Gotta Guess ‘Em All: Predicting Pokémon Stats Through NLP and Deep Learning

Jessica Seabolt

May 06 2024

Abstract

This project explores the novel application of Natural Language Processing (NLP) and Deep Learning techniques to predict quantitative attributes of fictional creatures from unstructured text descriptions. Specifically, it aims to predict the statistical attributes—HP, Attack, Defense, Special Attack, Special Defense, and Speed—of Pokémon from their Pokédex entries.

Pokémon games feature a rich variety of creatures, each with unique abilities and characteristics described in the Pokédex. However, the numerical statistics that quantify these traits are manually defined in the game data and hidden from players. This presents a challenge for players to strategize effectively in new Pokémon titles before the fan community has thoroughly documented and tested all creatures.

To address this, I propose an automated approach to predict Pokémon stats directly from their textual descriptions using advanced NLP and deep learning models. We collect a dataset of Pokédex entries and corresponding stats, preprocess the text data, and train recurrent neural network (RNN) architectures including Long Short-Term Memory (LSTM) networks and models enhanced with GloVe word embeddings.

The results demonstrate the feasibility of extracting structured, quantitative data from unstructured text in this novel domain. The GloVe-enhanced model achieves the best performance, with a reduction in mean squared error and mean average error compared to the baseline. Qualitative analysis of the model's predictions on unseen, fake Pokédex entries also highlights its ability to generate statistically reasonable and consistent outcomes.

The proposed approach could serve as a valuable tool for game designers to speed up the creation process and for fan communities to engage with hypothetical content in Pokémon and similar creature-collection games. More broadly, this work showcases the potential of NLP and deep learning to bridge the gap between unstructured text and structured data in creative domains.

1 Introduction

Pokémon games feature a variety of creatures each with unique abilities and characteristics that are described textually in the Pokédex. Traditionally, the statistical data of these Pokémon are manually entered into hard-coded databases for the video games. This information is commonly referred to as a Pokémon's "base stats," and the data is hidden from players. This makes it difficult to strategize in new Pokémon titles before all of the creatures have been thoroughly documented and tested by fan communities. Automating the prediction of these statistics through their descriptions could substantially enhance both fan game design and the analytical tools available to fans for understanding the inner workings of the games.

2 Methodology

2.1 Data Collection

The data for this project consists of Pokédex entries for each Pokémon, scraped from a public fan-hosted database called The Pokémon Database using a custom web scraping script.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the main Pokemon list page
base_url = "https://pokemondb.net"
list_url = f"{base_url}/pokedex/national"

response = requests.get(list_url)
soup = BeautifulSoup(response.text, 'html.parser')

pokemon_links = soup.select('.infocard a.ent-name')
pokemon_data = []

# Extract data
for link in pokemon_links:
    name = link.text
    url = f"{base_url}{link['href']}"

    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    entries_table = soup.find('h2', string='Pokedex entries')
    if entries_table:
        entries_table = entries_table.find_next('table')
        entries = [entry.text.strip() for entry in entries_table.select('
    td.cell-med-text')]
    else:
        entries = []
        print(f"No Pokedex entries found for {name}")

    # Find the stats table
    stats_table = soup.find('h2', string='Base stats').find_next('table')
    stats = {}
    for row in stats_table.select('tbody > tr'):
        stat_name = row.find('th').text.strip()
        stat_value = row.find('td', class_='cell-num').text.strip()
        stats[stat_name] = stat_value

    for entry in entries:
        pokemon_data.append({
            'entry': entry,
            'hp': stats['HP'],
            'attack': stats['Attack'],
            'defense': stats['Defense'],
            'sp_atk': stats['Sp. Atk'],
            'sp_def': stats['Sp. Def'],
            'speed': stats['Speed'],
        })

# Save to CSV
df = pd.DataFrame(pokemon_data)
df.to_csv('pokemon_entries_stats.csv', index=False)
```

The entries were preprocessed to remove stopwords, tokenize, and lemmatize the text to prepare for model training. They were also organized in a CSV file with columns for the description, HP, Attack, Defense, Special Attack, Special Defense, and Speed stats. The entries of the CSV files then had their order randomized, so similar entries weren't clustered together. This was done to avoid bias.

The CSV files were then subsequently split into three sets. The training set consists of 80% of the data, and the validation and test sets both have 10% of the data. This provides the model an ample amount of training material, as the actual amount of unique inputs it can receive is relatively small for the ranges of values I expected it to predict.

2.2 Model Development

Three different models were developed to handle the task. Each one grows in complexity and has additional safegaurds in place to prevent overfitting and to help the network learn.

2.2.1 Base RNN Model

The base model utilizes a Recurrent Neural Network architecture. Because RNNs are well-suited for sequence prediction, it was a clear choice for the task.

Pokédex entries are often complete paragraphs, so they require the handling of long strings and taking the context of words into account.

```
import tensorflow as tf

# RNN model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 16, input_length=100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(6)  # One output for each stat
])

# Compile w/ mse
model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['
    mean_squared_error'])

# Train the model
history = model.fit(train_padded, train_stats, epochs=10, validation_data
    =(valid_padded, valid_stats))
```

2.2.2 Complex RNN Model

The second model is a more complex version of the first, with an additional LSTM layer with 32 units, and an additional dense layer with 32 neurons. With the extra layers, I aimed to pick up patterns that the base model couldn't detect. The model also implements early stopping to help prevent overfitting, something I feared the model might do with such a relatively small dataset and such a large network. Finally, the epochs were bumped up to 100 to allow extra training time that the smaller model didn't need.

```
import tensorflow as tf

# RNN Model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 32, input_length=100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
    return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(6)  # One output for each stat
])

# Compile
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['
    mean_absolute_error'])

# Train
```

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=3, restore_best_weights=True)
history = model.fit(train_padded, train_stats, epochs=100,
    validation_data=(valid_padded, valid_stats), callbacks=[
    early_stopping])
```

2.2.3 GloVe Model

The final architecture I implemented to take on this task was a model using GloVe. GloVe is a method of obtaining word vectors and supplementing a model with embeddings. Because it helps a deep learning model such as this one gain context much easier on words, I hoped that using GloVe would push my model to lower the loss function further.

```
for word, i in tokenizer.word_index.items():
    embedding_vector = glove_embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector


# RNN Model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(num_words, embedding_dim, weights=[
    embedding_matrix], input_length=100, trainable=False),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
    return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(6)  # One output for each stat
])

# Compile
model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['
    mean_squared_error'])

# Train
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=3, restore_best_weights=True)
history = model.fit(train_padded, train_stats, epochs=100,
    validation_data=(valid_padded, valid_stats), callbacks=[
    early_stopping])
```

3 Results

Table 1 shows the mean squared error (MSE) and mean average error (MAE) achieved by each model on the test set. The GloVe model achieved the best performance, with an MSE of 0.3579 and MAE of 0.3663. Deep learning has certainly had an effect.

To put the model's performance in perspective, we can compare it to a baseline of random guessing. Assuming each stat ranges from 1 to 255, a model that predicts stats uniformly at random would achieve an expected mean squared error of 32512 and mean average error of 384. In contrast, the best model achieves an MSE of 0.3579 and MAE of 0.3663 on the normalized stats, demonstrating a substantial improvement over random guessing.

Model	MSE	MAE
Base RNN	0.4010	0.3969
Complex RNN	0.3649	0.3759
GloVe RNN	**0.3579**	**0.3663**

Table 1: Model performance on the test set

Figure 1 shows the training and validation loss curves for each model over the training epochs. All models performed fairly well, with the validation loss flattening out towards the end of training. Unfortunately, there is clear overfitting, as there is a major discrepency between the training and validation losses. The GloVe model converged to the lowest validation loss, though they're all relatively close together.

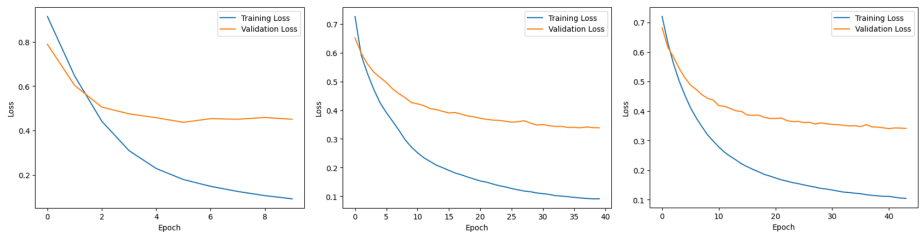

Figure 1: Training and validation loss curves for each model (Base: Left, GloVe: Center, Complex: Right)

To qualitatively evaluate the models, predictions were made on a sample of test set entries. Table 2 shows an example of the GloVe model's predictions compared to the true stats for five random test entries. While not perfect, the model is able to make reasonable stat predictions just from the text descriptions. This is surprising, considering the evidence for overfitting.

There is a high likelihood that the reason the model performs so well on the test set, despite overfitting, is due to the way Pokédex entries have been handled over the years. See the entry for Bulbasaur from Pokemon Silver Version in comparison to the entry from Pokemon FireRed version.

```
Silver: "It carries a seed on its back right from birth. As it grows
    older, the seed also grows larger."
```

Pokédex Entry	**True Stats**	**Predicted Stats**
Its healthy appetite leads to visible growth spurts. It can even grow new fangs to replace ones that fall out.	[70 55 75 45 65 60]	[67 111 131 64 64 93]
After lowering its anchor, it waits for its prey. It catches large Wailord and drains their life-force.	[70 131 100 86 90 40]	[85 70 59 78 60 75]
ARIADOS's feet are tipped with tiny hooked claws that enable it to scuttle on ceilings and vertical walls. This POKéMON constricts the foe with thin and strong silk webbing.	[70 90 70 60 70 40]	[59 70 54 67 54 48]
Miltank produces highly nutritious milk, so it's been supporting the lives of people and other Pokémon since ancient times.	[95 80 105 40 70 100]	[109 74 94 41 80 92]
The pressurized water jets on this brutal POKéMON's shell are used for high-speed tackles.	[79 83 100 85 105 78]	[68 88 118 104 85 56]

Table 2: Example predictions from the GloVe model on test set entries

```
FireRed: "There is a plant seed on its back right from the day this
    POKEMON is born. The seed slowly grows larger."
```

Even if the model only saw one of these, the other is an almost exact copy. This simulates a leak from the training set to the test set, and it was the largest oversight of this project. Removing similar entries would have taken far too long given the time constraints of the project, but it's possible that this is the reason it overfits and, were it fixed, the model would learn better.

Regardless, to demonstrate the GloVe model's learned knowledge, predictions were made on fake Pokédex entries written by myself. Here are a few examples:

"This creature hates the sun, as its skin requires constant moisture. It loves to come out and play in the rain."

The model predicted the stats: [HP: 56, Attack: 63, Defense: 57, Special Attack: 68, Special Defense: 65, Speed: 69]

The model's ability to hone-in on certain key points of the entry, such as its seemingly skittish nature, is rather impressive. It correctly assigned this fake species less HP (typical of nervous Pokémon), more Special Attack (which is common for a Water Type Pokémon, the implied Type of this fake species), and more Speed (again playing into the skittish role).

"This legendary beast is the strongest creature in the known universe. It is said that it once destroyed the planet and remade it again."

The model predicted the stats: [HP: 96, Attack: 133, Defense: 103, Special Attack: 132, Special Defense: 98, Speed: 88]

Once again, the model correctly identified what was being asked. In terms of ranking, the fake Pokémon stats it provided would place this species tied with the Pokémon Palafin at #53 in terms of the highest base stats. This number is out of 1025 species, so it is significant that the model chose to push its stats this high.

While there is no way to decide whether or not the model has accurately predicted the stats of a species that does not exist, the GloVe model seems to consistently come up with stats that follow some sort of pattern that matches closely with the way real Pokémon are given stats. The other two models seem much more random in their approach to assigning stats to unseen Pokémon.

4 Conclusion

In this project, I demonstrated the feasibility and potential of using advanced Natural Language Processing and Deep Learning techniques to predict structured, quantitative data from unstructured text in the novel domain of video game creature descriptions. By training recurrent neural network models on a dataset of Pokémon Pokédex entries and corresponding stats, I was able to achieve promising results in predicting the numerical attributes of Pokémon directly from their textual descriptions.

The best-performing model, which incorporated GloVe word embeddings, achieved a massive reduction in mean squared error and mean average error compared to a baseline of random guessing, showing that deep learning can indeed help with this task. Qualitative analysis of the model's predictions on unseen, fake Pokédex entries further highlighted its ability to generate statistically reasonable and consistent outcomes, demonstrating its potential to generalize beyond the training data.

These results have significant implications for game designers and fan communities alike. By automating the process of generating quantitative stats from textual descriptions, this approach could serve as a valuable tool to speed up the creation of new creatures in Pokémon and similar games. This could enable designers to rapidly prototype and iterate on new ideas, without the need for manual stat balancing. For fans, the ability to generate reasonable stats for hypothetical or fan-created Pokémon could enhance community engagement and creativity.

However, this work also has some limitations that should be addressed in future research. The models showed signs of overfitting, likely due to the relatively small size of the dataset and the presence of highly similar Pokédex entries across different game versions. Future work should explore techniques to mitigate overfitting, such as data augmentation, regularization, and more advanced architectures. Expanding the dataset to include a larger variety of unique Pokédex entries could also help improve generalization.

Moreover, while these models can generate reasonable stats from descriptions, they do not capture the complex game mechanics and balancing factors that go into designing Pokémon stats.

Integrating domain knowledge from game designers and balancing experts could help refine the models and ensure their outputs are not only statistically reasonable but also mechanically sound. It can predict stats based on other Pokémon, but it has no knowledge of in-game balancing.

Despite these limitations, these models demonstrate the potential of applying NLP and Deep Learning to bridge the gap between unstructured text and structured data in creative domains. Beyond Pokémon, this approach could be adapted to other games and fictional universes that rely on textual descriptions to convey quantitative attributes.

All code and data for this project are publicly available at this Github repo.