Journal of Geek Studies

jgeekstudies.org

# The Angry Video Game Model: exploring neural network architectures to predict videogame review ratings

Henrique Magalhães Soares

*Independent researcher. São Paulo, SP, Brazil.*
*Email: hemagso@gmail.com*

Videogame reviews are an essential part of the videogame industry today. From reviews by specialized gaming outlets such as IGN or Kotaku to individual reviews on platforms that sell the game themselves such as Steam, reviews are an important part of the gaming ecosystem. They allow players to identify which games are worth investing their hard-earned cash and their scarce leisure time.

Reviews have also been used as a way for customers to draw attention to an issue with the game. This practice, known as review bombing, has increased in recent years. One of the earlier high-profile examples is the review bombing of *Mass Effect 3* in 2012 due to its controversial ending, which led developer Bioware to later release an extended cut as a response (Gelbart, 2019). Other examples include the review bombing of *The Elder Scrolls V: Skyrim* following the announcement in 2015 of paid mods for the game, which ultimately led the game developer Bethesda to postpone this feature until 2017 (Gelbart, 2019), and the review bombing of *Pokémon Sword/Shield* due to the games not including every Pokémon from previous generations, among other issues (Kim & Liao, 2019).

## THE ANGRY VIDEO GAME NERD

Reviews can also be the source of a lot of fun. Numerous YouTube channels specialized themselves in making game reviews in an entertaining way, providing (sometimes) insightful commentary on the quality of games.

One of these shows is *The Angry Video Game Nerd*, a YouTube review comedy web series created by James Rolfe. To be clear, despite having increased in popularity on YouTube, the show predates it by more than a year. It was first released on Cinemassacre website on May 25, 2004 (Cinemassacre Productions LLC, n.d.). Its YouTube debut only happened on December 15, 2005 (Wikipedia, 2021).

In the show James Rolfe plays the persona of "The Nerd", an angry, short tempered and foul-mouthed character who reviews bad games (usually from the 16 or 32-bit console era) in an attempt to warn the viewer not to play them. The show is considered one of the pioneers of review videos on the internet, inspiring, for better or worse, many other content creators.

It is very common for reviews to include some kind of numerical or ordinal scale that allows customers to compare reviews or to aggregate reviews from different users into a single score. However, despite being reviews, Angry Video Game Nerd (henceforth AVGN) videos do not feature any numerical ratings that allow us to do that. In this article I aim to investigate whether ma-

**Cite this article as:** Soares, H.M. (2024) The Angry Videogame Model: exploring neural network architectures to predict videogame review ratings. Journal of Geek Studies 11(1): 7-34. https://doi.org/10.5281/zenodo.10900204

chine learning and sentiment classification techniques can help us overcome this limitation of the show's format.

**Note on profanity:** Being inspired by Rolfe's show to write this article, I will use many of his quotes to illustrate the techniques being applied here. While I used the actual verbatim text when running the models and predictions (as the removal of an expletive may change the meaning intended for the sentence), the quotes in print will not feature any of the Nerd's trademark profanity. That was done to keep this text accessible to all viewers, so I will instead use less offensive substitutes or *** in place of the expletives. These editorial changes will be marked by a gray highlight over the word.

## OBJECTIVES

This paper aims to establish a performance baseline for machine learning methods in predicting video game review scores, offering an introduction to the topic for those new to the field. While the methods described here are definitely outdated compared to recent advances brought by Large Language Models such as GPT-4 or Google Gemini, they still can be useful – both as simple and low-cost solutions for less complex language tasks and as educational tools. I believe the approach used in this article strikes a good balance between achieving a good performance and being easy to understand and follow along – this is a science communication journal, after all. (This still requires some basic understanding of linear algebra to follow along.) To facilitate this goal, all code used in this paper has been made available on Github (https://github.com/hemagso/avgm).

### Overview of this article

This article is structured in the following manner:

- First, I will present some common challenges and pitfalls encountered when dealing with natural language.
- Next, I will describe the data used in this article and detail all pre-processing done to it.
- I then discuss model architectures used in this article and present their performance and shortcomings.
- Finally, I present my conclusions on which model architecture works best for this dataset and suggest new lines of research based on this work.

## CHALLENGES FOR NLP

Before describing the work done here, I want to briefly discuss some of the challenges of working with Natural Languages from a machine learning perspective. This list is by no means exhaustive, but it should be enough to situate the reader on the challenges that must be overcome to obtain good performance for our machine learning models.

### Contextual words

Most languages have words whose meaning changes depending on where they are in the text and what other words accompany them. For example, consider the following sentence:

> *"I **ran** to the game store because we **ran** out of bad games to play."*

The highlighted word, "ran", has two different meanings in this sentence. On the first occurrence it means to quickly move yourself to another location, while in the second occurrence it is part of the "ran out" expression, which indicates that our supply of something was exhausted (in this case, bad games). Another example, more relatable for our sentiment analysis application, can be seen on the following sentence:

> *"You know what? This game is not bad."*

In this example the only way to accu-

rately assess the sentiment of the reviewers towards the game is to take "not bad" as a unit of meaning (probably indicating that the game is mediocre at best).

Machine learning models that analyze the text word by word would be unable to understand the intended meaning in these two examples. Methods that can account for the position of words in a sentence and their relation to each other are necessary when dealing with this kind of data.

**World knowledge dependence**

*"The trophy doesn't fit in the brown suitcase because it's too big."*

What is too big? The trophy or the briefcase? This might be a simple question for a human but consider how much knowledge not expressed in the sentence or on the meaning of the words themselves a person needs to answer this question. First, you need to know that briefcases can contain other things, while trophies cannot. Secondly, you need to know that an object can only be contained by a larger object.

This sentence is an example of a Winograd schema, an alternative to the Turing test proposed by Hector J. Levesque as a means to test for machine intelligence (Levesque et al., 2012). The sentences in a Winograd schema are obvious for a human reader, but exceedingly difficult to machines due to the large amount of world knowledge or indirect reasoning necessary to solve their ambiguity. (Humans usually will not even notice that there is any ambiguity at all!)

Although some recent methods have achieved accuracy rates of over 90% by exploiting extremely complicated deep neural networks and pre-trained transformer models (Kocijan et al., 2020), the Winograd schema stands as a good example of the subtle challenges in natural language processing.

**Ambiguity**

Consider the following sentence:

*"I went into the forest, where I found a bat."*

What did I find in the forest? Was it a small flying mammal, or a long piece of wood? Both answers are possible for this sentence, and without further context no correct answer can be given. Differently from Winograd schemas there is no prior world knowledge that can 100% disambiguate the meaning of this sentence. Humans might disambiguate it based on their prior beliefs on how likely each encounter is,[1] and machines can take a similar approach.

**Language detection**

Humans around the world speak lots of different languages. It is hard to pin down an exact number, as languages are constantly evolving and the distinction between a language and a regional dialect can be hard to define. As of the writing, there are 7,139 languages recognized by the Ethnologue, a reference publication on the topic (Eberhard et al., 2021).

Although NLP models can be trained on datasets consisting of multiple different languages (and some applications such as Machine Translation in fact require such datasets), it is often useful to split your problem into individual languages and then train specialist models for each. This way, your model does not need to learn how to deal with things like false friends – words that are written or sound similar in two different languages but mean completely different things. For example, "parente" in Portuguese is a false friend for "parent" in English: the former refers to any person belonging to your family, while the latter is more specific, referring only to your mother and father.

This, however, introduces another problem: how can we automatically detect the language of a text? I will discuss this problem in more detail below, but for now, let us

[1] Actually, no. They would probably ask me "What do you mean? Like, the animal?" instead of assuming something and risk making a fool out of themselves.

explore a few more challenges for NLP models.

### Spelling errors

People make mistakes, and written language is not an exception. While humans are rather good at correcting these mistakes (by noticing typos, erroneous pronunciation or context), machines are terrible at that. One of the first steps in almost all NLP models is tokenization, in which the text is split into small pieces that are mapped to a predefined set of tokens, and a misspelled word would not map to any of these, causing what is usually called an "Out-of-vocabulary" token – a token that the model never met before. The effect of these can be severe on predictions. Consider the following example:

*"This game is awful."*

This review might be tokenized into the following tokens, all of which are known by the model, having been assigned a sentiment score during training.

| Token | This | game | is | Awful |
|---|---|---|---|---|
| Score | +0 | +5 | +0 | -70 |

In this hypothetical example, the presence of the word "awful" allows our simple "Bag-of-words" model to estimate a negative sentiment for this review. But consider what would happen in the next misspelled example:

*"This game is aful."*

| Token | This | game | Is | aful |
|---|---|---|---|---|
| Score | +0 | +5 | +0 | ? |

The model has never met the word "aful", so it does not know what to make of it. It might assign it an average score, wrongly classifying the sentence as a neutral sentiment.

This problem can be partly alleviated by increasing the size of your token set to accommodate common misspellings of words. However, it is impossible to account for all occurrences, and there is a trade-off between model training time and performance and vocabulary size. A lot of research has gone into developing tokenization methods that can deal with this kind of issue, and some of them will be explored further below.

### Domain-specific vocabulary

Domain specific vocabulary happens when a specific word has a different meaning within a specific domain when compared to its everyday usage. These are quite common in science, requiring a reader to consider the topic of a text when deciding on the meaning of a word. For example, when dealing with set theory an "element" refers to an individual member of a set. If on the other hand we are discussing chemistry, an "element" refers to a chemical element, a substance consisting of atoms that have the same number of protons in their nucleus. When talking about a game, an "element" might refer to elemental spells, such as "Fire", "Air", or "Lightning", a feature common to many magic systems in games.

This problem is more severe when you are trying to build a generalist language model, as the model will need large volumes of data to learn how to differentiate between meanings. Since we are dealing with a narrow application in this article (sentiment classification for game reviews), we will not discuss this problem in depth.

### High dimensionality and sparsity

Finally, we discuss a problem that is not exclusive to NLP, but rather it is something that needs to be considered for most machine learning problems. The curse of dimensionality refers to phenomena that arise from dealing with high-dimensional spaces. In machine learning it is usually related to the fact that with an increase in dimensionality of your dataset there is an exponential increase in the amount of data required to cover all the space.

Let us work with a hypothetical example

to illustrate this problem. Imagine a classification model that takes as input 3 nominal variables, each with 10 categories. In this example, we would need at least 1,000 different training examples to cover the whole space. Now consider what would happen if instead we had 10 nominal variables with 10 categories each.

This curse of dimensionality affects text classification in a very particular way. One approach when modelling text data is to create dummy variables for each possible word:

$$D_w = \begin{cases} 1 & If\ word\ w\ is\ present \\ 0 & Otherwise \end{cases}$$

The English language, however, has a lot of words.[2] If we consider a small vocabulary set of just 1,000 words, the number of combinations needed to cover this space is greater than $10^{301}$.

Language space is also sparse, having most of the dummies described above assigned a value of zero. This happens because most texts use only a small subset of the vocabulary of the English language. Even *The Lord of the Rings*, a masterpiece of 481,103 words famous for its elaborate descriptions and flowery language uses only 15,493 distinct words (LotrProject, n.d.).

This high dimensionality and sparsity provide several challenges for training machine learning models. We will discuss "Word Embeddings", a common method for dealing with this problem further in this article.

## DATA & PREPROCESSING

### Data description

For this study I collected data from the review aggregator website Metacritic (www.metacritic.com). The data are comprised of 644,268 user reviews for 15,931 different videogames.[3] Each review is also associated with an ID that uniquely identify the user who made the review, as well as the publication date of that review on the website. Table 1 describes the fields available on the dataset.

**Table 1.** Dataset description.

| Name | Type | Description |
|---|---|---|
| game_url | URL | URL identifying the game |
| user | String | Username of the user who wrote the review |
| date | Date | Date the review was published on Metacritic |
| userscore | Integer | 0 to 10 integer score containing the score assigned by the user to the game being reviewed |
| review | String | String containing the unformatted review text |

Unfortunately, I cannot make the dataset available for further research as the contents of the review themselves are copyrighted by CBS Interactive, who owns Metacritic, as stated in their Terms of Use (https://cbsinteractive.com/legal/cbsi/terms-of-use/).

### Data preprocessing pipeline

*Language detection*

Metacritic is a website with global presence and users from a multitude of nationalities can post reviews there. Although most reviews are written in English, there are other languages represented on the dataset, and the website provides no structured data on what that language is. I utilized the *langdetect* python package (Nakatani, 2010) to identify the languages of the reviews, yielding a total of 46 different languages. The distribution of reviews among the top-10 most common languages on the dataset is shown in Table 2.

As expected, most of reviews are written in English. As I do not have enough data to train our model in multiple languages, I opted to work only with English reviews from this point forward.

[2] There are currently over 550 thousand entries on Wiktionary for English (Wiktionary, 2020), and native speakers usually have a vocabulary of around 10,000 words.

[3] This count considers games available in different platforms as entirely different games. For example, *Skyrim* on PC, PS4, Xbox, and Amazon Alexa counts as four different games.

**Table 2.** Language distribution of reviews.

| Language | # | Percent |
|---|---|---|
| English | 596,477 | 92.58% |
| Spanish | 16,25 | 2.52% |
| Russian | 12,043 | 1.87% |
| Portuguese | 7,32 | 1.14% |
| French | 2,202 | 0.34% |
| German | 1,743 | 0.27% |
| Italian | 1,234 | 0.19% |
| Somali | 695 | 0.11% |
| Polish | 584 | 0.09% |
| Turkish | 577 | 0.09% |
| other | 4530 | 0.70% |
| unknown | 613 | 0.10% |

*Train / validation / test split*

When training machine learning models, it is important to have a method to estimate your model performance in unseen data, that is, data that was not used to train the model. This avoids overfitting, a common problem when training large models in which the model ends up "learning the training data by heart" and performing poorly in unseen data.

Various methods exist to estimate the performance of the model on unseen data, but the simpler method is to simply holdout a fraction of your data, not using it to train your model parameters. This method, aptly called the holdout method, has the downside of reducing the amount of data available for the model to learn from. However, since I have enough data for my purpose, I decided to use it. I split the data into three different sets:

- Train set: data used to train the model weights through back-propagation;
- Validation set: data used to choose which model architecture is the best for this problem and to calibrate models hyperparameters;
- Test set: data used solely to estimate the performance of the final model on unseen data.

I reserved 10% of the dataset as the Validation set and 10% more as the Test set, leaving 80% of the data for model training. Table 3 shows the number of records in each set.

**Table 3.** Train / Validation / Test split.

| Dataset | Percentage | Samples |
|---|---|---|
| Train | 80% | 417,884 |
| Validation | 10% | 89,299 |
| Test | 10% | 89,13 |

*Tokenization*

The last step on our pre-processing pipeline is tokenization. As mentioned above, tokenization is the process through which we segment a text into a sequence of meaningful tokens. These tokens (after being converted into numerical id's that can be manipulated with math) are then fed into our machine learning models for training and predictions.

The choice of tokenization method can have a huge effect on how easy a model is to train, as it effectively sets the minimum level of detail from which a model can derive meaning. Hence, it is usually a trade-off between having a token that is large enough to convey sufficient information by itself and the overall number of distinct tokens in your dataset (also known as your vocabulary size). To illustrate this trade-off, consider the following choices for tokenization:

- Letter tokenization: each different letter and digit is a separate token;
- Word tokenization: each group of characters separated by whitespace or punctuation marks are a separate token;
- Sentence tokenization: each different sentence is a different token.

Let us use a quote from AVGN episode "Hong Kong 97" to illustrate the differences between these three methods: "*I've been called upon to take care of business once again. Apparently, there is a game worse than* Big Rigs. *WORSE than* Dr. Jekyll and Mr. Hyde. *WORSE than* CrazyBus *or* Desert Bus. *It is known as* Hong Kong 97, *and I've been getting requests for it up the butt*."

Table 4 shows the length of the sequences produced by each tokenization method, and the number of distinct tokens produced. Notice how the number of tokens needed to represent the text decreases with token complexity. This means that each token conveys more information.

**Table 4.** Statistics for different tokenization methods.

| Method | Length | Distinct Tokens |
|---|---|---|
| Letter | 257 | 32 |
| Word | 50 | 43 |
| Sentence | 5 | 5 |

On the other hand, the uniqueness of each token also increases with token complexity. Letters will be repeated quite often, as well as most words.[4] On the other hand, it is rare for full sentences to be repeated (and those which are probably phatic constructions or other types of uninformative sentences). This is a problem for us since we need many examples of a token to allow our model to learn how it should deal with it.

A lot of different methods of tokenization have been tested for natural language processing, and today most models use a sub-word unit approach. That method is somewhere between our letter tokenization and word tokenization. Sub-word methods have several useful properties that help us to better deal with misspellings and rare words, as we will see on the next section.

*Sub-word units*

Let us explain why we would want to use tokens smaller than a word with an example. Consider the following excerpt from AVGN episode "Plumbers don't wear ties": "*Oh, so is he a **plumber**? Well, the game's called **Plumbers** Don't Wear Ties, so I guess it makes sense: he's a **plumber**, and I don't see him wearing a tie…* [Images of John wearing a tie] *...WHAT THE HECK?! You can't even trust the darn title!*"

Take note of the two highlighted words, "plumber" and "plumbers". One tokenization option if to consider both as separate tokens. However, the model would see them as completely unrelated, and would need a lot of data to learn the relationship between them.

Another option is to create two separate tokens: "plumber" and "#s". The first token is just the word plumber by itself, and the second token is just the letter s (The # symbol indicates that this token is appended to another token to form a word). Table 5 compares the tokenization of these words.

**Table 5.** Word vs. sub-word tokenization examples.

| Word | Word units | Sub-word units |
|---|---|---|
| Plumber | ["Plumber"] | ["Plumber"] |
| Plumbers | ["Plumbers"] | ["Plumber", "#s"] |

In the sub-word representation both words share a token. In this way, the model does not need to learn that both words are related. It only needs to learn that the token "#s" usually means that the previous token is plural. And there are much more examples of plurals for the model to learn this than examples of the words "Plumber" and "Plumbers".[5] This is even more useful for rarer words. Consider the word "supernaturally", for example. There are many more examples of the word "supernatural" than "supernaturally", as can be seen in Figure 1, extracted from Google N-Gram viewer. It is easier for the model to learn the meaning of "supernatural" and then learn the meaning of "#ly" as the adverbial form from all other adverbs on the dataset than trying to learn the meaning of "supernaturally" by itself.

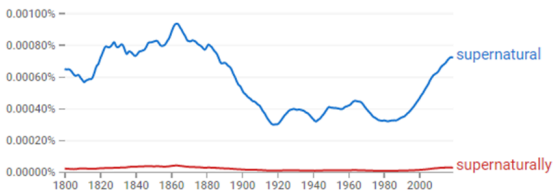



**Figure 1.** Occurrence over time for "supernatural" and "supernaturally".

[4] Unless we are dealing with rare words such as "gobbledygook" or "winklepicker". Yes, those are real words.

[5] Despite the existence of a very prolific game series with a plumber character.

For this work I opted to use the Word Piece tokenizer model. First proposed by (Wu et al., 2016) to address the problem of segmenting Korean and Japanese text,[6] this method was then adopted to automatically segment text into sub-word units. Its main advantage is being unsupervised, allowing us to learn the best token representation directly from the corpus and without the use of any annotated data. I trained the tokenizer on our train corpus to produce a vocabulary of 30,000 tokens, using the implementation available in the *tokenizers* python library (huggingface, 2021) with the following parameters:

- Normalization: I used the same normalizer as the one used by the BERT language model (Devlin et al., 2018). This normalizer replaces all types of whitespace characters by the common whitespace, replaces accented characters by their unaccented version, and applies lowercasing to all characters;
- Pre-tokenizer: I used the same pre-tokenizer as BERT, splitting on whitespace characters and punctuations to produce the first tokenization.

The details on the tokenization process are beyond the scope of this text. If you are interested in learning more, please check the accompanying *jupyter* notebooks available on Github (https://github.com/hemagso/avgm) where I go in more details about the process. To test the tokenization, let us check how it tokenized the following phrase:

> *"Feast your eyes on this accursed nonsense."*

['feast', 'your', 'eyes', 'on', 'this', 'accur', '#sed', 'nonsense', '.']

Everything seems to be working fine. Most common words consist of a single token, but the word "accursed" was split in sub-word units. With this step out of the way we can now proceed to discuss my modelling methodology.

The final step is converting the sequence of tokens into a sequence of numerical IDs, as models need things to be converted into numbers for them to be able to operate on them. For tokenizers, each unique word in the vocabulary is assigned during training a unique ID. In the case of the example above, it is:

> [16746, 1456, 4308, 1360, 1358, 4305, 5417, 6438, 15]

## METHODOLOGY

### Approaches to text classification

How can computers understand human languages? After all, computers are engineered to deal with numbers, and their language if one of numbers and symbols, rigid. Can computers understand the nuances of human language, with all its intricacies and beauty? Can a machine write a poem? Maybe.[7] But first we will need to help it turn language into math. In this section I will discuss different approaches that can be used to train text classifiers from labelled tokenized text.

#### *Bag-of-words models*

Let us go back to the sentence we tokenized above:

> [16746, 1456, 4308, 1360, 1358, 4305, 5417, 6438, 15]

The actual IDs here have absolutely no meaning and are completely arbitrary. The first step when modelling is deriving a useful representation of our data. One of the simplest options is calculating the count on

[6] Tokenization is a challenge in these languages because, in contrast to most languages based on the Latin script, Korean and Japanese words are not whitespace-separated. For example, can you spot the boundaries between words in the following text? 日本語を勉強しましたが本当に大変でした

[7] Surprisingly, yes (Lau et al., 2020), although others will have to judge its quality.

the text for a specific word and using this count as features for a classifier. This approach is called bag-of-words, and it is surprisingly effective for some application. Bag-of-word Naïve Bayes classifiers were one of the first effective spam filtering applications (Delany et al., 2013). This type of classifier works on the assumption that the mere presence of a word is informative about the dependent variable. In our review prediction problem, for instance, we could select words that are known to have a negative or positive sentiment to be part of our bag-of-words and use this to predict the sentiment for reviews:

- Positive words: good, great, excellent, masterpiece, incredible, marvelous;
- Negative words: bad, awful, terrible, trash, stupid.

This approach, however, has some flaws. It cannot take the context of the words into account, as all information about where the word is in the sentence is lost. For example, if we use the bag-of-words listed above we would not be able to properly classify the phrase "*This game is not bad.*" This is particularly important in cases where the word might not be informative by itself but is a powerful predictor when in context. In the sentences "*This game is **very** bad*" or "*This game is **slightly** bad*" the highlighted words are only informative in the presence of the word "bad". This weakness can be mitigated by building not a bag-of-words but a bag-of-n-grams. For example, we could calculate the counts of the 2-grams ("very", "bad") and ("slightly", "bad") and use those counts as features for our model. However, this starts to introduce a whole bunch of new challenges. How do I select the words in my bag of words? How can I find n-grams that are informative and should be included? The bag-of-words is a simple and surprisingly effective approach and you should definitely start with it before trying more complicated approaches – an advice that I will completely ignore in this article as I go forward to talk about Sequence Models.

*Sequence models*

So, how can I make use of the information provided by the order of the tokens in our sentence? Well, a good place to start is by not throwing it away at all. Sequence models consume the raw sequence of tokens as its input, allowing us to build architectures that can make use of the order information on the sentence.

However, this introduces a new problem. Models need not only be finite, but also of a fixed size. As we need to train the parameters in advance, the number of parameters and how they are related to each other need to be determined ahead of time. Text, however, can be of arbitrary length, and our model need to be able to deal with reviews such as "*It is good*" as well as "*This is an amazing piece of gaming history. The developers were probably inspired by God's angels when they were writing each single line of code of this masterpiece.*"

In this article we make use of recurrent architectures to solve this problem. Recurrent neural networks work by having an internal state of fixed size. An also fixed function is used to update this hidden state based on the current element of the input and the previous value. After applying this function on all elements of the sequence we are left with a fixed size state vector that can then be fed into a classifier to produce predictions. To illustrate how this type of model can work let us use a very simple mock example with a recurrent model composed by the following components:

- A 2-dimensional hidden state

  $h_t = (h_t^0, h_t^1)^T, h_0 = (0,0)^T$

- An input stream where:

  $$x_t = \begin{cases} 1 & if\ w_t\ in\ (Good, Amazing, Incredible) \\ 2 & if\ w_t\ in\ (Bad, Awful, Trash) \\ 3 & if\ w_t = Not \end{cases}, or\ 0\ otherwise$$

- An update function

  $h_t = \left(f_1(x_t, h_{t-1}), f_2(x_t, h_{t-1})\right)^T$ where

$$f_1(x,h) = \begin{cases} 1 & if\ x = 3 \\ 0 & otherwise \end{cases}$$

$$f_2(x,h) = \begin{cases} h^1 + 1 & if\ x_t = 1\ and\ h^0 = 0 \\ h^1 - 1 & if\ x_t = 1\ and\ h^0 = 1 \\ h^1 - 1 & if\ x_t = 2\ and\ h^0 = 0 \\ h^1 + 1 & if\ x_t = 2\ and\ h^0 = 1 \\ h^1 & otherwise \end{cases}$$

Before we go through an example, try to figure out what this recurrent rule does. What does $h_0$ represents? How about $h_1$? Let us run through some examples.

**Example 1:** "*This game is good*"

| t | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $w_t$ | | This | game | is | good |
| $x_t$ | | 0 | 0 | 0 | 1 |
| $h_t^0$ | 0 | 0 | 0 | 0 | 0 |
| $h_t^1$ | 0 | 0 | 0 | 0 | 1 |

**Example 2:** "*This game is bad*"

| t | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $w_t$ | | This | game | is | bad |
| $x_t$ | | 0 | 0 | 0 | 2 |
| $h_t^0$ | 0 | 0 | 0 | 0 | 0 |
| $h_t^1$ | 0 | 0 | 0 | 0 | -1 |

**Example 3:** "*This game is not good*"

| t | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $w_t$ | | This | game | is | not | good |
| $x_t$ | | 0 | 0 | 0 | 3 | 1 |
| $h_t^0$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $h_t^1$ | 0 | 0 | 0 | 0 | 0 | -1 |

**Example 4:** "*This game is not bad*"

| t | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $w_t$ | | This | game | is | not | bad |
| $x_t$ | | 0 | 0 | 0 | 3 | 2 |
| $h_t^0$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $h_t^1$ | 0 | 0 | 0 | 0 | 0 | 1 |

Notice that both negative sentiment examples got a negative $h^1$ by the end, and both positive sentiment examples got a positive one, despite examples 3 and 4 expressing those sentiments using a negation clause. The model was able to do that because it used $h^0$ as a memory of whether or not the previous word in the sequence was a negation word, allowing it to properly assess the sentiment of the words good or bad in context.

Of course, this is a toy example that only serves to illustrate the mechanism through which recurrent models can understand context. In practice, it is almost impossible to interpret the update function and the meaning of each element of the state vector in the way we did here. However, we can learn this function and representations from the data! This is the basic principle behind Recurrent Neural Networks, the method of choice for this article.

*Word embeddings*

Until now we have been using the index for the word as a categorical feature for our model, representing them by their indices. In practice, categorical features are usually represented by an encoding scheme known as one-hot encoding, where an indicator variable indicates the presence of a category:

$$x_{OH} = (I_1 \quad I_2 \quad \dots \quad I_N)^T$$

$$I_k = \begin{cases} 0 & If\ x \neq k \\ 1 & If\ x = k \end{cases}$$

This can work fine for small vocabulary of tokens, but as the vocabulary increases, we quickly start facing the problems of high-dimensionality and sparsity mentioned above. For our 30,000 words vocabulary, each element of our sequence (that is, every sub-word unit for our samples) would need to be represented by a 30,000-long vector of a single "one" and 29,999 "zeros".

One way of dealing with this problem is by using word embeddings. Word embeddings reduce the size of the representation of each word by replacing the long and sparse Boolean (only zeros and ones) vectors by smaller and dense (containing any real number). The nice thing about embeddings is that not only they can be trained from your data, but they can also be learned from unlabeled data. Below we illustrate an example for an embedding of size 4:

$$w_t = your$$
$$x_t = 1456$$
$$x_{OH} = (0, 0, 0, \dots, 0, 1, 0, \dots, 0)^T$$
$$x_E = (0.24, 3.71, -0.87, -1.33)^T$$

Embedding has many interesting properties, and there is a lot of research on methods to build embeddings. For a detailed explanation of embeddings, I recommend the work of Alammar (2019).

I will start with a simple sequence model. This will establish a baseline performance level for this task and justify some design choices that we will make going forward. It is also good practice when dealing with a new application: we start with the simplest model and build it up to address weak points identified along the way.

*Model design*

Our first model will be a Vanilla Recurrent Neural Network. The model has the architecture shown in Figure 2. Do not worry right now about what exactly a Recurrent Layer is; we will get into more detail about it later on.

| Network description | Output | #Weights |
|---|---|---|
| **Embedding Layer**<br>Vocabulary Size = 30,000 \| Embedding Size = 512 | 512 | 15,360,000 |
| **Recurrent Layer**<br>Hidden Size = 512 \| Bidirectional = Yes | 1024 | 1,050,624 |
| **Fully Connected Layer**<br>Output Size = 11 | 11 | 11,275 |
| **Activation Function**<br>Type: Log Softmax | 11 | 0 |
| | | 16,421,899 |

**Figure 2.** Simple model architecture.

Figure 2 describes the parameters of each layer, such as Embeddings Sizes and Hidden Sizes. I also noted the output tensor size (which is useful to wrap your head around on how each layer transforms its input) and the number of weights on each layer (which will be particularly useful when we are comparing different model architectures).

*Model training*

I trained this model on the train dataset described above. The training ran for 20 epochs, and at the end of each epoch performance metrics were collected both for the training set and the validation set. The model was trained using Negative Log Likelihood Loss, with the Adam optimizer (Kingma & Ba, 2015) with default parameters (Learning rate = 0.001; $\beta_1$=0.9; $\beta_2$=0.999) used for gradient descent.

*Model evaluation*

To evaluate the model, I used the following model level metrics:

- Loss: the Negative Log Likelihood value;
- Exact Accuracy (ACC): the percentage of ratings that were perfectly predicted by the model;
- Accuracy ± 1 (ACC1): the percentage of ratings that were wrong by at most 1 rating;
- Mean Absolute Error (MAE): the average distance between the true rating and the predicted rating.

I also evaluated the following class level metrics to assess the quality of the prediction:

- Recall: the percentage of records with a certain rating that were predicted with said rating;
- Precision: the percentage of records predicted with a certain rating that were indeed of that rating.

All metrics were calculated for both train and validation datasets.

*Model level metrics*

Let us start by looking at the model level metrics, and how they varied during training (Fig. 3). We can see that on average all metrics improved with training (Fig. 3), although there is a lot of variation on both the training and validation set. This could be an indication that our model is having trouble learning and that we might need a larger or more sophisticated model.

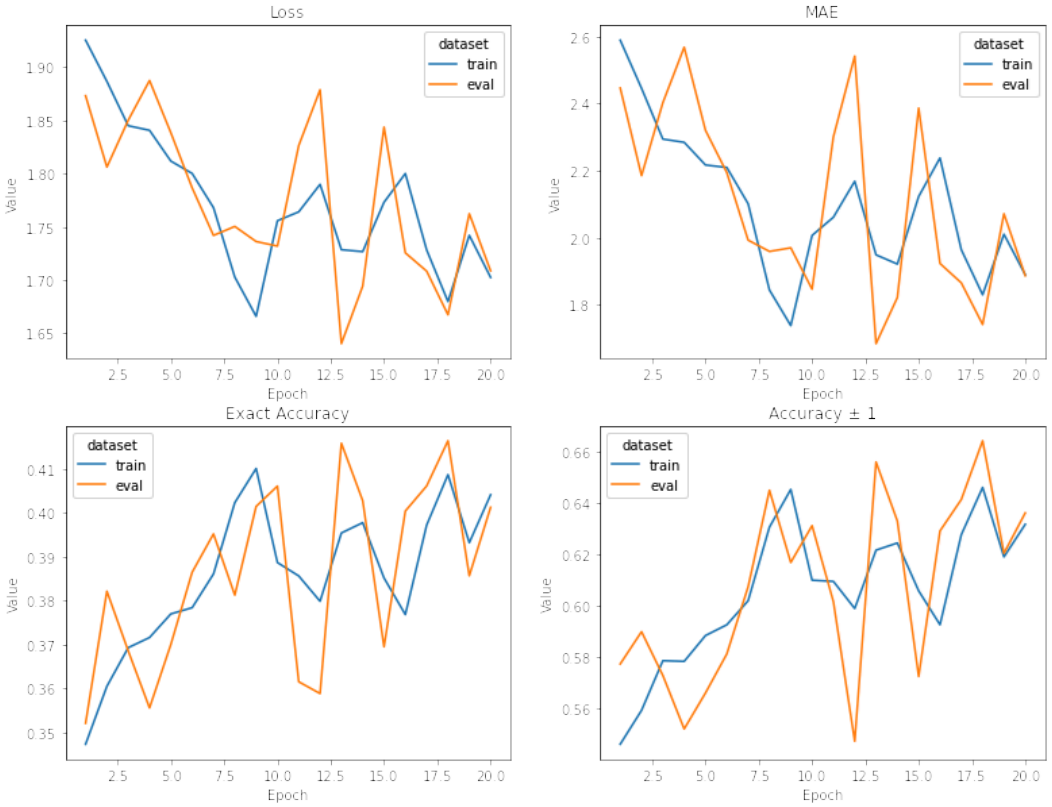



**Figure 3.** Model level metrics for a simple model over training epochs.

Using loss as our selection criteria, we see that the model achieved the best generalization (performance on unseen data) on Epoch 13:

| Loss | MAE | Exact Accuracy | Accuracy ± 1 |
|---|---|---|---|
| 1.640418 | 1.683351 | 41.6% | 65.6% |

So, how good is this model? Since this is the first application on this dataset, we do not have any established benchmarks. In this case, it is useful to look at the performances achieved by other models on similar datasets.

State-of-the-art (SOTA) performance on the IMDB dataset (a dataset with movie reviews and associated sentiment) showed 96.21% accuracy (NLP-progress, 2021). So, our model is awful, right? Wow, not so fast! The IMDB dataset collapses the rating measurement scale, classifying all ratings 6 and below as negative, and all ratings 7 and above as positive. That reduces the task to a binary classification! So, our accuracy metrics are not comparable with the IMDB dataset.

The most comparable benchmark I could find was the Yelp dataset (a dataset with reviews extracted from www.yelp.com), which has a 5-level measurement scale for ratings. SOTA for this application achieves 72.8% accuracy. This indicates that, yes, my model is probably bad and that we should probably use a better architecture. (This was already indicated by the volatile loss training curve, but it is always nice to have further evidence.) Before trying to build a new model, however, let us explore a bit more this first attempt – we might learn some other useful things to incorporate into new attempts.

*Class level metrics*

Let us now look on class level metrics. These metrics will allow us to know if our model has a good performance predicting all ratings, or if for some reason it predicts some ratings better than others.

Figure 4 shows both Recall and Precision metrics for each of our classes, and we can

immediately notice something funky is going on. The model seems to perform way better when predicting ratings 0 or 10. Recall for both these ratings is high, indicating that we correctly "retrieve" 80% of these ratings. However, Precision is way lower, indicating that this outstanding recall may in fact be caused by the model favoring these two ratings instead of the other 9 possibilities. Why would that be?

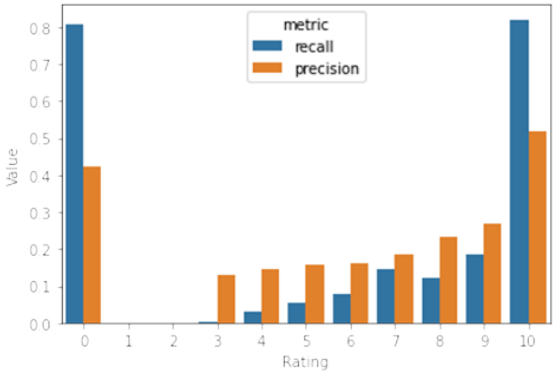



**Figure 4.** Precision and recall metrics for the simple model.

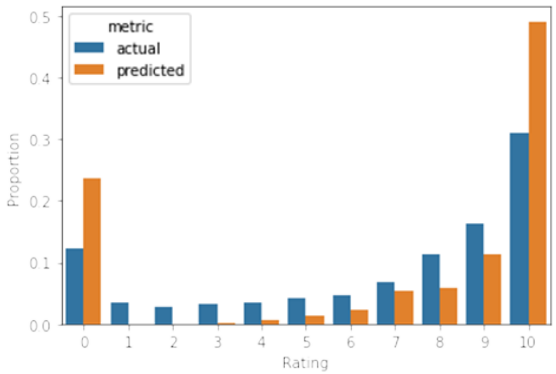



**Figure 5.** Actual and predicted class distribution for the simple model.

A quick look on the distribution of our data reveals the issue. Figure 5 shows the distribution of ratings in our dataset along the distribution of our model's predictions. The thing is: review ratings usually have an unbalanced distribution. In our case, over half the reviews is either 10 ("*This is the best game ever.*") or 0 ("*I hate this game with the power of a thousand suns.*"). Consequently, our optimization process ends up prioritizing getting those two ratings in detriment of the others, polarizing our reviews even more.

Situations like this are an example on why we should not only analyze model level metrics but also use class level metrics in the analysis. I will discuss which design decisions we can make to avoid this issue in later sections.

*Individual predictions*

It is also always useful to look at individual predictions made by our model. You might get qualitative insights that you would not notice from the aggregated data. To that end, I picked three sentences from three different AVGN episodes.

**Dr. Jekyll and Mr. Hyde:** "*You'd think I'm jokin', like I'm trying to be funny or somethin'. But, no, the fact that that game exists is a horrible abomination of mankind. That game is so freaking horrible, and I am not kidding*" (AVGN, 2010).

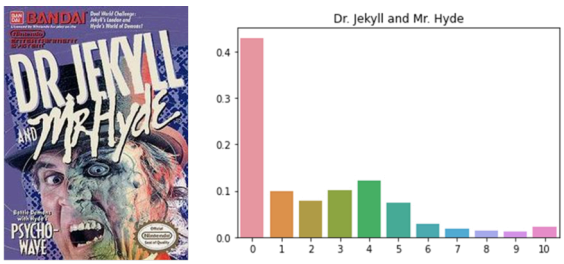



**Figure 6.** Cover art (source: Wikipedia) and predicted scores for the game *Dr. Jekyll and Mr. Hyde*.

Here we can see that the model captured the overall sentiment of the sentence, with the largest probability being assigned to rating 0, with a longer tail towards intermediate ratings.

**Earthbound:** "*I am blown away. That was one of the craziest games I've ever played. Sure, it has flaws but I think it does belong on the list of mandatory Super Nintendo games*" (AVGN, 2018).

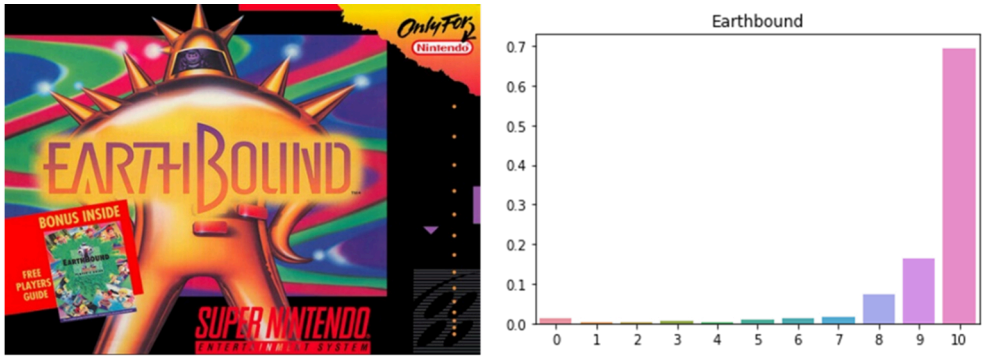



**Figure 7.** Cover art (source: Wikipedia) and predicted game scores for the game *Earthbound*.

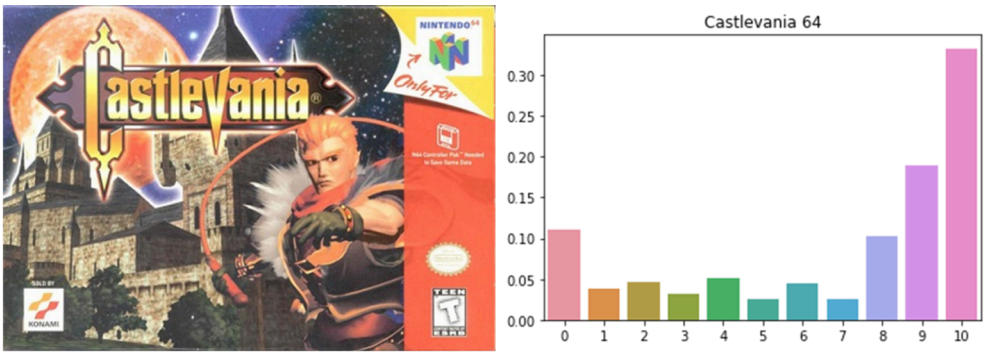



**Figure 8.** Cover art (source: Wikipedia) and predicted game scores for the game *Castlevania 64*.

Here we can see that the model is very sure of the positive sentiment of the review, assigning most of the probability to a 10 rating.

**Castlevania 64:** "*The graphics are good, for Nintendo 64 standards, but I find them unappealing, because it's the beginning of the 3D age, and they haven't perfected it yet. It's that awkward period between the old and the new*" (AVGN, 2009).

This is a mixed review, and it shows one of the flaws that the model has right now. Note how there are peaks on both rating 10 and rating 0, with a valley in between. Isn't this weird? How can the model assign a high probability for both 10 and 0, and not to anything in between?

This happens because the model has no idea that there is an order associated to the ratings. It has no idea that if the probability for a rating 10 is high, the probability for a rating 0 should be low. It treats the ratings as an unordered categorical scale, also known as a nominal scale. Below, I discuss how we can make the model aware of the order of the scale, and what trade-offs that entails.

### Model design choices

As we saw in the previous section there are several issues with the current model: poor performance overall vs. SOTA benchmarks; poor Recall and Precision on intermediate ratings; unawareness of the ordinal nature of the ratings.

All these issues stem from design decisions we made when building our model. In this section, I will present which decisions those are and discuss options to improve the model.


**Figure 9.** The XKCD model design approach (source: https://xkcd.com/1838/).

*Target variable measurement scale*

Not all measurements are created equal. Consider the following measurements associated to myself:

- my country of residence is Brazil;
- I am the eldest son in my family;
- I live near latitude -23.6 and longitude -46.7.
- as of the writing of this article, I am 32 years old.

There are different things that I can do with each of those measurements. I can compare my country of residence to another person, but I cannot calculate what "twice my country of residence" would be. You can know that my age is greater than my brothers', but without any extra information you cannot know by how much. These are examples that show that there are different types of measurements, and it is useful to be aware of that when building machine learning models.

There has been some work in statistics and measurement theory to create definitions for the different types of measurement. For instance, Stevens (1946) proposed a four-level measurement scale (Nominal, Ordinal, Interval, and Ratio). Other researchers, such as Mosteller & Tukey (1977) and Chrisman (1998), proposed more sophisticated classifications, with 7 and 10 different levels, respectively. I found in practice however that Stevens' taxonomy works well to discuss machine learning. But what exactly is each kind of measurement?

- **Nominal:** nominal measurement scales differentiate between values based on their identity. Other than that, no other comparisons can be done on measurement scales. For example, you cannot rank order them or calculate the difference between them. In the examples above "Country of residence" is a nominal scale variable. You can say that Brazil is different from the United States of America, but you cannot rank order them[8] or calculate the difference between Brazil and USA.[9] Other examples of nominal scales are Gender, Language, and Favorite Book.

- **Ordinal:** Ordinal measurement scales are like nominal scales, but they have an intrinsic order associated to them. You still cannot calculate the difference between them, but you can determine if one is greater than another, allowing one to rank order them. In the examples above "eldest" is an ordinal measurement scale. You know that by being the eldest my age is greater than my middle and youngest brothers', but you cannot know by how much. Other examples of ordinal scales are Likert scales that are commonly used in surveys to measure agreement level, and star ratings on Amazon.com reviews.

- **Interval:** Interval measurements are something that most of us might call a numerical measurement. We can not only compare and rank them, but also calculate the difference between two

[8] You can rank order them on other associated measurements, such as GDP, population, or HDI, but in those cases the measurements being ranked are those indices, not the countries themselves.

[9] Although you could argue that this difference is at least a couple of caipirinhas.

values. However, interval scales have an arbitrary zero value and, as such, their ratios are not meaningful. My location in latitude and longitude is an example of an interval scale. You can say that the difference between my latitude and someone located in Cambridge, MA, is 66 degrees. But it makes no sense to say that the ratio between my latitude and the latitude of someone in Cambridge, MA, is -0.56. Other examples of interval scales include temperatures on both Fahrenheit and Celsius scales.[10]

- **Ratio:** Ratio measurements are like interval measurements, but their scale has a well-defined and usually non-arbitrary zero scale so that calculating ratios make sense. In the examples above, my age is a ratio scale. It makes sense to say that I am twice as old as my brother. Other examples of ratio measurements include income and temperatures measured on the Kelvin scale.

Now that we know the four types of measurement scales, which one do you think best applies to videogame ratings in our dataset? We can say that one rating is greater than another, so nominal is out of the picture. But is the difference between two ratings meaningful? And more than that, is it consistent across the scale? Consider the following two completely unrelated and hypothetical cases:

- after much deliberation, you decided to increase your rating for this article from 2/10 to 3/10;
- after much deliberation, you decided to increase your rating for this article from 9/10 to 10/10.

Do you feel that the increase in rating in both cases is the same? Most people would argue no. The first increase changed the article from a bad article to a slightly "less bad" article. The second case, on the other hand, elevated it from a very good article to perfection! (Thank you, by the way.) However, people calculate metrics such as average scores all the time and, strictly speaking, you should never do that to ordinal scales! What gives?

The fact of the matter is that this is a controversial topic (see Knapp, 1990) into which we will not delve further. In this article we will compare the choice between modelling ratings as a nominal variable (in which the model is unaware of order) and as an ordinal variable (in which order is considered). It is possible to also model this target variable as an interval scale, although we need to take some extra care to avoid out of domain problems for our predictions (for instance, our model assigning a rating of 13 or -3 for a game).

*Class weights*

Review ratings (outside specialized media) have an exceedingly unbalanced distribution, as it is quite common for people to give a game a 10 if they liked it, or a 0 if they disliked it. This makes our model care more about getting 0's and 10's right than getting other ratings right, as we saw in the precision and recall metrics for our simple model (Fig. 5). Although this is the approach that maximizes overall accuracy, this can sometimes lead to useless models (see Box 1 for an example).

One way to correct for this is to assign weights for each observation, increasing the importance of the ones that are less frequent. This way, even though there are less reviews with a rating of 7, getting one of them wrong will "hurt" more (from a loss function standpoint) than getting a score of 10 wrong.

Take note that this will decrease our model's accuracy on the unbalanced dataset, but it will probably yield a more useful model in the end.

[10] And the Rankine, Rømer, Delisle, Réaumur or any other weird temperature scale that you might be using and that is not Kelvin.

**Box 1. When can maximizing accuracy be bad for your model?**
We discussed above how maximizing accuracy might not be in our best interests and might even produce useless models. But how come is that? Isn't higher accuracy always better? Not in the case of highly unbalanced datasets. Imagine you are building a machine learning classification model to detect a rare pulmonary disease based on thorax X-ray images. Less than 0.1% of people examined have the disease, which is a highly unbalanced dataset, but you decide to ignore my warning and just feed the raw data to the model. You get a model with 99.9% accuracy, which seems amazing! Until you realize it is just predicting that no one has the disease. This model might have amazing accuracy, but it is completely useless for diagnostics.

*Model architecture*

Another choice when designing our models is the architecture that will be used for our neural network. The architecture describes how each individual neural connects to one another. Good architecture allows us to reduce the model's size by exploiting some feature of the problem being addressed.

In previous sections, we hand-waved our simple model architecture, just saying it was a Recurrent Neural Network (RNN). In this section we explain this architecture in more detail and also introduce two other architectures: the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU).

- There will be a little bit of math on this section, so we better get our notations straight:
- upper case letters represent 2-D tensors (also known as matrices);
- lower case letters represent 1-D tensors (also known as vectors);
- the operator represents the Hadamard product (also known as element-wise multiplication);
- the symbol represents the hyperbolic tangent function;
- the symbol represents the sigmoid function.

With that out of the way let us describe our model architectures.

*Recurrent Neural Networks (RNN)*

The Vanilla RNN is one of the simplest examples of a sequence model there is. The model has a hidden state $h_t$ which is updated at each time step of the sequence, based on the input value at that time ($x_t$: for language models, this is usually some form of embedding representation of the input tokens) and on the value of the hidden state from the previous step ($h_{t-1}$). This update is done by the following expression:

$$h_t = \phi_h(W_{ih} \cdot x_t + b_{ih} + W_{hh} \cdot h_{t-1} + b_{hh})$$

where: $W_{ih}$ and $b_{ih}$ are tensors that describe how the input updates the hidden state; $W_{hh}$ and $b_{hh}$ are tensors that describe how the previous hidden state updates the current one.

These tensors are shared among all time steps in the sequence. This update rule can be represented by the following computation graph (Fig. 10; the bias terms $b_{ih}$ and $b_{hh}$ were omitted for brevity).

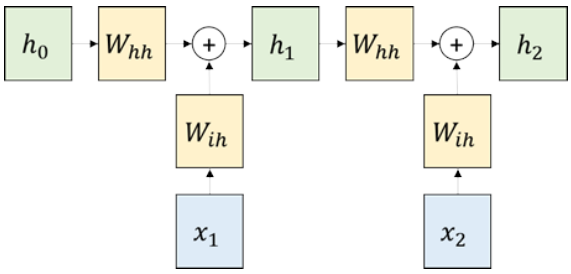



**Figure 10.** RNN computation graph.

Note that at each time step, the new state is calculated as a combination of the current input and the previous state. This way, the

model has an internal memory that allows it to remember elements seen in the past. However, Vanilla RNNs have poor performance on long sequences due to its inability to "judge" if an input needs to be remembered or not.[11] For a Vanilla RNN both the word "and" (a common uninformative stop word) and the word "awful" (a highly informative word for game reviews) are the same in terms of whether they should be remembered by the internal hidden state.

*Long Short-Term Memory (LSTM)*

LSTM neural networks are a recurrent architecture proposed by Hochreiter & Schmidhuber (1997) to improve on the long-term dependencies problem seen in vanilla RNNs. This is done by introducing an internal memory cell $c_t$ and some update gates:

- the input gate $i_t$ produces a scalar between 0 and 1 that judges how much influence the input should have on the internal memory cell. One can interpret this value as "what percentage of the input should I keep on the internal cell state?"
- the forget gate $f_t$ produces a scalar between 0 and 1 that judges how much influence should the previous cell memory state have on the new internal memory cell state. One can interpret this value as "what percentage of the previous cell state should I keep?"
- the output gate $o_t$ produces a scalar between 0 and 1 and judges how much influence should the internal memory cell have on the output value (the hidden state $h_t$). One can interpret this value as "what percentage of the cell state should I output?"

**Box 2. Sigmoids and hyperbolic tangents? What the heck are those?**

If you crossed paths with neural networks before you might have noticed that hyperbolic tangents and sigmoids show up a lot. Ever wondered why that is?

**Hyperbolic tangents**

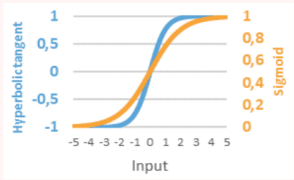
$$\phi_h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Neural networks work due to the presence of non-linearities between neurons. One common non-linearity (Inspired by the way neurons on the brain work) is an activation non-linearity – That is, the neuron does not respond to a stimulus until it reaches a certain threshold. The hyperbolic tangent function has this behavior and is differentiable and because of that it is widely used in neural networks.

**Sigmoids**

$$\sigma(x) = \frac{e^x}{1 + e^x}$$

Although sigmoid have a similar shape to hyperbolic tangents their use in neural networks is due to their [0, 1] domain, being the activation function of choice when dealing with probabilities.




These gates and their dynamic can be represented by the following expressions. Note how the use of the sigmoid function guarantees the scalar [0, 1] domain on the output of each gate.

$$i_t = \sigma(W_{ii} \cdot x_t + b_{ii} + W_{hi} \cdot h_{t-1} + b_{hi})$$
$$f_t = \sigma(W_{if} \cdot x_t + b_{if} + W_{hf} \cdot h_{t-1} + b_{hf})$$
$$o_t = \sigma(W_{io} \cdot x_t + b_{io} + W_{ho} \cdot h_{t-1} + b_{ho})$$
$$g_t = \phi_h(W_{ig} \cdot x_t + b_{ig} + W_{hg} \cdot h_{t-1} + b_{hg})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \phi_h(c_t)$$

I will not describe all tensors here, as the notation is analogous to the one used for RNNs. Again, all weight tensors W and b are shared among all time steps. However, as the gates depend on the input and on the hidden state, the LSTM can learn to weight the importance of different inputs and

[11] This is an oversimplified analogy, but I find it helpful to understand how clever design of networks and exploiting characteristics from your application can facilitate training. In theory, Vanilla RNNs can model arbitrarily long-term dependences on the input sequence. However, the finite precision of computers leads to numerical problems when training them via back-propagation, as the error gradients tend to vanish or explode as we move through time.

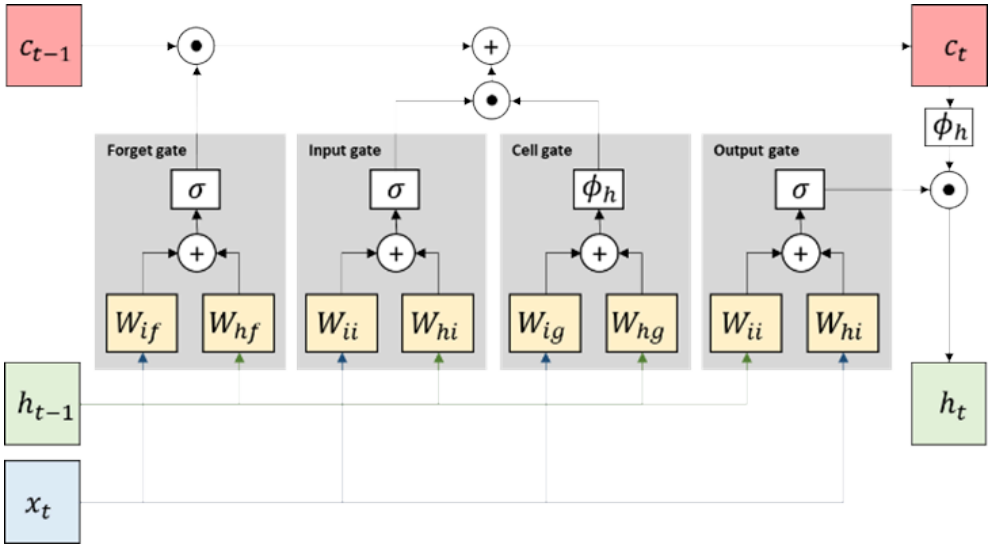



**Figure 11.** LSTM computation graph. For brevity I represent only a single time step here and, as before, omit the bias tensors.

which inputs are worth remembering. As with RNNs, we can represent the update rule with a slightly more complicated computation graph (Fig. 11). LSTMs have seen a lot of success in a wide range of applications, from speech recognition to beating human players in the popular game Dota 2 (OpenAI, 2019).

*Which models are you training, after all?*

After all the considerations made in this section, I am finally ready to present which models I ran for this article. I decided to work with the following options for the design decisions we just discussed:

- **Target measurement scale:** Nominal and Ordinal;
- **Class weights:** Unbalanced and Balanced;
- **Model architecture:** Vanilla RNN (hidden size = 256) and LSTM (hidden size = 128).

I will try all combinations between these design decisions, yielding a total of 8 different models, training every combination for 20 epochs. Note that I am using different hidden sizes between the RNN and LSTM. I did this to keep model capacity constant between architectures so that we can attribute any improvement to the change in architecture itself. If we do not do this, we would not be able to distinguish between an improvement due to the architecture and an improvement due to the increase on the number of weights of the model.

## RESULTS

After training all 8 models I can pick the best one before scoring the Nerd's reviews. To that end, I will inspect the model level metrics and class level metrics we discussed above.

### Model performance

*Selecting the best epoch*

For each model, I needed to select the best epoch before comparing their performance. This happens because although the performance on the training set will usually get better and better as you train your model, the same cannot be said about the performance on the validation set, which is the one that matters. After a while, performance on the validation set can start to de-

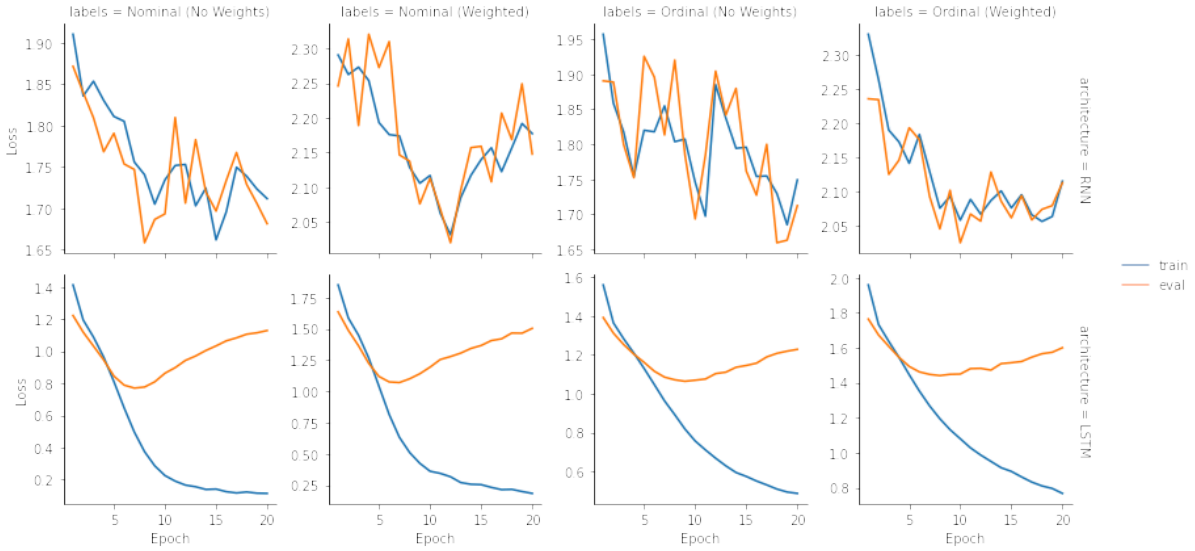



**Figure 12.** Loss over training time for RNN and LSTM.

grade, which is an indicative that you might be overfitting your training data that and getting worse at generalizing to unseen data.

With that in mind, I used the Negative Likelihood loss value for that epoch as my selection criteria for the model. Figure 12 shows how this metric behaved over training time.

As we've seen with the simple model before, both training and validation Loss are highly volatile for models using the Vanilla RNN architecture (Fig. 12). This is a strong indicative that this model is too simple for our problem, and that it might take too much training time and data for it to achieve a good performance. The LSTM architecture, on the other hand, fared way better, displaying a trend that is common for machine learning models: a constant decline on the training Loss as the model gets better and better at predicting the data it already saw and a V-shaped behavior for the validation loss as after a point the model starts to overfit the data (Fig. 12).

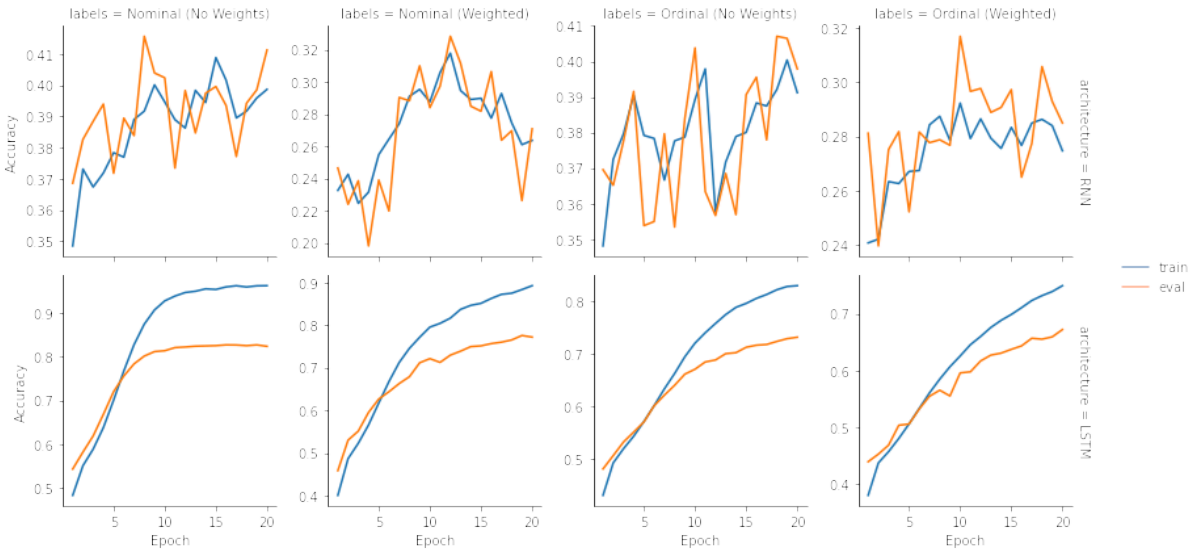



**Figure 13.** Accuracy over training time for RNN and LSTM.

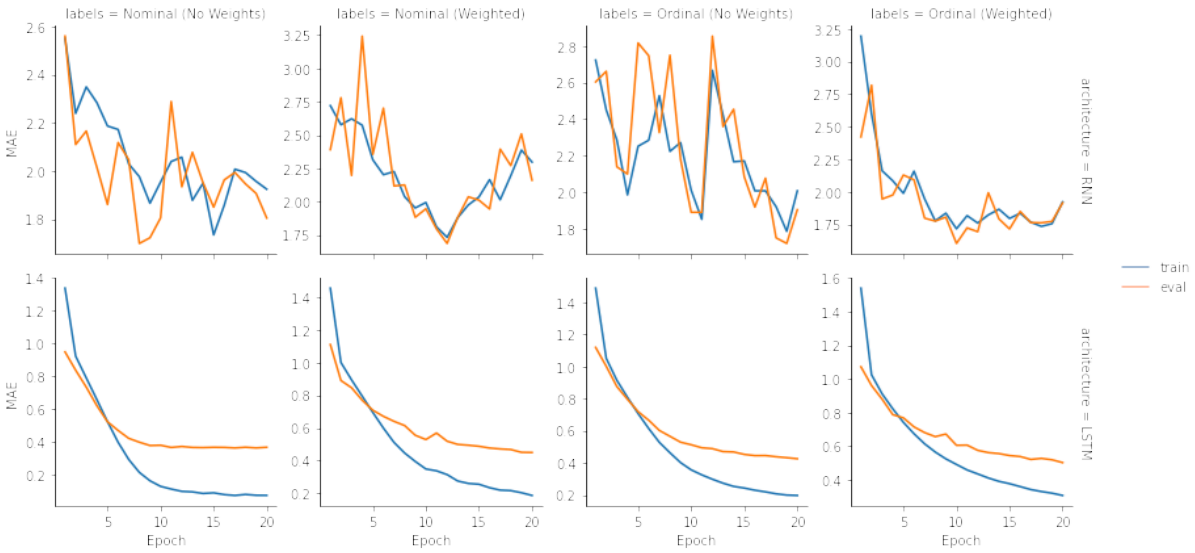



**Figure 14.** Mean Absolute Error (MAE) over training time for RNN and LSTM.

The best model will be the one with the lowest validation loss. So, let us also take a look on the Accuracy and Mean Absolute Error (MAE) metrics before continuing (Figs. 13 and 14, respectively).

Notice that, unlike what was seen for loss both accuracy and MAE are stable or even improve for the validation set beyond the point in which the model started overfitting. This happens because Accuracy and MAE take into account only the final predicted value, while the loss also considers how confident the model is on the prediction. Take the following examples:

- true review rating is 7, and the model predicted 9 with 55% accuracy;
- true review rating is 7, and the model predicted 9 with 98% accuracy.

Accuracy and MAE would be equally impacted by these examples, while the Loss would be much more affected by the second example then by the first. As the model overfits the training data it tends to get ever more confident in its predictions, which in turn makes its wrong predictions "hurt" more and more. For this article I opted to pick the epoch with the best loss for each model. The results are summarized in Table 6.

*Selecting the best model*

Now that we have the best epoch for each model, we can proceed to select the best overall model. Since we want it to have good performance predicting all classes, I will use the precision and recall metrics,

**Table 6.** Best epoch for each model.

| Architecture | Scale | Balanced? | Epoch | Loss | Acc. | Acc+1 | MAE |
|---|---|---|---|---|---|---|---|
| RNN | Nominal | No | 8 | 1.66 | 41.6% | 65.0% | 1.70 |
| RNN | Nominal | Yes | 12 | 2.02 | 32.8% | 61.0% | 1.69 |
| RNN | Ordinal | No | 18 | 1.66 | 40.7% | 63.1% | 1.75 |
| RNN | Ordinal | Yes | 10 | 2.05 | 27.9% | 54.0% | 1.78 |
| LSTM | Nominal | No | 7 | 0.77 | 78.4% | 91.3% | 0.42 |
| LSTM | Nominal | Yes | 7 | 1.07 | 66.3% | 87.2% | 0.64 |
| LSTM | Ordinal | No | 10 | 1.07 | 67.2% | 90.5% | 0.52 |
| LSTM | Ordinal | Yes | 8 | 1.44 | 56.5% | 88.2% | 0.66 |

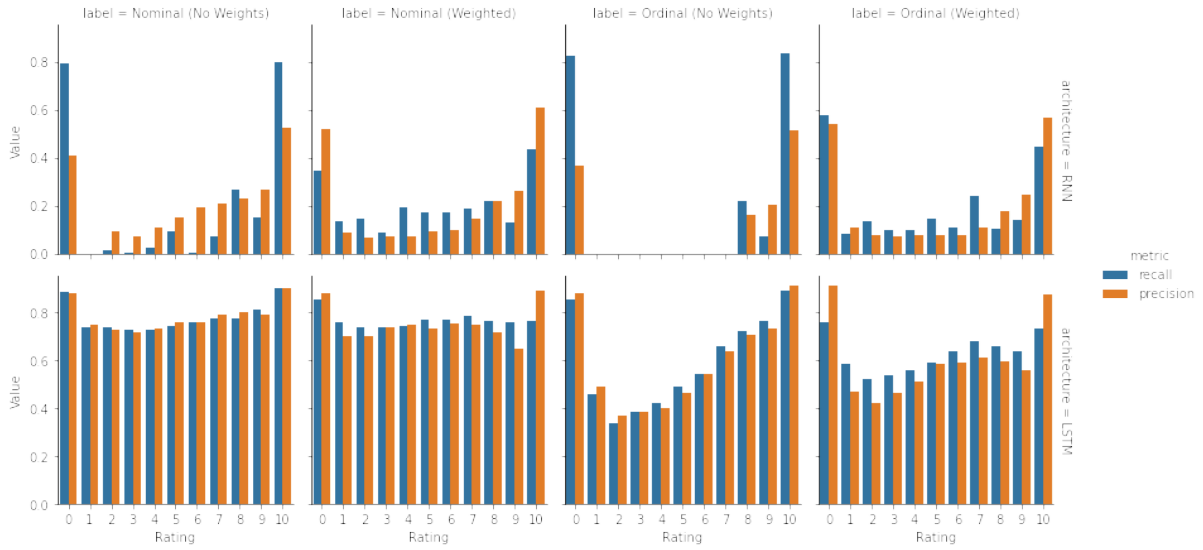



**Figure 15.** Precision and recall for each model.

combined into a single metric using a Macro-averaged F1-score described by the formula below:

$$F_1 = \frac{2}{N} \cdot \sum_{i=1}^{N} \frac{P_i \cdot R_i}{P_i + R_i}$$

where *Pi* and *Ri* are the precision and recall for score class *i*, and *N* is the overall number of classes. (The macro-averaged F1-score is essentially the simple average between the individual F1- scores for each class, which in turn is the harmonical mean between precision and recall.) To calculate that let us check what are the values for precision and recall for all the models (Fig. 15).

We can immediately see – even before calculating any metrics – that the LSTM Nominal models perform better than all other variations (Fig. 15). Interestingly this architecture seems to have been less affected by the unbalanced dataset than the others, though it is hard to tell by inspecting the chart whether the balanced or unbalanced model performs better. Table 7 shows the F1-score for each model.

The unbalanced LSTM Nominal had the best performance based on the F1 criteria (Table 7). But the difference between it and the ordinal model was not that big; and remember, our motivation for testing an ordinal model was to make our predictions reflect better the ordinal nature of reviews. Let us revisit those three examples from above before making any decisions.

**Table 7.** F1-score by model. (For some models, recall or precision were both 0, which makes the F-1 undefined. In this case I considered the F-1 score to be zero for that class).

| Model architecture | Scale | Balanced? | F-Score |
|---|---|---|---|
| RNN | Nominal | No | 0.19 |
| RNN | Nominal | Yes | 0.19 |
| RNN | Ordinal | No | 0.13 |
| RNN | Ordinal | Yes | 0.18 |
| LSTM | Nominal | No | 0.78 |
| LSTM | Nominal | Yes | 0.76 |
| LSTM | Ordinal | No | 0.59 |
| LSTM | Ordinal | Yes | 0.61 |

**Dr. Jekyll and Mr. Hyde (Fig. 16):** *"You'd think I'm jokin', like I'm trying to be funny or somethin'. But, no, the fact that that game exists is a horrible abomination of mankind. That game is so freaking horrible, and I am not kidding"* (AVGN, 2010).

**Earthbound (Fig. 17):** *"I am blown away. That was one of the craziest games I've ever played. Sure it has flaws but I think it does be-*

*long on the list of mandatory Super Nintendo games*" (AVGN, 2018).

**Castlevania 64 (Fig. 18):** "*The graphics are good, for Nintendo 64 standards, but I find them unappealing, because it's the beginning of the 3D age, and they haven't perfected it yet. It's that awkward period between the old and the new*" (AVGN, 2009).

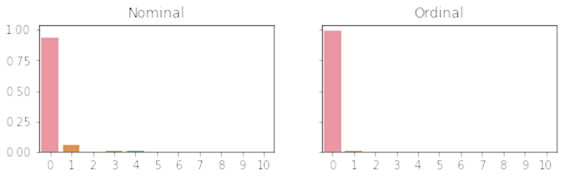



**Figure 16.** Predicted scores for *Dr. Jekyll and Mr. Hyde* on both the nominal and ordinal model.

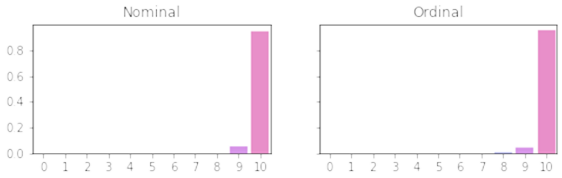



**Figure 16.** Predicted scores for *Earthbound* on both the nominal and ordinal model.

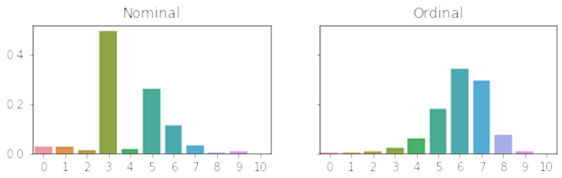



**Figure 16.** Predicted scores for *Castlevania 64* on both the nominal and ordinal model.

For both *Dr. Jekyll and Mr. Hyde* (Advance Communication Co., 1988) and *Earthbound* (Ape / HAL Laboratory, 1994) we see remarkably similar output from both models. That is to be expected; both are very polarized reviews. However, the output for *Castlevania 64* (Konami Computer Entertainment Kobe, 1999) is different, and we see from this example that the model is better able to take into account the ordinal nature of the data. Although this did not improve the accuracy of the model (in fact, it decreased exact accuracy by almost 10 percentage points), I decided that it was a price worth paying, especially when considering that our off-by-one metric is very similar on both models.

### Predicting ratings for *Angry Video Game Nerd* reviews

Well, it has come to this. After all this work we will finally be able to assign proper review scores to the reviews made by the *Angry Video Game Nerd*. For this section, I used the episode transcripts available on the AVGN Wiki (https://avgn.fandom.com/). As for which episodes to test, I opted to score the Top 10 AVGN episodes, as selected by the Nerd himself (Table 8). I also added some of my favorite episodes that were not on the top 10 (Table 9).

So, how did it go? Figure 19 shows the predicted probability for each review, as well as the expected value calculated by averaging the classes.[12]

**Table 8.** Top 10 AVGN episodes.

| # | Episode name | URL |
|---|---|---|
| 1 | R.O.B. | https://www.youtube.com/watch?v=vYm_UaYVSzc&t=0s |
| 2 | Mario 3 | https://www.youtube.com/watch?v=Y5wK5Z23hoo&t=0s |
| 3 | Mega Man | https://www.youtube.com/watch?v=Q3iEn5rzMnw&t=0s |
| 4 | Jekyll and Hyde Re-Revisited | https://www.youtube.com/watch?v=EjXn5qiM8Zw&t=0s |
| 5 | Crazy Castle | https://www.youtube.com/watch?v=wai9CnvatBo&t=0s |
| 6 | Ninja Gaiden | https://www.youtube.com/watch?v=6t2YvyLqw3c&t=0s |
| 7 | Seaman | https://www.youtube.com/watch?v=-IV8hCvsXy0&t=0s |
| 8 | How the Nerd Stole Christmas | https://www.youtube.com/watch?v=iMINBv_Dqvs&t=0s |
| 9 | Berestein Bears | https://www.youtube.com/watch?v=LB3CybXl8rs&t=0s |
| 10 | Dick Tracy | https://www.youtube.com/watch?v=t9nxIUhZCCw&t=0s |

[12] Yes, exactly what I said you cannot do with an ordinal variable. But I did it anyway. It is a useful metric! Now you see why this is such a hotly debated topic.

**Table 9.** Some of my favorite AVGN episodes, listed in no particular order.

| # | Episode name | URL |
|---|---|---|
| 11 | Hong Kong 97 | https://www.youtube.com/watch?v=M_aXcH1zDEE |
| 12 | Plumbers Don't Wear Ties | https://www.youtube.com/watch?v=DyaF_gCKWsI |
| 13 | Castlevania (Part 1) | https://www.youtube.com/watch?v=Hfo6hoN0PUw |
| 14 | Battletoads | https://www.youtube.com/watch?v=UD7k4mTThLY |
| 15 | Ghost N' Goblins | https://www.youtube.com/watch?v=94Y6y1MOoEo |
| 16 | Earthbound | https://www.youtube.com/watch?v=LZ5nX0FTH6Q |
| 17 | Silver Surfer | https://www.youtube.com/watch?v=gvnRBywkUZ0 |
| 18 | Ikari Warriors | https://www.youtube.com/watch?v=bByE7n2AJj4 |
| 19 | Big Rigs: Over the Road Racing | https://www.youtube.com/watch?v=h6DtVHqyYts |
| 20 | Tiger Electronic Games | https://www.youtube.com/watch?v=_u5dtBtG9yU |


**Figure 19.** Angry Video Game Model predictions for *Angry Video Game Nerd* reviews.

Well, most things seem to make sense. Games such as *Mario 3* (Nintendo R&D4, 1988), *Mega Man* (Capcom, 1987) and *Earthbound* have almost perfect scores, which make sense given the praise the Nerd gives them in the reviews. On the other hand, games such *Hong Kong 97* (HappySoft, 1995) and *Big Rigs: Over the Road Racing*

(Stellar Stone, 2003) have terrible scores. But wait… what is this? *Dr. Jekyll and Mr. Hyde* has a score of 5.4? This cannot be right. And what about *Castlevania* (Konami, 1986) getting a score of 1.2? The Nerd had mostly praise for this game. What is happening here? WHAT IS THE MODEL THINKING?

**Figure 20.** What were they thinking? (source: AVGN).

To answer this question, I had to dig deeper. Something in those reviews is throwing off the model.

### What is the model thinking?

Machine learning models' predictions are extremely complicated to explain and this one is no different. The model is essentially a black box, and a lot of effort has been put into it to understand why it made a prediction. Machine learning is being used to allocate investments, predict credit risk, score test results, and even to drive cars. In all these applications, being able to explain a model decision is highly desirable, be it from a legal standpoint (to explain to a customer why they were denied credit) or from a safety standpoint (to understand why an autonomous vehicle thought it could drive through a barn[13]).

There is a lot of research being done in model explainability. Some works, such as the LIME method proposed by (Ribeiro et al., 2016), work by deriving proxy models; that is, simpler, linear and locally bound approximations of the full model that can be easily explained. Another approach is to use salience mapping, in which parts of the input are occluded from the model, and the variations in the predicted output can give us some insights on what is influencing the prediction. The latter approach works well for our problem, as we can feed parts of a longer review to the model to get a localized sentiment for a sentence.

To investigate the predictions of our model, I ran them again multiple times. Each time, a small sliding window of 5 lines was cut out from the review, and the expected score was calculated by the model using only the text in the sliding window ("Only") and using everything in the review but the text on the sliding window ("Except"). Figure 21 shows the results of this analysis for *Dr. Jekyll and Mr. Hyde* and *Castlevania* (part 1).

We can see that the predicted sentiment can vary a lot throughout the review, as

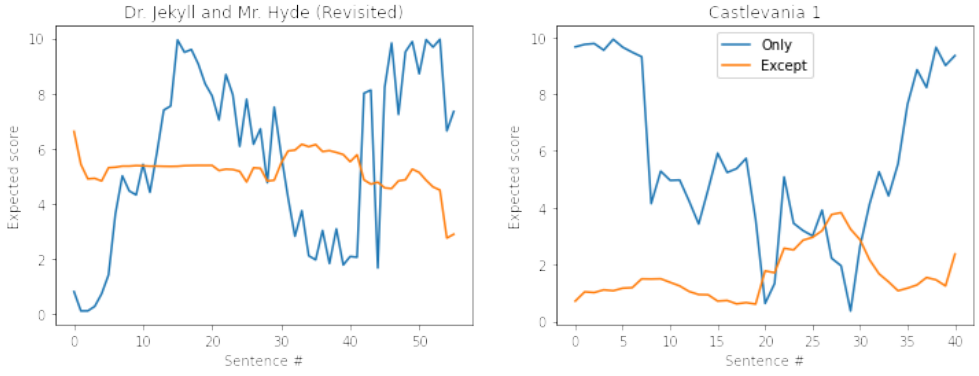



**Figure 21.** Salience mapping for two AVGN episodes.



[13] Maybe it was trained using *Big Rigs: Over the Road Racing*?

shown by the wild variations observed in the blue line. However, the removal of a couple of five lines usually has little effect on the review, with a few exceptions. In *Dr. Jekyll and Mr. Hyde*, removing the beginning of the review significantly increased the score, while removing the ending significantly decreased it. Let us take a close look on those sentences: "[The episode begins with a black-and-white clip; the first few seconds from the original Dr. Jekyll and Mr. Hyde review from 2004. The Nerd's voice can be heard over this.] *In May of 2004, I gave a warning about a game called Dr. Jekyll and Mr. Hyde. I made it perfectly clear: DO NOT PLAY THIS GAME. But from what I understand... people have played it! They didn't listen. But it wasn't their fault... I only showed about one minute of footage from the game, and even though I talked about it at great length, it didn't do any good...* [The Nerd drinks some Rolling Rock.] *I called it a piece of* ****. *I called it an awful pile of steaming goat* ****. *But that was honoring it. I could've said anything, it wouldn't have mattered. I could've taken a* **** *on it, but my own* **** *would have been offended to lay on this loathsome piece of FILTH! Just the thought of covering this thing in doo-doo is like encasing it in gold! I curse the day I ever laid eyes on it. I curse the plastic that holds this abomination. My words are insufficient in describing the total insult to humanity that this "game" has provided! Everything that I've ever said and anything that anybody else has ever said is NOT enough! It MUST be shown.* [He drinks more Rolling Rock.]" (AVGN, 2010).

Well, okay. I see why this might have such a large effect on the expected score. No surprises here. Now let us look at the ending sentences: "*No...!* [The scene fades to black and fades back in a blur. The Jekyll-to-Hyde transformation music from the game plays as the Nerd wakes up, back in his room, where he first transformed. He resumes playing the game, and has an epiphany.] *The Nerd: I think I get it. Why, it's the best game ever made. It's more than a game... it exposes the dual nature of the human spirit. The only way to win the game is to be Jekyll, but you wanna be Hyde so you can shoot* ****. *You see, it's a constant battle between good and evil, and Jekyll must stay farther along his path than Hyde. If Hyde gains the lead, then evil will triumph over good, and that's the true conflict to the human soul. And to deny the evil completely, would only force it into the subconscious mind, like a city broken into different social classes. People don't wanna step outside their own boundaries, like Jekyll wandering into the wrong section of town. He's unwelcome. Nevertheless, he must abide by his own good nature. No wonder the cane doesn't work. The game does not reward you for acting upon your malevolent intentions. It's a proposed guideline for a set of morality rules to be programmed in real life! It uses the Victorian era as a fundamental depiction of outward respectability and inward lust. It's a metaphor for social and geographical fragmentation. It eludes the Freud theory of repression, in which unacceptable desires or impulses are excluded from the conscious mind, and left to operate on their own... in the unconscious.*" (AVGN, 2010).

Here we can see something interesting. Taken at face value, the Nerd's words seem to sing high praise for the game. He makes it sounds almost like a transcendent experience. But it is all good old sarcasm. Apparently, the model did not learn enough to be able to identify the true meaning of the Nerd's words.

The *Castlevania* example is less interesting. The Nerd starts talking about how good it was when it was launched, and the impact it had in his life. But after that he proceeds to talk about all the frustrating parts of the game, and the model seem to have found that part more relevant. The bump in the orange line correspond to comments by the Nerd on puns made on the game credits sequence. Apparently, the model really hates puns and the reviews are being pun-inshed for it. "*Hmm...Trans Fisher? It reminds me of Terence Fisher, the director of many of the* Hammer Horror *films. That's a funny coincidence. Oh wait... Vran Stoker? Like Bram Stoker, the author of* Dracula*? Wha-- Christopher Bee?! Is it a joke? I don't get it. Are they saying Christopher Lee is like a bee?* [Bee with a face like Christopher Lee's comes buzzing by] *No, they can't mean that. This is probably just a series of strangely coincidental typos.* [The Nerd notices another

name] *Belo Lugosi? Boris Karloffice? They're just* ******* *around. Love Chaney Jr.? Mix Schrecks? Green Stranger?! Is this supposed to be funny? Like just take a celebrity's name and change it around? That's like if I took the name 'Stephen Spielberg', and called him 'Stephen Jeelberg'. Like, that's not funny, that's kindergarten level! No, kindergarten students don't find that funny! Aliens don't find that funny! Well anyway, that's Castlevania for you. Good game, but holy* **** *is it hard. Now as promised, we're gonna plow through the rest of 'em, all the old-school Castlevania games. The ones that I grew up with* – [The 'What a horrible night to have a curse' box from *Castlevania II: Simon's Quest* appears in front of the Nerd, interrupting him. The box disappears a few seconds later, and a day-to-night transition in the style of said game is shown. The nighttime music plays and the Nerd's room looks darker than before. The Nerd notices the *Castlevania II: Simon's Quest* cartridge]" (AVGN, 2009).

## CONCLUSION

Can a model truly capture the full emotion of an AVGN review? Maybe time will tell. In this article, however, I demonstrated that Recurrent Neural Networks, in particular the LSTM architecture, have a good performance on the videogame review rating prediction task when compared to other sentiment analysis benchmarks. I also demonstrated that the model trained on this dataset can produce coherent ratings[14] for the reviews performed by the Angry Video Game Nerd, barring issues of dealing with sarcasm.

## FURTHER WORK

As discussed in the start, this work focused on simpler model architectures to make it more approachable. More advanced architectures such as Transformers might be able to better deal with long term dependences. I also only trained this model on the review dataset data, and I did not make use of transfer learning at all. Pretraining on a larger and more general dataset might improve the quality of the embeddings. Testing contextual embeddings such as BERT (Devlin et al., 2018) might also improve how the model deals with context dependence ambiguity. Finally, training the model on longer reviews might improve its performance on the AVGN dataset, as the length of an AVGN episode is several times longer than the average length of a user review.

## REFERENCES

**Anonymous.** (2021) Sentiment analysis on IMDB. Papers with Code. Available from: https://paperswithcode.com/sota/sentiment-analysis-on-imdb (Date of access: 03/Jan/2022).

**Alammar, J.** (2019). The Illustrated Word2vec. GitHub. Available from: https://jalammar.github.io/illustrated-word2vec/ (Date of access: 03/Jan/2022).

**Chrisman, N.R.** (1998) Rethinking levels of measurement for cartography. Cartography and Geographic Information Systems 25: 231–242.

**Cinemassacre Productions LLC.** (n.d.) Cinemassacre FAQ. Available from: https://cinemassacreold.website-us-east-1.linodeobjects.com/faq/ (Date of access: 06/Apr/2021).

**Delany, S.J.; Buckley, M.; Greene, D.** (2013) SMS spam filtering: methods and data. Expert Systems with Applications 39: 9899–9908.

**Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K.** (2018) BERT: pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv. Available from: https://arxiv.org/abs/1810.04805 (Date of access: 03/Jan/2022).

**Eberhard, D.M.; Simons, G.F.; Fennig, C.D.** (2021) Ethnologue: Languages of the World. SIL International, Dallas.

**Gelbart, B.** (2019) A history of review bombing. Gamerant. Available from: https://gamerant.com/mass-effect-borderlands-3-modern-warfare-review-bombs/ (Date of access: 06/Apr/2021).

**Hochreiter, S. & Schmidhuber, J.** (1997) Long

[14] This is opinion. Unfortunately, without ground truth ratings provided by the Nerd himself we might never be able to go beyond such qualitative statements.

short-term memory. Neural Computation 9: 1735–1780.

**Kim, A. & Liao, S.** (2019) Why fans aren't happy with Pokémon Sword and Shield developer Game Freak. CNN. Available from: https://edition.cnn.com/2019/11/16/tech/nintendo-pokemon-sword-shield-trnd/index.html (Date of access: 06/Apr/2021).

**Kingma, D.P. & Ba, J.** (2015) Adam: a method for stochastic optimization. ICLR, San Diego.

**Knapp, T.R.** (1990) Treating ordinal scales as interval scales: an attempt to resolve the controversy. Nursing Research 39: 121–123.

**Kocijan, V.; Lukasiewicz, T.; Davis, E.; Marcs, G.; Morgenstern, L.** (2020) A review of Winograd schema challenge datasets and approaches. arXiv. Available from: https://arxiv.org/abs/2004.13831 (Date of access: 03/Jan/2022).

**Lau, J.; Cohn, T.; Baldwin, T.; Hammond, A.** (2020) This AI poet mastered rhythm, rhyme, and natural language to write like Shakespeare. IEEE Spectrum. Available from: https://spectrum.ieee.org/artificial-intelligence/machine-learning/this-ai-poet-mastered-rhythm-rhyme-and-natural-language-to-write-like-shakespeare (Date of access: 03/Jan/2022).

**Levesque, H.J.; Davis, E.; Morgenstern, L.** (2012) The Winograd schema challenge. Thirteenth International Conference on Principles of Knowledge Representation and Reasoning 2012: 552–561.

**LotrProject.** (n.d.) Word count and density. LotrProject. Available from: http://lotrproject.com/statistics/books/wordscount (Date of access: 06/Jun/2021).

**Mosteller, F. & Tukey, J.W.** (1977) Data analysis and regression: a second course in statistics. Addison-Wesley, Boston.

**Nakatani, S.** (2010). Language Detection Library for Java. SlideShare. Available from: https://www.slideshare.net/shuyo/language-detection-library-for-java (Date of access: 03/Jan/2022).

**NLP-progress.** (2021) Sentiment analysis. NLP-progress. Available from: http://nlp-progress.com/english/sentiment_analysis.html (Date of access: 03/Jan/2022).

**OpenAI.** (2019) Dota 2 with large scale deep reinforcement learning. arXiv. Available from: https://arxiv.org/abs/1912.06680 (Date of access: 03/Jan/2022).

**Ribeiro, M.T.; Singh, S.; Guestrin, C.** (2016) "Why Should I Trust You?": explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2016: 1135–1144.

**Stevens, S.S.** (1946) On the theory of scales of measurement. Science 103: 677–680.

**Wikipedia.** (2021). YouTube. Wikipedia. Available from: https://en.wikipedia.org/wiki/YouTube (Date of access: 06/Apr/2021).

**Wiktionary.** (2020) Statistics. Wiktionary. Available from https://en.wiktionary.org/wiki/Special:Statistics (Date of access: 06/Jun/2020).

**Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; et al.** (2016) Google's neural machine translation system: bridging the gap between human and machine translation. arXiv. Available from: https://arxiv.org/abs/1609.08144 (Date of access: 03/Jan/2022).

## Acknowledgements

I want to thank James Rolfe for creating the web series that inspired this work, and for providing hours of quality entertainment while I was growing up. I am still subscribed to his channel, and I do not aim to change that any time soon.

## About the author

**Henrique Soares** is an engineer and machine learning enthusiast who, fortunately, grew up playing better games than the ones featured in AVGN. When he is not working on unconventional applications of machine learning, Henrique spends way too much time watching YouTube web series.