Ppt2

Gradient problem이란 RNN backpropagation 시 gradient가 너무 작아지거나 반대로 너무 커져서 학습이 제대로 이루어지지 않는 문제입니다. 수식을 살펴보면 저번주에 보았듯이 vanilla RNN 셀 t번째 시점의 hidden state 수식이 이거였고 t번째 hidden state에 대한 t-1번째 hidden state의 gradient는 h\_t를 h\_t-1에 대해 미분하면 되므로 저렇게 됩니다. Chain rule을 이용하면 i번째 시점에서의 손실 J\_i에 대한 h\_1의 gradient는 이렇게 되고 미분 식마다 W\_h가 나오므로 W\_h가 i-j만큼 곱해집니다.

Ppt3

Norm 성질에 의해 방금의 식이 이렇게 표현되고 결국 W\_h matrix의 L2 norm이 W\_h의 가장 큰 고유값이 되고 그 말은 i번째 시점에서의 손실에 대한 hidden state의 gradient의 L2 norm은 절대적으로 W\_h의 L2 norm 크기에 달려있다는 뜻입니다. 만약 W\_h의 가장 큰 고유값이 1보다 작다면 위의 식에서 보다시피 W\_h가 계속 곱해지게 되는데 그러다보면 gradient가 빠르게 사라지게 됩니다. 이 때문에 발생하는 문제가 vanishing gradient problem입니다.

Ppt4

Vanishing gradient가 문제가 되는 이유는 weight가 가까이 위치한 dependency에 맞게 학습을 하고 멀리 떨어진 dependency에 대해서는 학습을 못하게 됩니다. 예시를 보면 긴 문장이 input으로 들어왔을 때 마지막에 올 단어가 ticket이라는 것을 첫번째 줄의 ticket으로 유추할 수 있지만 vanishing gradient 문제로 멀리 떨어진 단어들과의 dependency를 학습하지 못하게 되어 ticket이 아닌 가까이 위치한 printer로 잘못 유추해버린 것을 볼 수 있습니다. 또한, gradient가 소실되었을 때, 정말로 미래에 과거가 영향을 미치지 않아서 소실된건지, 파라미터 값이 잘못 설정되어서 소실된건지 구분할 수 없게 됩니다.

Ppt5

Exploding gradient problem은 앞의 식에서 W\_h가 1보다 크면 비정상적으로 큰 값이 되면서 결국 발산해버리는 것입니다. 이 문제는 gradient clipping으로 해결할 수 있습니다. Gradient clipping은 gradient가 일정 threshold를 넘어가면 gradient값의 L2 norm값으로 나눠주는 방식입니다. 쉽게 말하면 파라미터를 update할 때 gradient가 정상적인 값보다 너무 크다고 판단되면 scale down해주는 방법입니다.

Ppt6

Gradient vanishing 문제를 해결하는 방법으로 ResNet, DenseNet, HighwayNet이 있습니다. ResNet은 input x에 convolutional layer을 지나고 나온 결과를 더해줌으로써, 과거의 내용을 기억할 수 있게 하는 것입니다. 과거의 학습 내용을 보존하고 추가적으로 학습하는 정보를 더해주므로 gradient가 사라지는 문제가 해결될 수 있습니다. DenseNet은 이전 layer들의 feature map을 계속해서 다음 layer의 입력과 연결하는 방식입니다. ResNet은 feature map끼리 더하기를 해주는 방식이라면 DenseNet은 feature map끼리 concatenation시키는 방식입니다. HighwayNet은 ResNet과 비슷하고 transform gate와 carry gate를 이용하여 output이 input에 대해 얼마나 변환되고 옮겨졌는지 표현하는 모델인데 LSTM에서 영감을 받은 모델입니다. Vanishing gradient는 여러 분야에서 일반적인 문제이지만 특히 RNN의 경우 동일한 weight matrix를 반복적으로 곱하므로 특히 더 불안정합니다. 이를 해결하기 위해 제시된 모델로 LSTM, GRU 모델이 있습니다.

Ppt7

LSTM은 RNN의 vanishing gradient문제로 발생하는 장기 의존성 문제를 해결하기 위해 RNN에서 메모리를 분리하여 따로 정보를 저장함으로써 한참 전의 데이터도 함께 고려하여 output을 만들어내는 모델입니다. LSTM의 핵심 아이디어는 이전 단계의 정보를 memory cell에 저장해서 흘려보내는 cell state인데 input, forget, output gate들을 이용하여 정보의 반영여부를 결정해줍니다. 이제 단계별로 세 개의 gate에서 어떤 일이 알아보는지 알아보겠습니다.

Ppt8

먼저 forget gate에서 어떤 정보를 잊고 어떤 정보를 반영할지 결정을 합니다. t번째 시점에서의 x값과 t-1시점에서의 hidden state를 입력받아서 sigmoid activation function을 통해 0에서 1사이의 값을 출력합니다. 출력한 값이 0에 가깝다면 불필요한 정보를 모두 지워버린다는 것이고 1에 가깝다면 이 정보를 많이 반영한다는 뜻입니다. 다음에는 input gate에서 새로운 정보가 cell state에 저장될지를 결정합니다. Forget gate와 마찬가지로 t번째 시점에서의 x값과 t-1시점에서의 hidden state를 입력받으며 input gate layer와 update gate layer가 존재합니다. Input gate는 sigmoid 함수에 의해 0에서 1사이의 값을 출력하여 현재의 정보를 반영할 것인지 결정하고 update gate는 cell state에 더해질 후보값들의 벡터를 만듭니다.

Ppt9

Forget gate에서 결정된 과거의 정보 삭제 유무와 input gate에서 결정된 현재 input값 반영 유무가 더해져서 cell state의 입력값으로 들어가게 되고 이렇게 cell state가 update됩니다. 마지막으로, output gate에서 cell state를 바탕으로 필터링한 값을 최종 output으로 반환합니다. Cell state에서의 식을 보면 만약 forget gate가 1로, output gate가 0으로 설정되면 t시점에서의 cell state는 이전의 정보가 완전히 보존되는 채로 hidden state를 update하기 때문에 cell의 정보가 완전하게 보전될 것이므로 장기의존성문제를 해결할 수 있습니다. 하지만, LSTM도 vanishing/exploding gradient문제가 아예 없다고 보장할 수는 없다고 합니다.

Ppt10

GRU는 LSTM의 강점을 가져오되, 불필요한 복잡성을 제거한 모델입니다. 매 time step마다 input x\_t와 hidden state h\_t는 있지만 cell state는 존재하지 않고 사실상 hidden state에 합쳐집니다. 이 모델도 LSTM처럼 gate들을 통해서 정보의 흐름을 통제하는데 update gate와 reset gate가 있습니다. Reset gate는 과거의 정보를 적당히 리셋시키는 것이 목적이고 update gate는 LSTM의 forget gate와 input gate가 합쳐진 것으로 보시면 됩니다. 전체적인 순서를 보자면, 먼저 reset gate를 통해 직전의 hidden state에 곱함으로써 직전의 hidden state 값을 그대로 이용하지 않고 reset을 해서 현 시점이 x값을 통해 현 시점의 candidate를 계산함으로써 임시적인 hidden state를 만듭니다. Update gate를 통해 구한 현시점과 과거 시점의 정보양 비율을 결정하고 1-z\_t에 이전 hidden state를 곱하고 z\_t에 현 시점의 candidate값을 곱해서 최종 hidden state를 계산합니다. GRU도 LSTM과 비슷한 방식으로 update gate에서 나온 값을 0으로 설정하면 이전 hidden state값이 계속 보존되므로 gradient vanishing문제를 어느정도 해결했다고 볼 수 있습니다.

Ppt11

또 다른 RNN으로는 bidirectional RNNs와 Multi-layer RNNs이 있습니다. Bidirectional RNNs은 left, right 두 방향으로 모두 정보를 이용하기 위해서 고안되었습니다. Forward rnn과 backward rnn으로 이루어져 있는데 Forward RNN은 말그대로 정방향으로 입력받아 hidden state를 생성하고 backward RNN은 역방향으로 입력받아 hidden state를 생성합니다. 분리된 weight를 가지고 있는 forward RNN과 backward RNN을 학습한 후 각 hidden state를 concat해서 최종적인 표현을 형성합니다. 나중에 배울 BERT 모델이 이를 기반으로 만들어졌다고 합니다. Multi-layer RNN은 RNN을 여러층으로 사용한 모델입니다. 더 복잡한 특성을 학습할 수 있고 lower RNN에서는 lower level의 feature들을, higher RNN에서는 higher level의 feature들을 학습할 수 있다고 합니다. 보통 2~4개 정도의 layer를 쌓으며 BERT 같은 transformer-based network의 경우 24개의 layer까지 사용할 수 있다고 합니다.

Ppt12

Machine translation이란 말 그대로 기계를 통해 번역을 하는 문제입니다. 1950년 대 냉전 시기에 소련과 미국이 항상 서로의 통신이나 기밀 문서들을 자동으로 번역해서 해석하기를 원하면서 machine translation이 처음으로 등장합니다. 주로 rule-based 방식으로 두 개의 언어 사전을 통해 각 단어가 대응되는 것을 찾아서 번역하는 단순한 방식입니다. 그 이후 rule-based 방식이 아닌 data를 통해 확률 분포를 학습하여 확률을 가장 높이는 방법으로 번역하는 방법이 등장합니다. 이를 statistical machine tranlsation이라 합니다. 실제로는 bayes theorem을 사용하여 이 식의 두 개의 부분을 계싼하는 방식으로 수행합니다. Translation model은 평행한 데이터를 통해서 확률을 계산하는데 작은 단어나 구를 번역하는 역할을 합니다. 뒷부분의 language model은 monolingual데이터를 통해서 계산하고 좋은 문장이나 좋은 구조를 뽑아내는 역할을 합니다. 번역을 하려면 많은 양의 병렬 데이터가 필요하고 조건부확률을 만들어주는 translation model을 어떻게 학습시키는가가 중요한데 이 때 필요한 방법이 alignment 정렬입니다. 정렬은 우선, 두 문장 병렬처리된 문장에서 특정 단어쌍들의 대응관계를 말합니다. 하지만 정렬에서 가장 큰 문제는 두 개의 언어는 각각 특성이 달라 단어들의 순서나 품사의 순서가 다르고 1대1 대응이 어려울 수 있다는 것입니다. 이 4개의 예시를 보시면 모두 1대1 정렬이 잘되지 않고 one-to-many, many-to-one, many-to-many로 된 것을 보실 수 있습니다. 이렇게 다양한 상황에서 정렬을 학습시킬 수 있는 방법은 일단 첫번째로 무차별로 대입하는 것입니다. 가능한 모든 y를 열거하여 모든 확률을 계산해 보는 것인데 굉장히 오래 걸리며 비용이 많이 소모될 것입니다. 또 다른 방법으로는 Heuristic 알고리즘인데 너무 낮은 확률들은 계산하지 않고 높은 확률 위주의 가지들로 다른 가지들은 쳐내면서 가는 방식입니다. 이를 decoding이라고 부르는데 디코딩이 이따가 나올 신경기계번역 sequence-to-sequence에서 설명드리겠습니다. Statistical machine translation의 단점은 이와 같습니다. 위와 같은 한계점을 극복하게 된 모델이 바로 sequence-to-sequence, 신경기계번역입니다.

Ppt14

딥러닝이 nlp분야에서 굉장히 기여를 많이 하는 직접적인 계기가 된 모델입니다. 이 모델은 단일 신경망으로 기계번역을 하고 두 개의 RNN인 인코더 RNN과 디코더 RNN으로 이루어져 있습니다. 인코더 rnn에서 소스 문장을 마지막에 인코딩시켜서 디코더 rnn에 넘겨줍니다. Rnn은 단방향, 양방향, lstm 등 다 될 수 있습니다. 디코더 rnn은 인코딩에 따라 첫번째 출력을 얻고 다음에 나올 단어의 확률분포 argmax를 취합니다. 다음 단계에서 디코더에 전 단계의 단어를 넣고 다시 피드백하여 argmax를 취하고 단어를 맞춥니다. 이런 방법을 계속 반복하여 목표 문장을 찾다가 디코더가 종료 토큰을 생성하면 중단합니다. 시퀀스 투 시퀀스는 긴 텍스트를 짧은 텍스트로 요약이 가능하다는 점, 전의 단어가 다음 단어로 이어지며 문맥적인 것을 파악할 수 있다는 점, 자연어를 코드로 생성할 수 있다는 점 등에서 기존 기계번역보다 훨씬 더 유용합니다. 시퀀스 투 시퀀스 모델을 요약하자면 ‘조건부 언어 모델’인데 디코더가 다음 타깃을 계측하는 조건부 언어 모델로 학률을 직접 계산합니다. X가 주어지면 y를 뽑아내고 그 첫번째 y를 x와 다시 input으로 넣고 계속 반복하여 conditioning을 겁니다. 시퀀스 투 시퀀스 모델은 아까 배웠던 병렬 망뭉치로 학습을 시키고 시스템을 전반적으로 최적화합니다. 인코더 rnn에 문장을 넣고 디코더 rnn의 모든 단계에서 다음에 올 확률분포 y hat의 손실을 계산합니다. 손실은 교차 엔트로피 등으로 계산할 수 있고 손실들을 이런 방식으로 모두 얻은 후 이를 평균 내어 최종 loss값을 구합니다. 역전파는 전체 시스템에 걸쳐 흐릅니다. 시스템을 전체적으로 최적화하는 것이 학습의 포인트인데 개별적으로 학습하는 것은 좋지 않고 end-to-end방식으로 학습하는 것이 문장 요소를 모두 고려해 좋은 결과물을 도출한다고 합니다. 미리 정의된 최대 길이에서 최대한 짧은 문장을 채우는 방법으로 진행합니다.

Ppt15

시퀀스 투 시퀀스 모델에서는 greedy decoding이라는 문제가 발생합니다. 각 단계에서 가장 확률이 높은 단어를 선택하는데 선택하게 되면 다시 수정할 수 없으며 돌아갈 수 없게 됩니다. 이를 greedy decoding이라고 합니다. 이를 해결할 수 있는 방법으로 철저하게 가능한 모든 언어 번역 공간을 검색하는 것이 있습니다. 모든 경우의 수를 다 계산하는 것이므로 부담이 됩니다. 그래서 일종의 검색 알고리즘인 빔 검색 디코딩 방법이 제시되었습니다. 디코더의 각 단계에서 k개의 가장 가능성이 높은 부분 번역을 선택하여 가지치기 하는 방식입니다. 빔 검색 방법이 최적의 솔루션을 찾아준다고 보장하지는 못하지만 전부다 계산하는 것보다는 훨씬 효율적입니다.

Ppt16

예시를 보시면 k를 2로 설정했으므로 각 단계마다 2개의 가능성이 높은 단어를 선택한 것을 볼 수 있고 한 가설이 end 토큰을 생성하고 끝이 나면 그것은 따로 두고 다시 다른 가설들을 탐색하기 시작하기 때문에 다시 돌아올 수 없어 수정할 수 없는 문제를 해결할 수 있습니다. Beam search도 무한정 가설을 세울 수 없기 때문에 한계를 정해줘야 하는데 첫 번째는 미리 정한 시간 단계 t에 도달하면 멈추는 방법, 두 번째는 완료된 가설 수를 정해 놓고 이에 도달하면 중지하는 방법입니다. 이렇게 완성된 가설 모음 중 가장 높은 점수를 받은 것을 선택합니다. 긴 가설일수록 낮은 점수를 받게 되어서 기준이 애매해질 수 있는데 이를 해결하는 방법으로는 오른쪽 수식과 같이 점수를 time step 수로 나누어 평균을 구하여 평균 중에서 높은 점수를 선택하면 됩니다.

Ppt17

NMT의 장단점을 살펴보겠습니다. 장점으로 일단 기존 기계번역보다 sequence를 고려하기 때문에 좀 더 문맥을 고려하고 단어와 구 사이의 유사성을 잘 이용하여 더 나은 성능을 제공한다는 것입니다. 두번째는 개별적으로 최적화되어야 하는 기존 기계번역과는 달리 역전파가 전 시스템에 걸쳐서 진행되기 때문에 단순하고 편리합니다. 세번쨰는 feature engineering이 필요없고 모든 언어쌍에 같은 방법으로 동일한 아키텍쳐로 적용이 가능하기 떄문에 그저 병렬 말뭉치만 찾으면 되서 인간의 노력이 덜 필요합니다. 단점으로는 기계번역은 디버그가 가능하지만 NMT는 쭉 흘러가는 형태이기 때문에 어디서 오류가 발생했는지 표시하기 어렵습니다. 두번째는 특정 요류를 발견하고 수정하고 제어할 사후 규칙을 만들기 어렵다는 것입니다. 기계 번역은 그 부분만 따로 뗴어와서 제어할 수 있는데 NMT는 flow가 전반적으로 흘러가기 때문에 특정 부분만 수정하기 어렵습니다. NMT의 한계점으로는 목표 어휘에 없는 단어 생성이 불가하고 학습데이터와 테스트데이터 사이에 도메인이 일치하지 않으면 성능이 떨어진다는 한계점이 있습니다. 또한, 한 문장을 인코딩하는 것이기 때문에 긴 텍스트의 경우 문맥을 계속 유지하여 오류가 생길 수 있으며 리소스가 부족한 언어 쌍의 경우 성능이 떨어질 수 있다는 문제가 있습니다. 더 많은 문제점으로는 오른쪽 그림을 보시면 paper jam은 종이가 프린터에 끼였다는 의미인데 jam으로 번역하여 관습적인 언어, 상식에 대한 개념이 없다는 것입니다. 두번째로는 편향되게 번역할 수 있다는 것인데 간호사의 주어를 she, 프로그래머의 주어를 he로 마음대로 할당하였음을 볼 수 있습니다. 또한, 단순하게 아무 단어를 반복한 문장을 멋대로 해석하는 문제가 발생하기도 합니다. 이러한 시퀀스 투 시퀀스의 한계점을 보완하기 위해 나온 방법이 attention입니다.

Ppt18

Sequence to sequence에서는 문장의 정보가 끝에서 다 인코딩 되어버리는 즉, 정보가 쏠려버리는 병목현상이 나타납니다. 소스 문장에 대한 정보를 끝에서 다 캡처하도록 강조하며 여기에 너무 많은 압력이 가해지게 됩니다. Attention은 단계별로 집중을 하게 만들어 한꺼번에 집중이 가해지는 병목현상을 해결해줍니다. 디코더의 첫 단계에서 인코더의 첫 단계와 내적을 시켜 스칼라 값을 뽑아 attention score를 구합니다. Attention score란 현재 디코더의 시점 t에서 단어를 예측하기 위해 인코더의 모든 은닉 상태 각각이 디코더의 현시점의 은닉 상태 states와 얼마나 유사한지를 판단하는 점수입니다.

Ppt19

그렇게 4개의 attention score를 가져와서 소프트맥스에 넣고 확률분포를 얻습니다. Bartchart로 알아본 결과, 대부분의 확률 질량이 첫번째 단어로 갔음을 알 수 잇고 그렇게 ‘he’라는 단어를 먼저 만듭니다. 그렇게 얻은 attention 분포는 인코더의 hidden states의 가중치가 됩니다. Attention 분포와 인코더의 hidden states를 가중합하여 attention output을 뽑고 그 attention output과 디코더의 hidden state를 결합하여 y1 hat이라는 결과 he를 도출하게 됩니다. Attention 출력을 사용하여 다음 단어에 영향을 주고 그런 식으로 두 번째 디코더에도, 세번째 디코더에도 같은 것을 반복합니다. 각 단어에는 분포가 있고, 기존 NMT보다 훨씬 더 유연하고 부드러운 정렬과 사고를 하게 됩니다. 각 인코딩의 각 단계마다 집중을 해주어 정보의 흐름을 마지막에 집중시키지 않는 것도 확인할 수 있습니다. 또한 문장이 길어질수록 기울기 소실 문제가 발생하는데 attention은 각 단계마다 집중을 할 수 있어 기울기 소실 문제도 해결하였습니다. NMT는 flow가 전반적으로 진행되기 때문에 어디서 오류가 발생했는지 추적하기 어렵지만 attention은 인코딩의 각 단계마다 집중하여 보면서 어디서 오류가 발생했는지 확인할 수 있습니다.

Ppt20

Machine translation이 제대로 이루어졌는지 확인하는 대표적인 지표는 ‘BLUE’점수입니다. BLUE란 기계에 의해 번역된 문장과 사람이 작성한 문장을 비교해서 유사도를 측정하여 성능을 측정하는 지표입니다. 보시다시피 NMT의 성능이 굉장히 가파르게 증가했다는 것을 볼 수 있고 현재까지도 연구가 진행중이라고 합니다. BLUE는 이 3가지 요소로 이루어져 있고 보통은 n-gram길이를 4까지 활용하는 BLUE4를 활용한다고 합니다.

(그 아래는 ppt읽기)