Principais contribuições do artigo “A neural probabilistic language model”, de Bengio et al

Aluno: Leandro Carísio Fernandes

* Modelo de linguagem estatístico
* A probabilidade de uma sequência (por exemplo, ) é igual a “probabilidade de ocorrer a palavra 1” multiplicada pela “probabilidade de ocorrer a palavra 2 dado que a primeira palavra ocorreu” multiplicada pela “probabilidade de ocorrer a palavra 3 dado que as duas primeiras palavras ocorreram”...
* Primeira aproximação: n-gramas -> a probabilidade é aproximadamente igual àquela considerando apenas as n-1 últimas palavras
* A quantidade de parâmetros livres para um vocabulário de tamanho V é igual a Vn – 1. Por exemplo, com V=10.000 e n=10 -> ~1040
* A proposta de resolver esse problema é:
  + Representar cada palavra do vocabulário como vetores m dimensionais (por exemplo, m=30)
  + Considerar que a entrada é uma sequência de tamanho n-1 e tentar prever a n’ésima palavra. Assim, a sequência já é conhecida no treinamento
  + Da forma como é feita, a expectativa é que os vetores generalizem para terem papéis sintáticos e semânticos semelhantes (por exemplo, quarto/sala devem estar mais próximos entre si do que um artigo)
  + A novidade do paper é essa: “learned distributed feature vector”
* O cálculo dos log-probs é feito assim:
* Para implementar no PyTorch, a entrada é uma sequência de (n-1) palavras.
* Primeiro pegamos a sequência de palavras e passamos para uma camada nn.Embedding(vocab\_size, m), onde m é o tamanho do vetor de embedding
* Em seguida, passamos por uma camada nn.Linear para o cálculo de d + Hx. Tem entrada de tamanho (n-1)\*m e saída de tamanho h
* A saída do passo anterior é entrada para um nn.Tanh() ou nn.ReLU()
* Nesse ponto temos a opção de fazer W = 0 e usar a saída anterior como entrada para uma camada nn.Linear com entrada de tamanho h e saída de tamanho vocab\_size. Isso vai calcular b + Utanh(d+Hx)
* Se W != 0, precisamos de duas camadas pra terminar o cálculo. Uma nn.Linear de entrada (n-1)\*m e saída vocab\_size para calcular b + Wx e outra nn.Linear, sem bias, de entrada de tamanho e saída de tamanho vocab\_size para calcular Utanh(d+Hx)
* Contribuições: 1) durante o treino a rede aprende o probabilidade das palavras do modelo de linguagem (p(w) ~ y) ao mesmo tempo em que treina os embeddings; 2) proposta de treino em paralelo usando CPU; 3) facilidade em considerar UNK tokens.