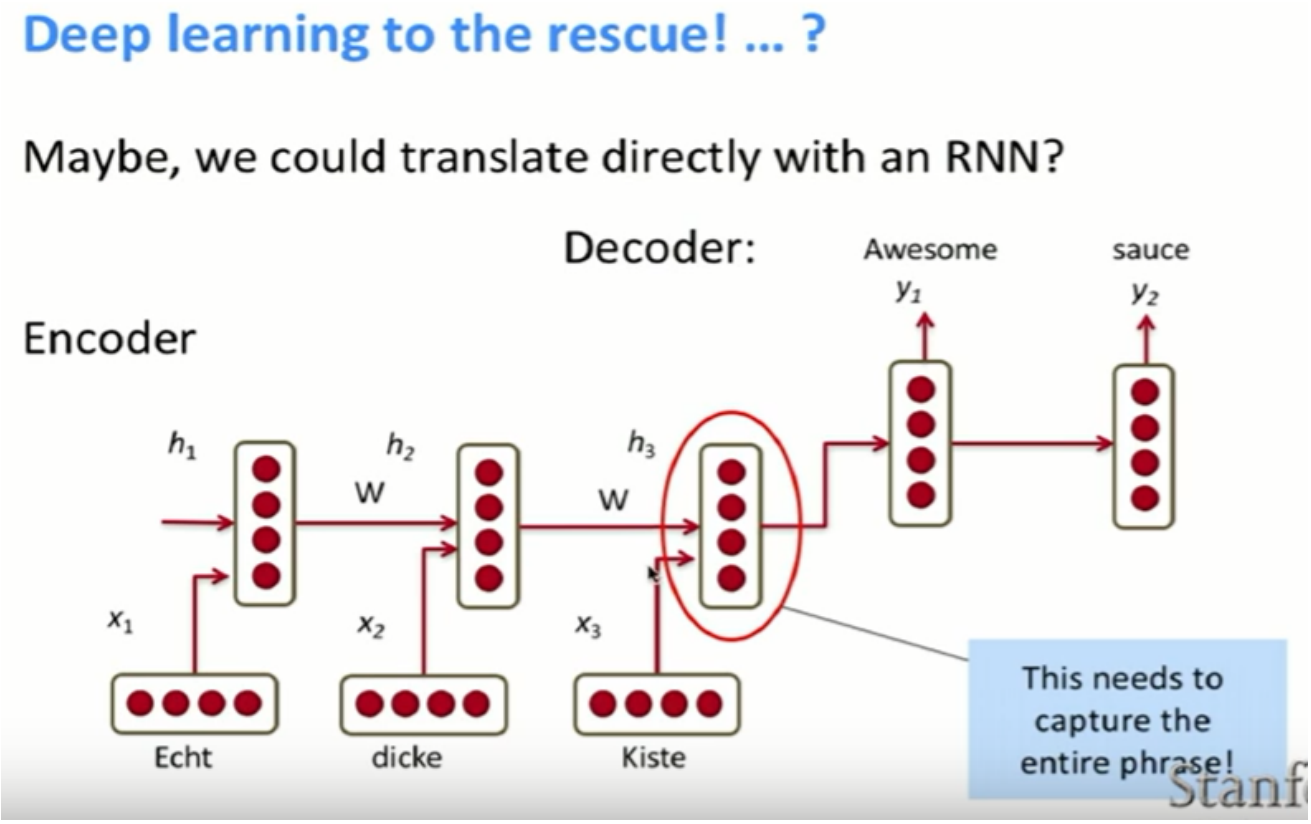
Lecture 9: Machine Translation and Advanced Recurrent LSTMs and GRUs

출처: https://www.youtube.com/watch?v=QuELiw8tbx8&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6&index=9&t=10s

traditional machine transaction은 생략했다



- 간단한 single recurrent neural network이다. 독일어에서 영어로 번역한다고 해보자! word vector들($\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}$)과 softmax classifier를 갖는다. 만약 독일어 문장의 끝에 와서 더 이상 인풋이 없을 때부터, 번역을 한다.
- 마지막 vector는 전체 구의 정보를 갖고 있어야 한다. 하지만 슬프게도...5~6개 이전 단어부터는 그렇게 하지 못한다.

MT with RNNs – Simplest Model

Encoder: $h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$

Decoder: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)}h_{t-1}\right)$

$$y_t = softmax\left(W^{(S)}h_t\right)$$

Minimize cross entropy error for all target words conditioned on source words

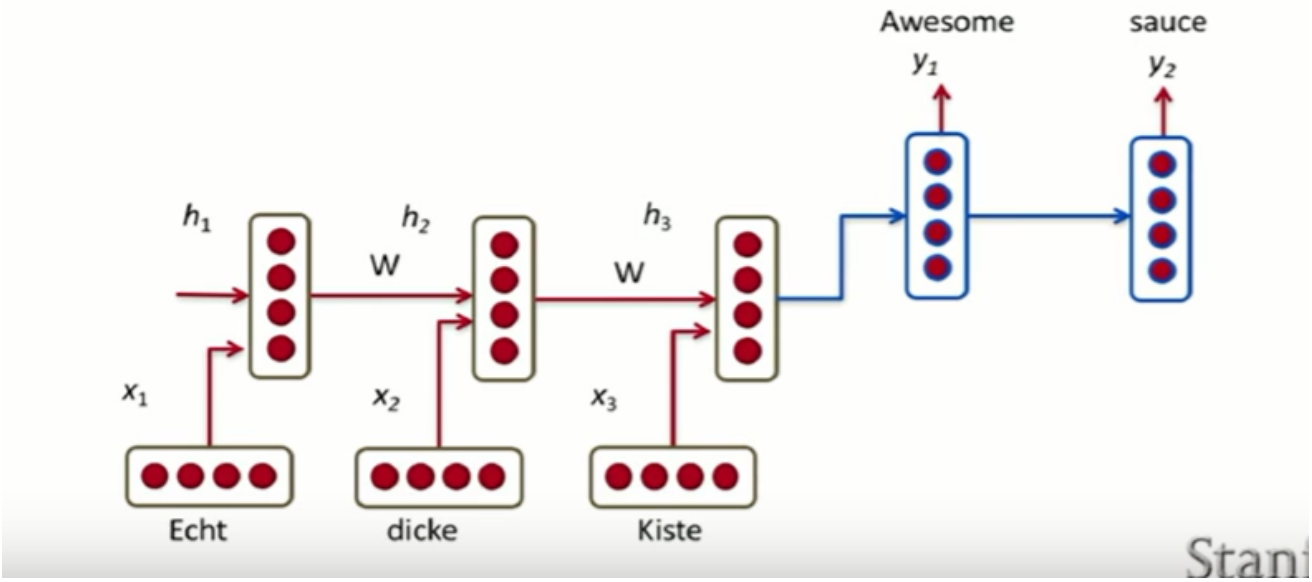
$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y^{(n)}|x^{(n)})$$

(ϕ는 각각의 vector를 위한 똑같지 않은 W matrix가 있다는 뜻이다.)

- 일반적인 RNN이 있다고 가정해보자.
- encoder에서 h_t는 linear network이다. 여기서 이전 hidden state와 현재 word vector(x_t) matrix vector product를 곱한다.
- decoder에서 h_t는 final form은 아니고 그냥 간단한 형태를 나타낸 것이다. decoder에서는 input이 없으므로, $W^{(hx)}x_t$를 없애준다. 단지 이전의 hidden state를 참고하면서 쭉 이동하는 거다. y_t는 각각의 time step에서의 softmax output이다.
- cross entropy를 통해 error를 최소화해준다.
- machine transaction보다는 상대적으로 간단하지만, 좀 더 간단하게 만들 필요가 있다.

RNN Translation Model Extensions

1. Train different RNN weights for encoding and decoding



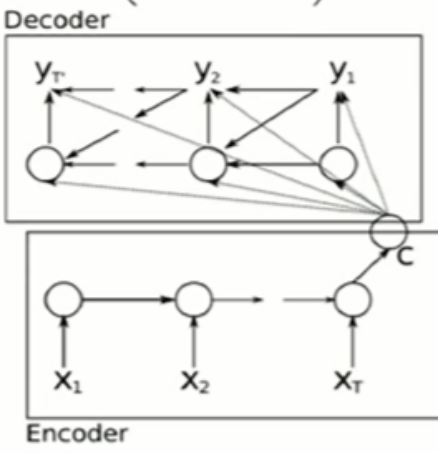
- 첫번째 단계는 encoding과 decoding에 각각 다른 weight를 주는 것이다. encoding과 decoding에 같은 W를 쓰는 게 아니라, encoding에서 쓰는 W와 decoding에서 쓰는 W를 구분하는 것이다.

RNN Translation Model Extensions

Notation: Each input of ϕ has its own linear transformation matrix. Simple: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)}h_{t-1}\right)$

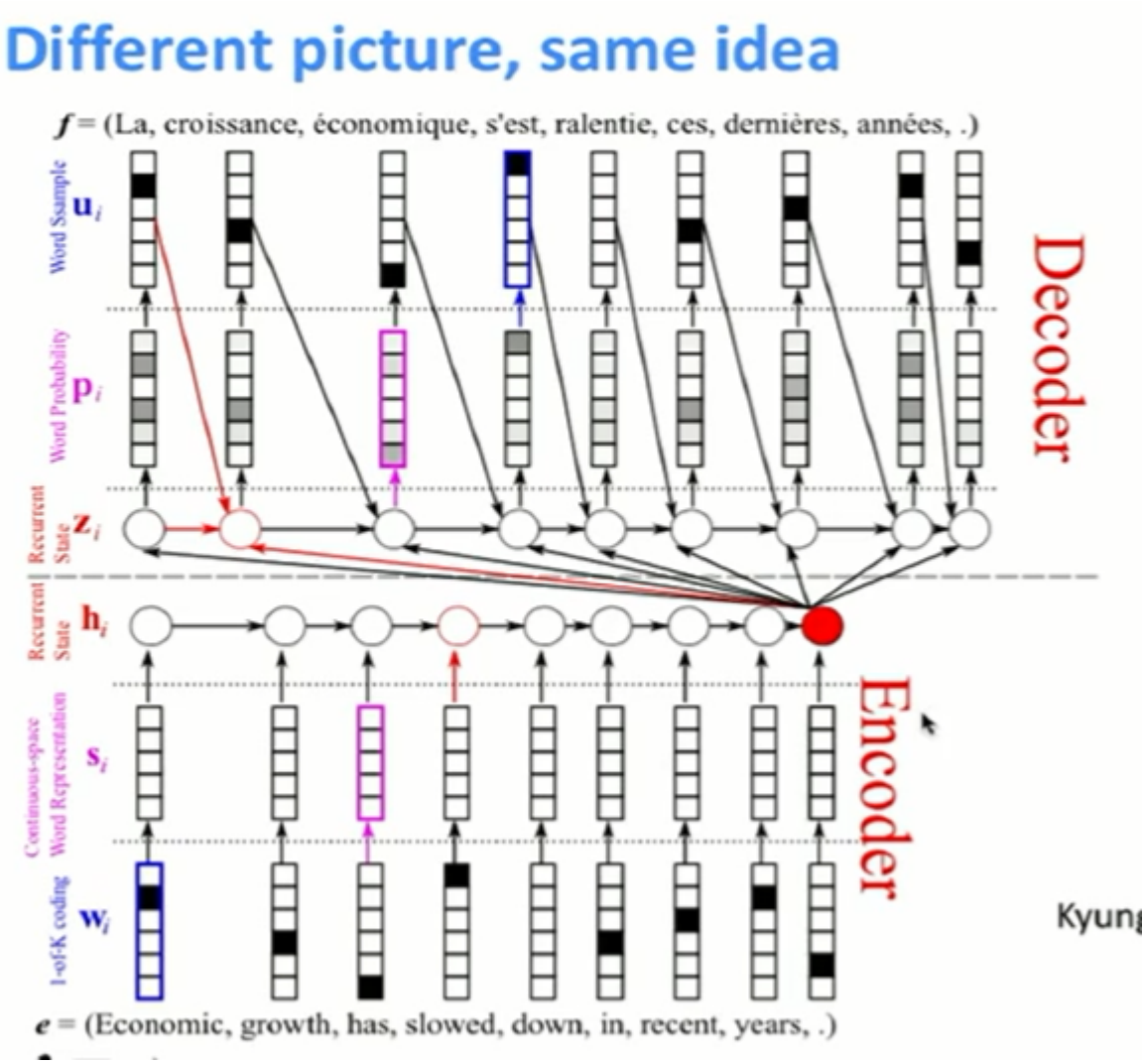
2. Compute every hidden state in decoder from
   - Previous hidden state (standard)
   - Last hidden vector of encoder $c=h_T$
   - Previous predicted output word y_{t-1}

$$h_{D,t} = \phi_D(h_{t-1}, c, y_{t-1})$$



Cho et al. 2014

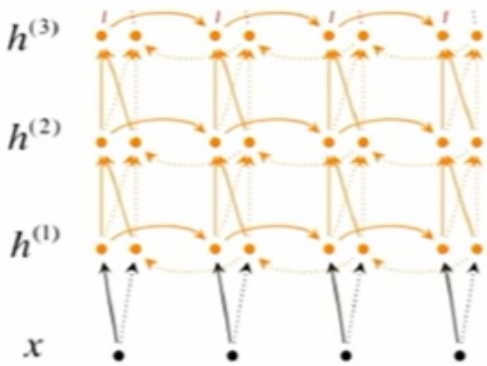
- 두번째 단계는 decoder의 모든 hidden state에서 이전 hidden state, encoder에서 전체 구의 정보를 가지고 있었던 마지막 hidden vector, 이전에 예측한 output word에 대한 matrix vector product를 한다. y_{t-1}를 파라미터로 넣는 이유는 실제로 이렇게 해보니 model에서 단어를 반복하는 일이 줄어들었다고 한다...!



- 이전 슬라이드에 있던 그림과 똑같지만, 좀 더 잘 파악하게 시각화한 그림이다.
- k개의 단어를 one-hot vector롤 바꿔준다. 그 다음에 word embedding을 만들어 준다.
- encoder의 hidden state를 쭉 통과한 다음, 가장 마지막 output vector를 decoder의 파라미터로 넣어준다.
- 그리고 위에서 봤던 것처럼, t-1번째 hidden state, encoder의 마지막 vector, y_{t-1}을 t번째 hidden state의 파라미터로 넣어준다.

RNN Translation Model Extensions

3. Train stacked/deep RNNs with multiple layers
4. Potentially train bidirectional encoder



5. Train input sequence in reverse order for simpler optimization problem: Instead of A B C → X Y, train with C B A → X Y

- 세번째 단계는 여러 layer를 쌓는 stacked RNN을 이용하는 것이다.
- 네번째는 흔하지 않지만 bidirectional encoder를 이용하는 것이다.
- 다섯번째는 input 문장의 순서를 바꿔서 넣는 방법이다. 위의 예시를 통해 설명하자면, A는 X로 바뀔 확률이 크고 B는 Y로 바뀔 확률이 크기 때문이다. 번역할 단어와 번역될 단어를 좀 더 가깝게 둔다고 생각하면 된다. 그래서 이전 강의에서 언급했던vanishing gradient가 덜 나타나게 할 수 있다.

6. Main Improvement: Better Units

- More complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU) introduced by Cho et al. 2014 (see reading list)
- Main ideas:
  - keep around memories to capture long distance dependencies
  - allow error messages to flow at different strengths depending on the inputs

- 이제 GRU에 대해 살펴볼 거다.
- GRU의 main idea는 오랜 시간 동안 기억을 저장하고 있는 것이다. 그리고 input에 따라 다른 강도로 error message를 흘려보내는 것이다.

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

- Compute reset gate similarly but with different weights

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

- GRU에서는 일반적인 RNN과 달리, 먼저 gate를 계산한다. gate들은 h_t와 마찬가지로 hidden state와 똑같은 길이를 가지고 있는 vector이다.

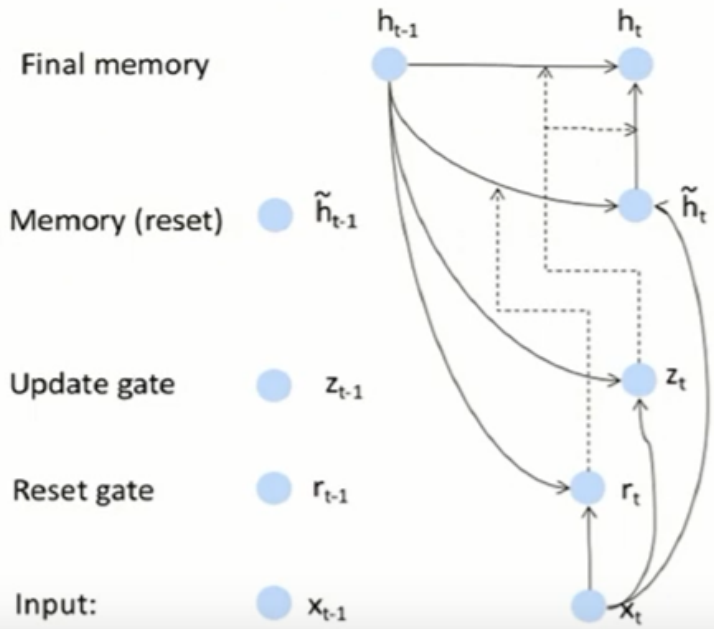
- 현재 input word vector와 hidden state를 이용해서 update gate를 계산한다. 그리고 다른 W를 가지고 reset gate를 계산한다.
- 일반적인 RNN과 달라지는 점은 σ가 sigmoid 함수라는 것이다. 그래서 z_t의 element들은 0과 1사이의 값이 된다. 이 값들을 확률로 해석할 수 있다.

GRUs

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

- reset gate와 Uh_{t-1}을 element wise로 곱해준다. 그리고 Wx_t와 더해준다. $\tilde{h_t}$는 intermediate memory content이다.
- 만약 reset gate가 거의 0에 가깝다면, 이전의 기억(이전의 계산)을 거의 무시한다.
- 이렇게 하는 이유를 감성 분석을 예로 들어 보겠다. 영화평에서의 감성 분석을 한다고 치자. 만약 줄거리에 대한 요약이 쭉있고 마지막쯤에 "~재밌었다."라고 한다면, 줄거리는 중요하지 않고 "재밌었다"라는 단어만 중요하다. 그렇다면 굳이 줄거리에 대한 정보를 기억할 필요가 없으니, 모델에게는 줄거리, 즉 이전의 정보를 모두 잊어버리게 하는 것이다.
- 마지막 memory에서는 z_t의 값에 따라 이전의 기억을 좀 더 저자할지 현재의 기억을 좀 더 저장할지를 결정할 수 있다. 만약 z_t가 모두 1이면 $h_t = h_{t-1}$이므로, 단지 이전의 기억을 카피하는 것이다.
- 감성 분석으로 다시 예를 들면, "I love this movie, it's beautiful love story."라는 영화평에서 love가 중요한 것이지 love story가 중요한 게 아니다. 일반적인 RNN에서는 중요도를 고려하지 않고 모든 time step마다 update를 한다. 그래서 중요한 정보가 점차 사라지게 된다.
- model은 언제 reset을 하고 update를 할지를 학습하게 된다.

Attempt at a clean illustration



$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

Stanfo

- 오른쪽 수식을 보면서 왼쪽의 그림을 따라가면 무리없이 이해가 될 것이다.

GRU intuition

- If reset is close to 0, ignore previous hidden state → Allows model to drop information that is irrelevant in the future

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!
- Units with short-term dependencies often have reset gates very active

Stanfo

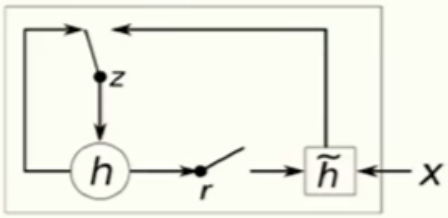
- 만약 reset을 0에 가깝게 한다면 이전 hidden state를 무시한다.
- z는 과거의 state가 현재에 얼마나 중요한 지를 결정한다. 만약 z가 1에 가깝다면, 그냥 이전의 hidden state를 copy하는 것이다. 그렇다면 vanishing gradient가 줄어들게 된다.

GRU intuition

- Units with long term dependencies have active update gates z

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- Illustration:



- Derivative of $\frac{\partial}{\partial x_1} x_1 x_2$? → rest is same chain rule, but implement with **modularization** or automatic differentiation

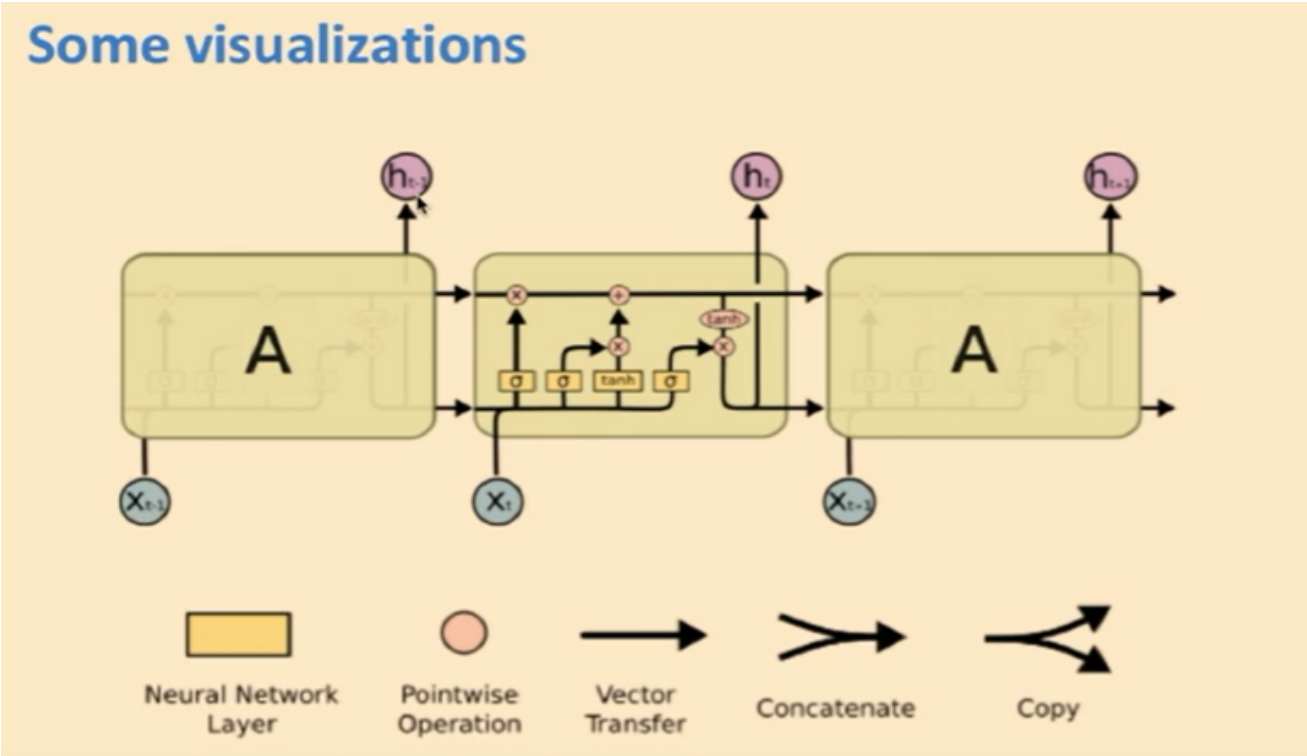
- 회로같은 걸 공부해본 사람이라면 위처럼 보는 것도 괜찮을 것이다.
- $r_t \circ U^{(r)} h_{t-1}$ 의 미분 값을 알고 있으므로 $\frac{\partial}{\partial x_1} x_1 x_2$의 값을 알 수 있다. 하지만 하나하나 계산하는 게 아니라 모듈을 쓰거나 자동화를 하는 것 같다.
- 굳이 update와 reset으로 나누는 이유는 어떤 정보를 기억하고 유지할 때 다른 메카니즘을 가질 수 있기 때문이다.

Long-short-term-memories (LSTMs)

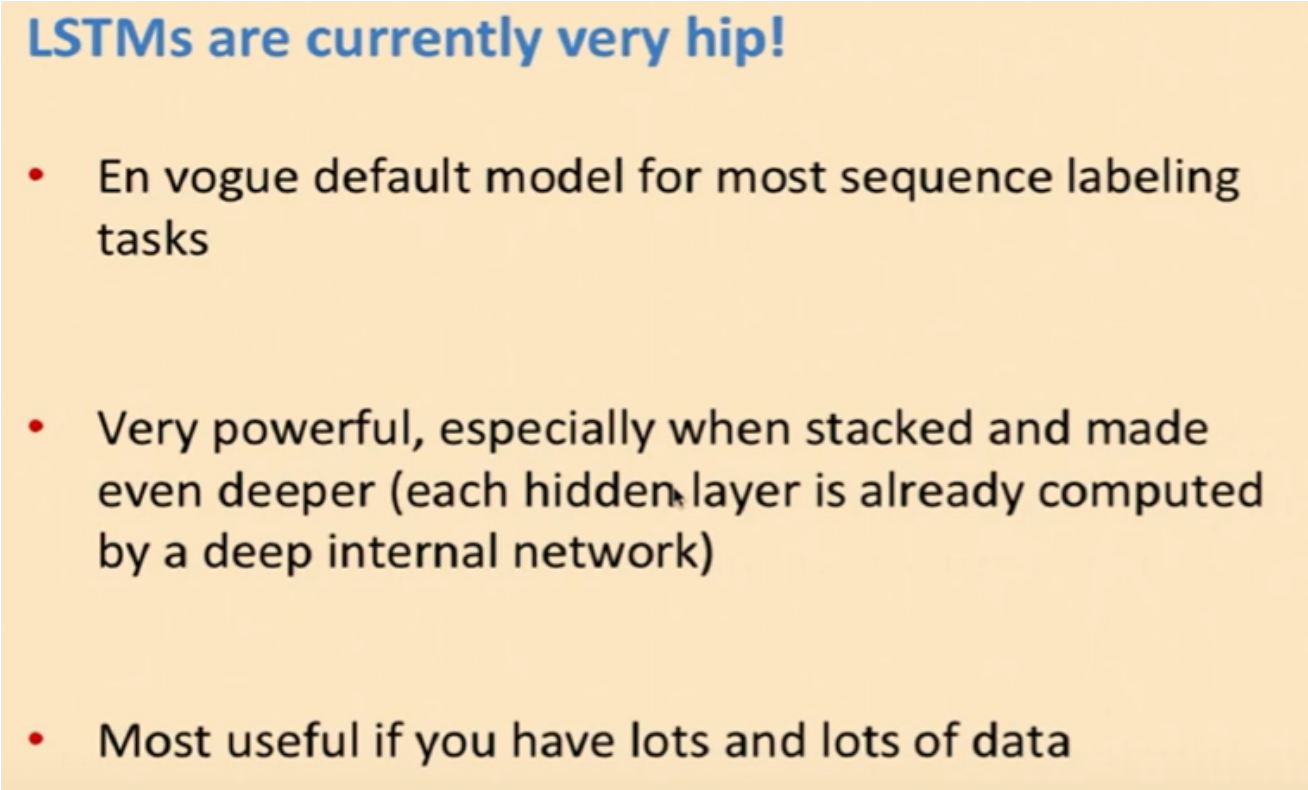
- We can make the units even more complex
- Allow each time step to modify
  - Input gate (current cell matters) $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$
  - Forget (gate 0, forget past) $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$
  - Output (how much cell is exposed) $o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$
  - New memory cell $\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$
- Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- Final hidden state: $h_t = o_t \circ \tanh(c_t)$

- LSTM은 GRU보다 더 많은 gate를 갖기 때문에 더 복잡하다.
- input gate는 현재의 vector(x_t)를 얼마나 중요한 지를 결정한다.

- forget gate는 이전의 vector(h_{t-1})을 얼마나 잊지 않을 건지를 결정한다. 만약 forget gate의 값이 0이면 하나도 기억하고 싶지 않다는 뜻이다.
- output gate는 현재 가지고 있는 정보를 이용해서 예측을 하는 게 중요한지 아니면 정보를 계속 갖고 있는 게 중요한지를 판단한다. 만약 현재 time step의 특정한 cell이 중요하지는 않지만 점점 더 중요해지는 것이라면, 해당 final softmax의 instance로 내놓지 않고 계속 그 정보를 가지고 있는다. 즉, 해당 time step에서만 잊혀진다.
- new memory cell은 GRU랑 비슷하게 현재의 정보를 더 저장할지 이전의 정보를 더 저장할지를 결정한다.
- 사실 4개의 gate의 parameter는 다 똑같고 3개는 sigmoid이고 1개는 tanh라는 차이밖에 없다.
- final memory cell과 final hidden state를 계산할 때, 위의 gate를 이용한다.
- final memory cell에서는 forget gate와 input gate를 이용한다. GRU처럼 단지 c와 1-c를 이용하는 게 아니라 두 개의 메카니즘을 이용한다. forget gate는 이전의 memory를 얼마나 기억할 건지를 결정하고 input gate는 현재의 memory를 얼마나 기억할 건지 결정한다. 만약 i_t가 1이면 현재의 memory를 모두 기억한다.
- final hidden state에서는 c_t를 계속 가지고 있을 지 아니면 해당 time step에서 예측을 하는데 이용할 건지를 결정한다. 이렇게 하는 이유는 해당 softmax classifier과 예측을 하는데 관련이 없는 정보를 받아들이는 것을 차단할 수 있다.



- 사실 비교적 간단한 시각화이지만 알아보기가 좀 어렵다...



- LSTM은 layer가 쌓이고 점점 깊어질 수록 강려크하다고 한다.

이후는 성능비교, 현황과 같이 부가적인 것이라 생략한다.