Predicting Movie Revenue with Neural Networks: An Analysis of Textual and Numerical Features

Lior Binnenboim - 208741405 - lior.bin@campus.technion.ac.il

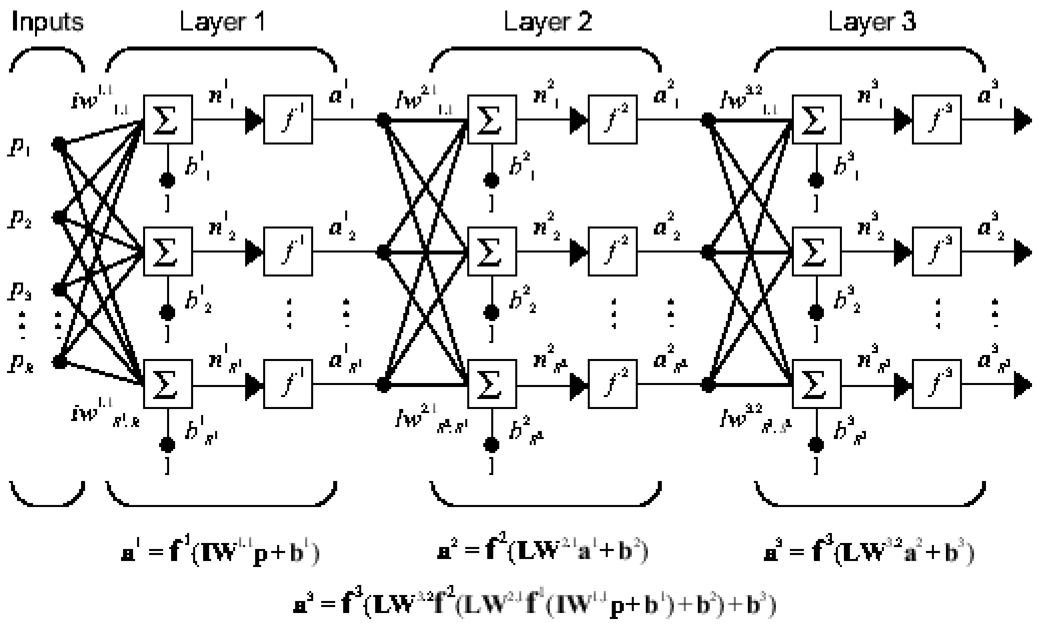
Roee Hadar - 206390098 - roee.hadar@campus.technion.ac.il

Introduction

The entertainment industry, particularly the world of cinema, is characterized by its inherent unpredictability. While the magic of storytelling lies at the heart of every film, the financial success of a movie can be elusive, influenced by a myriad of factors that extend beyond artistic merit. In this dynamic and ever-evolving landscape, the ability to predict a movie's box office revenue accurately has become an invaluable asset for studios, producers, and investors. The challenge of revenue prediction hinges on several critical variables, including but not limited to budget allocation, genre selection, film run-time and more. Our main goal is harnessing the power of machine learning, and NLP in particular, to extract insights from each movie textual descriptions and numerical features and use that to predict the film's success. We aspire to develop a tool that facilitates the exploration of how fine-tuning a movie's features can serve as a creative guide to produce films that deeply resonate with audiences, ultimately leading to heightened commercial success.
At the heart of our methodology lies the utilization of neural networks to learn intricate relations among the features. Additionally, we employ modern NLP techniques to study a numerical representation of the textual features' context. We aspire for this project to not only enhance the comprehension of prediction task involving both textual and numerical data, but also bring innovation in the field of predictive analytics within the entertainment industry.

Neural Networks Background

A neural network is a computational model inspired by the human brain, designed to solve complex tasks by learning patterns from data. At its core, a neural network is composed of interconnected layers of neurons or units. When every neuron in one layer is connected to every neuron in the next layer, this arrangement is referred to as a "fully connected layer".
In a fully connected layer, each neuron in the current layer receives input from every neuron in the previous layer. This connection is what allows information to flow from the input layer, through the hidden layers, ultimately leading to the output layer. Each of these connections has a specific weight associated with it.


Inputs
Layer 1
Layer 2
Layer 3
$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1)$
$\mathbf{a}^2 = \mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{a}^1+\mathbf{b}^2)$
$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{a}^2+\mathbf{b}^3)$
$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$

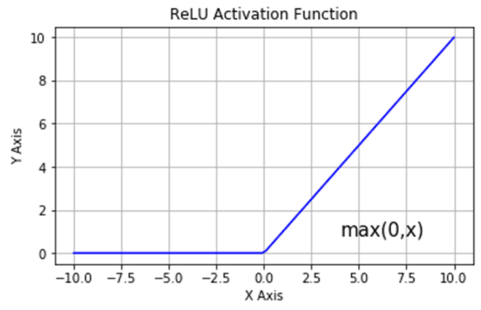
Mathematically, the transformation performed by a fully connected layer can be expressed as follows:

$\text{Output} = \text{Activation}\ (\ \text{Input} \times \text{Weights} + \text{Bias}\ )$

Here, "Input" represents the input values, which is a vector of the sample's features. "Weights" is the weight matrix controlling the linear transformation, and "Bias" is an additional parameter to shift the output.

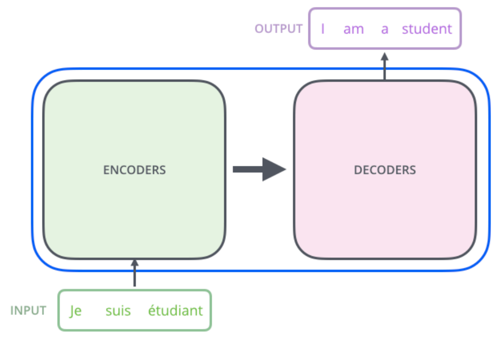
The activation function is applied to introduce non-linearity into the transformation.

One such non-linear activation function, which is the most popular in recent years, is the Rectified Linear Unit (ReLU), defined as $f(x) = \max(0, x)$. Despite its simplicity, ReLU introduces non-linearity into the network. A neural network with only linear operations (e.g., using only a linear activation) is essentially a linear model, regardless of its depth. ReLU helps the network learn and approximate non-linear patterns in the data.



Transformers Background

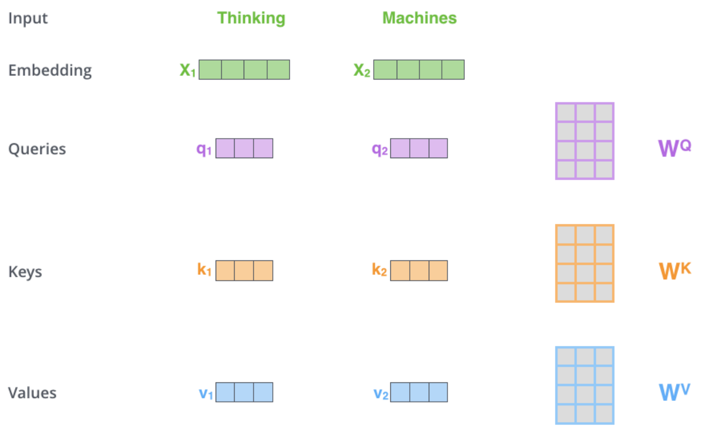
The Transformer architecture is a groundbreaking framework for natural language processing (NLP) tasks and has found applications in various domains. The architecture is composed of two main components: the encoder and the decoder. The Encoder's primary function is to process the input sequence and create a representation of it that captures the contextual information of the sequence. The Decoder's role is to generate an output sequence from the context representations created by the Encoder. It does this while maintaining the ability to attend to relevant parts of the input sequence. For example, a high level view of a transformer for the task of language translation:



Like in many natural language processing tasks, to conduct mathematical operations on the input text, it's essential to generate vector representations for the words. These representations, known as embeddings, aim to encapsulate the meaning and relationships among words. We will delve further into the details of this embedding algorithm in the upcoming section when discussing our solution.

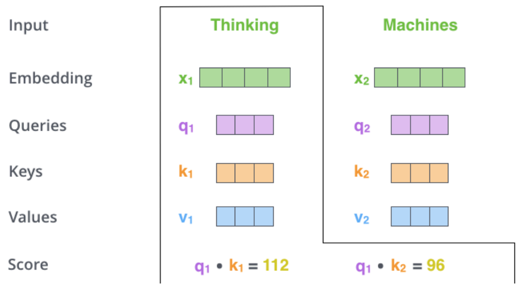
Encoder

The encoder itself is composed of two parts: the self-attention mechanism and fully connected neural networks. Self-attention computes attention scores for every word pair within the input sequence, and these scores encapsulate the connections between those words. In essence, self-attention allows the model to examine different positions in the input sequence, contributing to a deeper comprehension of how the current word influences the context.
The first step in self-attention is to create three vectors from each of the words' embeddings, a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the embedding by three matrices which are part of the transformer learned variables.

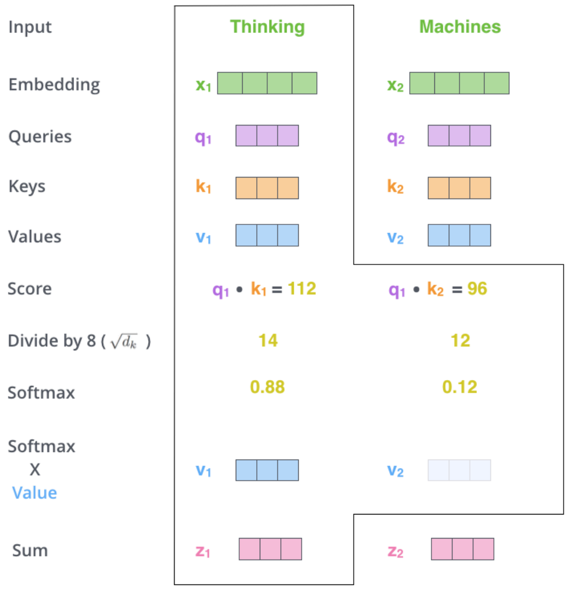


The second step in calculating self-attention is to calculate a score. The score determines how much focus to place on other parts of the input sentence as we process a word at a certain position. We take the dot product between the query vector of the current word and the key vector of each word in the input. For example, having the word

"Thinking" be the current focus and used as the query:



The scores are scaled and passed through a softmax function to obtain a probability distribution that indicates the importance or weight of each word for the current word. Softmax is a mathematical function that takes a vector of numbers and transforms it into a probability distribution so they're all positive and add up to 1. More formally, given a vector $\bar{x} = [x_1, x_2, \ldots, x_n]$, the softmax function computes the probability p_i for each x_i as follows: $p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$. Finally, we multiply each value vector by its softmax weight and sum them all up, giving us a vector representing the relationship between the word in focus and all other words in the input.



In practice, this calculation is done in matrix form for faster processing, as each computation can be done independently of the others. Those context vectors are passed to fully connected layers, this is done to learn non-linear relations between the different vectors, giving us a higher-level abstraction of the information captured by the self-attention mechanism.

Decoder

The decoder is structured into three components: a self-attention layer, a cross-attention layer, and a fully connected layer. It generates outputs sequentially, utilizing all preceding outputs to inform the generation of the subsequent ones. Consequently, within the self-attention layer, each word or embedding in consideration only looks back at the preceding outputs. This is in contrast to the encoder, which considers all of them simultaneously, achieved through the use of a mask.

The output from the encoder is transformed into a collection of attention vectors known as K (keys) and V (values). These vectors are employed by the decoder's cross-attention layer, enabling the decoder to focus its attention on relevant segments within the input sequence.

The generation process halts either when the decoder generates a special output signifying the conclusion (such as an end-of-sentence token in text generation) or when the overall output reaches a predetermined size.

The Suggested Solution

Building upon the material introduced in the previous part, our solution unfolds across various stages. We collect and process a dataset, build a model architecture involving transformers and neural networks for solving the NLP and regression tasks, and ultimately create a tool for predicting the revenue of a new unseen movie. The following sections provide a detailed exploration of our comprehensive solution.

Data Collection

The foundational step in addressing the challenge of predicting movie revenue involves the assembly of a comprehensive dataset. Currently, the most reputable source of film data is the platform IMDB, and as such, most of our data is gathered from there. Each entry in our database spans a multitude of features, including but not limited to the movie title, overview, budget, runtime, genre, rating/vote average and the revenue label. We came across many datasets that were of interest while gathering the data for the prediction task. The following were the primary considerations that helped us choose the datasets that fit our needs the best:

- Data Relevance and Coverage - Determined by the dataset characteristics/features, relevance to the purpose, and the extent of its coverage.
- Reliability and Credibility - Indicated by the specific data provided within the dataset, which was similar to other datasets tested, and by being referenced in other works of similar nature.
- Size of the Dataset - The most important aspect of the used dataset is the size of usable data, since the quality of the prediction model is dependent on having a large robust and labeled dataset. The size is determined by the amount of samples left in the dataset after the preprocessing step mentioned in the next section.

Our unprocessed dataset contains 722871 movies. This amount will decrease drastically after the preprocessing step, as many entries are not useful for the task at hand.

Data Preprocessing

With our dataset in hand, the next crucial phase involves the preprocessing of samples to refine and optimize the data for effective model training. This step encompasses a series of actions designed to enhance the relevance of our dataset:

- **Removing Entries with Faulty Data** - Entries corresponding to movies with exceptionally low revenue or budget are pruned from the dataset. This curation ensures that our model focuses on patterns and trends relevant to commercially viable movies. Another limitation of our architecture forces us to remove entries with non-English data.

- **Text Feature Aggregation** - The textual richness of a movie's title and overview is distilled into a single feature named "description". This consolidation simplifies the representation of textual elements for seamless integration into our predictive model.

- **One-Hot Encoding for Genre** - The genre feature undergoes a transformation through one-hot encoding, creating a set of binary features. This is done to transform the categorical values of the genre into numerical input.

- **Handling Zero Values in Numerical Features** - For each numerical feature, such as budget, rating and runtime, entries with zero values are replaced with the average value of the respective column. This step mitigates the impact of missing or unrealistic data points.

- **Scaling Values** - Budget and revenue columns are scaled by dividing them by a factor of 1 million to avoid working with large values in our code. Moreover, all numerical values go through the standard scaler (z-score) in order to standardize features, the formula is $z = \frac{x - \mu}{\sigma}$, where μ is the mean and σ is the standard deviation. The scale of input features can significantly impact the learning process, large and out of scale numerical values can introduce challenges for the optimization algorithms used during the training of our neural networks. The model optimizer, like gradient descent, is responsible for updating weights based on gradients during a process called back propagation. Large input values can result in large gradients, causing the model to oscillate or converge slowly. Also, if the different features work in different scales, the integrity of the learned weights might be compromised, as values in the scale 0-1 will have much less impact on the computation compared to values in scale 1M-100M.

- **Description Tokenization and Encoding** - When working with textual data in the context of neural networks, it's essential to convert the raw text into a numerical format that can be effectively processed by the model. Tokenization is the process of breaking down a text into smaller units, typically words or subwords. Encoding in this context refers to the process of converting the list of tokens into a numerical vector. This process allows us to store the description data in a format compatible with tensors, a fundamental data structures used in machine learning, enabling parallel processing and efficient matrix operations. To perform the tokenization and encoding we use the DistilBERT tokenizer, a tool specifically designed for tokenizing text in the context of transformer models. The central feature of this tokenizer lies in its utilization of the WordPiece tokenization strategy. In this approach, words are systematically broken down into smaller units, often consisting of single or pairs of characters. Through an iterative process, the tokenizer intelligently merges frequently occurring pairs of consecutive units, gradually constructing larger subwords within our vocabulary. Subsequently, each of these units is assigned a unique numerical identifier. This encoding process involves matching the text to these smaller units, allowing for an effective numerical representation of the original words.

At the end of the preprocessing step, we remain with 8224 entries.

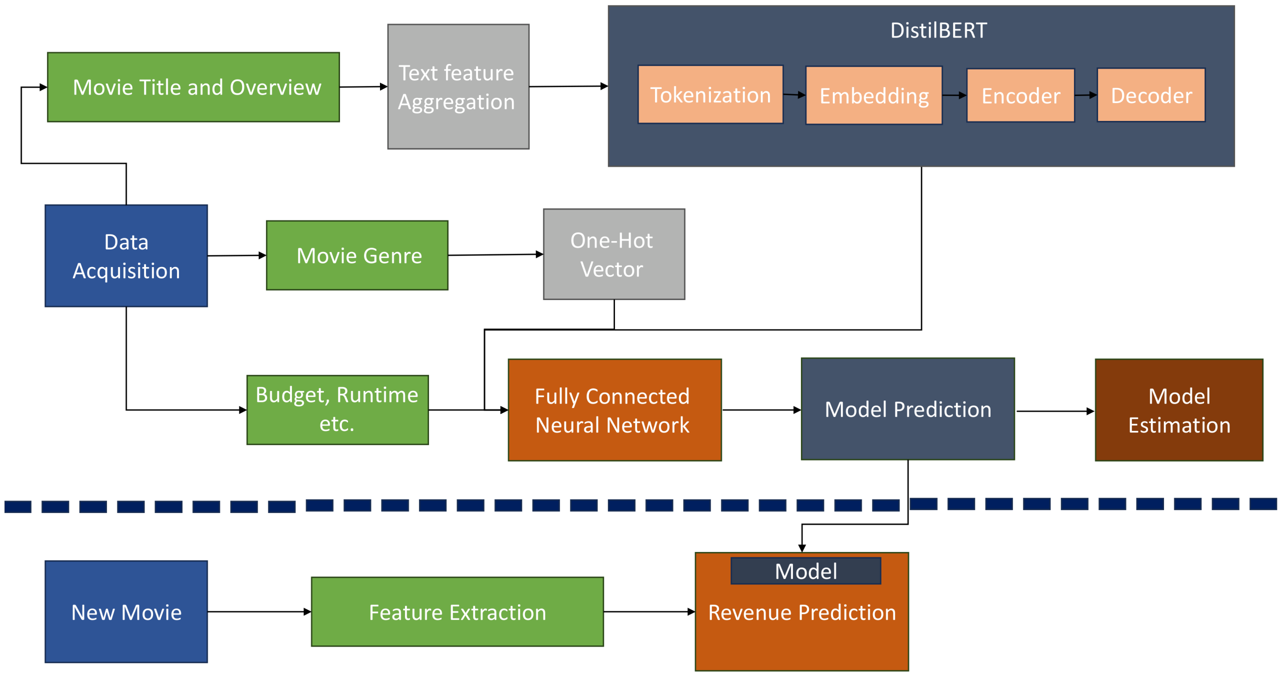
Model Architecture

For each movie in the dataset, our model processes two primary sets of features: the textual data encapsulated in the description feature, and the numerical data comprising all the other features. As a reminder, the description is stored as an encoded vector computed by the DistilBERT tokenizer. The first step in our model involves processing the encoded description through the DistilBERT transformer model, yielding a numerical vector that captures the contextual meaning of the description. The second step is to concatenate said output vector with the other numerical features vector derived from all other non-textual attributes. The final stage is to pass the concatenated vector through a series of fully connected layers. The initial layer processes the combined features and applies the Rectified Linear Unit (ReLU) activation function to introduce non-linearity. The output from the first fully connected layer is subsequently fed into the second fully connected layer, where the ReLU activation function is again applied. The final fully connected layer processes the output from the second layer and reduced the vector dimension to one, producing the prediction for the movie's revenue. Stacking multiple fully connected layers potentially increases the expressive power of the model, allowing it to learn more complex relations. Moreover, each layer adds a potential bias to our model, allowing it to account for systematic shifts or offsets in the data.

How We Leverage DistilBERT Transformer-Based Model

Recalling our tokenizer's role, it addresses the challenge of representing textual descriptions numerically. Through this tokenizer, we encode each description into a 2D numerical vector (tensor), enabling efficient matrix operations within our network. However, a new challenge emerges: Similar descriptions might receive entirely distinct encodings. Ideally, we desire similar contextual meanings to be reflected in similar numerical vectors. This is where embeddings come into play, they are vectors designed to encapsulate the meaning of words in a manner that positions semantically related words closer together in the vector space. Creating effective embeddings typically demands an extensive dataset. Fortunately, the DistilBERT model alleviates this constraint. Trained on a colossal dataset comprising billions of words, DistilBERT features an embedding layer at the beginning of its architecture. Leveraging this layer enables us to capture the semantic essence of textual descriptions. Now, similar descriptions can manifest closer proximity in the numerical vector space, enriching our model's understanding of contextual similarities.

Diagram of the Architecture



The Training Process

During the training phase, our model undergoes a series of steps aimed at optimizing its performance for the task of predicting movie revenues. This section provides a broad view of the training process without delving into the specific experiments or detailed results, which will be covered comprehensively in subsequent sections. The training is divided into the following steps:

1. The comprehensive dataset undergoes a split into training and test sets, following a 75-25 percentage division.

2. Each dataset is organized into batches, grouping the data into smaller sets of a fixed batch size. These batches are stored as tensors, facilitating efficient parallel matrix computations.

3. The mean squared error loss function is employed for the regression task of revenue prediction, and the Adam optimizer is used for weight updates. Adam (adaptive moment estimation) is an extended version of SGD (stochastic gradient descent). While SGD maintains the same learning rate for all weight updates during

training, Adam adjusts the learning rate adaptively for each parameter in the model based on the history of gradients calculated for that parameter. This allows the optimizer to converge faster and more accurately, especially in high-dimensional parameter spaces.

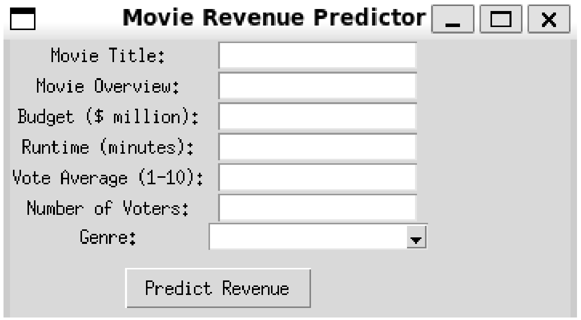
4. The training process iterates over the dataset multiple times, with each pass through the entire training data referred to as an epoch. Batches are processed in each epoch, and for every batch, revenue is computed. Subsequently, the loss is calculated against the true labels (gold labels) of the batch. The weights are updated after each batch through backpropagation.

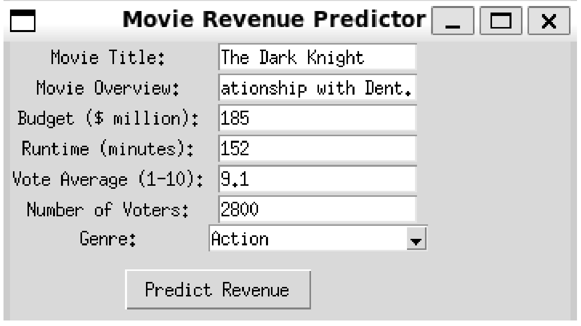
The Evaluation Process

Upon completion of training, the model's performance is assessed on previously unseen data, the test dataset (25% of the preprocessed data). This evaluation occurs in batches, where the loss is computed for each batch. The final metric returned is the average loss per movie. This evaluation provides insight into the model's predictive accuracy, measured in millions of dollars, by assessing the average difference between prediction and true movie revenues. Our final model, after taking the experiments results into account, reached an average loss of 119 million dollars on the test set.

Example of Usage

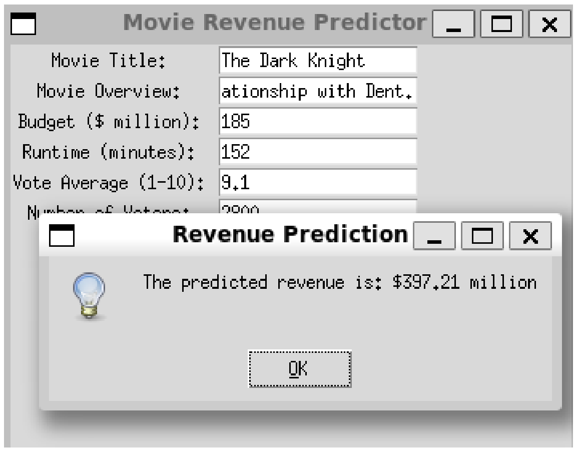
We have created a simple GUI application which is loading a pretrained model into memory, creating an interactive environment for testing different values for the film features and predicting its revenue.





Movie Revenue Predictor

Movie Title:	The Dark Knight
Movie Overview:	ationship with Dent.
Budget ($ million):	185
Runtime (minutes):	152
Vote Average (1-10):	9.1
Number of Voters:	2800
Genre:	Action


Movie Revenue Predictor

Movie Title: The Dark Knight

Movie Overview: ationship with Dent.

Budget ($ million): 185

Runtime (minutes): 152

Vote Average (1-10): 9.1

Number of Voters: 2800

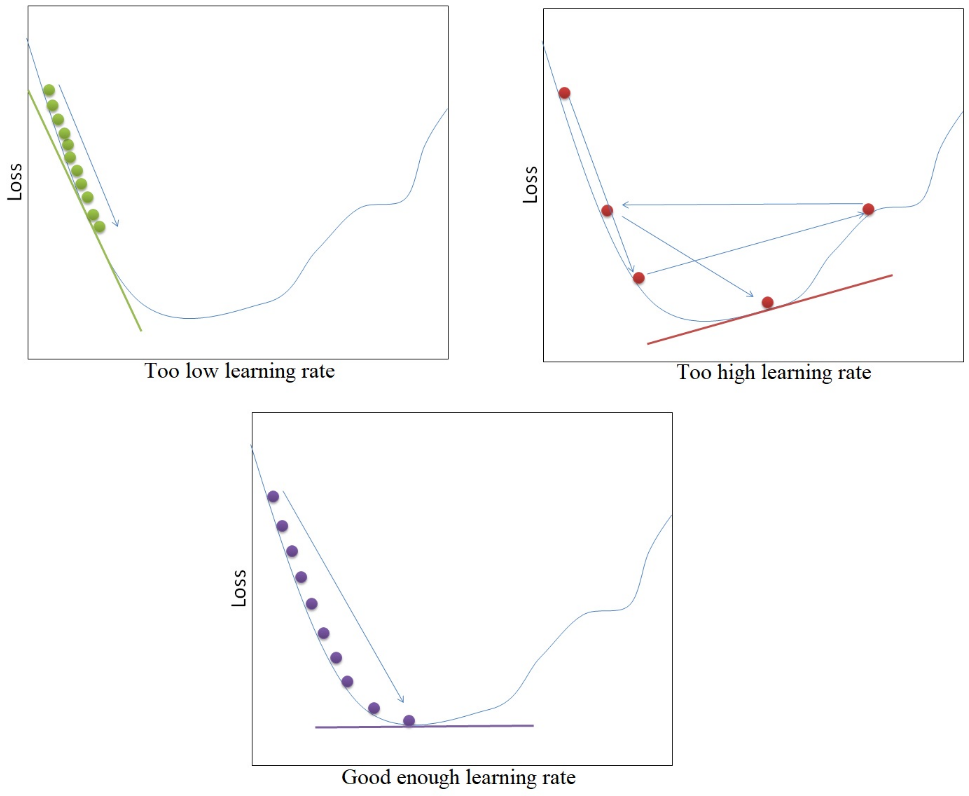
Revenue Prediction

The predicted revenue is: $397.21 million

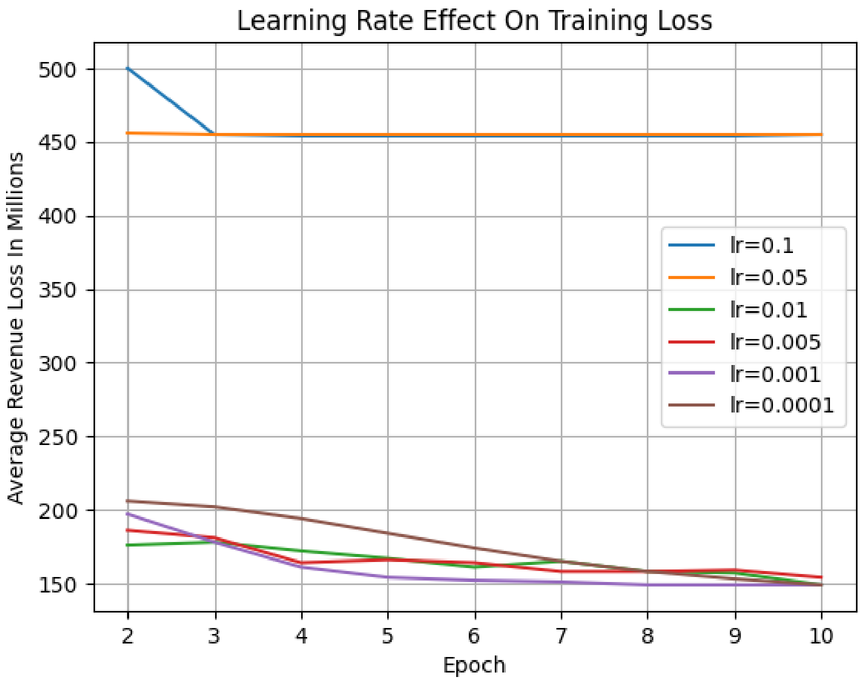
Experiments

Learning Rate Experiment

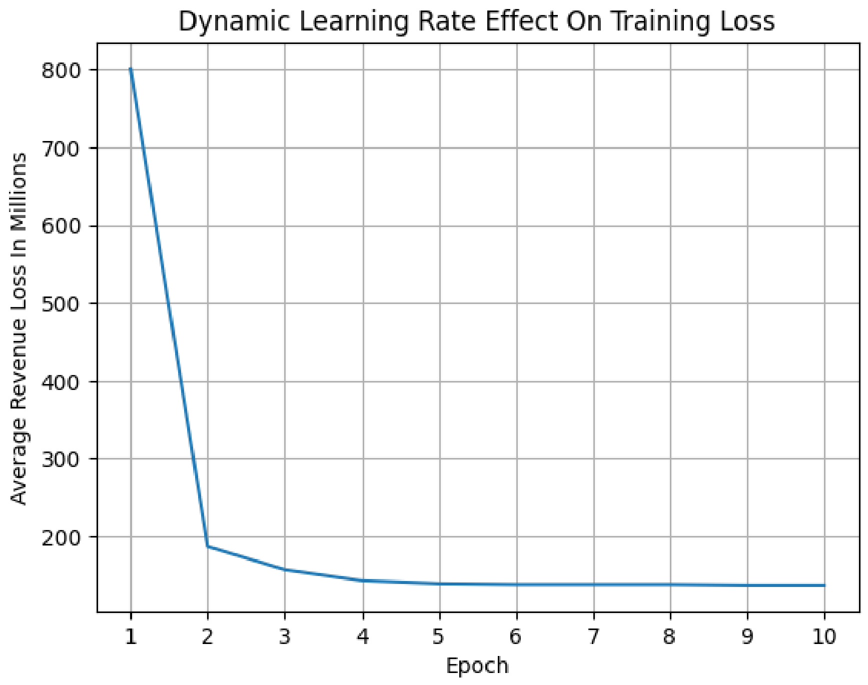
The aim of this experiment is to identify the most advantageous learning rate, a crucial parameter utilized by the optimizer to adjust model weights during training. An optimal learning rate strikes a balance between achieving rapid convergence and maintaining stable learning dynamics. The following illustration demonstrates the problem with unbalanced learning rate while searching for the minimum point using gradient descent:



Our initial step involves defining a pragmatic range for exploration. For each specified learning rate, the model undergoes training for 10 epochs, and the loss is recorded at the conclusion of each epoch. The following graph shows for each learning rate curve the recorded training loss, average difference between prediction and revenue label in millions, per epoch:



The initial epoch's loss is omitted from the graph due to exceptionally high values that distort the scale and compromise readability. Notably, learning rates of $0.1, 0.05$ rapidly approach a lower bound of approximately 450 by the third epoch. Conversely, learning rates of $0.01, 0.005, 0.001, 0.0001$ reach a lower bound of around 150, but with differing rates of convergence, larger values reach the bound more swiftly than smaller ones. These observations suggest that extended training with a smaller learning rate may yield a more favorable lower bound. However, the considerable time taken for each epoch and the marginal improvements discourage an increase in epochs, given hardware limitations and project time constraints. To address this, we experimented with a dynamic learning rate, decrementing the current value by a factor of $\sqrt{10}$ every epoch. The ensuing graph illustrates the outcomes, initialized with a learning rate of 0.01:



We attained a lower bound of 137, representing a minor enhancement, achieved within the same epoch count. This bound was reached after 5 epochs, even with the utilization of smaller learning rate values in subsequent epochs, signifying our arrival at a local minimum within the search space.

Summary

In the evaluation section, our refined model demonstrated substantial progress, achieving an average loss of 119 million dollars on the test set. Additionally, when predicting revenues for recently released movies, absent from our initial dataset, the model exhibited a commendable 60% accuracy, using the same loss metric. These promising outcomes underscore the potential of our approach, warranting further investment in extensive data collection. Given the limited training samples, we are confident that with large scale dataset, our model can be tailored for practical deployment in the film industry.

Future Work

Numerous intriguing possibilities await exploration to augment the effectiveness of the predictive model. These prospects encompass additional feature engineering, experimentation with alternative model architectures, implementation of data augmentation, and more. The primary aim of the proposed future work is to elevate prediction accuracy, all the while upholding interpretability, robustness, and practical applicability within the dynamic landscape of the film industry. The following outlines and elucidates potential avenues for extending this work:

- Ensemble Learning - Implement techniques in ensemble learning, such as combining multiple models through bagging, boosting, or stacking. This approach aims to create a prediction model that is not only more accurate but also more robust.
- Transfer Learning - Explore the potential of pre-training the DistilBERT model on an extensive movie-related dataset before utilizing it as a contextual feature extractor for textual information. This strategy has the potential to enhance the model's understanding of interrelations between films in the industry, leading to more accurate predictions.
- Feature Engineering - Conduct experiments in feature engineering by generating intricate features derived from existing ones in the database. This could involve exploring non-linear combinations of two or more features, studying covariance between features, and other advanced techniques.
- Time-Series Analysis - Incorporate time-series analysis or include time-related features to account for the temporal aspect of movie revenues. This approach aims to capture trends and seasonality in movie revenue patterns, providing a more nuanced understanding of how revenue evolves over time. Few examples include:
  - Lagged Features - representing past values of features and revenues, allowing the model to consider the impact of previous time periods on current predictions.
  - Time-Based Features - include additional features relating to time, such as special events like holidays, possibly capturing temporal patterns and trends.
  - Compute rolling statistics such as rolling mean or rolling standard deviation to smooth out short-term fluctuations and highlight long-term tendencies in the revenue time series.