benchmark datasets. In that model, researchers might download the dataset and just work on that fixed dataset. And this mindset has led to tremendous progress in machine learning so no complaints at all about this mindset, but if you are working on a production system then you don't have to keep the dataset fixed. I often edit the training set or even the test set if that's what's needed to improve the data quality to get a production system to work better. So, what are practical ways to do this effectively, not an ad hoc way, but systematic frameworks for making sure you have high quality data?

### 1.3.3 Modelling

After you've collected your data set, the next step is modeling, in which you must select and train the model and perform error analysis. The three key inputs that go into training a machine learning model are:

* The code that is the algorithm or the neural network model architecture that you might choose
* You also must pick hyperparameters
* There's the data and running the code with your hyperparameters on your data gives you the machine learning model.

I found that in a lot of research work or academic work you tend to hold the data fixed and vary the code and maybe vary the hyperparameters to try to get good performance. In contrast, I found that for a lot of product teams, if your main goal is to just build and deploy a working valuable machine learning system, I found that it can be even more effective to hold the code fixed and to instead focus on optimizing the data and maybe the hyperparameters for a high performing model.  And I found that rather than taking a model-centric view of trying to optimize the code to your fixed data set for many problems, you can use an open-source implementation of something you download off GitHub and instead just focus on optimizing the data. So, during modeling, you must select and train some model architecture.

Error analysis can then tell you where your model still falls short. It can tell you how to systematically improve your data, maybe improve the code too. But often if error analysis can tell you how to systematically improve the data, that can be a very efficient way for you to get to a high accuracy model. And part of the trick is you don't need to collect more data all the time, it can be expensive, but if error analysis can help you be more targeted in exactly what data to collect, that can help you be much more efficient in building an accurate model.

### 1.3.4 Deployment

 Finally, when you have trained the model and when error analysis seems to suggest it's working well enough, you're then ready to go into deployment. Take speech recognition. This is how you might deploy a speech system. You have a mobile phone. This would be an edge device with software running locally on your phone. That software taps into the microphone to record what someone is saying. Maybe for voice search and in a typical implementation of speech recognition, you would use a VAD module. VAD stands for a voice activity detection. And it's usually a relatively simple algorithm. Maybe a learning algorithm and the job of the VAD allows the smartphone to select out just the audio that contains hopefully someone speaking so that you can send only that audio clip to your prediction server. And in this case maybe the prediction server lives in the Cloud. This would be a common deployment pattern. The prediction server then returns both

* the transcripts to the user, so you can see what the system thinks you said
* And the search results if you're doing voice search.

And the transcript and search results are then displayed in the frontend code running on your mobile phone. So, implementing this type of system would be the work needed to deploy a speech model in production even after it's running though you still need to monitor and maintain the system. So, here's something that happened to me once: My team had built a speech recognition system, and it was trained mainly on adult voices. We pushed into production, ran it in production and we found that over time more and more young individuals, teenagers, you know, sometimes even younger seem to be using our speech recognition system and the voices of very young individuals just sound different. And so, my speech system's performance started to degrade. We just were not that good at recognizing speech as spoken by younger voices. And so, we had to go back and find a way to collect more data and other things to fix it. So, one of the key challenges when it comes to deployment is **concept drift or data drift**, which is what happens when the data distribution changes, such as there are more and more young voices being fed to the speech recognition system. And knowing how to put in place appropriate monitors to spot such problems and then also how to fix them in a timely way is a key skill needed to make sure your production deployment creates a value you hope it will.