Executive Summary

In the following report, we summarize our system design choices for the implementation of a neural network-based recommendation system. We cover the areas of data acquisition and database software, analytics platform and modeling frameworks, and the design of the computation and communications system.

In all areas, we have opted for open source software, where possible, and sought a balance between expected performance and implementation costs. Through our research and choices, we have designed a system that will allow us to get “up and running” at moderate investment levels, while still providing the ability to scale in the face of rapid growth.

Data and Database Software

Required Data Sources

Our initial model was built using data from the Internet Movie Database (IMDB, hereafter) and MovieLens.org. As we move into our commercial phase, the IMDB API is unfortunately no longer available to us, as their terms and conditions specifically state that the database can only be accessed and used for personal, non-commercial efforts. After researching our options, we have chosen to instead use the Open Movie Database (OMDB, hereafter) as our primary data source. The OMDB provides rich content, offering the following data about each movie:

- Title and release date
- MPAA rating
- Runtime
- Genre

- Director and writer
- Actors
- Plot summary (both full and condensed)
- Language
- Country of primary release
- Awards
- User and critic ratings, as well as vote counts, from IMDB, Rotten Tomatoes, and Metacritic

MovieLens.org provides API access to its content, so we will continue to utilize that source. To quickly capture new releases (and the change in public/critic opinion that follows a release), we will update these data sources on Monday and Friday early mornings (before 4 AM) each week. This will allow us to capture new releases (generally released at midnight Friday morning), as well as the “buzz” from the first weekend of ticket sales.

Database Design Options

As discussed in our previous report, the graph model is our recommendation for this project. Graph models can handle many-to-many relationships better than our above options. While relational models are an option at lower levels of complexity, our stated goal of reaching millions of users and large organizations such as Netflix mean that it is more than likely our levels of complexity with outstrip acceptable levels of efficiency than can be expected from that model.

Graph Database Selection

We narrowed our choices to either GraphDB or Neo4j for our graph database architecture. Neo4j is the most widely used graph database for enterprise, due largely to its speed, ease-of-adoption, and intuitive query language. Neo4j's ease-of-use, however, is offset by its lack of native support for semantic connections-- the ability to produce "extracted" knowledge, not just the top-level connections between two nodes (Anadiotis, 2017).

GraphDB is an RDF graph model that does allow those semantic, extracted layers of knowledge to be mapped; however, this additional complexity leads to computational expense when it comes to analyzing those connections.

We ultimately chose Neo4j for several reasons: 1) it has a widely known commitment to open source, one of the key requirements of this project; 2) It is optimized for high query/modeling rates, which we will experience as our scales ramps up; and, 3) it has the ability to import RDF graphs and make use of SPARQL, the RDF query language of choice. Considering the flexibility inherent in Neo4j, it seemed to be the obvious choice.

Analytics and Modeling Software

While Python will be our modeling language of choice and Tensorflow will be our underlying framework, we still need to choose the framework that will serve as our primary modeling and implementation environment. The two most popular frameworks for neural network design and implementation are Keras and PyTorch, each of which would prove an acceptable solution.

According to Migdal and Jakubanis at DeepSenseAI (2018), "Keras may be easier to get into and experiment with standard layers, in a plug & play spirit," while "PyTorch offers a lower-level approach and more flexibility for the more mathematically-inclined users." In short, Keras offers faster startup when it comes to building models and requires less specification (e.g. it automatically assigns tasks to the CPU and GPU, while PyTorch requires those tasks to be assigned manually), but it lacks the ability to fine-tune every detail of the network's processes.

Our proof-of-concept will be built in Keras, while our scaled production system will be built in PyTorch. This allows us to show leadership and investors the viability of our model (in code that is easier to follow), while giving us the ability to adjust that model in PyTorch to give it the best possible performance in production.

Computing and Communications Systems

To implement and deliver our recommendation system, we will built out two distinct cloud environments: one to handle to data ingestion and analytics, and one to deliver the customer application. We have chosen the Google Cloud Platform (GCP) as our cloud environment for two reasons: 1) low startup cost; and, 2) native support for our TensorFlow, Keras, and PyTorch frameworks. The low startup cost (free tiered hours as we develop, low cost as we scale) allows us to get our service to market with minimal investment, while still having the ability to rapidly scale up as we attract users. Google is the main research and investment sponsor behind TensorFlow, and its platform is heavily optimized for its use.

Our analytic environment will ingest the data from the OMDB and MovieLens APIs into a dedicated graph model server. The model will be trained on a separate server environment, optimized for neural network computations via parallel GPUs. Finally, the model results will be stored on a cloud server optimized for high request traffic. This way, the computational load from training such a complex network will not affect calls to the recommendation network, nor will they affect processing or maintenance on the main data servers.

Once the network has produced its results, the training and model results will be passed to an API for delivery into the customer application environment. The customer application environment consists of a web server, a database server, and an application server. These environments contain customer information and behavior and allow the client applications (both the client browser and mobile application) to access the API, generate recommendations, and pass player behavior back into the analytic environment. Figure 1 shows a simple design for this overall environment.

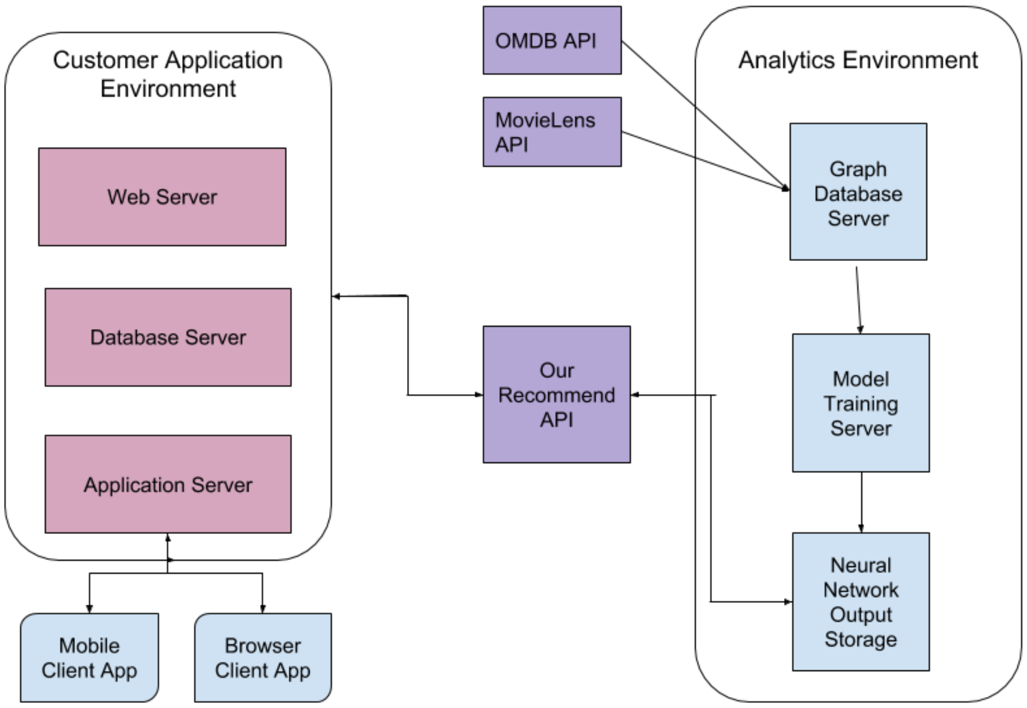

Figure 1

Summary

As seen above, we have designed a functional, scalable system with low startup overhead. By making use of open source software and cloud environments, we can build out our product with moderate investment, while still retaining the ability to meet high user demands. We identified our proposed system architecture, database environments, and development frameworks. By building upon this set of design choices, we position ourselves to become a leader in the recommendation systems market.

References

Kleppmann, M. (2017.) *Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems.* Sebastopol, Calif.: O'Reilly. [ISBN-13: 978-1449373320]

Migdal P. and Jukabanis R. (2018, June 26). Keras or PyTorch as your first deep learning framework. *deepsense.ai.* Retrieved from http://deepsense.ai

Anadiotis G. (2017, May 19). Graph databases and RDF: It's a family affair. Retrieved from https://www.zdnet.com

Imdb.com, Can I use IMDb data in my software?. Retrieved from https://help.imdb.com/

Google.com (2018, august 16). Building a Recommendation System in Tensorflow: Overview. Retrieved from https://cloud.google.com.