Processamento de Linguagem Natural

Modelos Neurais em PLN

Marlo Souza[1]
[1]Universidade Federal da Bahia - Brasil

31 de julho de 2024

Sumário

Redes Neurais Artificiais
Aplicando RNA para PLN
Arquitetura Codificador-Decodificador (Encoder-Decoder)
Mecanismo de Atenção
Arquitetura Transformer

Redes Neurais Artificiais

O neurônio

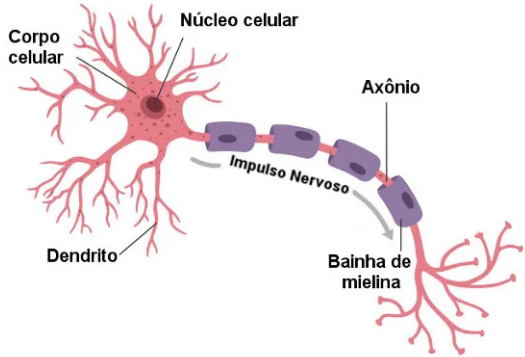


Figura: Representação simplificada de um neurônio

Um modelo matemático para o neurônio

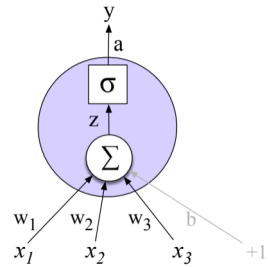


Figura: Um modelo matemático para o funcionamento do neurônio

Um modelo matemático para o neurônio

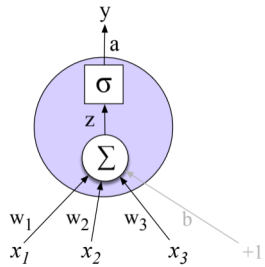


Figura: Um modelo matemático para o funcionamento do neurônio

$$y = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b)$$

Um modelo matemático para o neurônio

Podemos então compreender um neurônio como uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ tal que $y = f(x_1, \cdots, x_n) = f_1(\sum_i^n w_i x_i + b)$, com $f_1 : \mathbb{R} \to \mathbb{R}$.

Um modelo matemático para o neurônio

Podemos então compreender um neurônio como uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ tal que $y = f(x_1, \cdots, x_n) = f_1(\sum_i^n w_i x_i + b)$, com $f_1 : \mathbb{R} \to \mathbb{R}$.
f_1 é chamada então de função de ativação do neurônio e descreve a condição de disparada (o processamento realizado) por aquele neurônio.

Um modelo matemático para o neurônio

Podemos então compreender um neurônio como uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ tal que $y = f(x_1, \cdots, x_n) = f_1(\sum_i^n w_i x_i + b)$, com $f_1 : \mathbb{R} \to \mathbb{R}$.
f_1 é chamada então de função de ativação do neurônio e descreve a condição de disparada (o processamento realizado) por aquele neurônio.
Podemos também descrever o termo $z = \sum_i^n w_i x_i + b$ de forma vetorial como $z = w \cdot x + b$, com $w = \langle w_1, \cdots, w_n \rangle$, $x = \langle x_1, \cdots, x_n \rangle^T$.

Funções de ativação

As funções de ativação descrevem o processo de funcionamento de um neurônio, numa rede neural, ajudam a definir o viés de aprendizado das mesmas.

Funções de ativação

As funções de ativação descrevem o processo de funcionamento de um neurônio, numa rede neural, ajudam a definir o viés de aprendizado das mesmas.As propriedades dessas funções vão definir aspectos computacionais importantes, como a expressividade da rede e sua complexidade computacional.

Funções de ativação

As funções de ativação descrevem o processo de funcionamento de um neurônio, numa rede neural, ajudam a definir o viés de aprendizado das mesmas.As propriedades dessas funções vão definir aspectos computacionais importantes, como a expressividade da rede e sua complexidade computacional.
É necessário que a função de ativação escolhida não seja linear! (por quê?)

Funções de ativação

As funções de ativação descrevem o processo de funcionamento de um neurônio, numa rede neural, ajudam a definir o viés de aprendizado das mesmas.As propriedades dessas funções vão definir aspectos computacionais importantes, como a expressividade da rede e sua complexidade computacional.
É necessário que a função de ativação escolhida não seja linear! (por quê?)
Algumas funções comuns:

- Função sigmóide (σ)
- Função tangente hiperbólico ($tanh$)
- Função Linear Unitária Retificada ($ReLU$)

Funções de ativação: sigmóide

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Funções de ativação: sigmóide

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

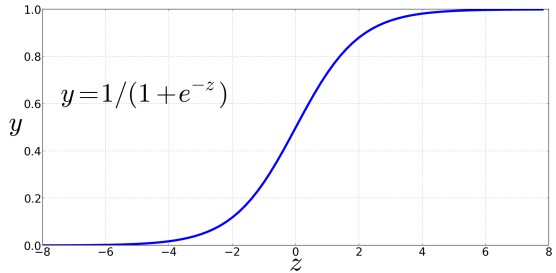


Figura: Gráfico da função sigmóide

Funções de ativação: tangente hiperbólico

$$y = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Funções de ativação: tangente hiperbólico

$$y = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

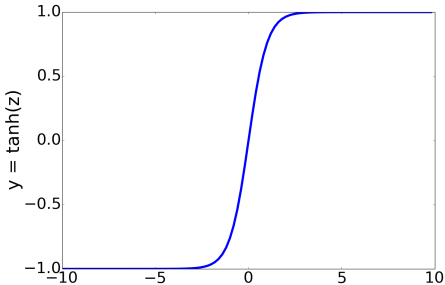


Figura: Gráfico da função tangente hiperbólico

Funções de ativação: ReLU

$$y = ReLU(z) = max(z, 0)$$

Funções de ativação: ReLU

$$y = ReLU(z) = max(z, 0)$$

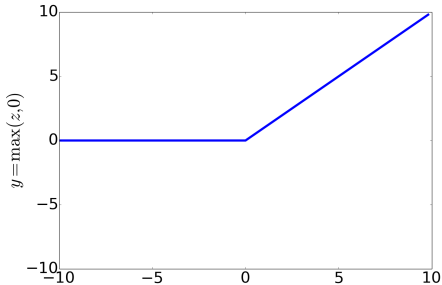


Figura: Gráfico da função ReLU

Redes neurais artificiais

11

Uma rede neural artificial é um grafo direcionado e rotulado em cada cada nó representa um neurônio (descrito por uma função de ativação) e cada aresta é rotulada com um peso de associação.

Redes neurais artificiais

Uma rede neural artificial é um grafo direcionado e rotulado em cada cada nó representa um neurônio (descrito por uma função de ativação) e cada aresta é rotulada com um peso de associação.

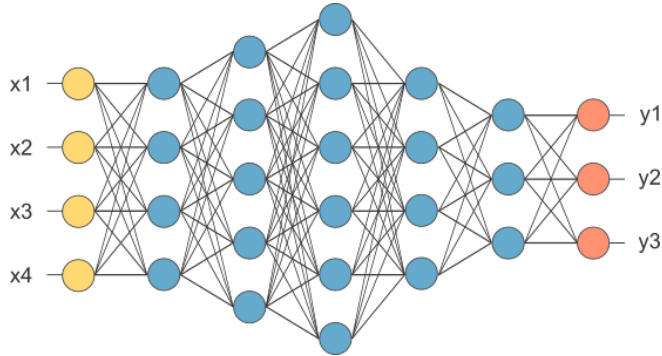


Figura: Representação de uma RNA

Redes Feedforward

12

Redes *feedforward*, redes densas, perceptron multicamadas, ou redes lineares, são RNA acíclicas com múltiplas camadas completamente conectadas.

Redes Feedforward

Redes *feedforward*, redes densas, perceptron multicamadas, ou redes lineares, são RNA acíclicas com múltiplas camadas completamente conectadas.
São o tipo mais básico de rede neural e, comumente, uma camada densa faz parte da maioria das arquiteturas empregadas na prática.

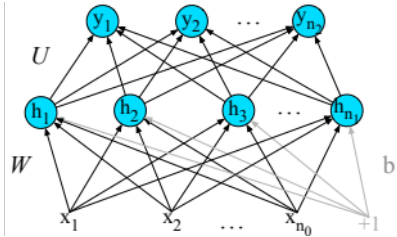


Figura: Rede Feedforward

Regressão Multinomial

13

Podemos implementar uma regressão multinomial usando redes neurais empregando funções softmax

Regressão Multinomial

Podemos implementar uma regressão multinomial usando redes neurais empregando funções softmax

$$y = \text{softmax}(Wx + b)$$

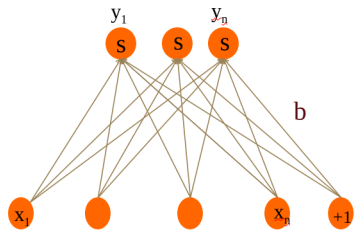


Figura: Camada Softmax

Regressão Multinomial

Podemos implementar uma regressão multinomial usando redes neurais empregando funções softmax

$$y = \text{softmax}(Wx + b)$$

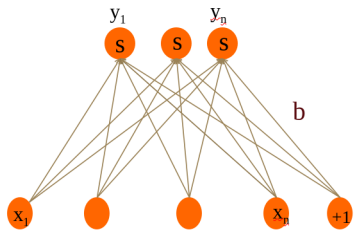


Figura: Camada Softmax

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

Aplicando RNA para PLN

Classificação de texto: Análise de sentimentos

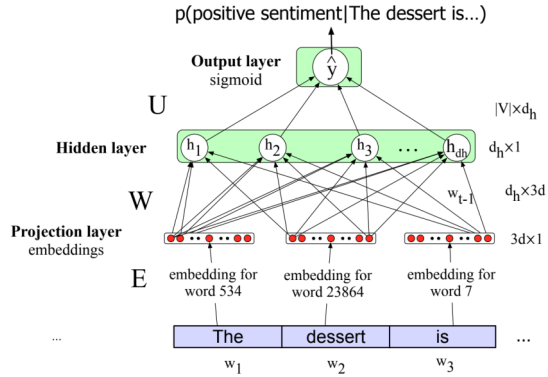
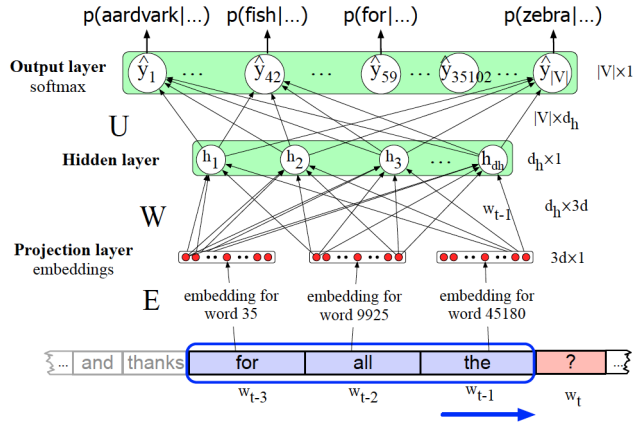


Figura: Uma rede neural para análise de sentimentos

Classificação de texto: Análise de sentimentos


```
class AS(nn.Module):
   def __init__(self):
      self.embeddings = nn.Embeddings(size_vocab,size_repr)
      self.hidden = nn.Linear(size_repr*number_words, number_neurons)
      self.hidden_act = nn.ReLu()
      self.out = nn.Linear(number_neurons,1)
      self.out_act = nn.Sigmoid()
   def forward(self,data)
      #data.shape == (1,n_words)
      embeddings = self.embeddings(data) #embeddings.shape ==(1,n_words,si
      embeddings = torch.cat(embeddings) #embeddings.shape ==(1,n_words*si
      z = self.hidden (embeddings) #z.shape == (1,number_neurons)
      z = self.hidden_act(z) #z.shape == (1,number_neurons)
      y = self.out # y.shape ==(1,1)
      y = self.out_act(y) #y.shape == (1,1)
      return y
```

Um modelo de linguagem neural



Vantagens sobre os modelos de n-gramas

Note que como o modelo aprende a partir das representações vetoriais de palavras (embeddings), não das sequências de palavras em si, o modelo pode explorar as similaridades entre palavras codificadas nelas para realizar suas predições.

Vantagens sobre os modelos de n-gramas

Note que como o modelo aprende a partir das representações vetoriais de palavras (embeddings), não das sequências de palavras em si, o modelo pode explorar as similaridades entre palavras codificadas nelas para realizar suas predições.

Assim, mesmo que uma sequência não tenha sido vista explicitamente nos dados de treino, o modelo pode avaliar sua probabilidade com base nessas similaridades.

Vantagens sobre os modelos de n-gramas

Note que como o modelo aprende a partir das representações vetoriais de palavras (embeddings), não das sequências de palavras em si, o modelo pode explorar as similaridades entre palavras codificadas nelas para realizar suas predições.

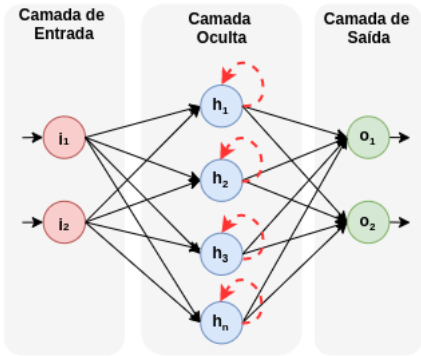
Assim, mesmo que uma sequência não tenha sido vista explicitamente nos dados de treino, o modelo pode avaliar sua probabilidade com base nessas similaridades.

- Tenho que lembrar de alimentar o gato hoje a noite (visto nos dados)
- Tenho que lembrar de alimentar o cachorro (não visto)

Redes Recorrentes e Sequências

Redes Neurais Recorrentes

Redes Recorrentes são aqueles em que acontecem ciclos na rede, i.e. em que a saída de um neurônio (ou camada) retroalimenta ele(a) mesmo(a).



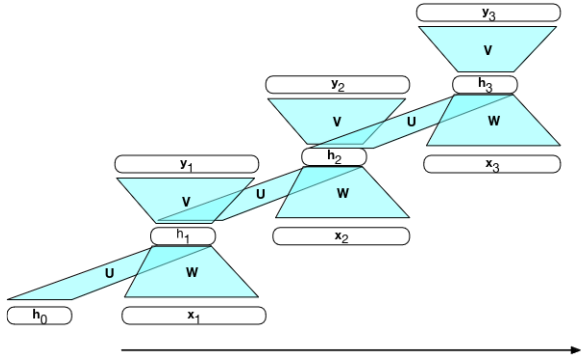
Rede Neural Recorrente

Redes Neurais Recorrentes

os enlaces recorrentes numa RNN funcionam como um mecanismo de memória, que mantém uma noção de estado na rede.

Redes Neurais Recorrentes

os enlaces recorrentes numa RNN funcionam como um mecanismo de memória, que mantém uma noção de estado na rede.A execução de uma RNN é implementada através da expansão iterada da rede a partir da saída da execução anterior.



treinamento de uma RNA

Treinando uma rede neural

O treinamento de uma rede neural se dá num processo iterativo sobre os dados de treinamento em duas etapas:

- Passo de programação direta (forward propagation)
- Passo de propagação retrograda (backward propagation)

Treinando uma rede neural

O treinamento de uma rede neural se dá num processo iterativo sobre os dados de treinamento em duas etapas:

- Passo de programação direta (forward propagation)
- Passo de propagação retrograda (backward propagation)

É necessário definir alguns ingredientes da metodologia de treino:

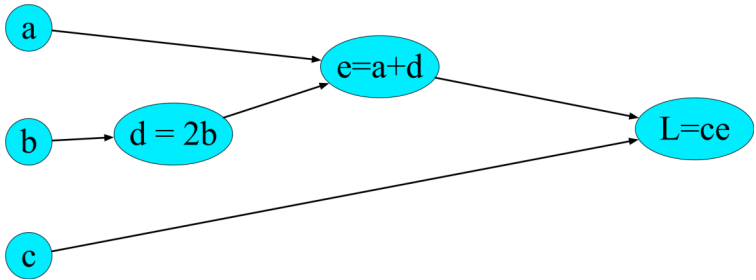
- Quantidade de épocas de treino, i.e. quantas vezes a rede processará o conjunto de treino para aprender
- A função de perda (*loss*), que representa a computação do erro cometido pela rede em seu processamento

Treinando uma rede neural: grafo de computação

Um grafo de computação determina uma ordenação topológica das operações necessárias a serem realizadas para se obter um determinado resultado.

Treinando uma rede neural: grafo de computação

Um grafo de computação determina uma ordenação topológica das operações necessárias a serem realizadas para se obter um determinado resultado.

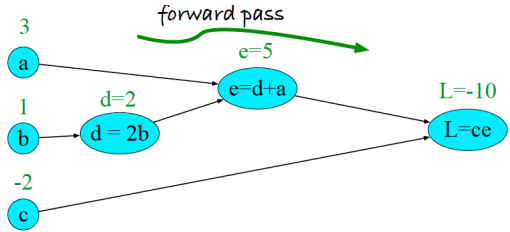


Treinando uma rede neural: propagação direta

Na propagação direta, as ações do grafo de computação são executadas em ordem, i.e. uma operação é executada somente quando todas as operações das quais ela depende já o foram.

Treinando uma rede neural: propagação direta

Na propagação direta, as ações do grafo de computação são executadas em ordem, i.e. uma operação é executada somente quando todas as operações das quais ela depende já o foram.



Treinando uma rede neural: cálculo do erro

Ao final da execução, a função de perda $L(y, \hat{y})$ é aplicada sobre o resultado encontrado e o resultado conhecido (rótulo).

Treinando uma rede neural: cálculo do erro

Ao final da execução, a função de perda $L(y, \hat{y})$ é aplicada sobre o resultado encontrado e o resultado conhecido (rótulo).
Para ajustar os pesos da rede neural, precisamos entender a contribuição de cada componente da rede ao erro obtido. Fazemos isso computando a derivada parcial de cada peso para a perda, i.e.

$$\frac{\partial L(y, \hat{y})}{\partial w_i}$$

Treinando uma rede neural: cálculo do erro

Ao final da execução, a função de perda $L(y, \hat{y})$ é aplicada sobre o resultado encontrado e o resultado conhecido (rótulo).
Para ajustar os pesos da rede neural, precisamos entender a contribuição de cada componente da rede ao erro obtido. Fazemos isso computando a derivada parcial de cada peso para a perda, i.e.

$$\frac{\partial L(y, \hat{y})}{\partial w_i}$$

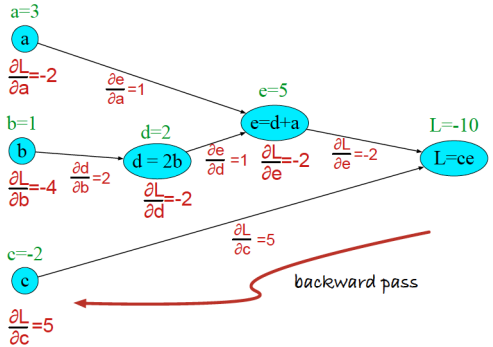
Tal valor será computado de forma numérica e propagado para trás usando a regra da cadeia $\frac{df}{dx} = \frac{df}{du} \cdot \frac{du}{dx}$

Treinando uma rede neural: propagação retrograda

Na propagação retrograda, calculamos as iterativamente as derivadas parciais do erro em relação aos pesos e atualizamos os pesos da rede, propagando para trás as mudanças.

Treinando uma rede neural: propagação retrograda

Na propagação retrograda, calculamos as iterativamente as derivadas parciais do erro em relação aos pesos e atualizamos os pesos da rede, propagando para trás as mudanças.



Aplicação: Geração de texto

Vamos para o Colab!