image shutterstock

Using fastai callbacks for efficient model training

Harnessing the power of early stopping and model save callbacks

Mark Ryan Jul 31 · 6 min read ★

When you train a deep learning model you want to get the most out of the resources that you are using to train the model. If you're using an environment like Paperspace Gradient where you pay by the hour, time is literally money. If you can train your model in less time you will save money. Even if you are using Colab and the meter isn't running, your own time is still valuable, so it's worthwhile to know how to get the most out of the time and capacity that you have available to train your deep learning model. In this article I'll describe two callbacks that you can use in fastai to ensure that your model training is as efficient as possible. The example that I describe in this article is explained in more detail in my Packt book Deep Learning with fastai Cookbook.

Two problems with training deep learning models

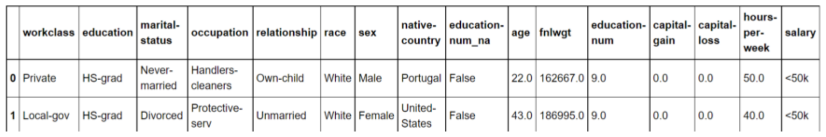
fastai shares a characteristic with Keras, the other commonly used high-level framework for deep learning. In both frameworks, the model training process is not efficient out of the box. By default, the model training process has the following problems:

1. The training process will keep going for as many epochs as you specify in the fit statement, even if the metric that you want to optimize for is no longer improving.
2. The model that you get at the end of the training process has the weights from the final epoch, even if there was an earlier epoch whose weights would result in a model with better performance.

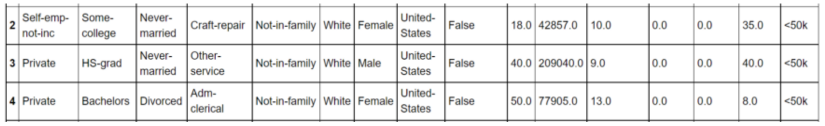
Luckily, both fastai and Keras include solutions to both of these problems in the form of **callbacks**. In the remainder of this article, I'll explain these callbacks in fastai. See chapter 6 of my Manning book Deep Learning with Structured Data for a similar description of how to use callbacks to control the model training process in Keras.

Baseline: train the model with no callbacks

To see the impact of fastai callbacks, we'll begin by training a model with no callbacks. This model is trained on the fastai curated dataset ADULT_SAMPLE, which contains details about individuals such as their years of education, marital status and occupation class.



	workclass	education	marital-status	occupation	relationship	race	sex	native-country	education-num_na	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary
0	Private	HS-grad	Never-married	Handlers-cleaners	Own-child	White	Male	Portugal	False	22.0	162667.0	9.0	0.0	0.0	50.0	<50k
1	Local-gov	HS-grad	Divorced	Protective-serv	Unmarried	White	Female	United-States	False	43.0	186995.0	9.0	0.0	0.0	40.0	<50k



2	Self-emp-not-inc	Some-college	Never-married	Craft-repair	Not-in-family	White	Female	United-States	False	18.0	42857.0	10.0	0.0	0.0	35.0	<50k
3	Private	HS-grad	Never-married	Other-service	Not-in-family	White	Male	United-States	False	40.0	209040.0	9.0	0.0	0.0	40.0	<50k
4	Private	Bachelors	Divorced	Adm-clerical	Not-in-family	White	Female	United-States	False	50.0	77905.0	13.0	0.0	0.0	8.0	<50k

ADULT_SAMPLE dataset

The goal of the model trained on the ADULT_SAMPLE dataset is to predict whether or not a given individual has a salary above or below 50 k.

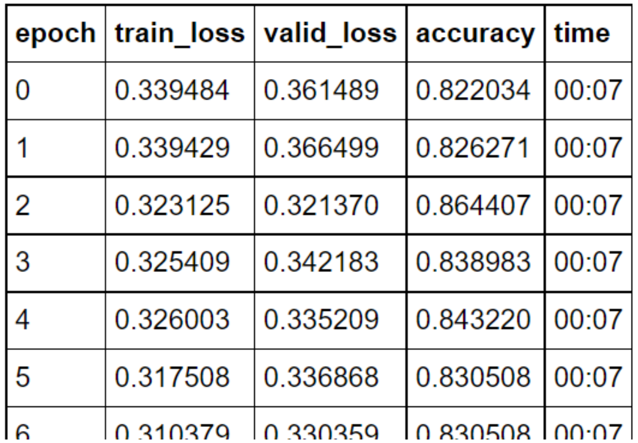
We start by training the model with no callbacks:

```python
set_seed(dls,x=42)
learn = tabular_learner(dls,layers=[200,100], metrics=accuracy)
learn.fit_one_cycle(10)
```

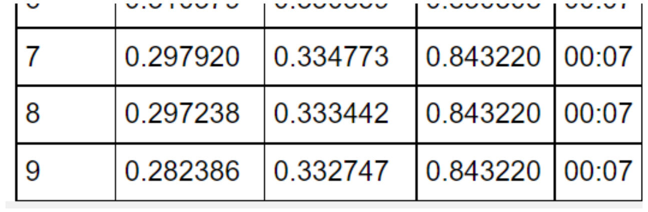
no_callbacks.py hosted with ♡ by GitHub   view raw

Note the call to set_seed(). We do this to get consistent results for each epoch between distinct training runs. This allows us to do an apples-to-apples comparison between training runs and highlight the impact of callbacks. If we didn't call set_seed() we would get inconsistent results between runs. For example, the accuracy on epoch 2 would be different for each training run.

Here is the output of the fit statement applied to the learner object with no callbacks:



epoch	train_loss	valid_loss	accuracy	time
0	0.339484	0.361489	0.822034	00:07
1	0.339429	0.366499	0.826271	00:07
2	0.323125	0.321370	0.864407	00:07
3	0.325409	0.342183	0.838983	00:07
4	0.326003	0.335209	0.843220	00:07
5	0.317508	0.336868	0.830508	00:07
6	0.310379	0.330359	0.830508	00:07



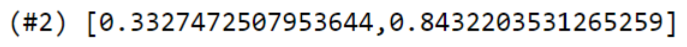
7	0.297920	0.334773	0.843220	00:07
8	0.297238	0.333442	0.843220	00:07
9	0.282386	0.332747	0.843220	00:07

Output of the fit statement for the baseline with no callbacks

The accuracy increased up to epoch 2 and then it drops in epoch 3 and oscillates for the remaining epochs up to epoch 9. When we run validate() on the learner object:

```
learn.validate()
```

The output shows that the accuracy for the trained model at the end of the training run is the accuracy for epoch 9:



```
(#2) [0.3327472507953644,0.8432203531265259]
```

output of validate() for the model trained with no callbacks

With no callbacks we hit both problems mentioned at the beginning of this article: the training run keeps going after the model has stopped improving, and the performance of the trained model that come out of the training run is worse than the best performance seen during the training run.

Add an early stopping callback to stop the training run when the model performance stops improving

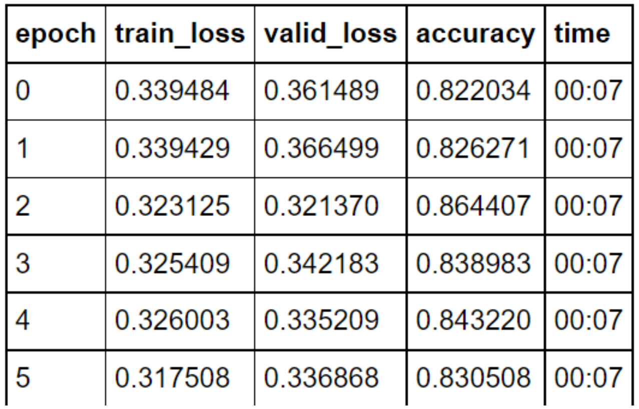
Now that we have established a baseline with no callbacks, let's add an early stopping callback to stop the training run once the model performance stops improving.

We'll use the same dataloaders object that we used for training the baseline model, but this time we will specify an early stopping callback in the fit statement:

```
set_seed(dls,x=42)
learn_es = tabular_learner(dls,layers=[200,100], metrics=accuracy)
learn_es.fit_one_cycle(10,cbs=EarlyStoppingCallback(monitor='accuracy', min_delta=0.01, patience=
```

early_stopping.py hosted with ♡ by GitHub view raw

The output of the fit statement now shows that the training run only goes up to epoch 5, even though 10 epochs were specified. The patience parameter was set to 3, so after each high-water mark for accuracy, the training process gets 3 epochs to improve. If it doesn't improve within 3 epochs after the high-water mark, the training process stops automatically.



epoch	train_loss	valid_loss	accuracy	time
0	0.339484	0.361489	0.822034	00:07
1	0.339429	0.366499	0.826271	00:07
2	0.323125	0.321370	0.864407	00:07
3	0.325409	0.342183	0.838983	00:07
4	0.326003	0.335209	0.843220	00:07
5	0.317508	0.336868	0.830508	00:07

Output of the fit statement with an early stopping callback

After the high-water mark for accuracy in epoch 2 the accuracy does not get better, so the training run stops 3 epochs later, after epoch 5.

When we run validate() on the learner object, the output shows that the accuracy for the model coming out of the training process is again the accuracy of the final epoch of the training run, even though earlier epochs had higher accuracy:

```
(#2) [0.33686837553977966,0.8305084705352783]
```

output of validate() for the model trained with an early stopping callback

Saving the best set of weights from the training process

Adding the early stopping callback was an improvement over the baseline because we didn't run so many unproductive epochs after the accuracy was no longer improving. However, we still got less-than-optimal accuracy in the final trained model, so there's still room for improvement. To get optimal accuracy, we will add a model saving callback to ensure that the trained model coming out of the training process has the best accuracy from the training run.

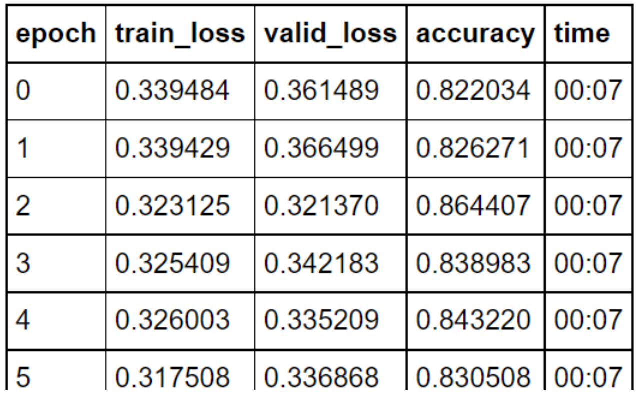
Again we'll use the same dataloaders object that we used for training the baseline model, but this time we will specify two callbacks in the fit statement. We will also set the path for the learner object to a writeable directory so that the model can be saved during the training process:

cbs=[EarlyStoppingCallback(monitor='accuracy'),SaveModelCallback(monitor='accuracy')]

```python
set_seed(dls,x=42)
learn_es_sm = tabular_learner(dls,layers=[200,100], metrics=accuracy)
keep_path = learn_es_sm.path
# set the model path to a writeable directory. If you don't do this, the code will produce an erro
#learn_es_sm.path = Path('/notebooks/temp/models')
learn_es_sm.path = Path(model_path)
learn_es_sm.fit_one_cycle(10,cbs=[EarlyStoppingCallback(monitor='accuracy', min_delta=0.01, patie
# reset the model path
learn_es_sm.path = keep_path
```

es_and_sm_callbacks.py hosted with ♡ by GitHub view raw

Just like the model in the previous section, the output of the fit statement shows that the training run only goes up to epoch 5, even though 10 epochs were specified.

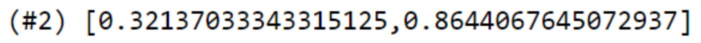


epoch	train_loss	valid_loss	accuracy	time
0	0.339484	0.361489	0.822034	00:07
1	0.339429	0.366499	0.826271	00:07
2	0.323125	0.321370	0.864407	00:07
3	0.325409	0.342183	0.838983	00:07
4	0.326003	0.335209	0.843220	00:07
5	0.317508	0.336868	0.830508	00:07

Output of the fit statement with an early stopping callback and a model save callbacs

Again, thanks to the early stopping callback, the training process stops automatically 3 epochs after the accuracy ceases to improve on the high-water mark in epoch 2.

What about the accuracy of the trained model at the end of the training process? Running validate() on the learner object for this model shows that the accuracy is not the accuracy from the final epoch of the training run. This time the accuracy matches what we saw in the high-water mark of epoch 2.



```
(#2) [0.32137033343315125,0.8644067645072937]
```

output of validate() for a model trained with an early stopping callback and a model save callback

With both callbacks, early stopping and model saving, we address both of the problems mentioned at the beginning of this article:

- Thanks to the early stopping callback, we avoid doing a bunch of epochs that are unproductive because the accuracy is no longer improving.
- Thanks to the model save callback, the trained model that comes out of the training process has the best accuracy that we saw during the training process.

Conclusion

By using the early stopping and model save callbacks in fastai you can get the most out of the training cycle. You will avoid burning up capacity on epochs where the model is not improving, and you can be sure that at the end of the training cycle you have the model with the best performance.

Here are some resources related to this article:

- Code examined in this article: https://github.com/PacktPublishing/Deep-Learning-with-fastai-

Cookbook/blob/main/ch8/training_with_tabular_datasets_callbacks.ipynb

- Book site: https://www.packtpub.com/product/deep-learning-with-fastai-cookbook/9781800208100
- Video on this subject: https://youtu.be/qkRok0e3yvs

Fastai Keras Deep Learning Callback Model Training