## AGENDA

1. **Fastai Framework :**
   **(Tabular, Vision and Collaborative filtering)**

2. **Fastai Vision Module :**
   **(Finding Optimal Learning Rate)**

3. **Hyperparameters Tuning :**
   **(Model Freezing and Unfreezing)**

Data Science Academy

I

Fastai Framework

(Tabular, Vision and Collaborative filtering)

Data Science Academy

## What is Fastai

- Fastai is a deep learning library which provides practitioners with high-level components that can quickly and easily provide state-of-the-art results in standard deep learning domains, and provides researchers with low-level components that can be mixed and matched to build new approaches. It aims to do both things without substantial compromises in ease of use, flexibility, or performance.

# Fastai

- Fastai cover how to treat following application using the high-level API:
    - Vision
    - Text
    - Tabular
    - Collaborative filtering

# Tabular Data with fastai

- The tabular functionality in fast.ai combines training, validation, and (optionally) testing data into a single TabularDataBunch object.
- This structure makes it possible to tune pre-processing steps on the training data and then apply them equally to the validation and test data.
- Thus, with fast.ai the process of normalizing, inputting missing values, and determining the categories for each categorical variable is largely automated.

# Fastai with Tabular Data

- To illustrate the tabular application, we will use the example of the Adult dataset where we have to predict if a person is earning more or less than $50k per year using some general data.

```
from fastai.tabular.all import *
df = pd.read_csv(path/'adult.csv')
df.head()
```

- Some of the columns are continuous (like age) and we will treat them as float numbers we can feed our model directly. Others are categorical (like workclass or education) and we will convert them to a unique index that we will feed to embedding layers. We can specify our categorical and continuous column names, as well as the name of the dependent variable in TabularDataLoaders factory methods.

# Fastai with Tabular Data

```
dls = TabularDataLoaders.from_csv(path/'adult.csv', path=path, y_names="salary",
    cat_names = ['workclass', 'education', 'marital-status', 'occupation', 'relationship',
'race'],
    cont_names = ['age', 'fnlwgt', 'education-num'],
    procs = [Categorify, FillMissing, Normalize])
```

- The last part is the list of pre-processors we apply to our data:

  - Categorify is going to take every categorical variable and make a map from integer to unique categories, then replace the values by the corresponding index.
  - FillMissing will fill the missing values in the continuous variables by the median of existing values (you can choose a specific value if you prefer)
  - Normalize will normalize the continuous variables (substract the mean and divide by the std)

## Fastai with Tabular Data

- We can define a model using the tabular learner method. When we define our model, Fastai, will try to infer the loss function based on our y names earlier.

```
learn = tabular_learner(dls, metrics=accuracy)
```

- And we can train that model with the fit_one_cycle method.

```
learn.fit_one_cycle(1)
```

| epoch | train_loss | valid_loss | accuracy | time |
|---|---|---|---|---|
| 0 | 0.369360 | 0.348096 | 0.840756 | 00:05 |

# Fastai with Tabular Data

- We can then have a look at some predictions:

```
learn.show_results()
```

| | workclass | education | marital-status | occupation | relationship | race | education-num_name | age | fnlwgt | education-num | salary | salary-pred |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5.0 | 12.0 | 3.0 | 8.0 | 1.0 | 5.0 | 1.0 | 0.324868 | -1.138177 | -0.424022 | 0.0 | 0.0 |
| 1 | 5.0 | 10.0 | 5.0 | 2.0 | 2.0 | 5.0 | 1.0 | -0.482055 | -1.351911 | 1.148438 | 0.0 | 0.0 |
| 2 | 5.0 | 12.0 | 6.0 | 12.0 | 3.0 | 5.0 | 1.0 | -0.775482 | 0.138709 | -0.424022 | 0.0 | 0.0 |
| 3 | 5.0 | 16.0 | 5.0 | 2.0 | 4.0 | 4.0 | 1.0 | -1.362335 | -0.227515 | -0.030907 | 0.0 | 0.0 |

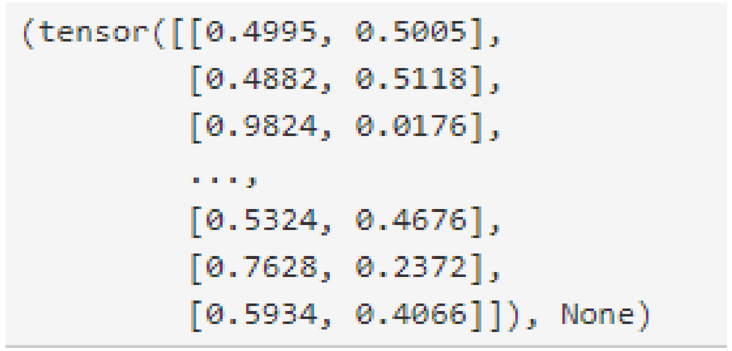
# Fastai with Tabular Data

- To get prediction on a new dataframe, you can use the test_dl method of the DataLoaders. That dataframe does not need to have the dependent variable in its column.

```
test_df = df.copy()
test_df.drop(['salary'], axis=1,
inplace=True)
dl = learn.dls.test_dl(test_df)
```

- Then Learner.get_preds will give you the predictions:

```
learn.get_preds(dl=dl)
```



```
(tensor([[0.4995, 0.5005],
         [0.4882, 0.5118],
         [0.9824, 0.0176],
         ...,
         [0.5324, 0.4676],
         [0.7628, 0.2372],
         [0.5934, 0.4066]]), None)
```

# Collaborative Filtering with fastai

- Collaborative filtering is an application of machine learning where we try to predict whether a user will like a particular movie or product. We do so by looking at the user's previous buying habits.

```
ratings.head()
```

| | user | Movie | rating | Title |
|---|---|---|---|---|
| 0 | 196 | 242 | 3 | Kolya(1996) |
| 1 | 63 | 242 | 3 | Kolya(1996) |
| 2 | 226 | 242 | 5 | Kolya(1996) |
| 3 | 154 | 242 | 3 | Kolya(1996) |
| 4 | 306 | 242 | 5 | Kolya(1996) |

# Collaborative Filtering with fastai

- We can then build a DataLoaders object from this table. By default, it takes the first column for user, the second column for the item (here our movies) and the third column for the ratings. We need to change the value of item_name in our case, to use the titles instead of the ids.

```
dls = CollabDataLoaders.from_df(ratings, item_name='title',
bs=64)
```

- In all applications, when the data has been assembled in a DataLoaders, you can have a look at it with the show_batch method:

```
dls.show_batch()
```

| | user | title | rating |
|---|---|---|---|
| 0 | 181 | Substitute, The (1996) | 1 |
| 1 | 189 | Ulee’s Gold (1997) | 3 |
| 2 | 6 | L.A. Confidential(1997) | 4 |
| 3 | 849 | Net, The(1995) | 5 |

# Collaborative Filtering with fastai

- fastai can create and train a collaborative filtering model by using collab_learner:

```
learn = collab_learner(dls, n_factors=50, y_range=(0, 5.5))
```

- To train it using the 1cycle policy, we just run this command

```
learn.fit_one_cycle(5, 5e-3,
wd=0.1)
```

| epoch | train_loss | valid_loss | Time |
|---|---|---|---|
| 0 | 0.967653 | 0.942309 | 00:10 |
| 1 | 0.843426 | 0.869254 | 00:10 |
| 2 | 0.733788 | 0.823143 | 00:10 |

## II
## Fastai Vision Module –
## Finding Optimal Learning Rate

Data Science Academy

# Fastai Vision (Single-label Classification)

- For computer vision task, fastai has fastai.vision module, which can be imported as follows:

```
from fastai.vision.all import *
```

- For this task, we will use the Oxford-IIIT Pet Dataset that contains images of cats and dogs of 37 different breeds. We will first show how to build a simple cat-vs-dog classifier, then a little bit more advanced model that can classify all breeds.

- The dataset can be downloaded and decompressed with this line of code

```
path = untar_data(URLs.PETS)
```

- Get_image_files is a fastai function that helps us grab all the image files (recursively) in one folder.

```
files = get_image_files(path/"images")
```

# Fastai Vision (Single-label Classification)

- We can then define an easy label function:

```
def label_func(f): return f[0].isupper()
```

- To get our data ready for a model, we need to put it in a DataLoaders object. Here we have a function that labels using the file names, so we will use ImageDataLoaders.from_name_func

```
dls = ImageDataLoaders.from_name_func(path, files, label_func,
item_tfms=Resize(224))
```

- We have passed to this function the directory we're working in, the files we grabbed, our label_func and one last piece as item_tfms: this is a Transform applied on all items of our dataset that will resize each image to 224 by 224, by using a random crop on the largest dimension to make it a square, then resizing to 224 by 224. If we didn't pass this, we would get an error later as it would be impossible to batch the items together.

# Fastai Vision (Single-label Classification)

- Then we can create a Learner, which is a fastai object that combines the data and a model for training, and uses transfer learning to fine tune a pretrained model in just two lines of code:

```
learn = cnn_learner(dls, resnet34, metrics=error_rate)
learn.fine_tune(1)
```

| epoch | train_loss | valid_loss | error_rate | time |
|---|---|---|---|---|
| 0 | 0.148785 | 0.013430 | 0.006089 | 00:13 |

| epoch | train_loss | valid_loss | error_rate | time |
|---|---|---|---|---|
| 0 | 0.047292 | 0.013792 | 0.005413 | 00:16 |

- The first line downloaded a model called ResNet34, pretrained on ImageNet, and adapted it to our specific problem. It then fine tuned that model and in a relatively short time, we get a model with an error rate of 0.3%.

# Fastai Vision (Single-label Classification)

- If you want to make a prediction on a new image, you can use learn.predict

```
learn.predict(files[0])
```

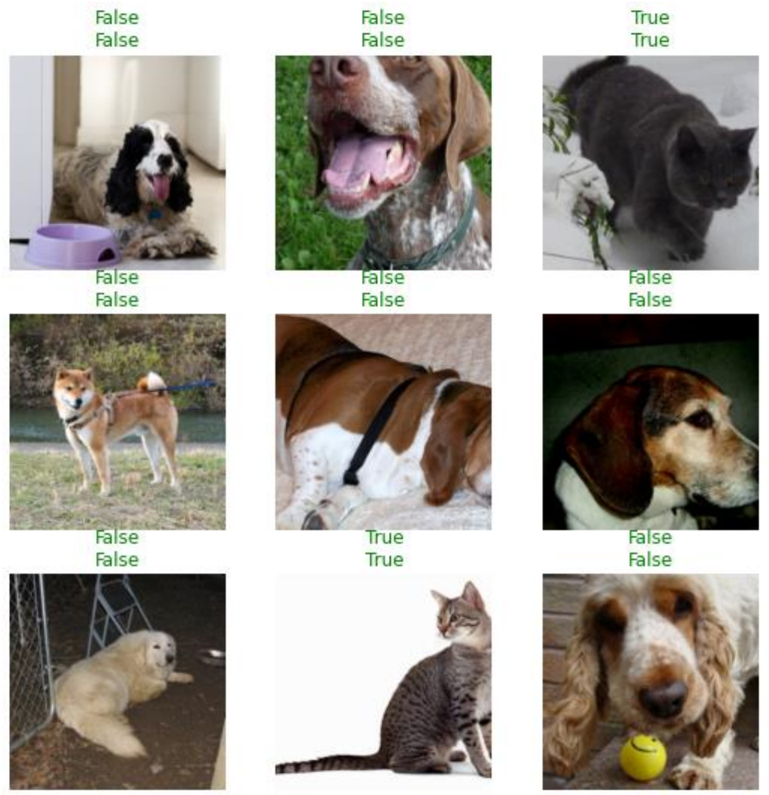
```
('False', TensorImage(0), TensorImage([9.9998e-01, 2.0999e-05]))
```

- The predict method returns three things: the decoded prediction (here False for dog), the index of the predicted class and the tensor of probabilities of all classes in the order of their indexed labels(in this case, the model is quite confifent about the being that of a dog).

# Fastai Vision
# (Single-label Classification)

- We can also have a look at some predictions with the show_results method.

```
learn.show_results()
```



# Fastai Vision (Multi-label Classification)

- Multi-label classification defers from before in the sense each image does not belong to one category. An image could have a person *and* a horse inside it for instance. Or have none of the categories we study.
- As before, we can download the dataset pretty easily:

```
path = untar_data(URLs.PASCAL_2007)
```

```
df = pd.read_csv(path/'train.csv')
df.head()
```

| | fname | labels | is_valid |
|---|---|---|---|
| 0 | 00005.jpg | chair | True |
| 1 | 00007.jpg | car | True |
| 2 | 000009.jpg | horse person | True |
| 3 | 000012.jpg | car | False |
| 4 | 000016.jpg | bicycle | True |

# Fastai Vision (Multi-label Classification)

- For each filename, we get the different labels (separated by space) and the last column tells if it's in the validation set or not. To get this in DataLoaders quickly, we have a factory method, from_df. We can specify the underlying path where all the images are, an additional folder to add between the base path and the filenames (here train), the valid_col to consider for the validation set (if we don't specify this, we take a random subset), a label_delim to split the labels and, as before, item_tfms and batch_tfms.

- Note that we don't have to specify the fn_col and the label_col because they default to the first and second column respectively.

```
dls = ImageDataLoaders.from_df(df, path, folder='train', valid_col='is_valid', label_delim='
',
                               item_tfms=Resize(460), batch_tfms=aug_transforms(size=224))
```

# Fastai Vision (Multi-label Classification)

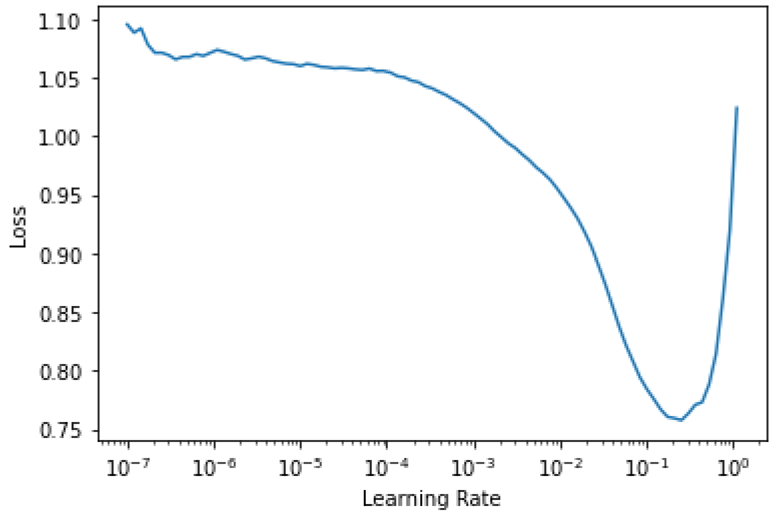
- Training a model is as easy as before: the same functions can be applied and the fastai library will automatically detect that we are in a multi-label problem, thus picking the right loss function. The only difference is in the metric we pass: error_rate will not work for a multi-label problem, but we can use accuracy_thresh.

```
learn = cnn_learner(dls, resnet50, metrics=partial(accuracy_multi,
thresh=0.5))
```

- We can use learn.lr_find to pick a good learning rate

```
learn.lr_find()
```

```
SuggestedLRs(lr_min=0.025118863582611
083, lr_steep=0.03981071710586548)
```



# Fastai Vision (Multi-label Classification)

- We can pick the suggested learning rate and fine-tune our pretrained model.

```
learn.fine_tune(2, 3e-2)
```

| epoch | train_loss | valid_loss | accuracy_multi | time |
|---|---|---|---|---|
| 0 | 0.437855 | 0.136942 | 0.954801 | 00:17 |

| epoch | train_loss | valid_loss | accuracy_multi | time |
|---|---|---|---|---|
| 0 | 0.156202 | 0.465557 | 0.914801 | 00:20 |
| 1 | 0.179814 | 0.382907 | 0.930040 | 00:20 |
| 2 | 0.157007 | 0.129412 | 0.953924 | 00:20 |
| 3 | 0.125787 | 0.109033 | 0.960856 | 00:19 |

## III
## Hyperparameters Tuning
## Model Freezing and Unfreezing

Data Science Academy

## Hyperparameters Tuning

- cnn_learner has parameters like learning_rate(lr), wd(weight decay), dp(dropout probability) that can be tuned. We can use Bayesian Optimization for tuning these parameters.

```
# lr = learning rate , wd = weight decay, dp = dropout probability

def fit_with(lr,wd,dp):

    # Create the model using a specified dropout and weight decay
    learn = cnn_learner(data, models.resnet18, metrics=accuracy, ps=dp)
    # Train the model for a specified number of epochs using a specified max learning rate
    learn.fit_one_cycle(1, max_lr=lr)
    # Plot the loss over time
    learn.recorder.plot_losses()
    # Save, print, and return the model's accuracy
    acc = float(learn.validate(learn.data.valid_dl)[1])
return acc
```

## Hyperparameters Tuning

- Next, we define our Bayesian optimizer using the rules set by the baysian-optimization library.

```
from bayes_opt import BayesianOptimization

# Bounded region of parameter space
pbounds = {'lr‘: (1e-4, 1e-2), 'wd':(1e-4,0.4), 'dp':(0.1,0.5)}

optimizer = BayesianOptimization(
    f=fit_with,
    pbounds=pbounds,
    verbose=2, random_state=1,
)
optimizer.maximize(init_points=2, n_iter=2,)
for i, res in enumerate(optimizer.res):
    print("Iteration {}: \n\t{}".format(i, res))

print(optimizer.max)
```

## Model Freezing and Unfreezing

- Freezing prevents the weights of a neural network layer from being modified during the backward pass of training. You progressively 'lock-in' the weights for each layer to reduce the amount of computation in the backward pass and decrease training time.

```
learn.freeze()
```

- You can unfreeze a model if you decide you want to continue training - an example of this is transfer learning: start with a pre-trained model, unfreeze the weights, then continuing training on a different dataset. When you choose to freeze is a balance between freezing early enough to gain computational speed-up without freezing too early with weights that result in inaccurate predictions.

```
learn.unfreeze()
```

Thank you!

Data Science Academy