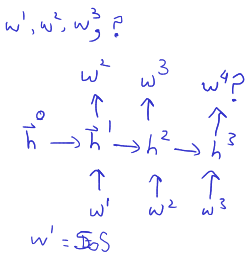
Language Modelling

(1) Text Prediction — Characters / Words

Given words $w^1, w^2, \dots, w^t$, predict next word w^{t+1}.



Input

- Consider the English language. Let English vocabulary V have K words.
- Each word can be represented by a K-dimensional vector with 1-of-K coding.
  - Tremendously inefficient. But how else can we convert a word into a vector of numbers that we can input into a neural network?
  - Each word vector is orthogonal to every other word vector. This is not realistic at all since words have similarities.

[0,1] [1,0]

Workaround: Project word vectors onto a lower-dimensional space via a projection/embedding matrix E of size $D \times K$ where $D < K$.

Matrix E can be learned.

Output

- Output layer = softmax on K neurons.
- $y_k^{t+1} = p(V_k \mid w^1, w^2, \dots, w^t)$
  = prob. of k-th word at time $t+1$ given previous words

$$\vec{y}^{\,t+1} = \begin{bmatrix} p(V_1 \mid w^1, \dots, w^t) \\ p(V_2 \mid w^1, \dots, w^t) \\ \vdots \\ p(V_K \mid w^1, \dots, w^t) \end{bmatrix} \quad K \times 1$$

$\vec{y}^{\,t+1}$: prob. distribution

Loss

Cross-entropy between output words and target words.

$$L\left(\underbrace{(\vec{d}^{\,1}, \vec{d}^{\,2}, \vec{d}^{\,3}, \dots, \vec{d}^{\,T})}_{\text{1-of-K coded vectors}}, \underbrace{(\vec{y}^{\,1}, \vec{y}^{\,2}, \vec{y}^{\,3}, \dots, \vec{y}^{\,T})}_{\text{prob. distribution vectors}}\right) = -\sum_{t=1}^{T}\sum_{k=1}^{K} \underbrace{d_k^t}_{\substack{1 \text{ for desired word at time } t \\ 0 \text{ for the rest.}}} \ln y_k^t = -\sum_{t=1}^{T} \ln y^t$$

y^t: index corresponding to w^{t+1}

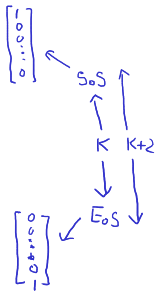
Training

Bprop using millions of sentences. Each sentence is 1 traing sample.

Sentence Language Generation

- Next word w^{t+1} is drawn from prob. dist. $\vec{y}^{\,t}$
- Feed w^{t+1} as next input and find $\vec{y}^{\,t+1}$.
- Repeat until ending symbol is drawn.

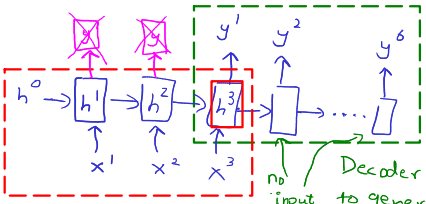
1-of-$(K+2)$ coding.



Language Translation

SoS Zaid slapped Khalid EoS → EoS زید نے خالد کو تھپڑ مارا SoS

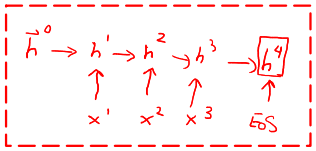
x^0 x^1 x^2 x^3 x^4 — y^7 y^6 y^5 y^4 y^3 y^2 y^1 y^0



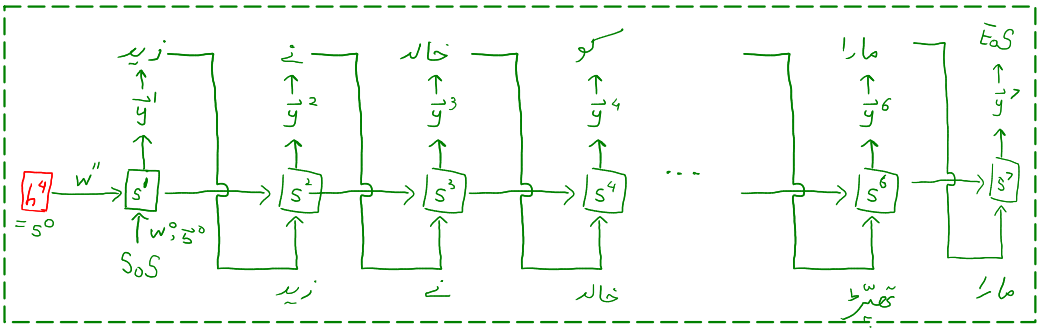
Encoder that produces $\vec{h}^3$ as the encoding of the whole input sentence (sequence of words)

Decoder that uses encoding $\vec{h}^3$ as input vector to generate the output sentence as a sequence of words.

A better decoder: make prob. dist $\vec{y}^{t+1}$ depend on word drawn from $\vec{y}^t$ as well.



Encoder



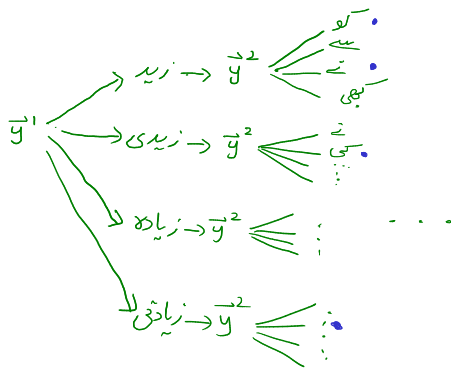
Decoder

Draw a sample from $\vec{y}^t$.

① Argmax $\vec{y}^t$

② Top-K ($\vec{y}^t$).

$$y_j^t = P(O^t = V_j \mid \underbrace{O^{t-1}, O^{t-2}, \ldots O^1}_{\text{all words output so far}}, \underbrace{I^1, I^2, \ldots, I^N}_{\text{all input words}})$$



Beam Search

- Terminate each branch when it produces EoS word.
- We let all Top-K branches continue until termination.
- From all the terminated branches pick the one with the highest overall probability.

Top-K $P(O^2 O^1 \mid I^1, I^2, \ldots, I^N) = \underbrace{P(O^2 \mid O^1, I^1, I^2, \ldots, I^N)}_{y^2_{O^2 \text{index}}} \underbrace{P(O^1 \mid I^1, I^2, \ldots, I^N)}_{y^1_{O^1 \text{index}}}$

Product rule: $P(AB \mid C) = P(A \mid BC)\, P(B \mid C)$

→ At time 2, this will give me overall probabilities of 16 branches.

→ From those 16 branches pick the top-4.

K