<https://pakalguksu.github.io/2020/03/01/CS224n-6%EA%B0%95-Language-Models-and-Recurrent-Neural-Networks/>

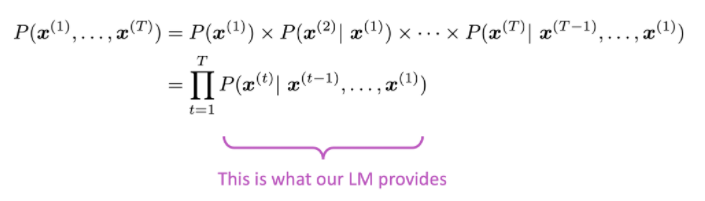
<https://nhlmary3.tistory.com/entry/RNN>

<https://aikorea.org/blog/rnn-tutorial-3/>

Language Modeling 은 다음에 어떤 단어가 올지 predict 하는 task이다.

즉, 단어들 x1, x2, .. , xt가 주어졌을 때 xt+1의 단어의 확률 분포를 계산하는 것이다.(확률 분포 != 확률 -> #가 올 확률 = ~ / \*가 올 확률 = ~ => 따라서 총 vocab V의 단어 수만큼 확률 분포를 계산할 수 있다.

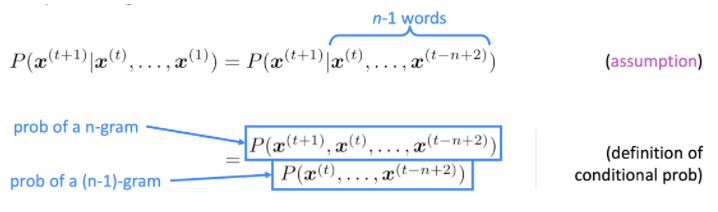
또한, Language Modeling을 통해 특정 문장에 확률을 할당할 수 있다. (여기서는 x1, x2 … xt 이 text) => 특정 문장에 확률을 부여할 수 있다는 것에 초점!



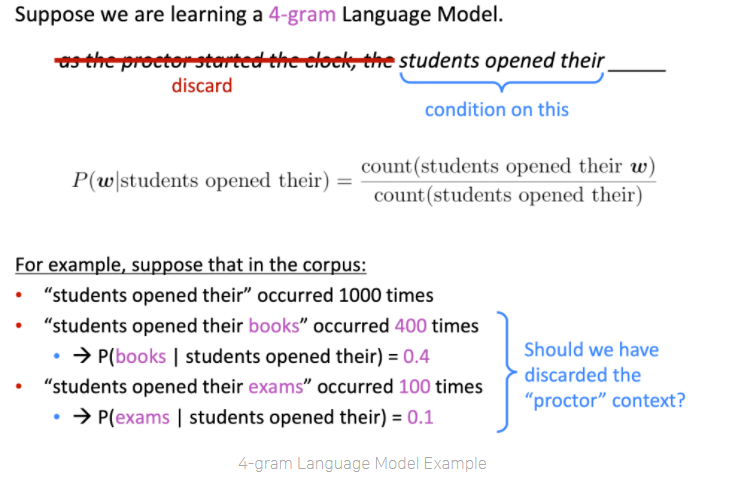
좌변 = 특정 문장의 확률 / 우변 = 조건부 확률 ?? 이거 다시 보기

N-gram Language Models

n-gram: 연속된 n개의 단어들을 의미(단어의 연속 = 문장 이니 문장으로 봐도 됨) / n-gram의 출현 빈도를 세어 다음 단어 예측에 사용



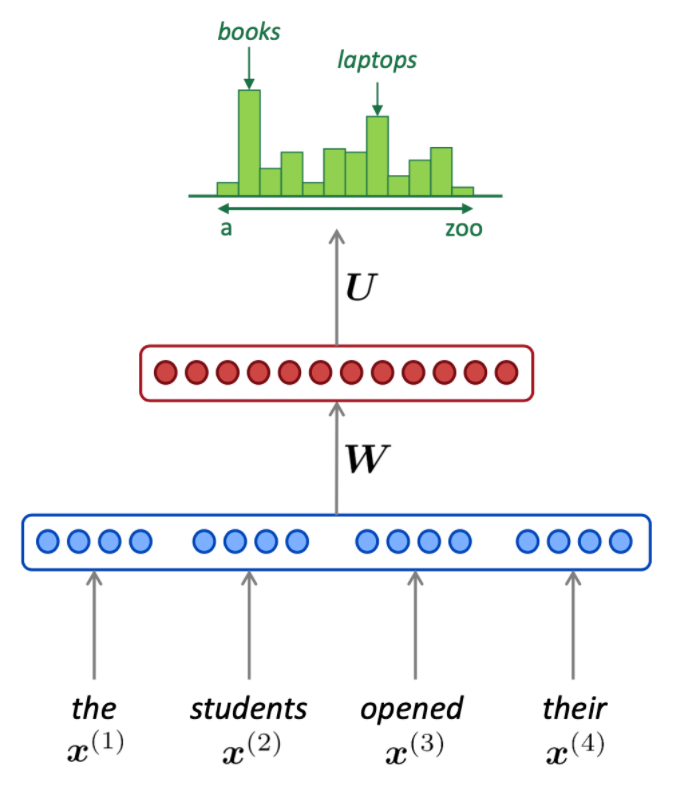
* 위에서 정의한 “문장에 확률을 할당할 수 있다” 라는 것을 통해 n-gram 문장이 나타날 확률과 (n-1)-gram 문장이 나타날 확률을 이용하면 => 현재 문장이 주어졌을 때 다음 단어가 올 확률을 계산할 수 있다. (n-gram 문장은 예측 단어까지 포함한다.)
* 단 n-gram 문장이 나타날 확률은 큰 corpus(text 여러 개들)에서 출현 빈도를 센다.



이렇게 통계적인 방법으로 다음 단어를 예측하고는 했다.(딥러닝 이전)

N-gram Problems

* Sparsity Problem: 특정 n-gram이 나타날 확률이 너무 낮을 때 위의 “students opened their”이라는 문장이 아예 안나타남 => 분모가 0 => 계산 불가 / “students opened their w”가 안나타남 => 확률이 0이라 계산하는, 가져오는 의미가 없다. <n-gram 문장 count를 인위적으로 늘린다 – smoothing / students opened their 대신 opened their를 센다> -> but 근본적인 문제 해결은 안됨.
* Storage Problem: Model은 count를 기반으로 해서 n-gram count 정보를 저장하는데 크기, 소모가 있다.
* Incoherence Problem: n-1개의 단어에만 영향을 받아 가정해서 이전 문맥은 반영하지 않아 놓치는 것이 있을 수 있다. => N의 크기를 늘리면 되지만 이렇게 되면 Sparsity, Storage 문제가 더 심해짐

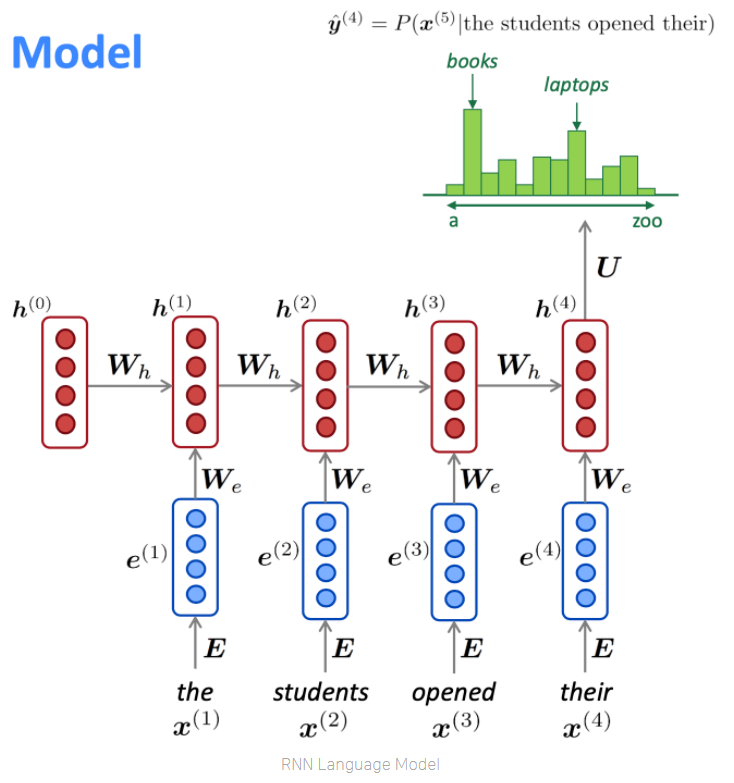


위는 가장 간단한 neural net 모델로 NNLM과 비슷하다고 생각하면 된다.

적절히 임베딩 한 후 forward 한다. 하지만 Fixed-window neural language model로 단어의 개수(input)이 한정적이다. Sparsity와 storage는 해결이 되지만 문제가 있다. (아예 계산 방법, 메커니즘이 달라서 자동적으로 두 문제가 해결된다.)

Window를 크게하는 데에는 한계가 있으며 전체적인 문맥을 읽을 수 없다. 또한 x1에 곱해지는 weight와 x2, …, x4에 곱해지는 weight가 다르다. => symmetry가 없다. = neural model이 비슷한 내용을 여러 번 학습한다.(weight 공유 못하고 비슷한 weight 계속 학습) = 학습 비효율성 => 따라서 Recurrent Neural Net이 등장한다.

RNN은 해당 단계의 input과 직전 단계의 hidden state에 적절한 weight를 곱해 계산, 또한 가장 중요한 특징은 weight를 공유한다는 것(symmetric)이다.

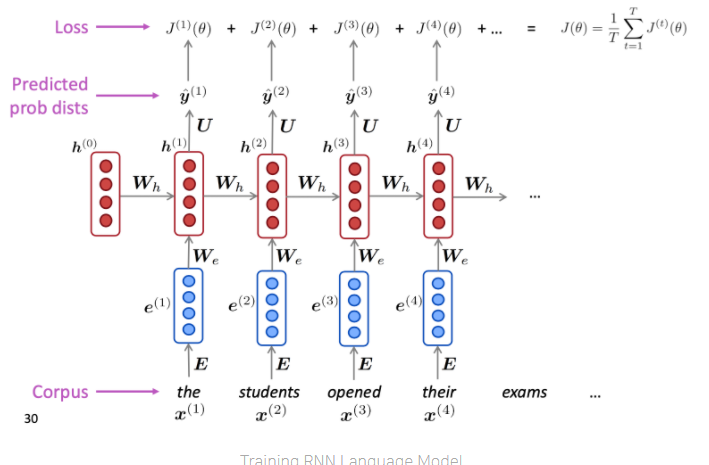


* Input length 상관없이 다음 단어 예측
* 이전 정보가 전해지므로 먼 곳의 context(먼 곳의 단어) 도 고려할 수 있다.
* Input이 길어져도 weight는 공유하기 때문에 model size는 증가하지 않으며 symmetric
* 전 단계의 계산이 완료돼야 하니 계산이 병렬적이지 않다. = 느림
* 또한, 이론적으로 먼 곳의 단어를 반영할 수는 있지만 vanishing gradient problem => context 반영되지 않는다. (후에 보완점 제안)

**Training RNN Language Model**

1. RNN model에 각 단어를 input으로 주고 모든 단계에서 다음 단어를 계산해 예측(파라미터로 곱해서)
2. 예측 단어 vs 실제 단어 차이의 cross-entropy로 loss
3. 모든 단계의 loss의 평균을 통해 전체 loss를 구함 (각 loss는 파라미터로 표현되기 때문에 gradient descent가 가능하다.)

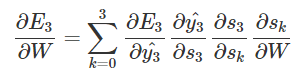
다만 단어 단위로 하면 계산이 많이 필요 => 문장이나 문서 단위로! => 첫 입력으로 ‘the’ 대신 ‘the students opened their exams’로 주는 것 / 또는 SGD로 optimize해도 좋다.



**RNN backprop**(다시) = BPTT = 각 시점에서의 에러들을 통한 gradient의 합을 최종 gradient로 여겨 공유되는 파라미터를 update 한다.

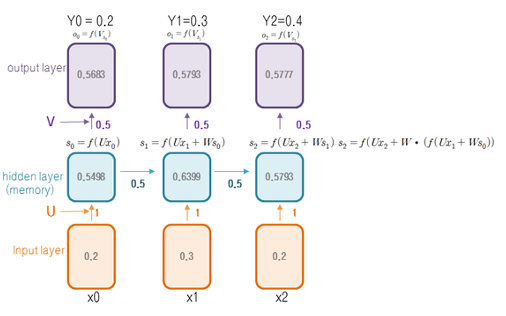
(U와 V에 대해서도 진행하고 업데이트 한다.

근데 또 각 Et에서 W에 대한, (U에 대한) 미분을 구할 때 이전 step에서의 파라미터가 걸리니 이걸 chain rule의 합으로 gradient를 표현할 수 있다.



위의 E3의 경우 s3가 s2, W에 걸리고 / s2는 W와 s1에 걸리니 연쇄적으로 해서 풀어보니 저렇게 복잡한 급수가 나온다.

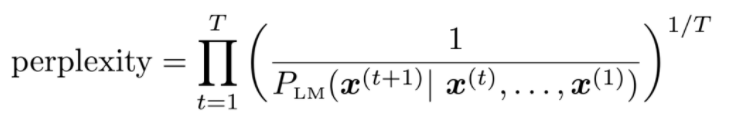
일단 전체 Error를 구한다. (MSE든 Cross Entropy Loss든)(위에 처럼) ?? 근데 어차피 자주 필요한건 time step 마다의 error 아닌가.



**Perplexity(다시)**

LM 성능 측정 척도이다. Perplexity가 낮을수록 좋은 language model.

LM을 통해 예측한 corpus(단어? 문장?)의 inverse를 corpus 길이로 normalize 해준 값 = cross-entropy의 exponential 값

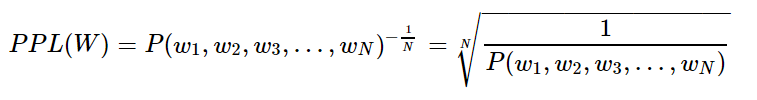


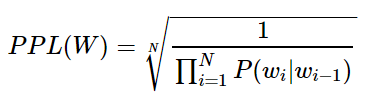
Extrinsic evaluation: 추가적 task로 평가 => 추가적 task를 좀 더 평가해야 되고 오래 걸림 => 모델 수 더 많아진다면 최악

intrinsic evaluation: 그 기능을 순수히 평가 / perplexity도 그 중 하나

perplexity는 낮을수록 좋고, PP 또는 PPL이라고도 한다.

문장의 확률을 최대화 = PPL을 최소화 / PPL은 단어의 수로 정규화 된 테스트 데이터에 대한 확률의 역수이다. / 문장 W의 길이가 N이라고 할 때



참고로 이식은 엄청 위쪽의 하늘색 highlight 참조  
이면 bi-gram

다음번에 RNN Problem + truncated RNN(추가자료) + LSTM 등등 다루기