Instituto de MATEMÁTICA E ESTATÍSTICA UFRGS

DEPARTAMENTO DE ESTATÍSTICA
E-mail: dest@mat.ufrgs.br

Trabalho de Conclusão de Curso

Composição Automática de Músicas utilizando Redes Neurais Recorrentes

Nicolas Mathias Hahn

19 de outubro de 2022

Nicolas Mathias Hahn

Composição Automática de Músicas utilizando Redes Neurais Recorrentes

Trabalho de Conclusão apresentado à comissão de Graduação do Departamento de Estatística da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para obtenção do título de Bacharel em Estatística.

Orientador: Prof. Dr. Guilherme Pumi

Porto Alegre
Outubro, 2022

Nicolas Mathias Hahn

Composição Automática de Músicas utilizando Redes Neurais Recorrentes

Este Trabalho foi julgado adequado para obtenção dos créditos da disciplina Trabalho de Conclusão de Curso em Estatística e aprovado em sua forma final pela Orientador e pela Banca Examinadora.

Orientador:_____
Prof. Dr. Guilherme Pumi, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul, Porto Alegre, RS

Banca Examinadora:

Prof. Dr. João Henrique Ferreira Flores, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul, Porto Alegre, RS

Porto Alegre
Outubro, 2022

"Since I have always preferred making plans to executing them, I have gravitated towards situations and systems that, once set into operation, could create music with little or no intervention on my part. That is to say, I tend towards the roles of planner and programmer, and then become an audience to the results."

Brian Eno (Alpern, 1995).

Resumo

O problema de composição musical automática é extensivamente explorado na literatura. Em geral, o objetivo final nesses trabalhos é a composição musical em si, de forma que os modelos utilizados são ajustados para que a música gerada seja adequada em algum sentido. Detalhes técnicos como os impactos que as modificações nos parâmetros têm na composição final são amplamente desconhecidos. Neste trabalho, temos por objetivo estudar o quão sensível é um modelo de rede neural recorrente, baseado em processamento de linguagem natural, construído para composição musical. Para tal tarefa, a mensuração será feita com a perplexidade, uma medida oriunda da teoria da informação. Por fim, as músicas são avaliadas, de forma subjetiva, em relação à musicalidade e à qualidade das peças musicais obtidas.

Palavras-Chave: Música, Composição Musical, Redes Neurais Recorrentes, Processamento de Linguagem Natural.

Abstract

The issue of automatically composed music has received a lot of attention in the literature. In general, the focus is on the composition itself, and models are tuned to produce useful results. Technical details such as the effects that adjustments to the model's parameters have on the composition, however, are still largely unknown. In this work, we investigate how sensitive is a recurrent neural network model, based on natural language processing, built for music composition. To accomplish this task, we apply a metric from information theory called perplexity. The composed works are then subjectively assessed for musicality and quality.

Keywords: Music, Music Composition, Recurrent Neural Networks, Natural Language Processing.

Sumário

1 Introdução **11**

2 Referencial Teórico **13**
2.1 Composição Algorítmica . . . 13
2.2 Redes Neurais Artificiais . . . 13
2.2.1 Redes Neurais e a Estatística . . . 14
2.2.2 Neurônio . . . 15
2.2.3 Função de Ativação . . . 16
2.2.4 Arquiteturas de RNAs . . . 17
2.2.5 Ajustando uma RNA . . . 20
2.3 Processamento de Linguagem Natural . . . 21
2.3.1 Corpus e Corpora . . . 22
2.3.2 Expressões Regulares . . . 22
2.3.3 Vocabulário e Tokenização . . . 22
2.3.4 Modelos de Linguagem . . . 22
2.3.5 Avaliação Intrínseca e Extrínseca . . . 23
2.3.6 Entropia e Perplexidade . . . 23
2.3.7 Word Embedding . . . 24
2.4 Notação ABC . . . 25
2.5 Web Scraping . . . 25

3 Estudo de Modelagem **27**
3.1 Implementação Computacional . . . 27
3.2 Bases de Dados . . . 28
3.3 Modelo . . . 28
3.3.1 Arquitetura . . . 29
3.3.2 Parâmetros e Ajuste da Rede . . . 29
3.3.3 Processo Gerador de Músicas . . . 29

4 Resultados **31**
4.1 Irish . . . 31
4.2 ABC Notation . . . 34
4.3 Músicas Geradas . . . 37

5 Considerações Finais **39**

Referências Bibliográficas **39**

Lista de Figuras

Figura 2.1: Modelo do k-ésimo Neurônio (adaptado de Hair et al., 2005; Haykin, 2009) . . . 15
Figura 2.2: Rede Camada Única e Múltipla (adaptado de Haykin, 2009) . . . 18
Figura 2.3: Rede Totalmente e Parcialmente Conectada (adaptado de Haykin, 2009) . . . 18
Figura 2.4: Diagrama de uma RNN *Vanilla* (adaptado de Goodfellow et al., 2016; Kamath et al., 2019) . . . 19
Figura 2.5: Diagrama de uma LSTM. Considere σ como a função de ativação *logit* (adaptado de Kamath et al., 2019). . . . 20
Figura 2.6: Exemplo de notação ABC convertendo em música (disponível em `abcnotation.com`). . . . 25

Figura 4.1: Perda dos modelos ajustados com Irish: curva verde representa $learning\_rate = 10^{-3}$ e roxo $learning\_rate = 10^{-5}$. . . . 32
Figura 4.2: Correlação entre parâmetros e métricas para Irish . . . 32
Figura 4.3: Irish: *tanh* (verde) vs *logit* (roxo) . . . 34
Figura 4.4: Perda dos modelos ajustados com ABC Notation: curva verde representa $learning\_rate = 10^{-3}$ e roxo $learning\_rate = 10^{-5}$. . . 35
Figura 4.5: Correlação entre parâmetros e métricas para ABC Notation . . . 36
Figura 4.6: ABC Notation: *tanh* (verde) vs *logit* (roxo) . . . 37

Lista de Tabelas

Tabela 4.1: Resultados dos experimentos referentes à base de dados Irish, com $vocab\_size = 64$, utilizando a função *tanh* como função de ativação. Em azul (vermelho), constam os dois melhores (piores) modelos em relação à perplexidade. 33
Tabela 4.2: Irish: *tanh* vs *logit* 33
Tabela 4.3: Resultados dos experimentos referentes à base de dados ABC Notation, com $vocab\_size = 125$, utilizando a função *tanh* como função de ativação. Em azul (vermelho), constam os dois melhores (piores) modelos em relação à perplexidade. 36
Tabela 4.4: ABC Notation: *tanh* vs *logit* 37

1 Introdução

Composição algorítmica, ou composição automática, refere-se ao processo de criação de músicas por meio de algum processo formal com pouca ou nenhuma intervenção humana. De acordo com Maurer (1999), podemos encontrar três metodologias diferentes que existem em composição algorítmica: estocástica, *rule-based* e inteligência artificial.

Wolfgang Amadeus Mozart (1756-1791) utilizou técnicas de composição algorítmica em sua obra *Musikalisches Wurfelspiel* (*Dice Music*), um jogo musical que envolvia atribuir um número para fragmentos de músicas, e combiná-los ao acaso, criando uma nova peça com as partes selecionadas. Além disso, John Cage (1912-1992), assim como Mozart, utilizou aleatoriedade em suas composições. Como exemplo, temos *Reunion*, uma performance em que músicas são geradas ao jogar xadrez em um tabuleiro equipado com um foto receptor: cada lance emitia um som e, portanto, a música resultante é única por jogo. Por fim, com o auxílio de computadores, David Cope (1941-) criou, em 1981, o EMI (*Experiments in Musical Intelligence*), um sistema baseado em grandes bases de dados com descrições de estilos de diferentes estratégias composicionais e, como complemento, o sistema também tinha a capacidade de criar as próprias regras composicionais por meio dos novos dados recebidos (Alpern, 1995; Maurer, 1999).

Na literatura, encontramos diversos exemplos de trabalhos envolvendo composição musical automática. Os dados utilizados para o objetivo de composição podem vir tanto em formato de áudio (como o de Kuang e Yang, 2021) quanto em formato de texto (como o de Agarwala et al., 2017). Ao mesmo tempo em que o trabalho de Colombo et al. (2016) é claro em sua relação com a composição algorítmica, de Souza e de Avila (2018) é evasivo. Também, observou-se um foco em criação de músicas e avaliação das composições musicais, seja pela estrutura musical seja pela opinião de ouvintes referente à geração (se era musicalmente plausível). Por fim, os trabalhos de Fernández e Vico (2013), Ji et al. (2020) e de Hernandez-Olivan e Beltran (2021) são um sucinto resumo de abordagens e técnicas de inteligência artificial (como redes neurais artificiais) utilizadas para composição automática de músicas, assim como as formas de avaliação.

O presente trabalho propõe-se a explorar o problema de composição algorítmica no contexto de modelagem, sendo as peças musicais geradas uma consequência dos modelos. Dessa forma, mesmo sem conhecimentos sobre teoria musical, é possível elencar um melhor modelo apenas por métricas objetivas. Por meio de técnicas de processamento de linguagem natural e de redes neurais artificiais, serão criados modelos capazes de composição com base em conjuntos de músicas na notação ABC.

Devido a isso, os dados utilizados serão restritos ao formato textual. Dada a natureza subjetiva da música, as peças musicais resultantes serão subjetivamente avaliadas, comparando-as de acordo com a percepção do autor.

2 Referencial Teórico

2.1 Composição Algorítmica

Composição algorítmica, ou composição automática, refere-se ao processo de criação de músicas por meio de algum processo formal com pouca ou nenhuma intervenção humana. O termo **algoritmo** é definido como um conjunto predeterminado de instruções com o objetivo de resolver um problema em específico com um número limitado de passos. O **problema** encarado por compositores é a composição musical, e as **instruções** predeterminadas sugerem que, uma vez que o processo é iniciado, a intervenção humana é substituída. Portanto, **composição automática** também descreve adequadamente esse tipo de composição musical, sendo que **automático** refere-se a "qualquer coisa que pode se mover ou agir por conta própria" (Alpern, 1995; Maurer, 1999).

De acordo com Maurer (1999), podemos encontrar três metodologias diferentes que existem em composição algorítmica:

- **estocástica:** abordagem mais simples. Envolve aleatoriedade e pode ser tão simples quanto gerar uma série de notas musicais ao acaso, bem como sortear uma ordem de fragmentos de músicas para ser executada pelo músico.
- ***rule-based*:** em vez de delegar decisões ao acaso como no método estocástico, um sistema *rule-based* define uma constituição ou gramática (sistema formal de princípios ou regras que as possíveis frases de uma linguagem são geradas) em que o sistema deve seguir, uma vez iniciado o processo de composição.
- **inteligência artificial (IA):** similar ao *rule-based* no sentido de ser baseado em uma gramática pré-definida. No entanto, sistemas de IA têm a capacidade de definirem sua própria gramática, que pode evoluir (isto é, ser modificada de uma forma automática) ao longo do processo.

Mais detalhes podem ser encontrados em Alpern (1995), Maurer (1999), Nierhaus (2009), Fernández e Vico (2013) e Hernandez-Olivan e Beltran (2021).

2.2 Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são modelos de aprendizagem de máquina inspirados no mecanismo de aprendizagem presente em organismos vivos (Aggarwal

et al., 2018). Nos seus primórdios, os algoritmos tinham o intuito de ser um modelo computacional capaz de imitar a aprendizagem ocorrida no cérebro (Goodfellow et al., 2016).

2.2.1 Redes Neurais e a Estatística

Muitos modelos de RNAs são similares ou idênticos a populares técnicas estatísticas como modelos lineares generalizados, regressão linear (simples e múltipla), regressão polinomial, regressão não paramétrica e análise discriminante (como a regressão logística), especialmente quando a ênfase é na predição de um fenômeno complexo em vez de sua explicação. Tal foco é devido às RNAs serem modelos **caixa-preta**, o que implica em perda de interpretabilidade (Sarle, 1994; Cheng e Titterington, 1994; Raul, 1996).

No contexto de RNAs, é possível definir dois procedimentos para a resolução de algum problema prático (aplicado): especificar a arquitetura da rede e treinar a rede com um conjunto de dados de treinamento. Paralelamente, no contexto estatístico, esses passos são equivalentes a especificar um modelo e estimar seus parâmetros dado um conjunto de dados. Posto isso, apresentamos uma lista de termos comumente utilizados na literatura de RNAs e as respectivas equivalências de acordo com a estatística:

- variáveis são chamadas de atributos (*features*);
- variáveis independentes são chamadas de entrada (*input*);
- valores preditos são chamados de saída (*output*);
- variáveis dependentes são chamadas de variáveis alvo (*target*) ou valores de treinamento;
- resíduos são chamados de erros;
- estimação é chamada de treinamento ou aprendizagem;
- os critérios de estimação são chamados de função perda, função custo ou função erro;
- observações são chamadas de padrões (*patterns*) ou de pares de treinamento (*training pairs*);
- parâmetros estimados são chamados de pesos (sinápticos);
- transformações são chamadas de links funcionais (*functional links*);
- interpolação e extrapolação são chamadas de generalização.

Os termos estatísticos **amostra** e **população** não demonstram conter equivalentes na literatura de RNAs. Por fim, os dados são comumente divididos em conjunto de treino e de teste (Cheng e Titterington, 1994).

2.2.2 Neurônio

Um neurônio (ou nó computacional) é uma unidade de processamento de informação fundamental para a operação de uma rede neural (Haykin, 2009). Podemos identificar três elementos básicos do modelo neuronal:

1. Um conjunto de **sinapses**, cada uma caracterizada por um peso ou força própria. O peso sináptico de um neurônio artificial pode estar em um intervalo que inclui valores positivos e negativos.

2. Um **somador** ou acumulador para somar os sinais de entrada, ponderados pelas respectivas sinapses de cada neurônio, constituindo uma **combinação linear**.

3. Uma **função de ativação** para restringir a amplitude de saída de um neurônio (fixando em um valor finito).

Também é incluído um intercepto no modelo, chamado de ***bias*** na literatura de *machine learning*, com o intuito de aumentar ou diminuir o sinal de entrada da função de ativação, dependendo do seu sinal.

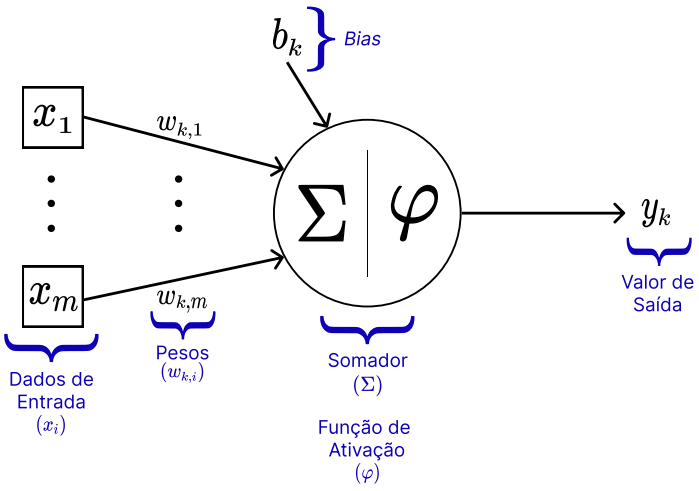

Figura 2.1: Modelo do k-ésimo Neurônio (adaptado de Hair et al., 2005; Haykin, 2009)

Dada uma sequência $x_1, \cdots, x_m$ de dados de entrada (valores observados) e uma função de ativação $\varphi(\cdot)$, a descrição do k-ésimo neurônio em uma rede neural nada mais é do que a determinação da função que leva o vetor de entrada $(x_1, \cdots, x_m)$ na k-ésima saída y_k. Uma configuração simples de um neurônio é dada pelo sistema de equações:

$$\begin{aligned} u_k &:= \sum_{i=1}^{m} w_{k,i} x_i; \\ y_k &:= \varphi(u_k + b_k), \end{aligned} \tag{2.1}$$

sendo b_k o *bias* e $w_{k,i} \in \mathbb{R}$ os pesos associados ao neurônio.

2.2.3 Função de Ativação

Uma função de ativação pode ser linear ou não linear. Seu objetivo em geral é transformar o valor de $v_k := u_k + b_k$ para um valor que esteja de acordo com a distribuição de y_k, como pode ser visto em (2.1). A escolha de uma particular função de ativação depende do problema que o neurônio almeja resolver. Em problemas de classificação de classe binária, ela é responsável por levar o valor de $v_k \in \mathbb{R}$ para o intervalo $(0, 1)$, possibilitando a predição das respectivas classes. Abaixo, seguem alguns exemplos:

- **linear:** $f : \mathbb{R} \to \mathbb{R}$ dada por

$$f(x) := \alpha x, \quad \alpha \in \mathbb{R}\backslash\{0\}.$$

  Observe que no caso da função linear, o neurônio está sempre ativado, sendo seu valor alterado pela constante α. Em particular, quando $\alpha = 1$, temos a função identidade e, consequentemente, um modelo de regressão linear.

- **sigmóide ou *logit***: $f : \mathbb{R} \to (0, 1)$ dada por

$$f(x) := \frac{1}{1 + e^{-x}}.$$

  Quando essa função de ativação é associada a uma rede, o neurônio é equivalente a uma regressão logística. Dessa forma, notamos que a função de ativação tem um papel semelhante à função de ligação no contexto de modelos lineares generalizados: relacionar o preditor linear ao respectivo valor esperado (McCullagh e Nelder, 1989; Sarle, 1994; Frei et al., 2020).

- **tangente hiperbólica (*tanh*):** $f : \mathbb{R} \to (-1, 1)$ dada por

$$f(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

  Em contextos que uma função sigmoidal é necessária (como a predição de uma variável binária), a *tanh* tipicamente desempenha melhor. Vale comentar que as funções de ativação sigmoidais são mais comuns em estruturas como as redes recorrentes do que em redes alimentadas adiante (Goodfellow et al., 2016).

- **ReLU (*Rectified Linear Unit*):** $f : \mathbb{R} \to [0, \infty)$ dada por

$$f(x) := \max\{0, x\}.$$

  De acordo com Goodfellow et al. (2016), é a recomendação padrão para redes neurais modernas. O autor justifica que, por ser uma função quase linear, são preservadas muitas propriedades que fazem os modelos lineares bons generalizadores.

- ***softmax***: $f : \mathbb{R}^n \to (0, 1)^n$ dada por

$$f(\boldsymbol{x}) := \left(\frac{e^{x_1}}{\sum_{j=1}^n e^{x_j}}, \cdots, \frac{e^{x_n}}{\sum_{j=1}^n e^{x_j}} \right), \quad \forall\, \boldsymbol{x} = (x_1, \cdots, x_n) \in \mathbb{R}^n.$$

  A função é tipicamente utilizada para transformar vetores reais em vetores a serem interpretados como probabilidades, sendo bastante útil em problemas de classificação.

Listas de funções de ativação comumente utilizadas podem ser encontradas em Hastie et al. (2008), Hagan et al. (2014), Aggarwal et al. (2018) e Academy (2022).

2.2.4 Arquiteturas de RNAs

Um único neurônio, mesmo com muitas entradas, usualmente não é suficiente para resolver problemas complexos, sendo necessários, por vezes, diversos operando paralelamente em uma estrutura que chamamos de **camada**. Temos que uma rede neural é constituída por três tipos básicos de camadas:

1. **entrada:** constituída por nós de fonte (consequentemente, não tem função de ativação), em que cada nó representa uma variável independente. Vale comentar que variáveis numéricas requerem apenas um nó, porém variáveis categóricas requerem um nó para cada categoria, semelhantemente ao processo de criação de variáveis dicotômicas ou *dummies*, mas sem a eliminação de uma categoria de referência.

2. **saída:** composta por neurônios, com a diferença que seu valor de saída é definitivo. No caso de um modelo preditivo, o valor será o resultado da previsão. Se o modelo é utilizado para classificação, então o valor resultante será empregado no processo de classificação (que será utilizado para determinar qual categoria foi classificada).

3. **intermediária ou oculta:** assim como a camada de saída, é composta por neurônios. Seus valores de entrada podem ser originados da camada de entrada ou da camada oculta anterior, bem como seus valores de saída podem alimentar a próxima camada oculta ou a camada de saída.

Posto isso, organizamos as redes em duas arquiteturas fundamentalmente diferentes (Hair et al., 2005; Haykin, 2009; Hagan et al., 2014): redes alimentadas adiante e redes recorrentes.

Redes Alimentadas Adiante

A nomenclatura desse tipo de arquitetura é devida ao fluxo de informação partir da camada de entrada até a camada de saída, de forma unidirecional (Goodfellow et al., 2016). Hastie et al. (2008) e Fan et al. (2021) comentam que é o tipo padrão de rede neural. A principal diferença entre uma rede de camada única e uma com múltiplas camadas é a existência (ou não) das camadas ocultas, sendo que a designação **camada única** se refere a uma camada de neurônios. Além disso, há a distinção entre a rede ser totalmente ou parcialmente conectada, no sentido de que os nós de uma camada são conectados a todos os nós da camada adjacente ou não (Haykin, 2009).

Redes Neurais Recorrentes

Redes neurais recorrentes (RNNs) são um tipo de redes neurais criadas para processar séries temporais e outros tipos de dados sequenciais (Fan et al., 2021). O que difere uma rede recorrente de uma rede alimentada adiante é seu laço de realimentação. Por exemplo, uma rede recorrente pode consistir de uma única camada

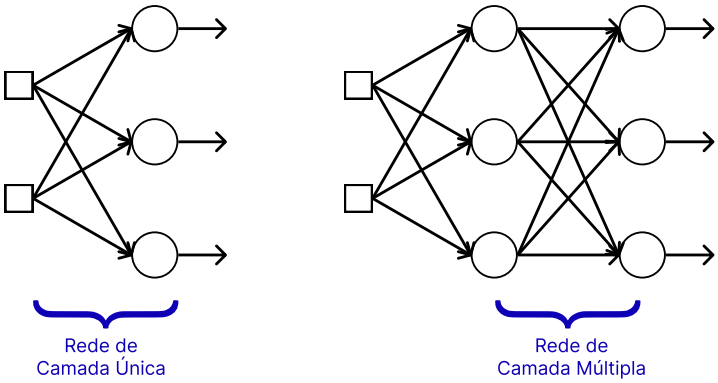

Figura 2.2: Rede Camada Única e Múltipla (adaptado de Haykin, 2009)

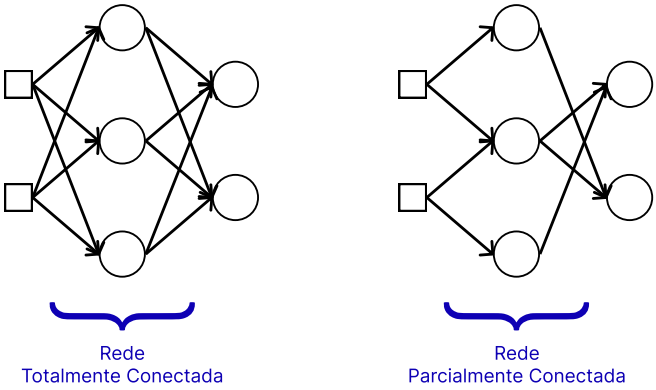

Figura 2.3: Rede Totalmente e Parcialmente Conectada (adaptado de Haykin, 2009)

de neurônios com cada neurônio alimentando seu sinal de volta para as entradas de todos os outros neurônios (Haykin, 2009).

RNN *Vanilla*

Sejam $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$ dados de entrada. Uma RNN *vanilla* modela o estado oculto (*hidden state*) $\boldsymbol{h}_t$ via

$$\boldsymbol{h}_t = f_{\boldsymbol{\theta}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t),$$

em que $f_{\boldsymbol{\theta}}$ geralmente é uma função não linear parametrizada por $\boldsymbol{\theta}$. Assim como em modelagem de séries temporais, há o compartilhamento dos parâmetros ao longo do tempo.

Dada uma sequência de entrada $\boldsymbol{x}_t, \cdots, \boldsymbol{x}_1$, pode-se utilizar $\boldsymbol{h}_t$ para fazer predições. Por exemplo, se a RNN é utilizada para modelagem estatística de linguagem

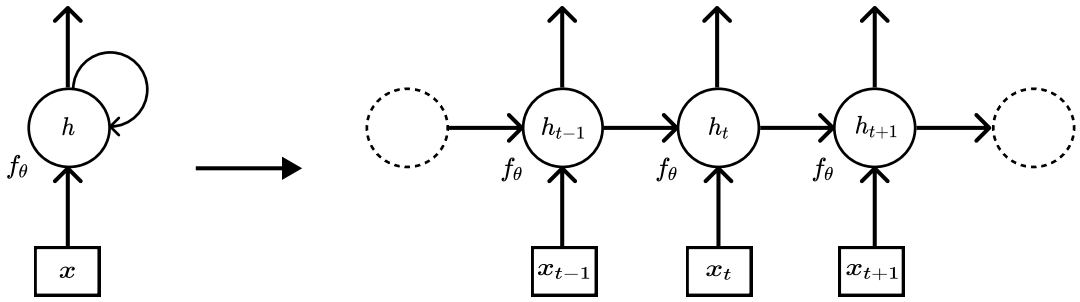

Figura 2.4: Diagrama de uma RNN *Vanilla* (adaptado de Goodfellow et al., 2016; Kamath et al., 2019)

(como prever as próximas palavras dadas as palavras anteriores), não é necessário armazenar todas as informações na sequência de entrada até o tempo t, apenas o suficiente para prever o resto da sentença.

Um déficit da RNN *vanilla* é sua dificuldade em capturar longa dependência nos dados. Esse problema é comumente relacionado ao fenômeno de dissipação (ou explosão) do gradiente, que ocorre no ajuste da rede. Dentre as variantes desenvolvidas para contornar esse problema, destacaremos a *long short-term memory* (Goodfellow et al., 2016; Fan et al., 2021).

LSTM - *Long Short-Term Memory*

De acordo com Goodfellow et al. (2016), as LSTM fazem parte de uma classe de modelos chamada de RNN fechadas (*gated RNN*). Os portões (*gates*), que também são camadas da rede neural, controlam o fluxo de informação, mantendo ou descartando o estado oculto $\boldsymbol{h}_t$ a cada passo temporal (Kamath et al., 2019).

Denote por $\odot$ o produto de Haddamard, isto é, a multiplicação elemento-a-elemento. O estado de célula $\boldsymbol{c}_t$ carrega informações da sequência (por exemplo, forma singular ou plural de uma sentença). O portão de esquecimento (*forget gate*) $\boldsymbol{f}_t$ determina por quanto tempo os valores de $\boldsymbol{c}_{t-1}$ são mantidos, o portão de entrada (*input gate*) $\boldsymbol{i}_t$ controla a quantidade de atualizações realizadas no estado de célula, e o portão de saída (*output gate*) $\boldsymbol{o}_t$ define quanta informação $\boldsymbol{c}_t$ revela para $\boldsymbol{h}_t$, de forma que a arquitetura da rede se torna

$$\begin{aligned} \boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{h}_t, \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \varphi(\boldsymbol{c}_t), \end{aligned}$$

em que $\varphi(\cdot)$ é uma função de ativação, usualmente a *tanh*. Além disso, os elementos desses portões têm valores no intervalo $(0, 1)$, sendo utilizada a função de ativação *logit* para garantir isso. O estado de célula $\boldsymbol{c}_t$ tem uma adição em sua fórmula, o que auxilia o algoritmo de retropropagação e, portanto, captura-se longa dependência nos dados (Goodfellow et al., 2016; Fan et al., 2021).

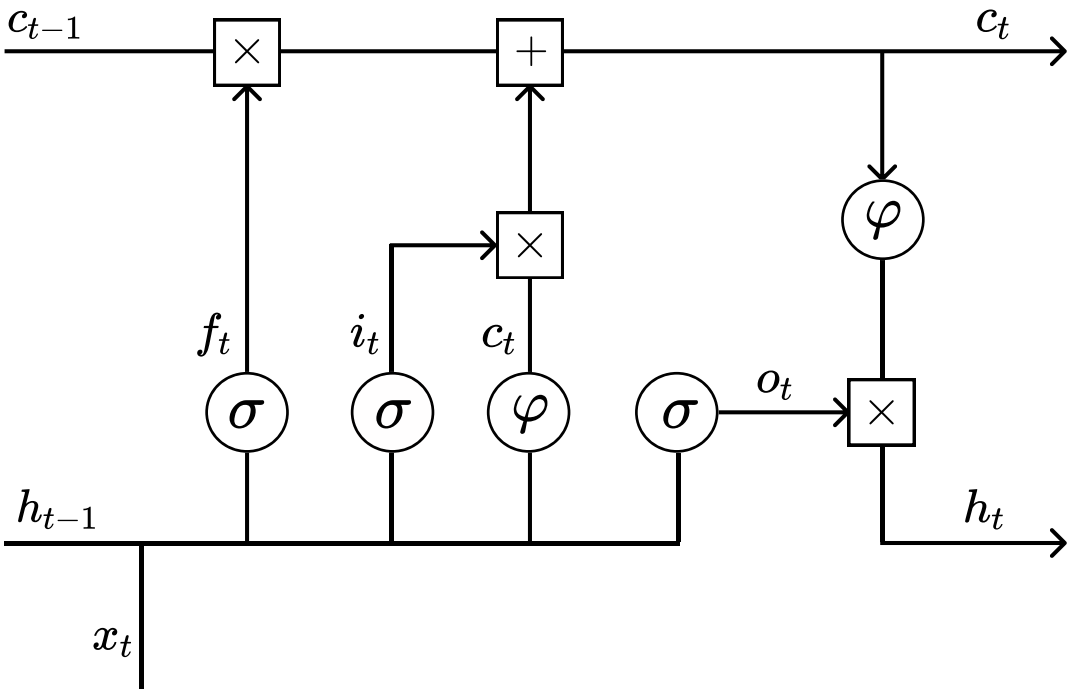

Figura 2.5: Diagrama de uma LSTM. Considere σ como a função de ativação *logit* (adaptado de Kamath et al., 2019).

Entende-se, portanto, que o diferencial de uma LSTM em relação a RNN *vanilla* são as suas células LSTM (*LSTM cells*), que contêm uma recorrência interna (*self-loop*), além da recorrência existente da RNN. Sendo assim, cada célula possui as mesmas entradas e saídas de uma RNN, mas há mais parâmetros e um sistema de portões controlando o fluxo de informação (Goodfellow et al., 2016).

2.2.5 Ajustando uma RNA

Algoritmos de aprendizagem de máquina, usualmente, envolvem alguma parte de otimização, referindo-se a minimizar (ou maximizar) uma função $f(x)$. Tal função pode ser chamada de **função perda** (*loss function*).

Seja $f : \mathbb{R}^n \to \mathbb{R}^n$ uma função diferenciável, e denote por $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ o gradiente de f no ponto $\boldsymbol{x} \in \mathbb{R}^n$. Os pontos críticos de f são as soluções das equações $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \mathbf{0}$, em que $\mathbf{0}$ denota o valor nulo em $\mathbb{R}^n$. Posto isso, podemos minimizar f por meio do método do gradiente descendente (***gradient descent***), também conhecido como ***steepest descent***, propondo um novo ponto

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}),$$

em que ϵ é chamada de **taxa de aprendizagem** (*learning rate*), um escalar positivo que determina o tamanho do passo em termos do valor do gradiente no ponto. O método converge quando todos os elementos do gradiente ficam abaixo de uma tolerância predeterminada (Goodfellow et al., 2016).

Uma das desvantagens do método do gradiente descendente é o uso de todo o conjunto de treino para o cálculo do gradiente. Devido a isso, o gradiente descendente

estocástico (***stochastic gradient descent* - SGD**) utiliza uma partição aleatória dos dados, denominada lote. Iniciando com um certo valor $\boldsymbol{\theta}_0$, o SGD ajusta os parâmetros $\boldsymbol{\theta}_t$ em direção ao gradiente negativo, e utilizando uma partição aleatória dos dados de forma iterativa para minimizar uma função perda. Pela Lei dos Grandes Números, o SGD aproxima-se do gradiente calculado. Por fim, temos que uma passagem pelo conjunto de dados de treinamento é denominada **época** (*epoch*), e após um determinado número de épocas, o treinamento é concluído (Goodfellow et al., 2016; Kamath et al., 2019; Fan et al., 2021).

ADAM (*adaptive moment estimation*) é um método de otimização do SGD direcionado a problemas de aprendizagem de máquina com grandes conjuntos de dados ou com espaço paramétrico com muitas dimensões. O método combina as vantagens do AdaGrad (Duchi et al., 2011) de lidar com esparsidade de gradientes (gradientes iguais a zero) e as vantagens do RMSProp (Tieleman e Hinton, 2012) de lidar com não estacionariedade. O detalhamento destes métodos fogem ao escopo deste trabalho. Mais informações podem ser encontradas em Kingma e Ba (2014).

O algoritmo de retropropagação (***backpropagation***) permite que a informação da função perda retorne através da rede neural, com o intuito de calcular o gradiente. O termo retropropagação refere-se ao método de retornar o erro de predição para calcular o gradiente e, por meio do SGD, os parâmetros da rede neural (no caso, os pesos) são ajustados. No contexto de RNN, os gradientes são calculados considerando, também, o índice $t \in T$ e, nesse caso, o algoritmo é chamado de retropropagação ao longo do tempo (***backpropagation through time* - BPTT**). Maiores detalhes podem ser encontrados em Haykin (2009), Goodfellow et al. (2016) e Fan et al. (2021).

2.3 Processamento de Linguagem Natural

O objetivo da linguística é ser capaz de caracterizar e de explicar as diversas observações linguísticas ao nosso redor, presentes em conversações, na escrita e em outras mídias. Parte disso está relacionado à compreensão das estruturas linguísticas pelas quais a linguagem se comunica e, devido a isso, foram propostas regras para a estruturação das expressões linguísticas. Tal abordagem tornou-se cada vez mais formal e rigorosa à medida que os linguistas exploravam gramáticas detalhadas que tentavam descrever o que seria um texto bem ou mal escrito (Manning e Schutze, 1999).

Processamento de Linguagem Natural (PLN), também conhecida como Linguística Computacional, foca na aplicação de métodos quantitativos e estatísticos para entender como seres humanos modelam linguagem, bem como abordagens computacionais para responder perguntas linguísticas. Ou seja, é a aplicação de métodos computacionais para modelar e extrair informações da linguagem humana (Kamath et al., 2019).

De acordo com Jurafsky e Martin (2021), é importante testar algoritmos em mais de uma linguagem, especialmente em linguagens com diferentes propriedades. Em contraste, segundo Bender (2019), há uma infeliz tendência atual: algoritmos de PLN serem desenvolvidos apenas em inglês. Mesmo quando algoritmos são desenvolvidos para outras linguagens, eles tendem a serem desenvolvidos para idiomas oficiais de grandes nações industrializadas (chinês, espanhol, alemão, etc.), limitando essas ferramentas. Vale também destacar que o texto em que essas ferramentas se-

rão aplicadas podem ser oriundos de fontes como notícias, livros, artigos científicos, reuniões de negócios, transcrições de programas televisivos ou filmes, textos jurídicos, entre outros. Em suma, ainda segundo Jurafsky e Martin (2021), é importante considerar quem produziu a linguagem, em qual contexto e para qual propósito.

2.3.1 Corpus e Corpora

Um *corpus* (plural *corpora*) é uma coleção de material textual (seja em um único documento ou uma coleção de documentos) construído de acordo com algum critério. Esse material é constituído com ao menos uma linguagem escrita (representação via símbolos de uma linguagem falada ou gestual). Em PLN, comumentemente temos um *corpus* com uma determinada quantidade de dados de algum domínio de interesse (como discursos, material literário, conversas em redes sociais, entre outros), sem necessariamente ter algum tipo de opinião de como ele foi construído. Dentre os cuidados que devemos ter ao selecionar um *corpus* ou ao relatar os resultados obtidos nas análises são se o tipo de texto é representativo ou se os resultados obtidos são úteis para o domínio de interesse (Manning e Schutze, 1999; Kamath et al., 2019).

2.3.2 Expressões Regulares

Expressões Regulares são particularmente úteis para pesquisar em textos, localizando determinados padrões em um *corpus*. Jurafsky e Martin (2021) comentam que, formalmente, uma expressão regular é uma notação algébrica utilizada para caracterizar um conjunto de *strings*. Além disso, uma função de busca com expressões regulares pode ser projetada para retornar todas as correspondências ao padrão definido, bem como apenas a primeira. Por fim, detalhamentos e demais definições referentes a expressões regulares estão disponíveis em Jurafsky e Martin (2021).

2.3.3 Vocabulário e Tokenização

Tokenização é o processo computacional de segmentar texto em unidades denominadas *tokens*, podendo ser palavras, números ou sinais de pontuação. Em seu processo mais simples, tokenização pode ser realizada ao separar texto por espaços em branco, induzindo um vocabulário ou um dicionário (Kamath et al., 2019; Jurafsky e Martin, 2021).

Segundo Manning e Schutze (1999), um método comumente utilizado em PLN é o mapeamento de palavras e de seus *tokens* para números (realizando a conversão do número para caractere quando necessário). Dentre as formas de realizar esse mapeamento, podemos citar uma tabela *hash* (uma função *hash* mapeia um conjunto de objetos para um intervalo específico de inteiros não negativos), que conterá um conjunto de *tokens* e seus respectivos índices. Por meio de tal tabela, pode-se obter tanto os índices quanto os *tokens* quando necessário.

2.3.4 Modelos de Linguagem

Um modelo estatístico de linguagem é aquele que atribui probabilidades para uma sequência de *tokens*. Por exemplo, pode-se determinar a probabilidade de uma determinada sequência por meio da probabilidade de cada *token* disponível nos *tokens* anteriores. Dentre suas aplicações, tem-se reconhecimento de fala, tradução

automática, marcação de parte da fala (*part-of-speech tagging*), reconhecimento de manuscrito, recuperação de informação, entre outras (Kamath et al., 2019; Jurafsky e Martin, 2021).

2.3.5 Avaliação Intrínseca e Extrínseca

Existem dois tipos de avaliação de um modelo de linguagem: extrínseca e intrínseca. Uma avaliação **extrínseca** envolve encapsular o modelo em uma aplicação e medir se houve melhorias. Por exemplo, em um contexto de reconhecimento de fala, o texto transcrito seria comparado com o discurso realizado. Apesar da avaliação extrínseca verificar qual ajuste melhora o desempenho do modelo, desenvolver um sistema completo para tal objetivo é usualmente caro, afetando sua viabilidade. Uma alternativa é a avaliação **intrínseca**, que consiste em utilizar uma métrica objetiva para mensurar as potenciais melhorias do modelo de linguagem. Basicamente, é um tipo de avaliação que mede a qualidade de um modelo independentemente de sua aplicação. No entanto, uma melhora na métrica intrínseca não necessariamente implica em uma melhora na avaliação extrínseca.

No caso de uma avaliação intrínseca de um modelo de linguagem, assim como em muitos modelos estatísticos, realizamos o ajuste por meio de um conjunto de treino, e medimos a qualidade do modelo pelo seu desempenho no conjunto de teste. Dado um *corpus*, podemos dividi-lo em treino e teste por meio de sequências de *tokens* (como palavras ou caracteres) presentes em partes sorteadas do texto. Mais detalhes podem ser encontrados em Goldberg (2017) e Jurafsky e Martin (2021).

2.3.6 Entropia e Perplexidade

Seja $p(w)$ a função massa de probabilidade de uma variável aleatória discreta W, associada a um conjunto de *tokens* (ou vocabulário) L. A **entropia** mede a quantidade de informação de uma variável aleatória, sendo definida por

$$H(p) := -E\Big(\log\big(p(W)\big)\Big) = -\sum_{w\in L} p(w)\log\big(p(w)\big).$$

No contexto de uma sequência de palavras de tamanho n, digamos $W_1, \cdots, W_n$, calcula-se a taxa de entropia (*entropy rate*) por

$$H_{rate} := -\frac{1}{n}\sum_{\boldsymbol{w}\in L^n} p(\boldsymbol{w})\log\big(p(\boldsymbol{w})\big).$$

Quando a verdadeira massa de probabilidade p que gerou os dados é desconhecida, pode-se utilizar a **entropia cruzada** (*cross-entropy*) para aproximar a entropia. Seja m um estimador para p (por exemplo, a massa de probabilidade empírica), a entropia cruzada é definida pela expressão

$$H(p,m) := \lim_{n\to\infty} -\frac{1}{n}\sum_{\boldsymbol{\omega}\in L^n} p(\boldsymbol{w})\log\big(m(\boldsymbol{w})\big).$$

Quando o processo estocástico $H(p,m)$ é estacionário e ergótico, podemos aproximá-lo via

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\sum_{\boldsymbol{\omega}\in L^n} \log\big(m(\boldsymbol{w})\big).$$

A intuição é que uma sequência de palavras suficientemente longa conterá outras sequências menores, e que cada uma dessas irá ocorrer recorrentemente na sequência mais longa de acordo com suas respectivas probabilidades.

É interessante observar que a entropia cruzada $H(p, m)$ é o limite superior da entropia $H(p)$. Devido a isso, temos que

$$H(p) \leq H(p, m).$$

Isso indica que pode-se utilizar m para estimar a verdadeira entropia de uma sequência de *tokens* gerados por p. Quanto maior for a acurácia de m, mais próximo $H(p, m)$ será de $H(p)$. Dessa forma, ao comparar dois estimadores, m_1 e m_2, o melhor será aquele com a menor entropia cruzada.

Perplexidade é uma medida oriunda da teoria da informação, comumente utilizada para a avaliação intrínseca de um modelo de linguagem, definida como

$$\mathcal{P}(\boldsymbol{W}) := e^{H(\boldsymbol{W})} = p(\boldsymbol{w})^{-\frac{1}{n}},$$

em que $\boldsymbol{W}$ é o vetor de tamanho n contendo a sequência de palavras e p é a função massa de probabilidade conjunta associada. Medidas menores de perplexidade são indicativas de uma predição melhor. Por fim, é importante destacar que a perplexidade de dois modelos de linguagem apenas podem ser comparadas se ambos utilizam o mesmo vocabulário (*corpus*). Maiores informações são encontradas em Manning e Schutze (1999), Kamath et al. (2019) e Jurafsky e Martin (2021).

2.3.7 Word Embedding

Em PLN, as palavras (ou *tokens*) são variáveis categóricas que precisam de uma codificação para que sejam utilizadas em um modelo estatístico. Seguem duas possibilidades:

- *one-hot encoding*: é criado uma variável indicadora (*dummy*) para cada categoria.
- *dense encoding (feature embeddings)*: cada categoria é embutida em um vetor d-dimensional.

O principal benefício da *dense encoding* é a capacidade de generalização, pois os valores do vetor denso são ajustáveis, possibilitando a captura de relações entre as diferentes palavras.

No contexto de RNA, podemos utilizar vetores esparsos (*one-hot encoding*) na camada de entrada e uma camada embutida (*embedding layer*) para obter o vetor denso que representará as categorias. Matematicamente, cada categoria f_i é mapeada a um vetor denso $d-$dimensional $\boldsymbol{v}(f_i)$. Seja um vocabulário de $|V|$ palavras, a coleção de vetores pode ser representada pela matriz $\boldsymbol{E}_{|V|\times d}$, em que cada linha corresponde a uma categoria. Considere $\boldsymbol{f_i}$ a representação *one-hot* da categoria f_i, i.e., um vetor $|V|$ dimensional em que todas os valores são zeros exceto por um índice de valor 1, correspondente à i-ésima categoria. Dessa forma, o produto $\boldsymbol{f_i E}$ irá "selecionar" a respectiva linha de $\boldsymbol{E}$. Posto isso, podemos definir $\boldsymbol{v}(f_i)$ como

$$\boldsymbol{v}(f_i) = \boldsymbol{f_i E},$$

em que a entrada da rede é a coleção de vetores *one-hot*. Apesar de elegante e bem definido matematicamente, uma implementação computacional eficiente não utiliza essa representação, mas sim uma estrutura de dados estilo tabela *hash*, mapeando cada categoria a seu respectivo vetor denso (Goldberg, 2017).

2.4 Notação ABC

Notação ABC é um sistema popular de notação musical baseada em texto para transcrever, publicar e compartilhar músicas folclóricas, particularmente de forma *online*. Criada e formalizada por Walshaw (1993), o autor mantém um website, `abcnotation.com`, com recursos como tutoriais, programas e coleções de músicas no formato (Walshaw, 2014).

A notação ABC é composta por duas partes: o cabeçalho, que fornece os metadados da música como o título e características da música, e o corpo, que detalha as notas musicais. Existem *tags* no cabeçalho como "T" (título) e "X" (número da música) que não afetam a síntese musical de nenhuma forma. Vale notar que a notação ABC define um mapeamento entre seus caracteres e símbolos específicos de uma partitura musical, o que permite a conversão de um formato para outro (Agarwala et al., 2017).

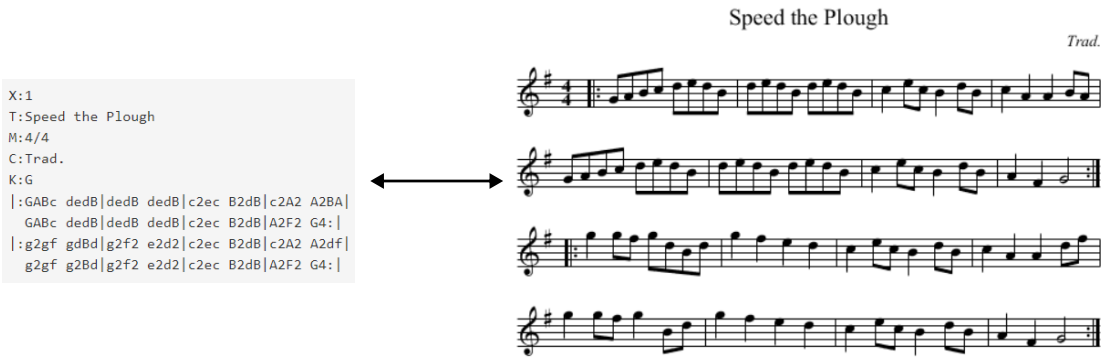

Figura 2.6: Exemplo de notação ABC convertendo em música (disponível em `abcnotation.com`).

2.5 Web Scraping

Web scraping é uma técnica de extração automática de dados de fontes online como *websites* (Farley e Pierotte, 2017; Khder, 2021). Seu desenvolvimento envolve dois programas customizados: um *crawler* e um *scraper*. O ***crawler*** sistematicamente coleta os dados da *internet* (especificamente, cria uma réplica da página visitada). Já o ***scraper*** extrai a informação relevante dos dados baixados e os armazena em uma base de dados, no formato e estrutura definidos pelo usuário (Lawson, 2015; Patil e Patil, 2016).

Grande parte dos *websites* contém um arquivo *robots.txt* para informar sobre quaisquer restrições sobre o uso da técnica em seus domínios. Essas restrições são

apenas sugestões, mas é recomendado que sejam seguidas, pois além de poderem conter informações sobre a estrutura do *website*, minimiza a chance do *crawler* ser bloqueado. Mais informações sobre o arquivo *robots.txt* estão disponíveis em `robotstxt.org` (Lawson, 2015).

Dentre as linguagens comumente utilizadas para o desenvolvimento do *crawler* e do *scraper* podemos citar R e Python. Detalhamentos sobre o assunto podem ser encontrados em Lawson (2015), Sirisuriya (2015), Patil e Patil (2016), Farley e Pierotte (2017) e Khder (2021).

3 Estudo de Modelagem

O estudo de modelagem visa a explorar o impacto de diferentes configurações de uma rede neural utilizada para composição musical. Ao modificar os parâmetros e hiperparâmetros, serão observados os impactos no ajuste da rede, bem como na métrica selecionada. Por fim, os diferentes modelos gerarão novas peças musicais, que serão avaliadas de forma subjetiva dado o desafio de mensurar uma música.

3.1 Implementação Computacional

O processo de obtenção de dados, bem como os modelos utilizados nesse trabalho foram desenvolvidos na linguagem de programação Python e na IDE (*Integrated Development Environment*) VS Code. As bibliotecas utilizados neste trabalho foram as seguintes:

- **Python**: versão 3.9.12;
- **TensorFlow**: versão 2.9.1;
- **NumPy**: versão 1.22.3;
- **regex**: versão 2.5.112;
- **sklearn**: versão 1.1.1;
- **bs4**: versão 4.10.0;
- **requests**: versão 2.27.1;
- **music21**: versão 7.3.3.

Além disso, as análises dos resultados foram realizados na linguagem R e na IDE RStudio:

- **R**: versão 4.1.2;
- **ggplot2**: versão 3.3.5;
- **patchwork**: versão 1.1.2;
- **dplyr**: versão 1.0.8.

Por fim, todos os códigos desenvolvidos nesse trabalho estão disponíveis em um repositório no GitHub do autor: `github.com/nmhahn/TCC_NMH`.

3.2 Bases de Dados

O trabalho foi realizado utilizando duas bases de dados: Irish e ABC Notation. Abaixo, seguem algumas explicações sobre cada uma:

- **Irish**: base de dados contendo 817 músicas folclóricas irlandesas no formato *.abc*. Foi utilizada a versão disponibilizada pelo Instituto de Tecnologia de Massachusetts (MIT) em seu GitHub[1]. No entanto, existem mais exemplos de fontes de músicas no formato[2].

- **ABC Notation**: inspirado pelo artigo de Agarwala et al. (2017), o autor quis realizar sua própria coleta de músicas no formato *.abc* do site abcnotation.com. Foi desenvolvido, na linguagem Python, um *crawler* e um *scrapper* para a obtenção dos dados. Após uma semana de execução, foram obtidos 184.900 arquivos no formato desejado contendo diversas informações referentes às peças musicais (como título, autor, tonalidade da música, entre outras). Por conseguinte, foi selecionada uma amostra aleatória simples de 5.000 músicas dessa base.

Depois, ambas as bases de dados utilizadas passaram por três procedimentos: tratamento, união e tokenização. Abaixo, alguns detalhes sobre cada uma:

- **tratamento**: um arquivo no formato *.abc* contém diversas *tags* com informações como título, subtítulo, autor, entre outras. Além disso, no arquivo também estão contidos caracteres representando letra de música, bem como caracteres de comentário. Como nenhum desses itens afetam diretamente a música, considerando que as notas musicais serão as mesmas, o autor resolveu, por meio de expressões regulares, remover esses caracteres, com o intuito de diminuir o ruído nos arquivos. Após os tratamentos, o arquivo final contém dados como índice (representando o início da música), tonalidade, andamento, vozes da música (no caso de músicas com múltiplos instrumentos) e as notas musicais (mantendo a estrutura da peça musical).

- **união**: todas as músicas foram "coladas", como se fizessem parte de um único texto. Esse processo possibilita que uma música inicie após o término da outra, pois temos, de certa forma, uma ordem de músicas (no caso, foi a ordem presente na base, sem aleatorização).

- **tokenização**: para cada caractere (*token*) presente, foi criado um único índice. Consequentemente, é possível que uma sequência de caracteres seja convertido para uma sequência única de índices, possibilitando a conversão de texto para número e vice-versa.

3.3 Modelo

Utilizou-se um modelo de RNN-LSTM, que foi ajustado com ambas as bases de dados de forma independente, ou seja, tem-se uma versão com a base Irish e outra com a base ABC Notation. Esse modelo não é original do autor, pois foi

[1] github.com/aamini/introtodeeplearning/blob/master/mitdeeplearning/data
[2] norbeck.nu/abc/links.asp

desenvolvido no laboratório de geração musical[3] ministrado pelo MIT. Além disso, foi feita uma divisão dos dados em 80% treino e 20% teste. No entanto, todo o processo de tratamento foi realizado antes dessa divisão.

3.3.1 Arquitetura

Como elucidado na seção 2.2.4, uma rede neural é constituída de camadas de neurônios e de nós de fonte. O modelo utilizado contém quatro camadas: entrada, *Embedding*, LSTM e *Dense*. A primeira contém o vocabulário (conjunto de *tokens*), sendo único para cada base. A segunda é responsável pelo *dense encoding*, identificando as possíveis relações entre os *tokens*. A terceira contém os neurônios LSTM, responsáveis pelo mapeamento temporal da sequência de elementos. Por fim, na última camada encontram-se as saídas da rede, sendo os resultados preditos.

3.3.2 Parâmetros e Ajuste da Rede

O processo de modelagem foi feito utilizando 2000 épocas no ajuste do modelo, tendo a entropia cruzada como função perda e a perplexidade como métrica de avaliação. Ademais, houve uma segmentação em duas etapas. Na primeira etapa, foi fixada a função de ativação *tanh* na camada LSTM, alterando-se os demais parâmetros e hiperparâmetros:

- o tamanho do vocabulário: *vocab_size*, sendo 64 para *Irish* e 125 para *ABC Notation*;
- número de células LSTM: $lstm\_units \in \{256, 1024\}$;
- a dimensão do vetor utilizado para *dense encoding*: $embedding\_dim \in \{256, 512\}$;
- a taxa de aprendizagem no ajuste: $learning\_rate \in \{10^{-3}, 10^{-5}\}$;
- comprimento da sequência obtida dos dados: $seq\_length \in \{50, 200\}$;
- tamanho do lote sorteado do texto único, via amostra aleatória simples sem reposição: $batch\_size \in \{4, 16\}$.

Consequentemente, foram ajustados 64 modelos, sendo 32 para cada base de dados. A segunda etapa do experimento consistiu em trocar a função de ativação para a *logit*, e avaliar se houve mudanças na perplexidade dos modelos. Devido ao volume de modelos, alterou-se apenas os dois piores e os dois melhores em termos da perplexidade.

3.3.3 Processo Gerador de Músicas

Tendo os modelos devidamente ajustados, foi possível realizar a composição algorítmica de novas peças musicais. O processo gerador consiste de cinco passos:

1. construir um modelo com os devidos parâmetros (*vocab_size*, *lstm_units*, *embedding_dim* e *batch_size*), mas fixar $batch\_size = 1$ para que haja apenas um vetor de entrada e um de saída;

[3] goodboychan.github.io/python/tensorflow/mit/2021/02/14/music-generation.html

2. fixado o modelo, carregam-se os pesos de um modelo similar ajustado previamente, sendo a única diferença entre eles o parâmetro *batch_size*;

3. o processo gerador envolve fornecer uma sequência de caracteres inicial, neste trabalho foi fornecida "X:" (*tag* que contém o número da música), pois é como qualquer música presente no arquivo *.abc* inicia;

4. de forma iterativa, o modelo estima um novo elemento para compor a sequência até atingir um comprimento definido (no caso, utilizou-se 1000);

5. via expressões regulares, são extraídos da sequência blocos de texto candidatos a músicas e, ao serem convertidos com sucesso, resultam em músicas.

4 Resultados

Cada base de dados será avaliada individualmente, considerando as diferentes configurações de parâmetros definidas e os impactos em ambas função perda (entropia cruzada) e métrica de avaliação (perplexidade). Como comentado na seção 2.3.6, dois modelos de linguagem são comparáveis apenas se compartilharem o mesmo vocabulário (único por base). Após, será explanado sobre as (potenciais) músicas geradas, bem como as percepções subjetivas do autor.

4.1 Irish

Inicialmente, avaliou-se a primeira etapa do estudo. A tabela 4.1 contém os resultados dos 32 modelos ajustados (identificados pela coluna *idx*) para a base de dados Irish, bem como os respectivos parâmetros e função de ativação *tanh*. Além disso, a figura 4.1 inclui a função perda calculada para cada época do ajuste.

De maneira geral, observou-se certos padrões no comportamento da função perda de acordo com a configuração da rede:

- No geral, o aumento do *batch_size* resultou na redução dos valores da função perda, bem como em uma menor variabilidade a cada época. Tal comportamento é coerente, dado que há mais dados utilizados no treino.
- *seq_length* tem comportamento e justificativa similares ao *batch_size*.
- O impacto do *learning_rate* na função perda é evidente, acusando um aumento na perda quando a taxa de aprendizagem é menor. Isso ocorre, pois, ao ser calculado o gradiente, o passo foi demasiado pequeno para o processo de otimização.
- O parâmetro *embedding_dim* não parece, em geral, afetar significativamente nos valores da função perda nos experimentos conduzidos.
- Aumentar os valores de *lstm_units*, geralmente, ocasionou em uma redução na função perda.

A figura 4.2 apresenta a matriz de correlação entre os parâmetros e as métricas, sendo o ρ de Spearman a medida de correlação. É notável que a matriz corrobora os padrões identificados. Destaque para a forte correlação negativa de -0.87 de *learning_rate* tanto com a perda quanto com a perplexidade. Ademais, é interessante

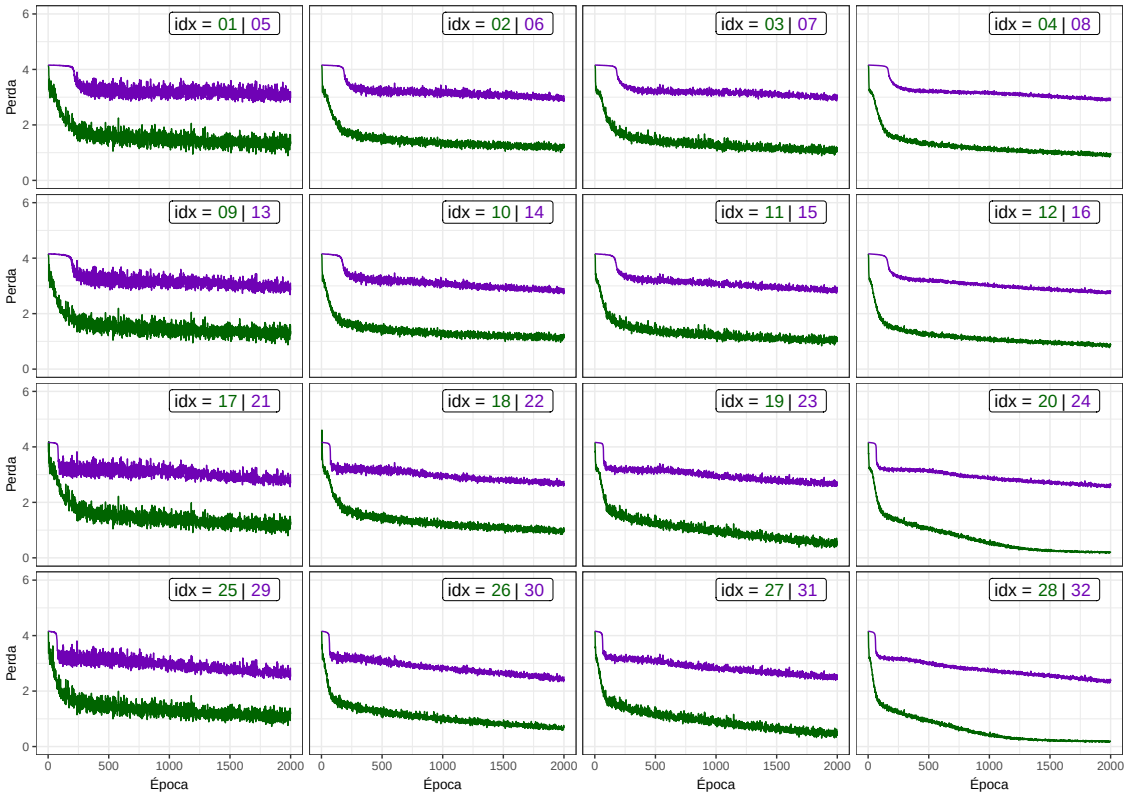

Figura 4.1: Perda dos modelos ajustados com Irish: curva verde representa $learning\_rate = 10^{-3}$ e roxo $learning\_rate = 10^{-5}$.

notar que *batch_size* apresentou uma (fraca) correlação positiva com a perplexidade, sendo um indicativo de tendência a sobreajuste do modelo.

Na segunda etapa, os modelos selecionados, em ordem decrescente para perplexidade, foram $idx \in \{5, 7, 11, 25\}$. Pela tabela 4.2, nota-se que as redes com função de ativação *logit* tiveram um desempenho pior quando comparadas à *tanh*.

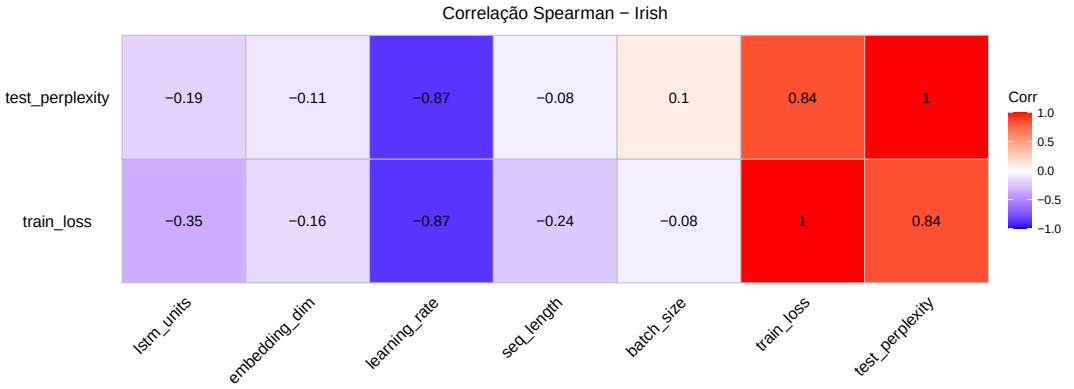

Figura 4.2: Correlação entre parâmetros e métricas para Irish

Tabela 4.1: Resultados dos experimentos referentes à base de dados Irish, com *vocab_size* = 64, utilizando a função *tanh* como função de ativação. Em azul (vermelho), constam os dois melhores (piores) modelos em relação à perplexidade.

idx	lstm_units	embedding_dim	learning_rate	seq_length	batch_size	train_loss	test_perplexity
1	256	256	10^{-3}	50	4	1.403	3.124
2	256	256	10^{-3}	50	16	1.296	4.205
3	256	256	10^{-3}	200	4	1.017	2.946
4	256	256	10^{-3}	200	16	0.973	3.054
5	256	256	10^{-5}	50	4	2.964	24.709
6	256	256	10^{-5}	50	16	3.060	17.931
7	256	256	10^{-5}	200	4	2.963	20.401
8	256	256	10^{-5}	200	16	2.939	18.316
9	256	512	10^{-3}	50	4	1.370	3.143
10	256	512	10^{-3}	50	16	1.235	4.034
11	256	512	10^{-3}	200	4	0.982	2.835
12	256	512	10^{-3}	200	16	0.897	3.050
13	256	512	10^{-5}	50	4	2.846	20.247
14	256	512	10^{-5}	50	16	2.922	15.660
15	256	512	10^{-5}	200	4	2.815	17.581
16	256	512	10^{-5}	200	16	2.801	15.721
17	1024	256	10^{-3}	50	4	1.298	2.896
18	1024	256	10^{-3}	50	16	1.011	3.999
19	1024	256	10^{-3}	200	4	0.615	2.884
20	1024	256	10^{-3}	200	16	0.223	4.689
21	1024	256	10^{-5}	50	4	2.692	16.002
22	1024	256	10^{-5}	50	16	2.778	13.960
23	1024	256	10^{-5}	200	4	2.645	14.593
24	1024	256	10^{-5}	200	16	2.649	13.348
25	1024	512	10^{-3}	50	4	1.171	2.752
26	1024	512	10^{-3}	50	16	0.662	4.381
27	1024	512	10^{-3}	200	4	0.567	3.019
28	1024	512	10^{-3}	200	16	0.196	4.381
29	1024	512	10^{-5}	50	4	2.522	13.508
30	1024	512	10^{-5}	50	16	2.526	11.069
31	1024	512	10^{-5}	200	4	2.460	12.176
32	1024	512	10^{-5}	200	16	2.404	10.387

Também, na figura 4.3 são comparadas as curvas de perda dos modelos com cada função de ativação. Observou-se que o ajuste dos modelos com *logit* apresentou um comportamento mais suave nos modelos $idx \in \{5, 7\}$, não ocorrendo a mesma queda abrupta próxima da época 250 observada na *tanh*. Por fim, pode-se concluir que a função *tanh* apresentou melhores resultados para a base Irish.

Tabela 4.2: Irish: *tanh* vs *logit*

idx	*train_loss*		*test_perplexity*	
	tanh	*logit*	*tanh*	*logit*
5	2.964	3.021	24.709	27.576
7	2.963	3.135	20.401	24.726
11	0.982	1.195	2.835	3.549
25	1.171	1.389	2.752	3.263

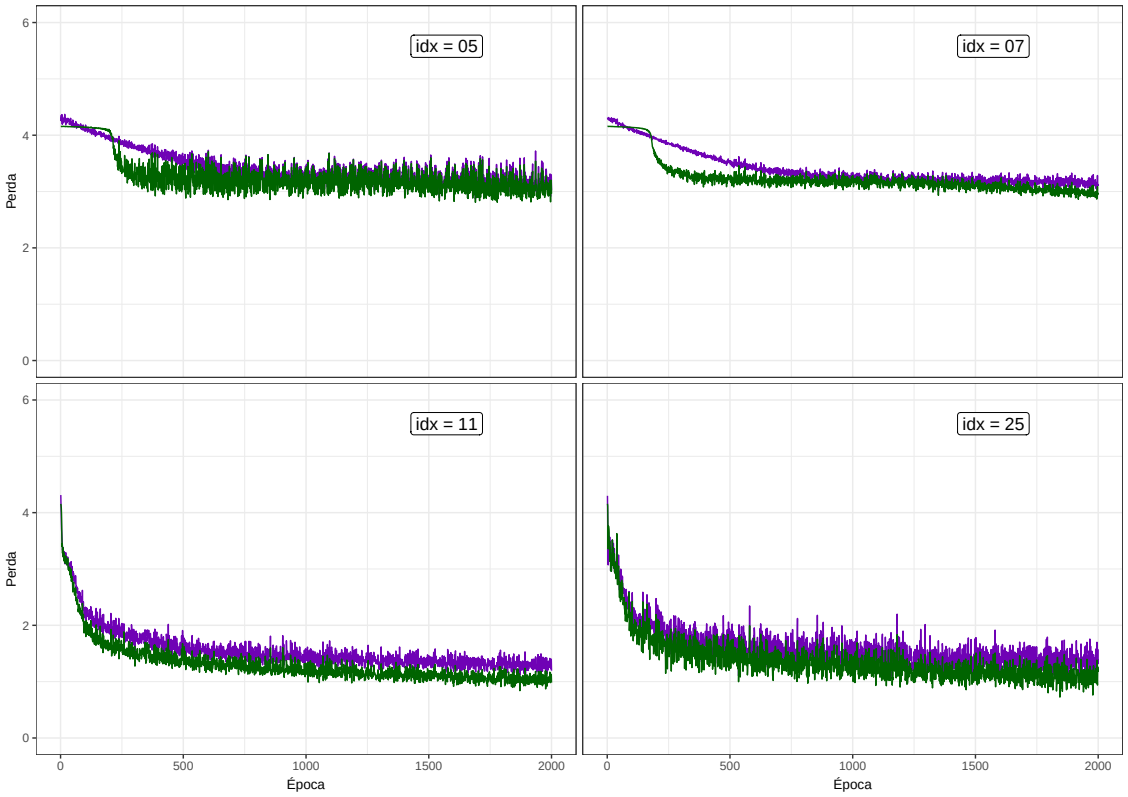

Figura 4.3: Irish: *tanh* (verde) vs *logit* (roxo)

4.2 ABC Notation

Por meio de uma abordagem similar a Irish, avaliou-se a primeira etapa para os casos de ABC Notation. A tabela 4.3 contém os resultados dos 32 modelos ajustados e na figura 4.4 constam as perdas calculadas ao longo das épocas dos respectivos ajustes.

De maneira geral, observou-se certos padrões no comportamento da função perda com a modificação de acordo com a configuração da rede:

- No geral, não é claro o impacto de *batch_size* na perda, mas permanece a redução de variabilidade a cada época.
- Diferentemente do observado em Irish, o aumento de *seq_length* apresentou, comumente, uma redução na função perda. Novamente, permanece a redução de variabilidade.
- Assim como notado em Irish, *learning_rate* tem um impacto expressivo na função perda, manifestando um aumento na perda quando a taxa de aprendizagem é menor.
- O parâmetro *embedding_dim* continuou sem expor modificações significativas na função perda.
- Permanece o comportamento demonstrado em Irish para *lstm_units*.

A figura 4.5 apresenta a matriz de correlação entre os parâmetros e as métricas, sendo o ρ de Spearman a medida de correlação. Uma vez mais, a matriz corrobora os padrões identificados, assim como a forte correlação negativa de -0.87 de *learning_rate* com a perda e com a perplexidade. Ademais, é interessante notar que *batch_size* apresentou uma (fraca) correlação negativa com a perplexidade, sendo o oposto observado na base Irish.

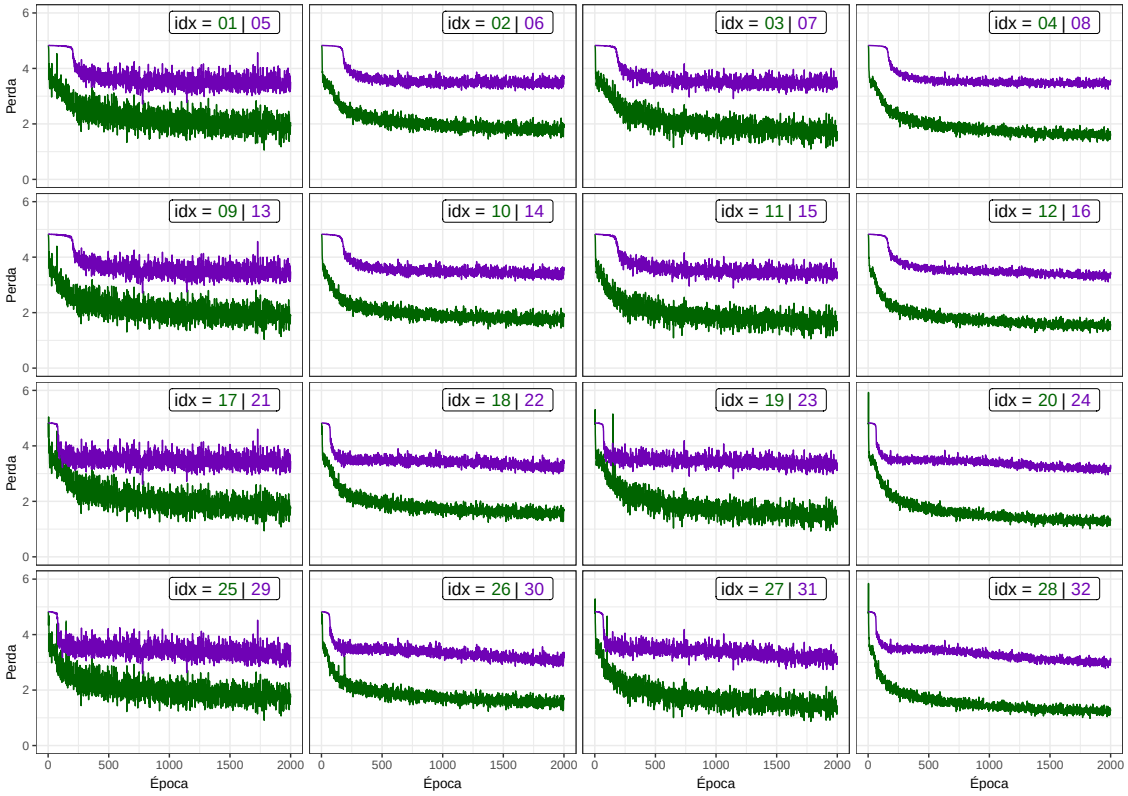

Figura 4.4: Perda dos modelos ajustados com ABC Notation: curva verde representa $learning\_rate = 10^{-3}$ e roxo $learning\_rate = 10^{-5}$.

Na segunda etapa, os modelos selecionados, em ordem decrescente para perplexidade, foram $idx \in \{8, 6, 20, 28\}$. Pela tabela 4.2, nota-se que as redes com função de ativação *logit* tiveram um desempenho pior quando comparadas à *tanh*.

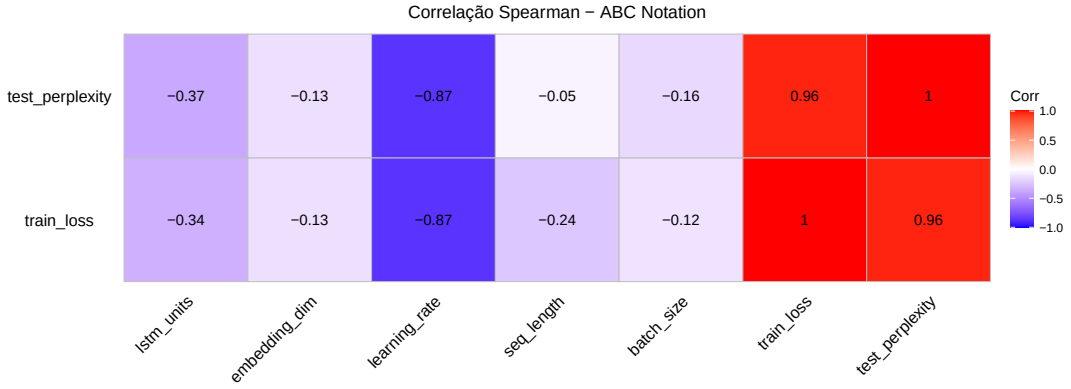

Figura 4.5: Correlação entre parâmetros e métricas para ABC Notation

Tabela 4.3: Resultados dos experimentos referentes à base de dados ABC Notation, com $vocab\_size = 125$, utilizando a função *tanh* como função de ativação. Em azul (vermelho), constam os dois melhores (piores) modelos em relação à perplexidade.

idx	lstm_units	embedding_dim	learning_rate	seq_length	batch_size	train_loss	test_perplexity
1	256	256	10^{-3}	50	4	2.058	6.572
2	256	256	10^{-3}	50	16	1.914	5.411
3	256	256	10^{-3}	200	4	1.731	5.763
4	256	256	10^{-3}	200	16	1.657	5.139
5	256	256	10^{-5}	50	4	3.517	29.208
6	256	256	10^{-5}	50	16	3.628	31.707
7	256	256	10^{-5}	200	4	3.508	31.001
8	256	256	10^{-5}	200	16	3.522	35.991
9	256	512	10^{-3}	50	4	1.924	5.949
10	256	512	10^{-3}	50	16	1.877	5.255
11	256	512	10^{-3}	200	4	1.675	5.505
12	256	512	10^{-3}	200	16	1.587	4.870
13	256	512	10^{-5}	50	4	3.483	28.005
14	256	512	10^{-5}	50	16	3.517	28.052
15	256	512	10^{-5}	200	4	3.435	29.033
16	256	512	10^{-5}	200	16	3.369	30.867
17	1024	256	10^{-3}	50	4	1.782	5.690
18	1024	256	10^{-3}	50	16	1.660	4.268
19	1024	256	10^{-3}	200	4	1.487	4.507
20	1024	256	10^{-3}	200	16	1.350	3.971
21	1024	256	10^{-5}	50	4	3.452	27.038
22	1024	256	10^{-5}	50	16	3.350	22.821
23	1024	256	10^{-5}	200	4	3.387	26.320
24	1024	256	10^{-5}	200	16	3.156	26.029
25	1024	512	10^{-3}	50	4	1.767	5.611
26	1024	512	10^{-3}	50	16	1.635	4.274
27	1024	512	10^{-3}	200	4	1.361	4.342
28	1024	512	10^{-3}	200	16	1.305	3.940
29	1024	512	10^{-5}	50	4	3.366	22.080
30	1024	512	10^{-5}	50	16	3.148	19.285
31	1024	512	10^{-5}	200	4	3.210	22.511
32	1024	512	10^{-5}	200	16	2.968	22.439

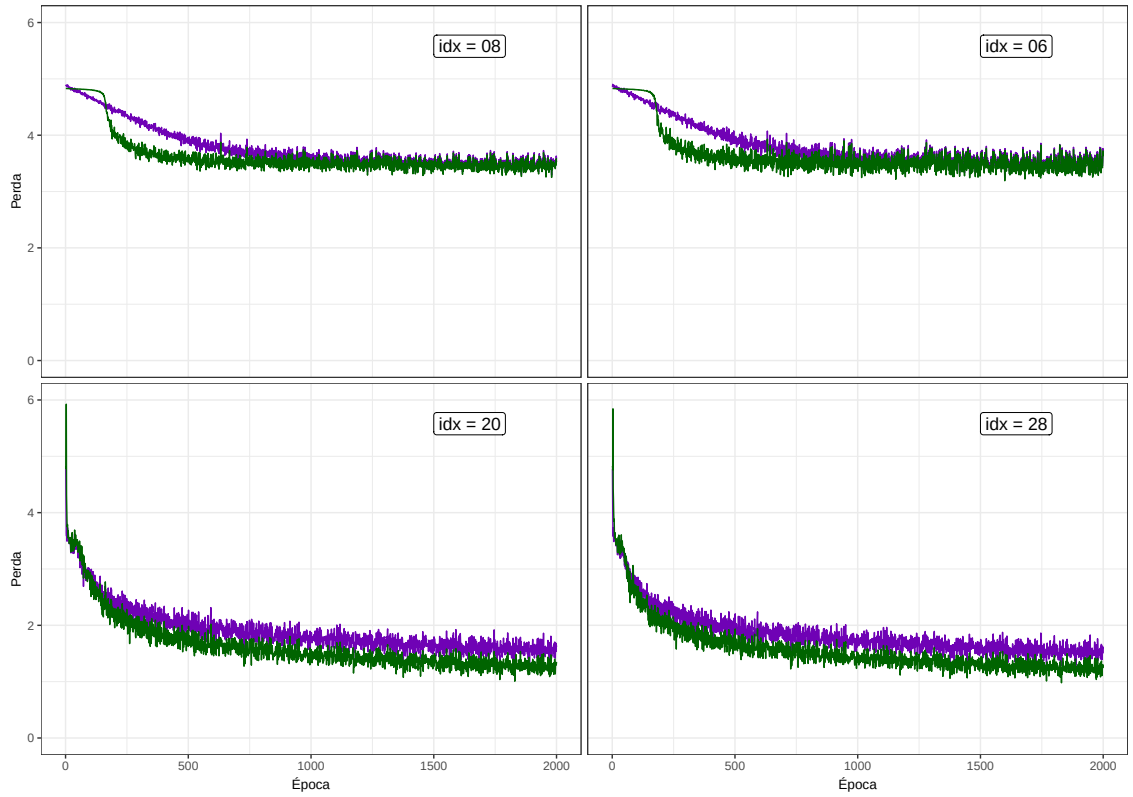

Figura 4.6: ABC Notation: *tanh* (verde) vs *logit* (roxo)

Tabela 4.4: ABC Notation: *tanh* vs *logit*

idx	*train_loss*		*test_perplexity*	
	tanh	*logit*	*tanh*	*logit*
8	3.628	3.670	31.707	33.038
6	3.522	3.571	35.991	37.849
20	1.350	1.616	3.971	5.142
28	1.305	1.585	3.940	4.960

Também, na figura 4.6 são comparadas as curvas de perda dos modelos com cada função de ativação. Observou-se que o ajuste dos modelos com *logit* apresentou um comportamento mais suave nos modelos $idx \in \{8, 6\}$, não ocorrendo a mesma queda abrupta próxima da época 250 observada na *tanh*. Por fim, pode-se concluir que a função *tanh* apresentou melhores resultados para a base ABC Notation. Nessa etapa, os resultados são similares aos vistos na base de dados Irish.

4.3 Músicas Geradas

Para cada modelo selecionado na segunda etapa, considerando ambas as bases e ambas as funções de ativação, foram geradas sequências de texto até compor dez blocos candidatos a músicas. Após isso, realizou-se a conversão dos arquivos *.abc* em música. Se a conversão é bem sucedida, uma nova peça musical é gerada; caso

contrário, o arquivo é invalidado. Por não ter sido imposta uma estrutura que garanta a conversão, nem todo *.abc* resultante gera uma música. Os resultados observados seguem:

- Apenas candidatos dos melhores modelos foram convertidos com sucesso (porém não todos). Isso indica que uma menor perplexidade contribui para que o texto produzido seja mais coerente para a conversão em música.
- Há uma maior taxa de conversão nos modelos utilizando *tanh* do que *logit*, sendo mais um indicativo de que a primeira apresenta desempenho superior à segunda.

Em síntese, foram geradas 24 músicas com Irish (10 para $idx = 11$ - com 6 para *tanh* e 4 para *logit* - e 14 para $idx = 25$ - com 8 para *tanh* e 6 para *logit*) e 14 com ABC Notation (8 para $idx = 20$ - 4 para cada função de ativação - e 6 para $idx = 28$ - com 5 para *tanh* e 1 para *logit*).

Ao escutarmos as músicas geradas para Irish, observa-se um certo padrão: diversos trechos são, via de regra, similares. É possível que o fator que justifique esteja relacionado com uma característica das músicas utilizadas no ajuste da rede neural. Além disso, é curioso observar que um dos arquivos *.abc* convertidos não tem, na realidade, nenhuma nota musical. Ou seja, foi criada a estrutura do arquivo sem nenhum preenchimento referente à música. Por fim, pode-se concluir que as peças musicais resultantes são musicalmente plausíveis.

No caso das músicas geradas com ABC Notation, o comportamento é mais interessante. Inicialmente, é clara a maior variabilidade utilizando essa base, sendo identificado estilos e até mesmos duas vozes sendo executadas simultaneamente (como mão direita e mão esquerda do piano). Ademais, uma das peças musicais geradas destoou das demais, apresentando combinações de notas aparentemente mais ruidosas. Novamente, surgiram arquivos convertidos sem música. Por conseguinte, temos, mais uma vez, resultados musicalmente plausíveis.

Os comentários feitos referente às composições geradas são as percepções do autor com seu limitado conhecimento musical. Para o leitor interessado, todo o material produzido para este trabalho está disponível conforme informado na seção 3.1, incluindo as músicas.

5 Considerações Finais

Este trabalho visou a explorar o problema de composição automática de músicas. Via modificações dos parâmetros, investigou-se a sensibilidade do modelo utilizado para a geração de arquivos *.abc* que, posteriormente, foram convertidos em novas peças musicais. Com o auxílio da medida perplexidade, foi possível mensurar se houve (ou não) mudanças nos resultados de cada configuração da rede neural. Também, observou-se que a função de ativação *tanh* teve desempenho superior na métrica para os cenários testados. Por fim, pode-se concluir que valores menores de perplexidade contribuem para a etapa de geração musical, posto que não houve nenhuma peça produzida pelas redes com resultados altos na métrica.

Considerando a natureza subjetiva da música, o autor comentou sobre suas próprias percepções referente às composições. Devido a isso, os pontos elencados podem conflitar com outros pontos de vista. Isso posto, observou-se um padrão nas peças musicais produzidas pela rede ajustada com Irish, sendo bastante similares em perfil de ritmo e sequência de notas musicais. No caso de ABC Notation, houve uma maior variabilidade de estilos em comparação, além do fato de ter surgido uma peça aparentemente mais ruidosa. De maneira sucinta, conclui-se que os resultados são musicalmente plausíveis.

É importante destacar que há muito a ser investigado no problema de composição algorítmica utilizando inteligência artificial. O presente trabalho concentrou-se em apenas um único modelo com uma única arquitetura. Assim sendo, extensões envolvem modificar a arquitetura contemplando o aumento do número de camadas, tal como outros perfis de redes, como a *transformer* (Vaswani et al., 2017). Além disso, devido aos indícios de sobreajuste dos modelos apresentados no trabalho, um tema interessante a ser explorado seria o dilema do viés e da variância (Geman et al., 1992). Por conseguinte, no contexto das composições, analisar detalhadamente características musicais almejando, talvez, peças com estilos específicos.

Referências Bibliográficas

Academy, D. S. (2022). Deep learning book. *Acesso em: 18 Junho. 2022*. Disponível em: http://www.deeplearningbook.org.

Agarwala, N., Inoue, Y., e Sly, A. (2017). Music composition using recurrent neural networks. *CS 224n: Natural Language Processing with Deep Learning, Spring*, 1:1–10.

Aggarwal, C. C. et al. (2018). Neural networks and deep learning. *Springer*, 10:978–3.

Alpern, A. (1995). Techniques for algorithmic composition of music. *Hampshire College*, 95(1995):120. On the web: http://hamp.hampshire.edu/adaF92/algocomp/algocomp.

Bender, E. (2019). The #benderrule: On naming the languages we study and why it matters. *The Gradient*.

Cheng, B. e Titterington, D. M. (1994). Neural networks: A review from a statistical perspective. *Statistical science*, pages 2–30.

Colombo, F., Muscinelli, S. P., Seeholzer, A., Brea, J., e Gerstner, W. (2016). Algorithmic composition of melodies with deep recurrent neural networks. *arXiv preprint arXiv:1606.07251*.

de Souza, V. A. A. e de Avila, S. E. F. (2018). Deep neural networks for generating music.

Duchi, J., Hazan, E., e Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Fan, J., Ma, C., e Zhong, Y. (2021). A selective overview of deep learning. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 36(2):264.

Farley, E. e Pierotte, L. (2017). Web scraping: An emerging data collection method for criminal justice researchers. *Justice Research and Statistics Association*.

Fernández, J. D. e Vico, F. (2013). Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582.

Frei, S., Cao, Y., e Gu, Q. (2020). Agnostic learning of a single neuron with gradient descent. *Advances in Neural Information Processing Systems*, 33:5417–5428.

Geman, S., Bienenstock, E., e Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58.

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis lectures on human language technologies*, 10(1):1–309.

Goodfellow, I., Bengio, Y., e Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hagan, M., Demuth, H., Beale, M., e De Jesus, O. (2014). Neural network design. *Neural Networks in a Softcomputing Framework*.

Hair, J. F., Black, W. C., Babin, B. J., Anderson, R. E., e Tatham, R. L. (2005). *Análise multivariada de dados*. Bookman Editora, 5 edition.

Hastie, T., Tibshirani, R., Friedman, J. H., e Friedman, J. H. (2008). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Haykin, S. (2009). *Neural networks and learning machines*. Pearson Education India, 3 edition.

Hernandez-Olivan, C. e Beltran, J. R. (2021). Music composition with deep learning: A review. *arXiv preprint arXiv:2108.12290*.

Ji, S., Luo, J., e Yang, X. (2020). A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions. *arXiv preprint arXiv:2011.06801*.

Jurafsky, D. e Martin, J. H. (2021). Speech and language processing. *US: Prentice Hall*, 3.

Kamath, U., Liu, J., e Whitaker, J. (2019). *Deep learning for NLP and speech recognition*, volume 84. Springer.

Khder, M. A. (2021). Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing & Its Applications*, 13(3).

Kingma, D. P. e Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kuang, J. e Yang, T. (2021). Popular song composition based on deep learning and neural network. *Journal of Mathematics*, 2021.

Lawson, R. (2015). *Web scraping with Python*. Packt Publishing Ltd.

Manning, C. e Schutze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Maurer, J. (1999). A brief history of algorithmic composition. https://ccrma.stanford.edu/~blackrse/algorithm.html. Acessado em: 25 Junho. 2022.

McCullagh, P. e Nelder, J. A. (1989). Generalized linear models. chapman and hall. *London, UK*.

Nierhaus, G. (2009). *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media.

Patil, Y. e Patil, S. (2016). Review of web crawlers with specification and working. *International Journal of Advanced Research in Computer and Communication Engineering*, 5(1):220–223.

Raul, R. (1996). Neural networks: a systematic introduction. *Journal of The Springer Science & Business Media*.

Sarle, W. S. (1994). Neural networks and statistical models. *Citeseer*.

Sirisuriya, D. (2015). A comparative study on web scraping. *Proceedings of 8th International Research Conference*.

Tieleman, T. e Hinton, G. (2012). Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 6.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., e Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Walshaw, C. (1993). *ABC2MTEX: An easy way of transcribing folk and traditional music, Version 1.0*.

Walshaw, C. (2014). A statistical analysis of the abc music notation corpus: Exploring duplication.