What is Online Machine Learning?

Max Pagels Follow
Apr 19, 2018 · 7 min read

During the start of my career, I was fortunate enough to work on a subfield of machine learning known as **online learning** (also known as **incremental** or **out-of-core learning**). Compared to "traditional" ML solutions, online learning is a fundamentally different approach, one that embraces the fact that learning environments can (and do) change from second to second. It's tricky to get right, but when applied correctly, the results you can achieve with online learning are nothing short of remarkable. In this post, I'll give a quick introduction to the technique.

Update 27/09/2019: lots of people have asked if there exist any purpose-built incremental learning libraries. Yes! Vowpal Wabbit is extremely powerful, and has been around for quite a while. For those that prefer scikit-inspired APIs, take a look at creme.

. . .

Our <insert name of AI product> gets better the more you use it!

Ever read about about some AI product or platform that claims the above, promising to get better over time? In the vast majority of cases, what's actually under the bonnet is, well, a bit underwhelming.

In a general sense, you need two things for machine learning: data and a suitable learning algorithm. The learning algorithm learns from/trains on your data and

produces a (hopefully) accurate model, typically used for prediction on new data. I'm oversimplifying things, but that's the core idea.

ML models, save for some exceptions, are static things. They are essentially collections of parameters. After you've trained a model, its parameters don't change. From a technical perspective, that's good news: if you want to serve predictions over an API, you can instantiate several instances of a model, place a load balancer on top of them, and pop round the pub for a pint to congratulate yourself on a job well done. Since model parameters don't change, you don't need to synchronise between model instances. It's horizontally scalable, almost trivially so. And if we're being honest, horizontally scalable is the best type of scalable.

But what about new data? If we just train a model once and never touch it again, we're missing out the information more data could provide us. This is especially important in environments where behaviours change quickly. Online shopping is one such environment: a product that is popular today may be all but forgotten tomorrow.

In order to react to new data and make an AI that learns over time, ML practitioners typically do one of two things:

1. They manually train on newer data, and deploy the resulting model once they are happy with its performance
2. They schedule training on new data to take place, say, once a week and automatically deploy the resulting model

99,99 per cent of the time, when someone claims "our AI gets better the more you use it", what they really mean is that they've gone for approach 2), scheduling the training of new models. In its most simple form, this could very literally be one line in a crontab file.

All this is well and good, apart from one glaring problem: even if you train new models each week, or even each day, you're still lagging behind. Your model is never fully up-to-date with current events, because it's trained on stale data. Ideally, what you want is **a model that can learn from new examples in something close to real time**. Not only *predict* in real time, but *learn* in real time, too.

. . .

I love using scikit-learn to play around with ML. All of the different algorithms in scikit-learn implement the same simple API, making it very easy to get up and running. For regression problems, I usually start with the SGDRegressor class. Here's how to train a simple model on dummy data (taken straight from the scikit documentation):

```
import numpy as np
from sklearn import linear_model
n_samples, n_features = 10, 5
y = np.random.randn(n_samples)
X = np.random.randn(n_samples, n_features)
clf = linear_model.SGDRegressor()
clf.fit(X, y)
```

The `fit()` method does all the training magic, resulting in a model we can use for prediction (in this case, predicting on one example):

```
clf.predict(np.random.randn(1, n_features))
```

In addition to the `fit()` method, the SGDRegressor also provides a `partial_fit()` method, so that you can incrementally train on small batches of data. In fact, all learning algorithms that are compatible with standard optimisation algorithms like (stochastic) gradient decent, adam, RMSprop, and so on have this capability.

Out of curiosity, let's see how long it takes to train on a single example using `partial_fit()`:

```
import numpy as np
from sklearn import linear_model

n_samples, n_features = 1, 500
y = np.random.randn(n_samples)
X = np.random.randn(n_samples, n_features)
clf = linear_model.SGDRegressor()

import time
```

```
start_time = time.time()
clf.partial_fit(X, y)
elapsed_time = time.time() - start_time
print(elapsed_time)

>>> 0.0008502006530761719
```

0.0009 seconds on my machine. That's quite fast. In fact, if we were to put our SGDRegressor behind a REST API and train on an example each time an HTTP request was made, factoring in, say, 10ms for request processing, we could handle about 520 requests a second, or about *45 million* requests a day.

This prompts an interesting question: given these numbers, would it be possible to learn from new examples in something close to real time? **Therein lies the potential of online learning: the second we see a new example, let's learn from it as fast as we can.** The faster, the better. In fact, because speed trumps everything else in online learning, we typically use simple learning algorithms over complex ones like neural networks. We strive for millisecond-level learning; everything else comes second.

ML purists might scoff at the idea of online algorithms for real time learning. Training a model can go wrong in lots of different ways: the algorithm itself might not be suitable, the model might fail to generalise well, the learning rate might be wrong, the regularisation might be too low or too high…the list goes on. Why on earth would we even attempt to learn immediately when there are no guarantees on what might happen?

The answer is simple: no matter how good a model is, or how much data you feed it, a model is still an imperfect representation of an environment. To make the best possible decisions right *now,* we can't afford to have a model that only knows about things that happened yesterday.

Consider the following example. Let's say we run a news website. We personalise our news by collecting data on what was clicked or not clicked, and by whom. Based on this information, we predict the types of news different visitors might like, and serve them relevant items.

One day, out of the blue, word gets out that the government is issuing a state of emergency, and will hold a press conference in an hour. Suddenly, *everyone* is interested in domestic affairs — even those who typically only read about sports or look at the funnies. When presented with a news piece about the conference, a huge percentage of the audience clicks it to learn more.

If you had gone the traditional route and batch trained your recommendation engine once a day, it would still be stuck offering the same type of content, even though the underlying world changed dramatically[1]. You *should* be serving up domestic news right now, but aren't because your system is too slow.

It gets worse: the following day, after the press conference and following a new training cycle, your engine would start actively recommending domestic news which, after 24 hours, isn't necessarily interesting any more. It's made two mistakes, both because it can't react fast enough.

That's the power of online learning: done properly, it can react in minutes or even seconds. With it, there is no such thing as "yesterday's news".

. . .

Bolding for emphasis, **implementing real time learning isn't easy**. If you place some learning algorithm behind an API and, god forbid, open it up to the Internet, there's an almost limitless number of ways it can go wrong. You might get lots of feedback (examples) from one thing but not another, leading to a skewed classes problem. You might've set your learning rate too high, causing your model to forget everything that happened more than a second ago. You might overfit, or underfit. Someone might DDoS your system, screwing up learning in the process. Online learning is prone to *catastrophic interference* — more so than most other techniques.

Online learning also requires an entirely different approach in terms of technical architecture. Since a model can, and will, change from second to second, you can't just instantiate several instances like you can with traditional techniques. It's not horizontally scalable. Instead, you are forced to have a single model instance that eats new data as fast as it can, spitting out sets of learned parameters behind an API. And the

second that one set in one process gets replaced by a new one, all other processes must follow suit immediately. It's an engineering challenge, because the most important part (the model) is only vertically scalable. It may not even be feasible to distribute between threads.

Learning immediately also requires fast access to new data. If you're lucky enough to get all the data you need for a single training example as part of an API call, you're good to go. But if something is not available client-side, you need to be able to grab that data from somewhere in milliseconds. Typically, that means using an in-memory store like Redis. "Big data" processing frameworks aren't of much help. If you want to to both batch and online learning, Spark isn't enough. If you do only online learning, Spark is useless.

I could go on for hours about the technical aspects, but the bottom line is this: online learning is an ML solution unlike any other, and it requires a technical approach unlike any other.

. . .

Summing up online learning isn't easy. It's not a single learning algorithm: in fact, lots of algorithms can learn online. It's also not fundamentally different in terms of how learning happens: you can use the same optimisation steps you always do. It doesn't even have a bombastic, sci-fi sounding name.

What online learning is is a fundamentally different way of approaching machine learning. It's an approach that embraces change, no matter how drastic. Its existence is predicated on the belief that since everything is in flux, we should stop seeking stationarity and instead start living in the moment.

A special thanks to Data Scientist Jarno Kiviaho, who I consider to be one of Finland's top authorities on this topic.

If you fancy playing around with the (admittedly simple) code in this post, it's available as a GitHub gist: https://gist.github.com/maxpagels/b9c9001f7e5b28a5742b81b02f7704e2

[1]: You could, theoretically, have a feature indicating a drastic event like this, but it's impossible to account for everything.

Machine Learning | Data Science | Reinforcement Learning | Supervised Learning