The Movie Database (TMDB) Rating Prediction Based on Several Factors

Emanuel Azcona

Is it possible to predict a movie's critic success before it is even released? Are there key/certain cast members (actors/actresses) or essential crew (directors, producers, etc.) that contribute to the critical success of a movie? In this project we investigate and create a an appropiate regression model for predicting a movie's IMDB critic rating based on different attributes like: cast, crew, and more.

In this project, critic rating is given to us in the form of a non-integer score between 0 and 10.

Dataset (TMDB)

The dataset was obtained from Kaggle under: https://www.kaggle.com/tmdb/tmdb-movie-metadata (https://www.kaggle.com/tmdb/tmdb-movie-metadata). Per a DMCA takedown request from IMDB, the original IMDB dataset was removed. In order to minimize the impact, Kaggle replaced the dataset with the TMDB dataset, which consists of a similar set of films and data fields in accordance with their terms of use (https://www.themoviedb.org/documentation/api/terms-of-use (https://www.themoviedb.org/documentation/api/terms-of-use)).

The dataset itself contains two $.\mathtt{CSV}$ files:

- `tmdb_5000_credits.csv`
- `tmdb_5000_movies.csv`

where the "credits" file contains crediting information about the 5000 movies in the datset (title, cast, crew) and the "movies" file contains the following information about each movie:

- budget
- voter rating
- voter count (number of people who voted)
- revenue
- genre(s)
- homepage
- keywords
- original language
- original title
- and way more.

For the purposes of this experiment, we're only going to focus on using the entire "credits" file and the budget/revenue columns of the "movies" file.

Loading Credits & Movies Files

For data manipulation throughout this experiment, we're going to rely on the Pandas library (available through `pip` or `conda`).

In [1]:
```
import pandas as pd # typical naming convention for abbreviating [pandas]
import ast          # I only use this library to convert strings into their
import numpy as np  # using the original NumPy library, not the autograd one

from matplotlib import pyplot as plt
%matplotlib inline
```

In [2]:
```
# load in dataset using pandas
creditsDF = pd.read_csv('tmdb_5000_credits.csv')
moviesDF = pd.read_csv('tmdb_5000_movies.csv')

# sanity check (compare number of samples in each .CSV file)
print('There are ' + str( creditsDF.shape[0] ) + ' movies in the "credits" f
print('There are ' + str( moviesDF.shape[0] ) + ' movies in the "movies" fil

# I wrote this as a sanity-check to see if every row in each .CSV file corre
# to the respective row in the other.
mistakeInData = False
for i in range( creditsDF.shape[0] ):
    if creditsDF['movie_id'][i] != moviesDF['id'][i]:
        mistakeInData = True
if mistakeInData:
    print('\nSomething is up dude. Double-check the data.')
else:
    print('\nDataset is legitimate. Proceed.')
```

```
There are 4803 movies in the "credits" file.
There are 4803 movies in the "movies" file.

Dataset is legitimate. Proceed.
```

In [3]:
```
# Overview of the first few samples for the movies dataframe
moviesDF.head()
```

Out[3]:

	budget	genres	homepage	id	keywords	original_la
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

In [4]:
```
# Overview of the first few samples for the credits dataframe
creditsDF.head()
```

Out[4]:

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

Cleaning and Manipulating Data to Create Clean Dictionaries of Movie Information

It's a little annoying to be manipulating two seperate dataframes related to one set of data. For that reason, a single dictionary is created to avoid confusion across datasets. The dictionary data structure was chosen because of its constant, $O(1)$, indexing runtime (I read some forums and documentation warning that Pandas indexing may not run, $O(1)$).

In the process of creating the dictionary, we must be careful to only extract and store information that is relevant and/or may be helpful to determining critic rating. One example of information that may not be helpful that is not included is the homepage to a particular movie.

Throughout the function below, I detail which features I decide to store during each iteration while parsing through the movie dataset.

I also made the decision to remove any movie in the dataset that has less voter counts than the overall median voter count. Why did I decide upon this median metric? I can not give a reasonable answer to that other than, it seemed reasonable. The motivation behind this is credited to some of the movies having unreasonably high (10) or low (0) ratings with only 1 or 0 voters.

In [5]:
```
def createMovieDictionary(creditsDF, moviesDF):

    # initialize dictionary of movies
    movies = {}

    # initiailize corresponding set of keys for the movies dictionary
    # I use a set to make sure that no movie has duplicate keys
    keys = set()

    # go through every movie in the dataset O(N)
    for i in range(moviesDF.shape[0]):

        budget = moviesDF['budget'][i]          # temporarily store the budg
        allVoteCounts = moviesDF['vote_count']  # temporarily store vote cou
        medVoteCount = np.median(allVoteCounts) # determine median of the vo

        # check to see if we're dealing with a non-zero budget movie
        if budget > 0:

            # only continue if the current number of votes is bigger than th
            if allVoteCounts[i] >= medVoteCount:

                cast = ast.literal_eval( creditsDF['cast'][i] ) # temporaril
                cast = set( [member['name'] for member in cast if member['or

                crew = ast.literal_eval( creditsDF['crew'][i] ) # repeat the
                crew = set( [member['name'] for member in crew if member['jo

                genres = ast.literal_eval( moviesDF['genres'][i] ) # repeat
                genres = set( [g['name'] for g in genres] )

                production_companies = ast.literal_eval( moviesDF['productio
                production_companies = set( [p['name'] for p in production_c


                # each movie entry of the movies dictionary is a dictionary
                # pertaining to that movie.

                # inner dictionary of each movie will have key-entry pairs i
                # - 'title': string representing name of the movie
                # - 'cast': Python list of cast members names' (all strings,
                # - 'crew': Python ....... crew ...........................
                # - 'budget': integer representing the total budget for maki

                # example:
                #
                # firstMovie = movies[19995]
                #   - firstMovie['title'] = 'Avatar'
                #   - firstMovie['budget'] = 237000000
                #   ....

                # temporarily save movie ID of the movie from id column of c
                movID = moviesDF['id'][i]

                # temporarily save title of current movie
                title = moviesDF['title'][i]
                # temporarily save the rating of the current movie
                rating = moviesDF['vote_average'][i]

                # append the set of keys with the current movie's ID
                keys.update({movID})

                movies[movID] = {'title': title}              # first entry
                movies[movID]['cast'] = cast                  # assign set c
                movies[movID]['crew'] = crew                  # assign set c
                movies[movID]['budget'] = budget              # assign budge
                movies[movID]['rate'] = rating                # assign ratin
                movies[movID]['genres'] = genres              # assign genre
                movies[movID]['prod'] = production_companies  # assign produ

    return movies, keys
```

In [6]:
```
movies, keys = createMovieDictionary(creditsDF,moviesDF)

# I'm paranoid, so I'm just deleting these dataframes now to free up some sp
del moviesDF, creditsDF

# Another sanity check
print("The movie dictionary has: " + str(len(movies)) + " entries." )
```

```
The movie dictionary has: 2302 entries.
```

Determine Distinct Items Pertaining to Features That Are Sets/Lists

Next, a function was created to create individual sets of the distinct cast, crew, genres, and production companies in the movies we extracted. We can utilize the Python $\texttt{set}(\cdot)$ method that creates a distinct set data structure for eliminating repeat cast or crew members.

In [7]:
```
def distinctThings(movies, key):

    # only return a set if the key entered is cast, crew, genres, or product
    if key in ['cast', 'crew', 'genres', 'prod']:

        distinct = set() # create distinction set

        for m in movies:                # iterate through all the movies in
            for member in movies[m][key]: # iterate through the list pertain
                distinct.update({member})
        return distinct
    else:
        return None
```

In [8]:
```
distinctCast = distinctThings(movies, 'cast')
distinctCrew = distinctThings(movies, 'crew')
distinctGenres = distinctThings(movies, 'genres')
distinctProd = distinctThings(movies, 'prod')
```

Create Rating Regression Model Based Only on

Budget

Simple 1-D Feature & 1-D Predictor Regression Model (Rating Based on Budget)

Time to play around with the data! To start things off, we can first create a simple regression model using a very minimal deep-feedforward network in Keras.

Before doing anything, two functions were created, createFeatures() and createPredictors(), that let us create feature/predictor matrices using NumPy. Keras by default takes in NumPy arrays as data inputs.

In [9]:
```
N_samples = len(movies)

def createFeatures(feature):

    abort = False
    # if we're only using the budget, we'll have a single column for X
    if feature is 'budget':
        N_features = 1

    # otherwise N_features is # of distinct possible cast, crew, etc.
    elif feature is 'cast':
        N_features = len(distinctCast)
    elif feature is 'crew':
        N_features = len(distinctCrew)
    elif feature is 'genres':
        N_features = len(distinctGenres)
    elif feature is 'prod':
        N_features = len(distinctProd)
    else:
        abort = True

    # if the number of features is still 0, then obviously we didn't input 
    if abort:
        return None, None
    else:
        # initialize  data array to array of zeros
        # in the case of the feature being anything else besides budget, we 
        X = np.zeros( (N_samples, N_features) )

        # iterate over keys set we created earlier (so we can iterate throug
        # "i" in this case will be the corresponding row to a movie (startin
        for i, ID in enumerate(keys):
            currentMovie = movies[ID]

            # budget results in a 1-dimensional feature fector
            if feature is 'budget':
                X[i, 0] = currentMovie[feature]

            # all other features result in NxM feature matrices (very sparse
            else:
                if feature is 'cast':
                    for j, member in enumerate(distinctCast):
                        if member in currentMovie[feature]:
                            X[i,j] = 1

                elif feature is 'crew':
                    for j, member in enumerate(distinctCrew):
                        if member in currentMovie[feature]:
                            X[i,j] = 1

                elif feature is 'genres':
                    for j, member in enumerate(distinctGenres):
                        if member in currentMovie[feature]:
                            X[i,j] = 1

                else:
                    for j, member in enumerate(distinctProd):
                        if member in currentMovie[feature]:
                            X[i,j] = 1

        return X

# short function for creating 1-dimensional predictor vector
def createPredictors():
    y = np.zeros((N_samples,1))
    for i, ID in enumerate(keys):
        currentMovie = movies[ID]

        y[i,0] = currentMovie['rate']
    return y
```

In [10]:
```
X = createFeatures('budget')
y = createPredictors()
```

In [11]:
```
# sanity check
print(X.shape, y.shape)
```

```
((2302, 1), (2302, 1))
```

Normalize Input Data (Optimization Trick to Reduce # of Epochs Required for Training)

Here, we utilize an optimization trick (normalizing the input data) that helps in reducing the number of epochs required for training. There are many methods for normalizing a vector. A common one being:

$$\vec{x}_{norm} = \frac{\vec{x} - \mu_x}{\sigma_x}$$

In this project, we use the Scikit-learn library which provides a normalization function that uses the l_2 norm for normalization.

$$\vec{x}_{norm} = \frac{\vec{x}}{||\vec{x}||_2}$$

In [12]:
```
from sklearn.preprocessing import normalize
```

In [13]:
```
Xnorm = normalize(X, axis = 0, norm = 'l2')
```

Using Keras (Tensorflow Backend)

Keras is a high-level neural network API, for Python, for use on top of Tensorflow, CNTK, or Theano. Since Theano updates will soon stop, and Tensorflow (although slow for the time being) is Google-backed and owned I used Keras with a Tensorflow backend.

In [14]:
```
import keras
from keras.models import Sequential # Very simple model-making in keras for
```

```
Using TensorFlow backend.
```

Rather than viewing the losses throughout training, a Keras callback was defined below to store loss history.

In [15]:
```
# self-defined callback for training in keras.
#
class LossHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.losses = []

    def on_epoch_end(self, epoch, logs={}):
        self.losses.append([logs.get('loss'), logs.get('val_loss')])
```

In [16]:
```
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
from sklearn.model_selection import KFold

# define 10-fold cross validation test harness
seed = 7
num_epochs = 20
folds =4

initialWeights = keras.initializers.RandomNormal(mean=0.0, stddev=0.07, seec

kfold = KFold(n_splits=folds, shuffle=True, random_state=seed)

cvscoresHistory = []

for k, (train, test) in enumerate(kfold.split(Xnorm, y)):
    currCVhistory = []
    model = Sequential()
    model.add(Dense(units=128, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=64, kernel_initializer= initialWeights, activatior
    model.add(Dense(units=32, kernel_initializer= initialWeights, activatior
    model.add(Dense(units=1, kernel_initializer= initialWeights, activation

    model.compile(loss='mean_absolute_error',
              optimizer = 'adam'
             )

    history = LossHistory()
    model.fit(Xnorm[train],
              y[train],
              validation_data = (Xnorm[test],y[test]),
              epochs=num_epochs,
              batch_size=64,
              verbose=1,
              callbacks = [history]
             )

    cvscoresHistory.append(history.losses)
```

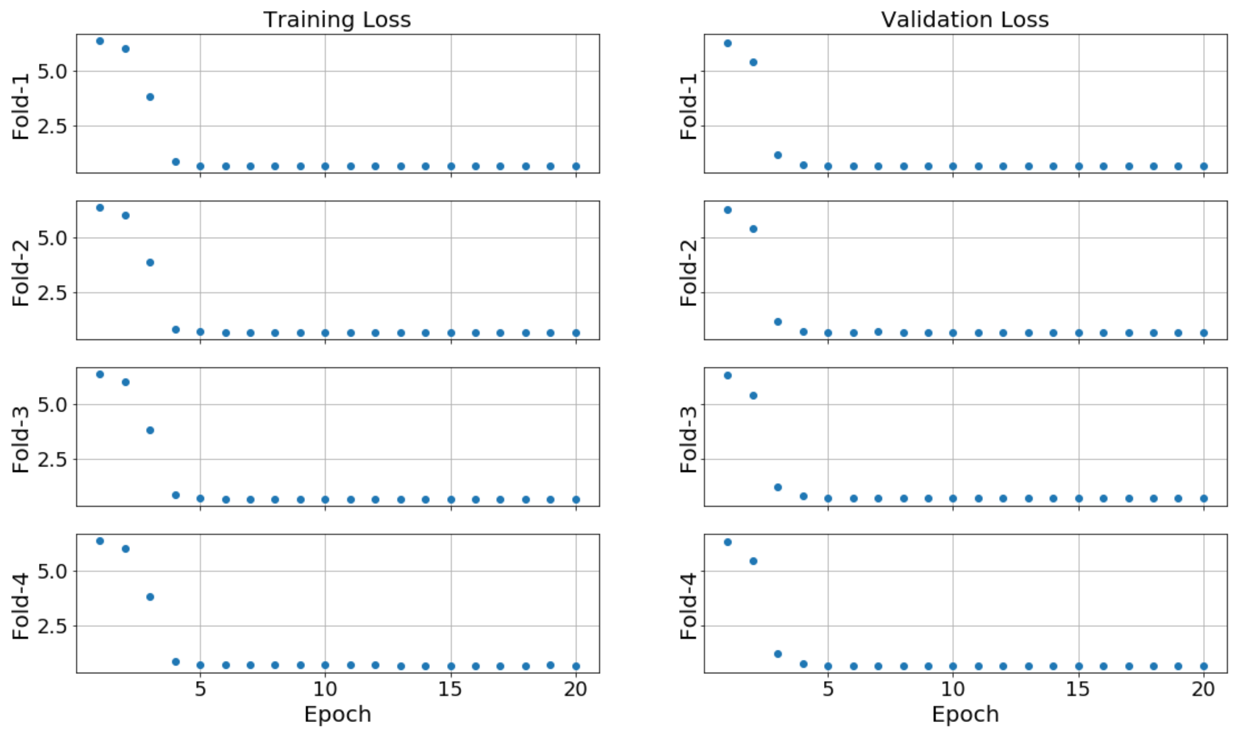
```
Train on 1726 samples, validate on 576 samples
Epoch 1/20
1726/1726 [==============================] - 0s - loss: 6.4098 - val_los
s: 6.3247
Epoch 2/20
1726/1726 [==============================] - 0s - loss: 6.0187 - val_los
s: 5.4272
Epoch 3/20
1726/1726 [==============================] - 0s - loss: 3.8333 - val_los
s: 1.1744
Epoch 4/20
1726/1726 [==============================] - 0s - loss: 0.8296 - val_los
s: 0.7059
Epoch 5/20
1726/1726 [==============================] - 0s - loss: 0.6707 - val_los
s: 0.6462
Epoch 6/20
1726/1726 [==============================] - 0s - loss: 0.6620 - val_los
```

```
s: 0.6468
```

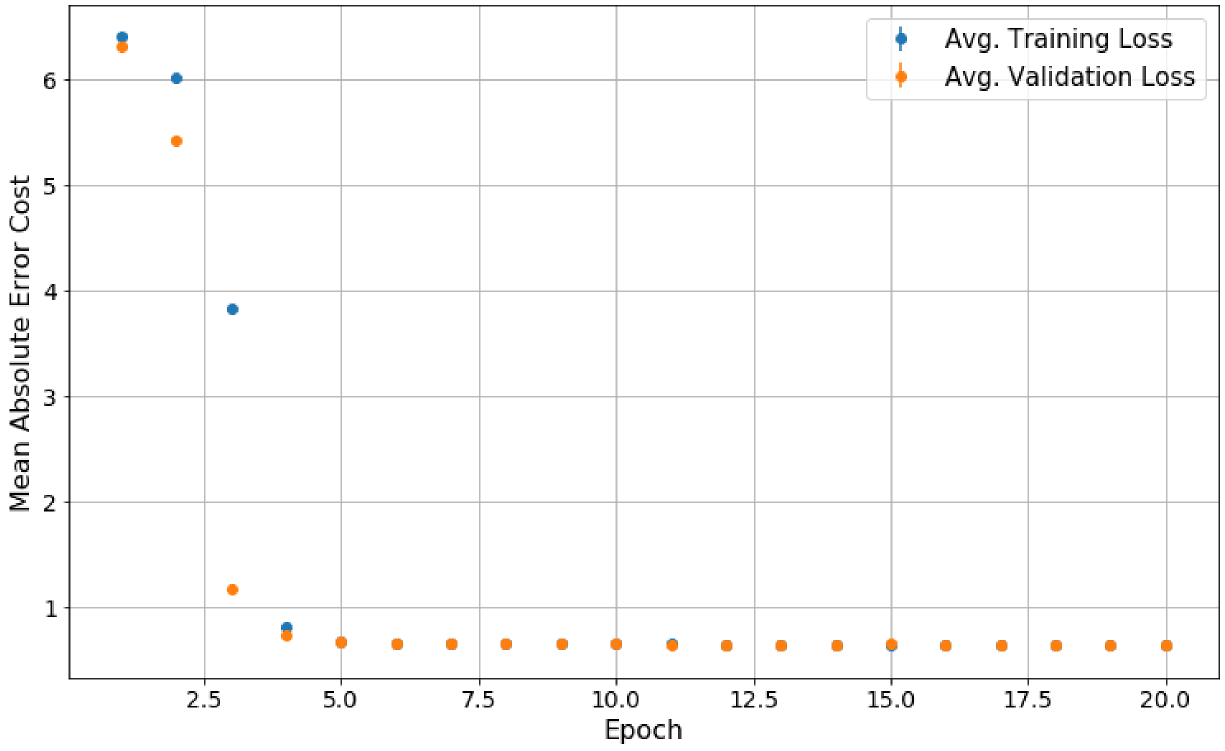
In [17]:
```
trainingLossHistory = np.asarray(cvscoresHistory)[:].T[0].T
validationLossHistory = np.asarray(cvscoresHistory)[:].T[1].T
```

In [18]:
```
meanTrainingLossHistory = np.mean(trainingLossHistory, axis = 0)
meanValidationLossHistory = np.mean(validationLossHistory, axis = 0)

stdTrainingLossHistory = np.std(trainingLossHistory, axis = 0)
stdValidationLossHistory = np.std(validationLossHistory, axis = 0)
```

In [19]:
```
iterations = [i+1 for i in range(num_epochs)]

f, axarr = plt.subplots(folds,2, sharex = True, sharey = True)
f.set_size_inches(18.5, 10.5)
axarr[0,0].set_title('Training Loss', fontsize = 20)
axarr[0,1].set_title('Validation Loss', fontsize = 20)
axarr[-1,0].set_xlabel('Epoch', fontsize = 20)
axarr[-1,1].set_xlabel('Epoch', fontsize = 20)

for i in range(folds):
    axarr[i,0].plot(iterations, trainingLossHistory[i], 'o' )
    axarr[i,0].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,0].tick_params(labelsize = 18)
    axarr[i,0].grid()

    axarr[i,1].plot(iterations, validationLossHistory[i],'o')
    axarr[i,1].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,1].tick_params(labelsize = 18)
    axarr[i,1].grid()
plt.show()
```



```
In [20]: f = plt.figure()
         f.set_size_inches(12.5, 7.5)
         plt.errorbar(iterations, meanTrainingLossHistory, stdTrainingLossHistory, fr
         plt.errorbar(iterations, meanValidationLossHistory, stdValidationLossHistory
         plt.tick_params(labelsize = 14)
         plt.legend(fontsize = 15)
         plt.xlabel('Epoch', fontsize = 16)
         plt.ylabel('Mean Absolute Error Cost', fontsize = 16)
         plt.grid(True)
         plt.show()
```



This Is Good News!

This is great! It seems that the validation loss and training loss are at a pass and stay constant after about 10 epochs of stochastic gradient descent. Since we decided to use the mean absolute error cost function, it is easy to see that after 10 epochs of training we our average error across when using budgets to predict ratings is off by $\approx \pm 0.6$. This error rate is on a scale with ratings that range between 0 and 10.

One very useful metric to determine is which set of weights for our network were the "most useful." One way to determine this is to sort the validation and training losses in lexicographical order.

Luckily, the NumPy library provides a function, lexsort(), that does lexicographical sorting and returns the indices of the order for the elements. In other words, we sort our loss data, first by order from smallest to largest validation loss, and then from smallest to largest training loss.

In [21]:
```
ind = np.lexsort( (meanTrainingLossHistory, meanValidationLossHistory) )
print('"ind" is an array that contains the indices (epoch #) of sorting (lea
print(ind)

print("\nTherefore, Epoch:" + str(ind[0]) + " would be a good place to stop
```

```
"ind" is an array that contains the indices (epoch #) of sorting (least t
o greatest) by first the validation loss and then by the training loss

[18 15 19 12 13 17 10 16 11 14  9  8  7  5  6  4  3  2  1  0]

Therefore, Epoch:18 would be a good place to stop training this model.
```

REPEAT NOW: Create Rating Regression Model Based Only Top 3 Paid Cast Members

In [22]:
```
X = createFeatures('cast')
Xnorm = normalize(X, axis = 0, norm = 'l2')
```

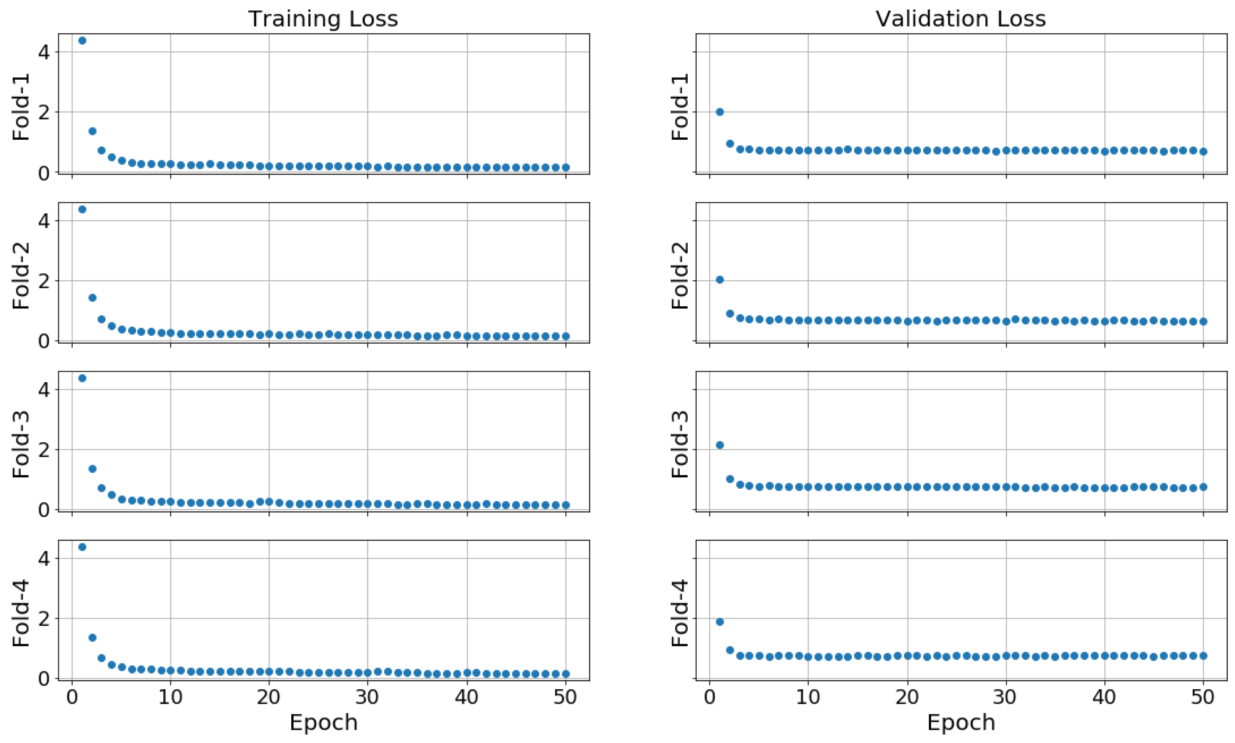
In [23]:
```
# define 10-fold cross validation test harness
seed = 7
num_epochs = 50
folds =4

initialWeights = keras.initializers.RandomNormal(mean=0.0, stddev=0.07, seec

kfold = KFold(n_splits=folds, shuffle=True, random_state=seed)

cvscoresHistory = []

for k, (train, test) in enumerate(kfold.split(Xnorm, y)):
    currCVhistory = []
    model = Sequential()
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=128, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=1, kernel_initializer= initialWeights, activation

    model.compile(loss='mean_absolute_error',
                  optimizer = 'adam'
                 )

    history = LossHistory()
    model.fit(Xnorm[train],
             y[train],
             validation_data = (Xnorm[test],y[test]),
             epochs=num_epochs,
             batch_size=64,
             verbose=1,
             callbacks = [history]
            )

    cvscoresHistory.append(history.losses)
```

```
Train on 1726 samples, validate on 576 samples
Epoch 1/50
1726/1726 [==============================] - 0s - loss: 4.3871 - val_los
s: 1.9884
Epoch 2/50
1726/1726 [==============================] - 0s - loss: 1.3664 - val_los
s: 0.9496
Epoch 3/50
1726/1726 [==============================] - 0s - loss: 0.7399 - val_los
s: 0.7582
Epoch 4/50
1726/1726 [==============================] - 0s - loss: 0.4855 - val_los
s: 0.7512
Epoch 5/50
1726/1726 [==============================] - 0s - loss: 0.3885 - val_los
s: 0.7249
Epoch 6/50
1726/1726 [==============================] - 0s - loss: 0.3213 - val_los
s: 0.7266
Epoch 7/50
```

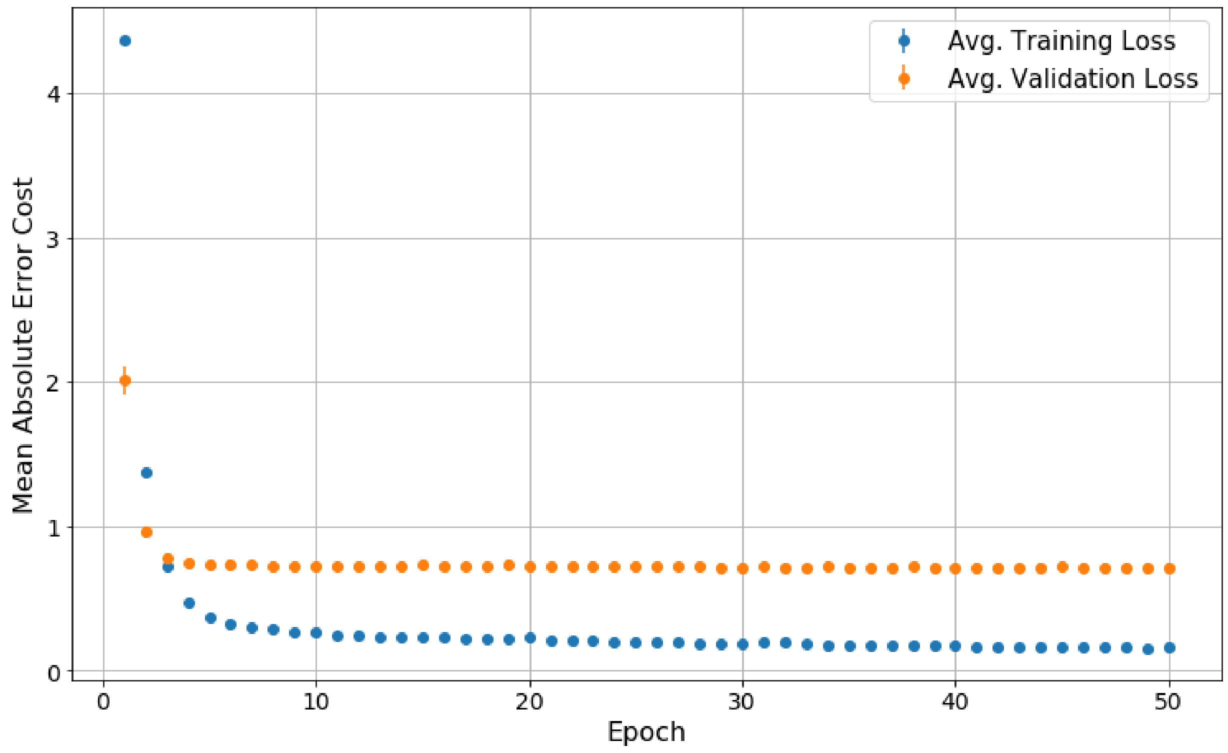
In [24]:
```
trainingLossHistory = np.asarray(cvscoresHistory)[:].T[0].T
validationLossHistory = np.asarray(cvscoresHistory)[:].T[1].T
```

In [25]:
```
meanTrainingLossHistory = np.mean(trainingLossHistory, axis = 0)
meanValidationLossHistory = np.mean(validationLossHistory, axis = 0)

stdTrainingLossHistory = np.std(trainingLossHistory, axis = 0)
stdValidationLossHistory = np.std(validationLossHistory, axis = 0)
```

In [26]:
```
iterations = [i+1 for i in range(num_epochs)]

f, axarr = plt.subplots(folds,2, sharex = True, sharey = True)
f.set_size_inches(18.5, 10.5)
axarr[0,0].set_title('Training Loss', fontsize = 20)
axarr[0,1].set_title('Validation Loss', fontsize = 20)
axarr[-1,0].set_xlabel('Epoch', fontsize = 20)
axarr[-1,1].set_xlabel('Epoch', fontsize = 20)

for i in range(folds):
    axarr[i,0].plot(iterations, trainingLossHistory[i], 'o' )
    axarr[i,0].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,0].tick_params(labelsize = 18)
    axarr[i,0].grid()

    axarr[i,1].plot(iterations, validationLossHistory[i],'o')
    axarr[i,1].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,1].tick_params(labelsize = 18)
    axarr[i,1].grid()
plt.show()
```



In [27]:
```
f = plt.figure()
f.set_size_inches(12.5, 7.5)
plt.errorbar(iterations, meanTrainingLossHistory, stdTrainingLossHistory, fr
plt.errorbar(iterations, meanValidationLossHistory, stdValidationLossHistory
plt.tick_params(labelsize = 14)
plt.legend(fontsize = 15)
plt.xlabel('Epoch', fontsize = 16)
plt.ylabel('Mean Absolute Error Cost', fontsize = 16)
plt.grid(True)
plt.show()
```



In [28]:
```
ind = np.lexsort( (meanTrainingLossHistory, meanValidationLossHistory) )
print('"ind" is an array that contains the indices (epoch #) of sorting (lea
print(ind)

print("\nTherefore, Epoch:" + str(ind[0]) + " would be a good place to stop
```

```
"ind" is an array that contains the indices (epoch #) of sorting (least t
o greatest) by first the validation loss and then by the training loss

[38 45 47 39 34 46 48 40 28 49 42 43 35 32 41 31 36 29 25 22 26 16 11 17
24
 37 20 23 21 15 19 44 27 33  7  9 12 10 13  8 30 18 14  5  4  6  3  2  1
0]

Therefore, Epoch:38 would be a good place to stop training this model.
```

But Does the Model Really Work?

To test this question, artificial data was created in which we cast only 1 of our cast members per movie. In other words, we create 1 movie for every actor/actress in our "distinctCast" array, and in each movie we only cast the corresponding cast member.

Using this array of data and a trained model, we can apply these artifical movies to our regression model and predict the ratings for these fake movies. From these fake movies we can sort their ratings from greatest to least and view who the "top choices" would be to get the highest voter rating average.

In [29]:
```
Xset = []
for i in range(0, len(distinctCast)):
    Xcurr = np.zeros((1,X.shape[1]))
    Xcurr[0,i] = 1
    Xset.append(Xcurr)


predictions = []
for castMember in Xset:
    predictions.append(float(model.predict(castMember)))

idx = np.argsort(predictions)
```

In [30]:
```
dList = list(distinctCast)

print('The best choices to make if you were to a cast only one "top-3" cast
for i in idx[-1:-20:-1]:
    print(list(dList)[i])
```

```
The best choices to make if you were to a cast only one "top-3" cast memb
er would be:

Brad Pitt
Denzel Washington
Joe Pesci
Tom Hanks
Leonardo DiCaprio
Elijah Wood
Vin Diesel
Robert De Niro
Clint Eastwood
Ralph Fiennes
Kristen Wiig
Joaquin Phoenix
Jared Leto
Ed Harris
Michael Caine
Bruce Willis
Alec Baldwin
Matt Damon
Kate Winslet
```

Given the state of the film industry today, the names that this model was able extrapolate are of various actors/actresses who have appeared in a lot of critically and commercially successful films, such as Brad Pitt, Tom Hanks, Leonardo Dicaprio, etc.

REPEAT NOW: Create Rating Regression Model Based Only On Director & Producer

In [31]:
```
X = createFeatures('crew')
Xnorm = normalize(X, axis = 0, norm = 'l2')
```

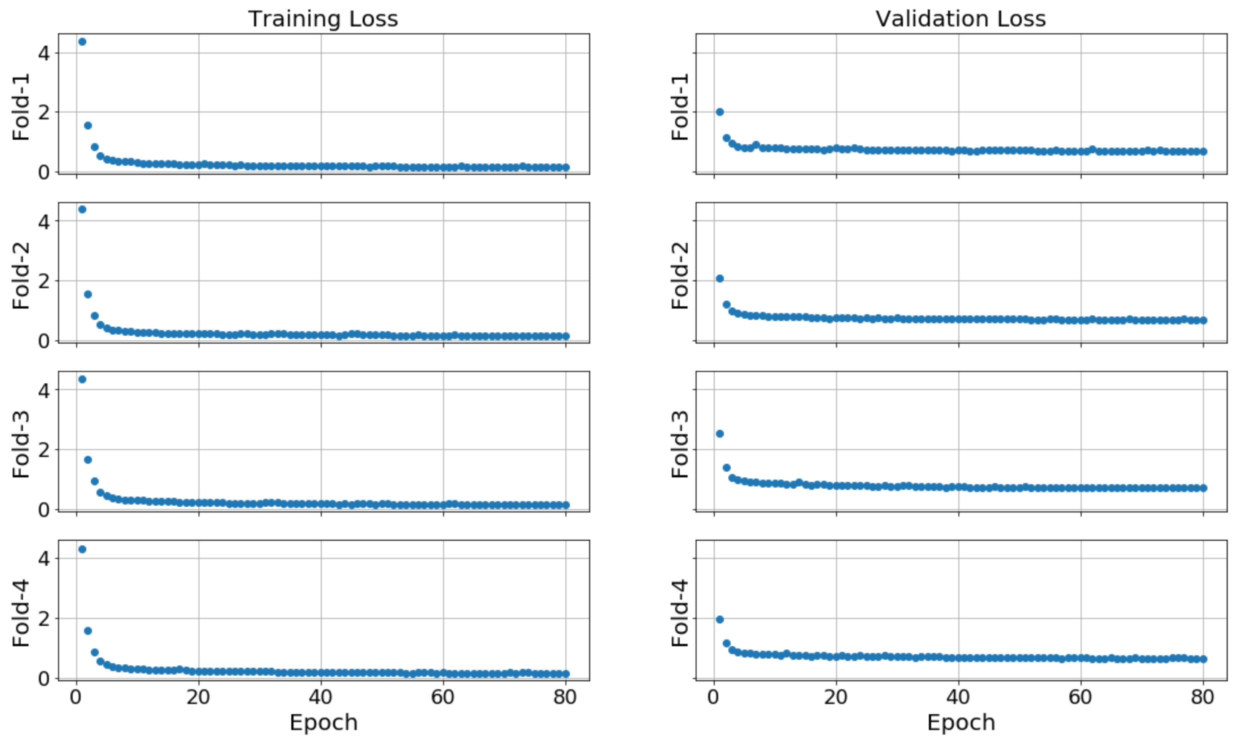
In [32]:
```
# define 10-fold cross validation test harness
seed = 7
num_epochs = 80
folds =4

initialWeights = keras.initializers.RandomNormal(mean=0.0, stddev=0.07, seec

kfold = KFold(n_splits=folds, shuffle=True, random_state=seed)

cvscoresHistory = []

for k, (train, test) in enumerate(kfold.split(Xnorm, y)):
    currCVhistory = []
    model = Sequential()
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=128, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=1, kernel_initializer= initialWeights, activation

    model.compile(loss='mean_absolute_error',
                  optimizer = 'adam'
                 )

    history = LossHistory()
    model.fit(Xnorm[train],
              y[train],
              validation_data = (Xnorm[test],y[test]),
              epochs=num_epochs,
              batch_size=64,
              verbose=1,
              callbacks = [history]
             )

    cvscoresHistory.append(history.losses)
```

```
1726/1726 [==============================] - 0s - loss: 0.1972 - val_los
s: 0.7289
Epoch 30/80
1726/1726 [==============================] - 0s - loss: 0.1972 - val_los
s: 0.7149
Epoch 31/80
1726/1726 [==============================] - 0s - loss: 0.1992 - val_los
s: 0.7205
Epoch 32/80
1726/1726 [==============================] - 0s - loss: 0.1947 - val_los
s: 0.7042
Epoch 33/80
1726/1726 [==============================] - 0s - loss: 0.1836 - val_los
s: 0.7098
Epoch 34/80
1726/1726 [==============================] - 0s - loss: 0.1879 - val_los
s: 0.7252
Epoch 35/80
1726/1726 [==============================] - 0s - loss: 0.1861 - val_los
s: 0.7114
```

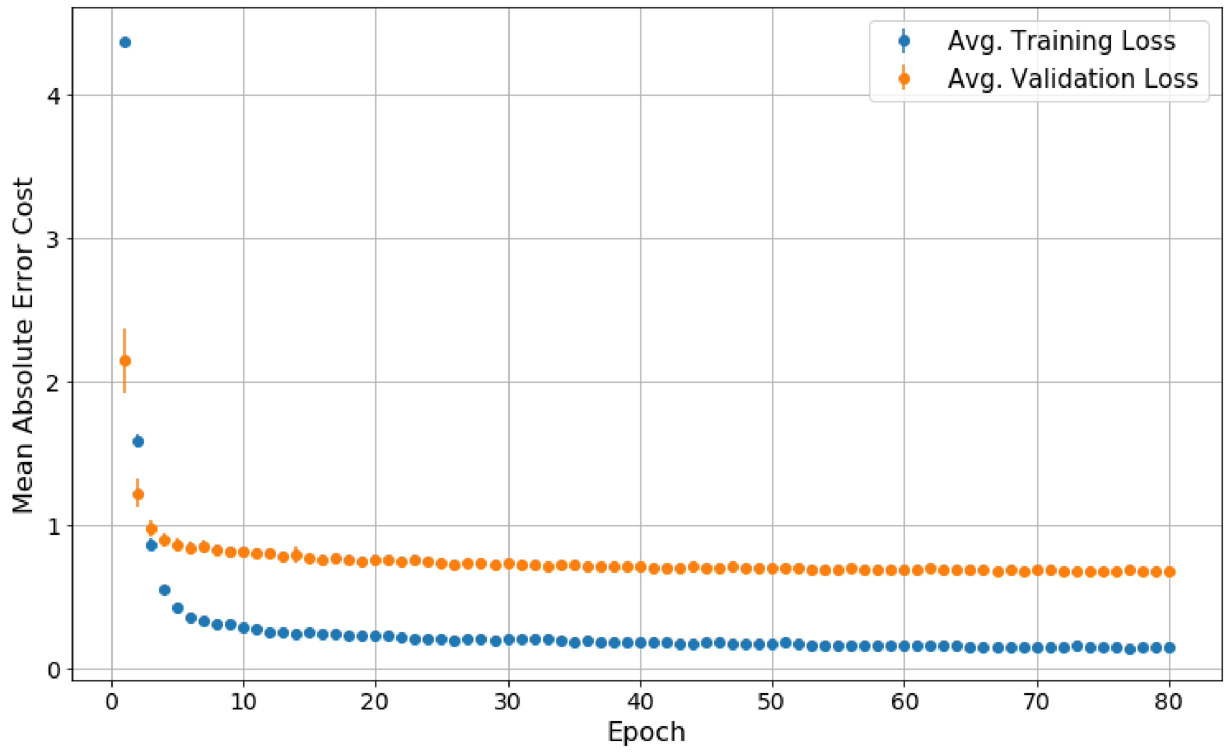
In [33]:
```
trainingLossHistory = np.asarray(cvscoresHistory)[:].T[0].T
validationLossHistory = np.asarray(cvscoresHistory)[:].T[1].T
```

In [34]:
```
meanTrainingLossHistory = np.mean(trainingLossHistory, axis = 0)
meanValidationLossHistory = np.mean(validationLossHistory, axis = 0)

stdTrainingLossHistory = np.std(trainingLossHistory, axis = 0)
stdValidationLossHistory = np.std(validationLossHistory, axis = 0)
```

In [35]:
```
iterations = [i+1 for i in range(num_epochs)]

f, axarr = plt.subplots(folds,2, sharex = True, sharey = True)
f.set_size_inches(18.5, 10.5)
axarr[0,0].set_title('Training Loss', fontsize = 20)
axarr[0,1].set_title('Validation Loss', fontsize = 20)
axarr[-1,0].set_xlabel('Epoch', fontsize = 20)
axarr[-1,1].set_xlabel('Epoch', fontsize = 20)

for i in range(folds):
    axarr[i,0].plot(iterations, trainingLossHistory[i], 'o' )
    axarr[i,0].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,0].tick_params(labelsize = 18)
    axarr[i,0].grid()

    axarr[i,1].plot(iterations, validationLossHistory[i],'o')
    axarr[i,1].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,1].tick_params(labelsize = 18)
    axarr[i,1].grid()
plt.show()
```



In [36]:
```
f = plt.figure()
f.set_size_inches(12.5, 7.5)
plt.errorbar(iterations, meanTrainingLossHistory, stdTrainingLossHistory, fm
plt.errorbar(iterations, meanValidationLossHistory, stdValidationLossHistory
plt.tick_params(labelsize = 14)
plt.legend(fontsize = 15)
plt.xlabel('Epoch', fontsize = 16)
plt.ylabel('Mean Absolute Error Cost', fontsize = 16)
plt.grid(True)
plt.show()
```



In [37]:
```
ind = np.lexsort( (meanTrainingLossHistory, meanValidationLossHistory) )
print('"ind" is an array that contains the indices (epoch #) of sorting (lea
print(ind)

print("\nTherefore, Epoch:" + str(ind[0]) + " would be a good place to stop
```

```
"ind" is an array that contains the indices (epoch #) of sorting (least t
o greatest) by first the validation loss and then by the training loss

[78 73 77 75 79 66 74 71 72 68 69 65 62 67 56 63 76 70 64 58 53 57 52 59
60
 54 48 49 55 61 51 47 42 41 50 40 45 44 43 46 38 39 36 37 32 35 31 33 34
28
 25 30 26 29 27 24 23 18 21 20 22 15 19 17 16 14 12 13 10 11  9  8  7  5
6
  4  3  2  1  0]

Therefore, Epoch:78 would be a good place to stop training this model.
```

But Does the Model Really Work?

To test this question, artificial data was created in which we cast only 1 of our crew members per movie. In other words, we create 1 movie for every director/producer in our "distinctCrew" array, and in each movie we only cast the corresponding cast member.

Using this array of data and a trained model, we can apply these artifical movies to our regression model and predict the ratings for these fake movies. From these fake movies we can sort their ratings from greatest to least and view who the "top choices" would be to get the highest voter rating average.

In [38]:
```
Xset = []
for i in range(0, len(distinctCrew)):
    Xcurr = np.zeros((1,X.shape[1]))
    Xcurr[0,i] = 1
    Xset.append(Xcurr)


predictions = []
for crewMember in Xset:
    predictions.append(float(model.predict(crewMember)))

idx = np.argsort(predictions)
```

In [39]:
```
dList = list(distinctCrew)

print('The best choices to make if you were to a cast only one "top" crew me
for i in idx[-1:-20:-1]:
    print(list(dList)[i])
```

```
The best choices to make if you were to a cast only one "top" crew member
would be:

Stanley Kubrick
Martin Scorsese
James Cameron
Frank Darabont
Christopher Nolan
Sergio Leone
James Mangold
Lasse Hallstr\u00f6m
Steven Spielberg
David Heyman
Frank Capra
David Lynch
Barry Mendel
Kevin Feige
Joe Wright
Frank Miller
Alfred Hitchcock
Lawrence Bender
Guillermo del Toro
```

Given the state of the film industry today, the names that this model was able extrapolate are of various directors/producers who have worked on a lot of critically and commercially successful films, such as Stanley Kubrick, Martin Scorsese, James Cameron, Christopher Nolan, etc.

REPEAT NOW: Create Rating Regression Model Based Only On Genres

In [40]:
```
X = createFeatures('genres')
Xnorm = normalize(X, axis = 0, norm = 'l2')
```

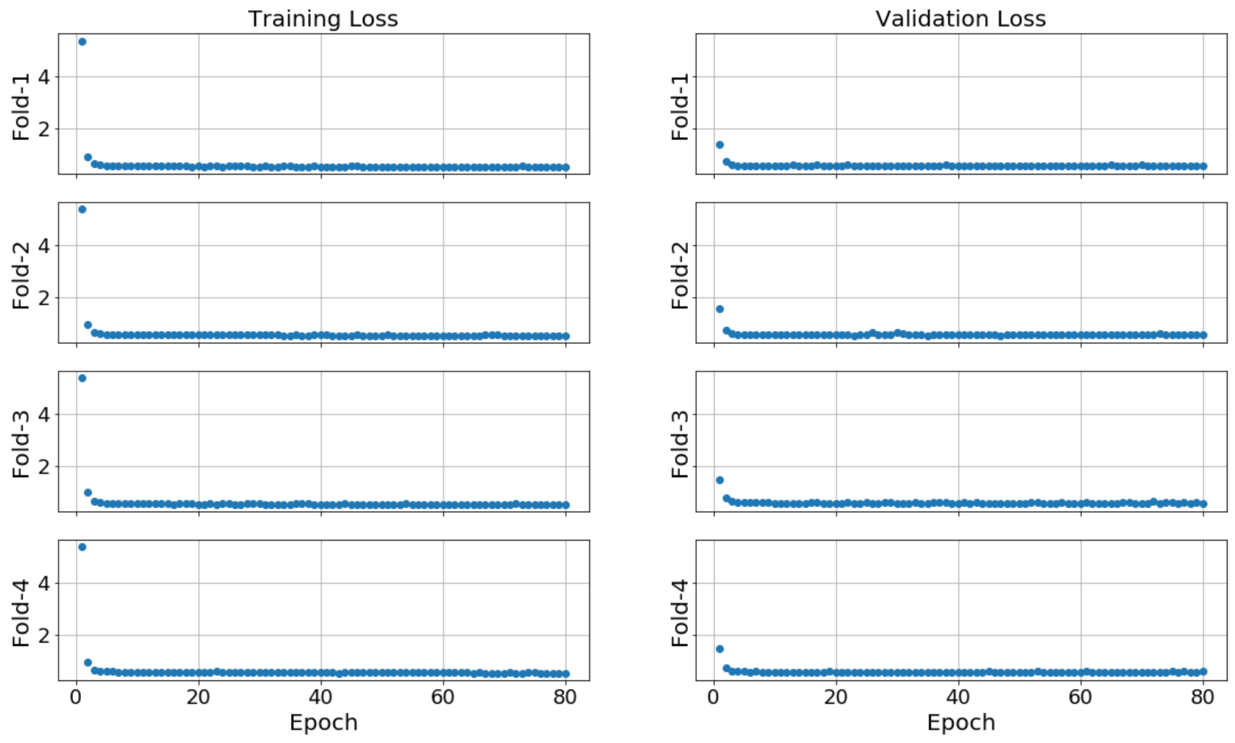
In [41]:
```
# define 10-fold cross validation test harness
seed = 7
num_epochs = 80
folds =4

initialWeights = keras.initializers.RandomNormal(mean=0.0, stddev=0.07, seec

kfold = KFold(n_splits=folds, shuffle=True, random_state=seed)

cvscoresHistory = []

for k, (train, test) in enumerate(kfold.split(Xnorm, y)):
    currCVhistory = []
    model = Sequential()
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=128, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=1, kernel_initializer= initialWeights, activation

    model.compile(loss='mean_absolute_error',
                  optimizer = 'adam'
                 )

    history = LossHistory()
    model.fit(Xnorm[train],
              y[train],
              validation_data = (Xnorm[test],y[test]),
              epochs=num_epochs,
              batch_size=64,
              verbose=1,
              callbacks = [history]
             )

    cvscoresHistory.append(history.losses)
```

```
Train on 1726 samples, validate on 576 samples
Epoch 1/80
1726/1726 [==============================] - 0s - loss: 5.3439 - val_los
s: 1.3937
Epoch 2/80
1726/1726 [==============================] - ETA: 0s - loss: 1.098 - 0s -
loss: 0.9283 - val_loss: 0.7327
Epoch 3/80
1726/1726 [==============================] - 0s - loss: 0.6532 - val_los
s: 0.6224
Epoch 4/80
1726/1726 [==============================] - 0s - loss: 0.6067 - val_los
s: 0.5992
Epoch 5/80
1726/1726 [==============================] - 0s - loss: 0.5853 - val_los
s: 0.5873
Epoch 6/80
1726/1726 [==============================] - 0s - loss: 0.5751 - val_los
s: 0.5750
Epoch 7/80
```

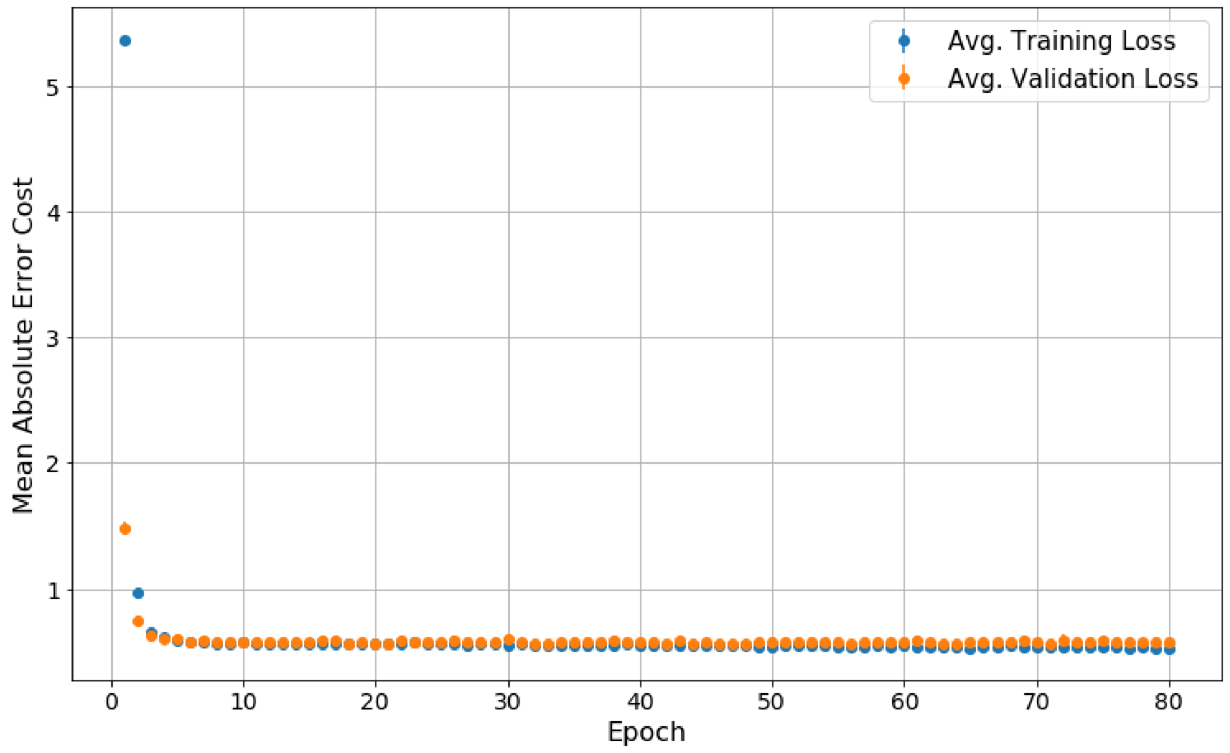
In [42]:
```
trainingLossHistory = np.asarray(cvscoresHistory)[:].T[0].T
validationLossHistory = np.asarray(cvscoresHistory)[:].T[1].T
```

In [43]:
```
meanTrainingLossHistory = np.mean(trainingLossHistory, axis = 0)
meanValidationLossHistory = np.mean(validationLossHistory, axis = 0)

stdTrainingLossHistory = np.std(trainingLossHistory, axis = 0)
stdValidationLossHistory = np.std(validationLossHistory, axis = 0)
```

In [44]:
```
iterations = [i+1 for i in range(num_epochs)]

f, axarr = plt.subplots(folds,2, sharex = True, sharey = True)
f.set_size_inches(18.5, 10.5)
axarr[0,0].set_title('Training Loss', fontsize = 20)
axarr[0,1].set_title('Validation Loss', fontsize = 20)
axarr[-1,0].set_xlabel('Epoch', fontsize = 20)
axarr[-1,1].set_xlabel('Epoch', fontsize = 20)

for i in range(folds):
    axarr[i,0].plot(iterations, trainingLossHistory[i], 'o' )
    axarr[i,0].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,0].tick_params(labelsize = 18)
    axarr[i,0].grid()

    axarr[i,1].plot(iterations, validationLossHistory[i],'o')
    axarr[i,1].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,1].tick_params(labelsize = 18)
    axarr[i,1].grid()
plt.show()
```



In [45]:
```
f = plt.figure()
f.set_size_inches(12.5, 7.5)
plt.errorbar(iterations, meanTrainingLossHistory, stdTrainingLossHistory, fm
plt.errorbar(iterations, meanValidationLossHistory, stdValidationLossHistory
plt.tick_params(labelsize = 14)
plt.legend(fontsize = 15)
plt.xlabel('Epoch', fontsize = 16)
plt.ylabel('Mean Absolute Error Cost', fontsize = 16)
plt.grid(True)
plt.show()
```



In [46]:
```
ind = np.lexsort( (meanTrainingLossHistory, meanValidationLossHistory) )
print('"ind" is an array that contains the indices (epoch #) of sorting (lea
print(ind)

print("\nTherefore, Epoch:" + str(ind[0]) + " would be a good place to stop
```

```
"ind" is an array that contains the indices (epoch #) of sorting (least t
o greatest) by first the validation loss and then by the training loss

[46 41 17 70 20 55 43 45 32 62 47 63 19 31 14 61 48 40 22 67 77 23 35  9
65
 75 13 59 26 78 38 57  7 49 33 39 66 53 79 18 54 27 36 34  8 44 12 56 50
73
 69 24 58 64 10  5 11 51 28 76 72 30 52 15 42 68 21 16 74 37 60  6 25 71
29
  4  3  2  1  0]

Therefore, Epoch:46 would be a good place to stop training this model.
```

But Does the Model Really Work?

To test this question, artificial data was created in which we only have 1 of these genres per movie. In other words, we create 1 movie for every genre in our "distinctGenres" array.

Using this array of data and a trained model, we can apply these artifical movies to our regression model and predict the ratings for these fake movies. From these fake movies we can sort their ratings from greatest to least and view what the "top choices" would be to get the highest voter rating average.

In [47]:
```
Xset = []
for i in range(0, len(distinctGenres)):
    Xcurr = np.zeros((1,X.shape[1]))
    Xcurr[0,i] = 1
    Xset.append(Xcurr)


predictions = []
for genre in Xset:
    predictions.append(float(model.predict(genre)))

idx = np.argsort(predictions)
```

In [48]:
```
dList = list(distinctGenres)

print('The best choices to make if you were to a cast only one top genre wou
for i in idx[-1::-1]:
    print(list(dList)[i])
```

```
The best choices to make if you were to a cast only one top genre would b
e:

War
Animation
Western
Drama
Romance
Mystery
Music
History
Crime
Fantasy
Family
Documentary
Adventure
Science Fiction
Thriller
Comedy
Action
Horror
TV Movie
```

Given the state of the film industry today, the order of genres that this model was able extrapolate corresponds to the critical and commercial success of our times today. Star Wars for example, contains the top 4 genres and can fit under the genres of:

- Mystery

- Romance
- Animation
- War
- Drama
- Fantasy
- and more.

REPEAT NOW: Create Rating Regression Model Based Only On Production Companies

In [49]:
```
X = createFeatures('prod')
Xnorm = normalize(X, axis = 0, norm = 'l2')
```

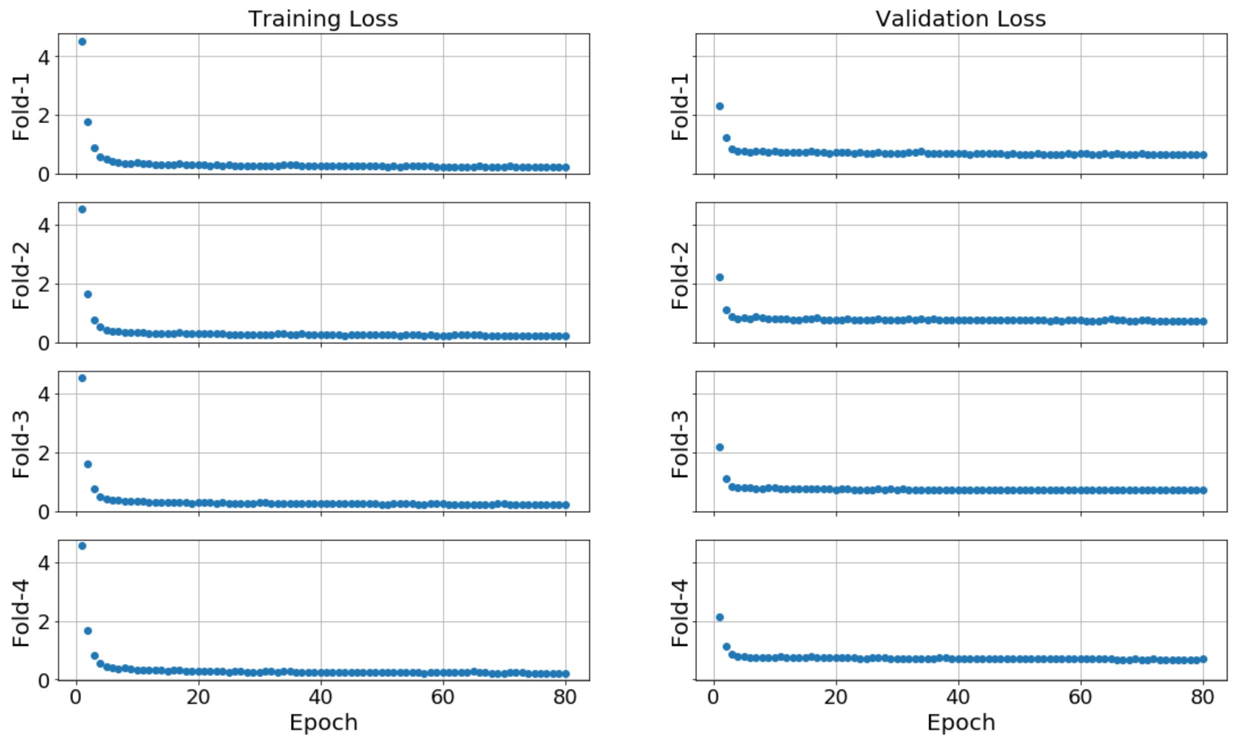
In [50]:
```
# define 10-fold cross validation test harness
seed = 7
num_epochs = 80
folds =4

initialWeights = keras.initializers.RandomNormal(mean=0.0, stddev=0.07, seec

kfold = KFold(n_splits=folds, shuffle=True, random_state=seed)

cvscoresHistory = []

for k, (train, test) in enumerate(kfold.split(Xnorm, y)):
    currCVhistory = []
    model = Sequential()
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=256, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=128, kernel_initializer= initialWeights, activatic
    model.add(Dense(units=1, kernel_initializer= initialWeights, activation

    model.compile(loss='mean_absolute_error',
                  optimizer = 'adam'
                 )

    history = LossHistory()
    model.fit(Xnorm[train],
              y[train],
              validation_data = (Xnorm[test],y[test]),
              epochs=num_epochs,
              batch_size=64,
              verbose=1,
              callbacks = [history]
             )

    cvscoresHistory.append(history.losses)
```

```
1726/1726 [==============================] - 0s - loss: 0.2780 - val_los
s: 0.6892
Epoch 30/80
1726/1726 [==============================] - 0s - loss: 0.2706 - val_los
s: 0.7055
Epoch 31/80
1726/1726 [==============================] - 0s - loss: 0.2732 - val_los
s: 0.6890
Epoch 32/80
1726/1726 [==============================] - 0s - loss: 0.2690 - val_los
s: 0.7390
Epoch 33/80
1726/1726 [==============================] - 0s - loss: 0.2745 - val_los
s: 0.7286
Epoch 34/80
1726/1726 [==============================] - 0s - loss: 0.2927 - val_los
s: 0.7672
Epoch 35/80
1726/1726 [==============================] - 0s - loss: 0.3195 - val_los
s: 0.6752
```

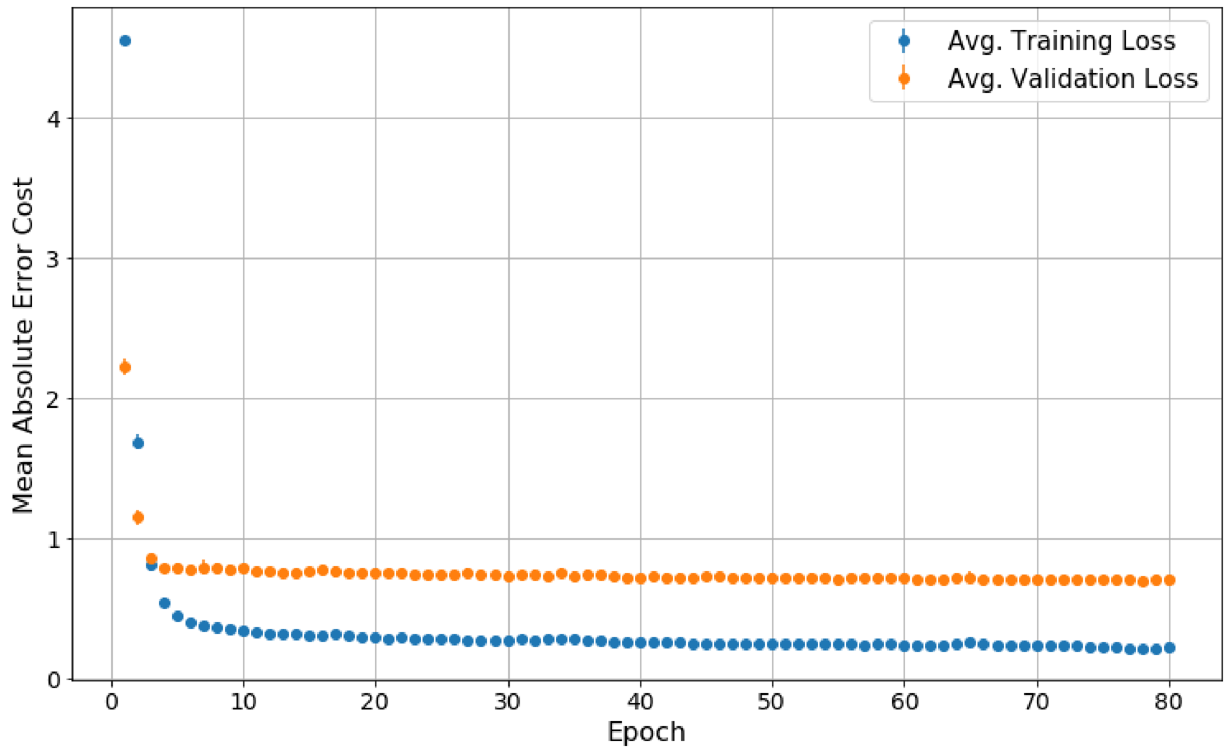
In [51]:
```
trainingLossHistory = np.asarray(cvscoresHistory)[:].T[0].T
validationLossHistory = np.asarray(cvscoresHistory)[:].T[1].T
```

In [52]:
```
meanTrainingLossHistory = np.mean(trainingLossHistory, axis = 0)
meanValidationLossHistory = np.mean(validationLossHistory, axis = 0)

stdTrainingLossHistory = np.std(trainingLossHistory, axis = 0)
stdValidationLossHistory = np.std(validationLossHistory, axis = 0)
```

In [53]:
```
iterations = [i+1 for i in range(num_epochs)]

f, axarr = plt.subplots(folds,2, sharex = True, sharey = True)
f.set_size_inches(18.5, 10.5)
axarr[0,0].set_title('Training Loss', fontsize = 20)
axarr[0,1].set_title('Validation Loss', fontsize = 20)
axarr[-1,0].set_xlabel('Epoch', fontsize = 20)
axarr[-1,1].set_xlabel('Epoch', fontsize = 20)

for i in range(folds):
    axarr[i,0].plot(iterations, trainingLossHistory[i], 'o' )
    axarr[i,0].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,0].tick_params(labelsize = 18)
    axarr[i,0].grid()

    axarr[i,1].plot(iterations, validationLossHistory[i],'o')
    axarr[i,1].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,1].tick_params(labelsize = 18)
    axarr[i,1].grid()
plt.show()
```



```
In [54]: f = plt.figure()
         f.set_size_inches(12.5, 7.5)
         plt.errorbar(iterations, meanTrainingLossHistory, stdTrainingLossHistory, fr
         plt.errorbar(iterations, meanValidationLossHistory, stdValidationLossHistory
         plt.tick_params(labelsize = 14)
         plt.legend(fontsize = 15)
         plt.xlabel('Epoch', fontsize = 16)
         plt.ylabel('Mean Absolute Error Cost', fontsize = 16)
         plt.grid(True)
         plt.show()
```



```
In [55]: ind = np.lexsort( (meanTrainingLossHistory, meanValidationLossHistory) )
         print('"ind" is an array that contains the indices (epoch #) of sorting (lea
         print(ind)

         print("\nTherefore, Epoch:" + str(ind[0]) + " would be a good place to stop
```

```
"ind" is an array that contains the indices (epoch #) of sorting (least t
o greatest) by first the validation loss and then by the training loss

[77 78 74 72 67 73 76 70 75 79 62 68 61 66 65 71 60 69 54 50 55 51 46 53
56
 57 59 48 43 49 47 58 63 52 41 39 42 64 38 45 40 44 34 37 32 29 35 28 30
24
 22 36 25 27 23 31 21 18 33 26 17 13 12 19 20 16 11 14 10  8  5 15  7  9
3
  6  4  2  1  0]

Therefore, Epoch:77 would be a good place to stop training this model.
```

But Does the Model Really Work?

To test this question, artificial data was created in which we only have 1 of these production companies per movie. In other words, we create 1 movie for every production company in our "distinctProd" array.

Using this array of data and a trained model, we can apply these artifical movies to our regression model and predict the ratings for these fake movies. From these fake movies we can sort their ratings from greatest to least and view what the "top choices" would be to get the highest voter rating average.

In [56]:
```
Xset = []
for i in range(0, len(distinctProd)):
    Xcurr = np.zeros((1,X.shape[1]))
    Xcurr[0,i] = 1
    Xset.append(Xcurr)


predictions = []
for prod in Xset:
    predictions.append(float(model.predict(prod)))

idx = np.argsort(predictions)
```

In [57]:
```
dList = list(distinctProd)

print('The best choices to make if you were to a cast only one "top" product
for i in idx[-1::-1]:
    print(list(dList)[i])
```

```
The best choices to make if you were to a cast only one "top" production
company would be:

Warner Bros.
Syncopy
WingNut Films
The Weinstein Company
Studio Ghibli
United Artists
Twentieth Century Fox Film Corporation
Sunswept Entertainment
Constantin Film Produktion
Winkler Films
Cloud Eight Films
Toho Company
Scion Films
Pixar Animation Studios
U-Drive Productions
Warner Bros. Animation
```

Given the state of the film industry today, the order of production companies that this model was able extrapolate corresponds to the critical and commercial success of our times today. For example, names such as:

- Warner Bros
- Syncopy

- WingNut Films
- United Artists
- The Weinstein COmpany
- Twentieth Century Fox
- and more.

Last And Final Interesting Thing to Attempt: Create Model That Uses All of These Modalities

In [58]:
```
Xb = createFeatures('budget')
Xca = createFeatures('cast')
Xcr = createFeatures('crew')
Xg = createFeatures('genres')
Xp = createFeatures('prod')

Xb_norm = normalize(Xb, axis = 0, norm = 'l2')
Xca_norm = normalize(Xca, axis = 0, norm = 'l2')
Xcr_norm = normalize(Xcr, axis = 0, norm = 'l2')
Xg_norm = normalize(Xg, axis = 0, norm = 'l2')
Xp_norm = normalize(Xp, axis = 0, norm = 'l2')
```

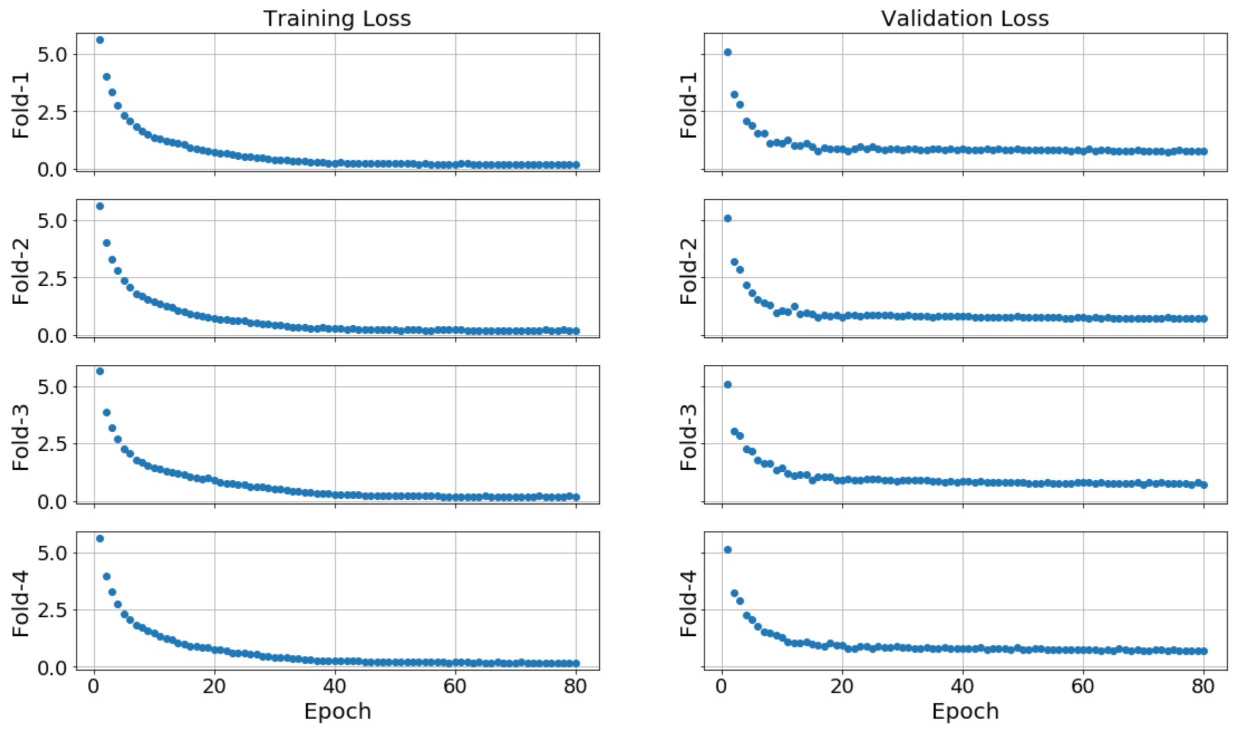
In [59]:
```
from keras.layers import Input, add
from keras.layers.normalization import BatchNormalization

# define 10-fold cross validation test harness
seed = 7
num_epochs = 80
folds =4

initialWeights = keras.initializers.RandomNormal(mean=0.0, stddev=0.07, seec

kfold = KFold(n_splits=folds, shuffle=True, random_state=seed)

cvscoresHistory = []

for k, (train, test) in enumerate(kfold.split(Xnorm, y)):
    currCVhistory = []

    budgetInput = Input(shape = (Xb.shape[1],))
    budgetBranch = Dense(units = 128, kernel_initializer = initialWeights, a
    budgetBranch = BatchNormalization()(budgetBranch)

    castInput = Input(shape = (Xca.shape[1],))
    castBranch = Dense(units = 128, kernel_initializer = initialWeights, act
    castBranch = BatchNormalization()(castBranch)

    crewInput = Input(shape = (Xcr.shape[1],))
    crewBranch = Dense(units = 128, kernel_initializer = initialWeights, act
    crewBranch = BatchNormalization()(crewBranch)

    genreInput = Input(shape = (Xg.shape[1],))
    genreBranch = Dense(units = 128, kernel_initializer = initialWeights, ac
    genreBranch = BatchNormalization()(genreBranch)

    prodInput = Input(shape = (Xp.shape[1],))
    prodBranch = Dense(units = 128, kernel_initializer = initialWeights, act
    prodBranch = BatchNormalization()(prodBranch)


    added = add([budgetBranch, castBranch, crewBranch, genreBranch, prodBran
    out = Dense(units = 1, kernel_initializer = initialWeights, activation='

    model = keras.models.Model(inputs = [budgetInput, castInput, crewInput,

    model.compile(loss='mean_absolute_error',
                  optimizer = 'adam'
                 )

    history = LossHistory()
    model.fit([Xb_norm[train], Xca_norm[train], Xcr_norm[train], Xg_norm[tra
              y[train],
              validation_data = ([Xb_norm[test], Xca_norm[test], Xcr_norm[te
              epochs=num_epochs,
              batch_size=64,
              verbose=1,
              callbacks = [history]
             )
    cvscoresHistory.append(history.losses)
```

```
1726/1726 [==============================] - 0s - loss: 0.4215 - val_los
s: 0.8692
Epoch 30/80
1726/1726 [==============================] - 0s - loss: 0.3851 - val_los
s: 0.8326
Epoch 31/80
1726/1726 [==============================] - 0s - loss: 0.3789 - val_los
s: 0.8445
Epoch 32/80
1726/1726 [==============================] - 0s - loss: 0.3582 - val_los
s: 0.8847
Epoch 33/80
1726/1726 [==============================] - 0s - loss: 0.3348 - val_los
s: 0.8354
Epoch 34/80
1726/1726 [==============================] - 0s - loss: 0.3488 - val_los
s: 0.8173
Epoch 35/80
1726/1726 [==============================] - 0s - loss: 0.3328 - val_los
s: 0.8811
```

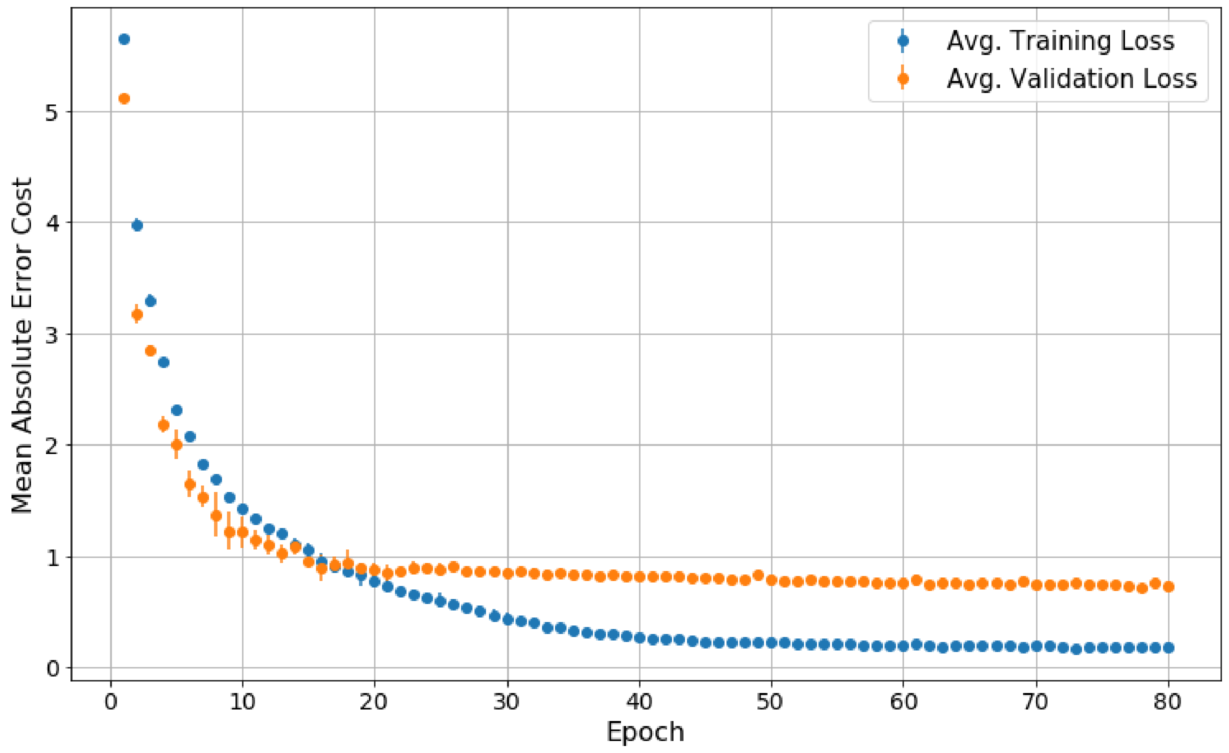
In [60]:
```
trainingLossHistory = np.asarray(cvscoresHistory)[:].T[0].T
validationLossHistory = np.asarray(cvscoresHistory)[:].T[1].T
```

In [61]:
```
meanTrainingLossHistory = np.mean(trainingLossHistory, axis = 0)
meanValidationLossHistory = np.mean(validationLossHistory, axis = 0)

stdTrainingLossHistory = np.std(trainingLossHistory, axis = 0)
stdValidationLossHistory = np.std(validationLossHistory, axis = 0)
```

In [62]:
```
iterations = [i+1 for i in range(num_epochs)]

f, axarr = plt.subplots(folds,2, sharex = True, sharey = True)
f.set_size_inches(18.5, 10.5)
axarr[0,0].set_title('Training Loss', fontsize = 20)
axarr[0,1].set_title('Validation Loss', fontsize = 20)
axarr[-1,0].set_xlabel('Epoch', fontsize = 20)
axarr[-1,1].set_xlabel('Epoch', fontsize = 20)

for i in range(folds):
    axarr[i,0].plot(iterations, trainingLossHistory[i], 'o' )
    axarr[i,0].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,0].tick_params(labelsize = 18)
    axarr[i,0].grid()

    axarr[i,1].plot(iterations, validationLossHistory[i],'o')
    axarr[i,1].set_ylabel('Fold-' + str(i+1), fontsize = 20)
    axarr[i,1].tick_params(labelsize = 18)
    axarr[i,1].grid()
plt.show()
```



In [63]:
```
f = plt.figure()
f.set_size_inches(12.5, 7.5)
plt.errorbar(iterations, meanTrainingLossHistory, stdTrainingLossHistory, fm
plt.errorbar(iterations, meanValidationLossHistory, stdValidationLossHistory
plt.tick_params(labelsize = 14)
plt.legend(fontsize = 15)
plt.xlabel('Epoch', fontsize = 16)
plt.ylabel('Mean Absolute Error Cost', fontsize = 16)
plt.grid(True)
plt.show()
```



In [64]:
```
ind = np.lexsort( (meanTrainingLossHistory, meanValidationLossHistory) )
print('"ind" is an array that contains the indices (epoch #) of sorting (lea
print(ind)

print("\nTherefore, Epoch:" + str(ind[0]) + " would be a good place to stop
```

```
"ind" is an array that contains the indices (epoch #) of sorting (least t
o greatest) by first the validation loss and then by the training loss

[77 79 76 69 71 67 74 75 64 73 70 61 78 66 65 63 59 57 72 62 58 68 56 54
55
 51 50 53 60 49 52 47 46 44 45 43 40 41 38 42 39 36 37 48 35 34 32 33 20
31
 29 28 27 21 26 30 19 24 18 15 23 22 25 16 17 14 12 13 11 10  9  8  7  6
5
  4  3  2  1  0]

Therefore, Epoch:77 would be a good place to stop training this model.
```