Texto e Word Vectors

Eduardo Adame

Redes Neurais

5 de novembro de 2025

Motivação

- Demonstramos como usar Redes Neurais com dados numéricos estruturados
- Imagens podem ser redimensionadas para um tamanho específico
- Valores de imagem são números (escala de cinza, RGB)
- Mas como trabalhamos com texto?
- **Problema 1:** Como lidar com sequências de texto (sequências de palavras com comprimento variável)?
- **Problema 2:** Como converter palavras em algo numérico?

Problema: Sequências de Comprimento Variável

- Com imagens, forçamos dimensões de entrada específicas
- Não é óbvio como fazer isso com texto
- Usaremos uma nova estrutura de rede chamada *"Rede Neural Recorrente" (RNN)* que será discutida na próxima aula
- Hoje focamos em: **como representar palavras numericamente**

Tokenização

- Necessário converter palavra em algo numérico
- **Primeira abordagem:** Tokenização
- Tratar como variável categórica com enorme número de categorias (codificação one-hot)
- Lidar com detalhes de maiúsculas, pontuação, etc.

“O gato de botas.”

↓

['o', 'gato', 'de', 'botas', '.', '<EOS>']

Tokenização - Construindo o Vocabulário

- Usar tokens para construir vocabulário
- Vocabulário é um mapeamento um-para-um de índice para token
- Geralmente representado por lista e dicionário

índice → palavra

```
['<EOS>',
 'o',
 'gato',
 'de',
 'botas',
 '.']
```

palavra → índice

```
{'<EOS>': 0,
 'o': 1,
 'gato': 2,
 'de': 3,
 'botas': 4,
 '.': 5}
```

Problemas com Tokenização

- Tokenização perde muita informação sobre palavras:
  - Classe gramatical (substantivo, verbo, etc.)
  - **Sinonímia** (palavras distintas com significado igual ou similar)
  - **Polissemia** (palavra única com múltiplos significados)
  - Contexto geral em que a palavra provavelmente aparece (ex.: "desemprego" e "inflação" são ambas sobre economia)
- Aumentar tamanho do vocabulário é difícil (requereria re-treinar o modelo)
- Comprimento do vetor é enorme $\rightarrow$ grande número de pesos
- Ainda assim, informação no vetor é muito **esparsa**

Vetores de Palavras (Word Embeddings)

- **Objetivo:** representar palavra por vetor m-dimensional (para m de tamanho médio, digamos, $m = 300$)
- Ter palavras "similares" representadas por vetores "próximos" neste espaço m-dimensional
- Palavras em domínio particular (economia, ciência, esportes) poderiam estar mais próximas entre si
- Poderia ajudar com sinonímia
  - ex.: "grande" e "enorme" têm vetores próximos
- Poderia ajudar com polissemia
  - "manga" (fruta) e "laranja" próximas em algumas dimensões
  - "manga" (roupa) e "bolso" próximas em outras dimensões

Vetores de Palavras - Vantagens

- Vetores seriam mais curtos e densos em informação, ao invés de muito longos e esparsos
- Requeririam menos pesos e parâmetros
- **Felizmente:** existem mapeamentos pré-treinados que podem ser baixados e utilizados
- Foram treinados em grandes corpora por muito tempo
- Vamos entender como foram desenvolvidos e treinados

O que Torna Duas Palavras Similares?

- **Ideia:** palavras similares ocorrem em contextos similares
- Para uma palavra dada, olhar palavras em uma "janela" ao redor dela
- Considerar tentar prever uma palavra dado o contexto
- Este é exatamente o modelo **CBOW** (Continuous Bag of Words)

"Todos são iguais perante a lei, sem distinção de qualquer natureza"

↓

(['iguais', 'perante', 'a', 'sem', 'distinção', 'de'], 'lei')

contexto palavra-alvo

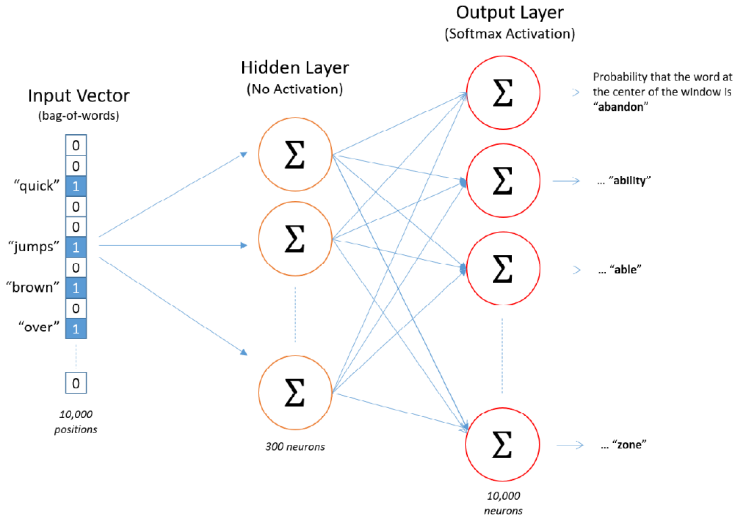
Modelo CBOW

Treinar rede neural em grande corpus de dados.

- Palavras de contexto (one-hot encoded)
- Camada oculta única (dimensão m)
- Palavra alvo (one-hot encoded)

Modelo CBOW - Extraindo Embeddings

Uma vez que a rede é treinada, os pesos → vetores de palavras.

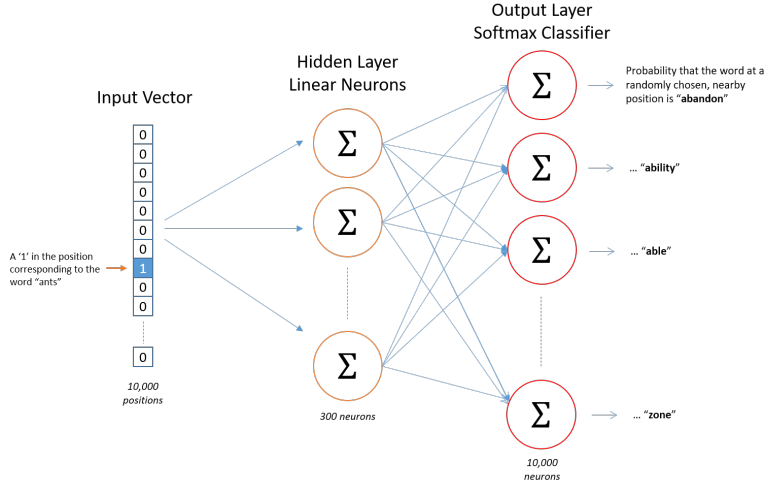


Modelo CBOW - Extraindo Embeddings

- A matriz de pesos entre entrada e camada oculta representa os word embeddings
- Cada linha corresponde ao vetor de uma palavra
- Dimensão: vocabulário $\times m$

Modelo Skip-gram

Mesma ideia, exceto que predizemos o contexto a partir do alvo.



Modelo CBOW - Extraindo Embeddings

- Palavra alvo (one-hot encoded)
- Camada oculta de Rede Neural
- Palavras de contexto (one-hot encoded)
- Geralmente funciona melhor que CBOW

Word2Vec (2013)

- *Distributed Representations of Words and Phrases and Their Compositionality*, Mikolov et al.
- Usa modelo Skip-gram para treinar em grande corpus
- Muitos detalhes para fazê-lo funcionar melhor:
  - Agregação de frases multi-palavras (ex.: "São Paulo")
  - Subamostragem (sobreamostrar palavras menos comuns)
  - Amostragem negativa (dar exemplos de palavras erradas à rede)
- Implementações disponíveis em `gensim`, `spaCy`
- Modelos pré-treinados disponíveis online

Propriedades Interessantes - Analogias

- Word2Vec captura relações semânticas e sintáticas
- Exemplo clássico de analogia:

$$\textbf{rei} - \text{homem} + \text{mulher} \approx \textbf{rainha}$$

$$\textbf{Paris} - \text{França} + \text{Brasil} \approx \textbf{Brasília}$$

- Operações vetoriais capturam relações conceituais
- Útil para tarefas de reasoning e NLP

GloVe (2014)

- **Global Vectors for Word Representation** (GloVe)
- Usa matriz de co-ocorrência com palavras vizinhas para determinar similaridade

$$J = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log(P_{ij}))^2$$

- $f \rightarrow$ frequência de palavra, com limite máximo
- $P_{ij} \rightarrow$ probabilidade das palavras i e j ocorrerem juntas
- Abordagem mais estatística que Word2Vec

GloVe - Disponibilidade

- GloVe é publicamente disponível
- Desenvolvido em Stanford: `https://nlp.stanford.edu/projects/glove/`
- Treinado em enormes corpora:
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab)
  - Common Crawl (42B tokens, 1.9M vocab)
  - Common Crawl (840B tokens, 2.2M vocab)
- Diferentes dimensionalidades disponíveis (50, 100, 200, 300)
- Amplamente usado em pesquisa e indústria

Comparação: Word2Vec vs GloVe

Aspecto	**Word2Vec**	**GloVe**
Abordagem	Preditiva (NN)	Baseada em contagem
Contexto	Local (janela)	Global (corpus todo)
Treinamento	Mais rápido	Mais lento
Memória	Menor	Maior (matriz)
Performance	Similar	Similar

- Na prática, performance similar na maioria das tarefas
- Escolha depende de recursos computacionais e corpus

Limitações dos Word Embeddings Estáticos

- **Problema principal:** uma palavra = um vetor único
- Não capturam contexto dinâmico:
  - "banco" em "banco de dados" vs "banco do parque"
  - Polissemia não é totalmente resolvida
- Não lidam bem com:
  - Palavras fora do vocabulário (OOV)
  - Morfologia e palavras compostas
  - Mudanças de significado ao longo do tempo
- Solução: **Embeddings Contextuais** (próximo tópico)

Evolução: Embeddings Contextuais (2018-2025)

- **ELMo** (2018): Embeddings de modelos de linguagem bidirecionais
- **BERT** (2018): Transformers bidirecionais
- **GPT** (2018-2023): Modelos autoregressivos
- **2025:** Modelos ainda mais sofisticados
  - Embeddings adaptam-se ao contexto da frase
  - "banco" terá representações diferentes dependendo do uso
  - Base para LLMs modernos (ChatGPT, Claude, etc.)
- **Diferença chave:** representação depende do contexto completo

Aplicações Práticas de Word Embeddings

1. **Análise de Sentimento**
   - Classificar opiniões como positivas/negativas
2. **Sistemas de Recomendação**
   - Encontrar produtos/conteúdos similares
3. **Tradução Automática**
   - Representar palavras em diferentes idiomas
4. **Busca Semântica**
   - Encontrar documentos por significado, não apenas palavras-chave
5. **Chatbots e Assistentes Virtuais**
   - Entender intenção do usuário

Avaliação de Word Embeddings

- **Avaliação Intrínseca:**
  - Testes de analogia (rei - homem + mulher = ?)
  - Tarefas de similaridade de palavras
  - Correlação com julgamentos humanos
- **Avaliação Extrínseca:**
  - Performance em tarefas downstream
  - Classificação de texto
  - Named Entity Recognition (NER)
  - Part-of-Speech (POS) tagging
- Benchmarks comuns: WordSim-353, SimLex-999, Google Analogy

Implementação Prática - Python

- **Bibliotecas principais:**
  - `gensim`: Word2Vec, FastText
  - `spaCy`: Embeddings pré-treinados integrados
  - `transformers` (HuggingFace): BERT, GPT
  - `tensorflow/keras`: Embedding layers
- **Carregar embeddings pré-treinados:**

```python
from gensim.models import KeyedVectors
# Carregar GloVe
model = KeyedVectors.load_word2vec_format(
    'glove.6B.300d.txt', binary=False)
# Similaridade
model.similarity('gato', 'cachorro')
```

Demonstração Interativa

- **TensorFlow Embedding Projector**
  - `https://projector.tensorflow.org`
  - Visualização 3D interativa de embeddings
  - Busca por vizinhos mais próximos
  - Visualização de relações analógicas
- **Experimente:**
  - Busque “king” e veja palavras similares
  - Use PCA ou t-SNE para redução dimensional
  - Explore diferentes datasets (Word2Vec, GloVe)

Recursos Adicionais

- **Papers Fundamentais:**
  - Mikolov et al. (2013) - Word2Vec
  - Pennington et al. (2014) - GloVe
  - Peters et al. (2018) - ELMo
- **Tutoriais e Cursos:**
  - Stanford CS224N: NLP with Deep Learning
  - Fast.ai: Practical Deep Learning for Coders
- **Datasets e Modelos:**
  - `https://nlp.stanford.edu/projects/glove/`
  - `https://code.google.com/archive/p/word2vec/`
  - HuggingFace Model Hub

Próxima Aula

- **Redes Neurais Recorrentes (RNN)**
  - Como processar sequências de comprimento variável
  - LSTM e GRU
  - Aplicações em processamento de texto
- **Preparação:**
  - Revisar conceitos de word embeddings
  - Familiarizar-se com notação de sequências
  - Instalar bibliotecas necessárias

Obrigado!

Dúvidas?