ЖЧ. Deep Learning for Computer Vision

Notes from lectures and questions to them. Summer 2022
2022-07-27_umich_DL4CV_lectures_1-13_notes.md
Course:

```
EECS 498-007 / 598-005
Deep Learning for Computer Vision
Fall 2019
```

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2019/schedule.html

See lectures 1-11 in lectures1-11

Lecutre 12. Recurrent Neural Networks (RNN)

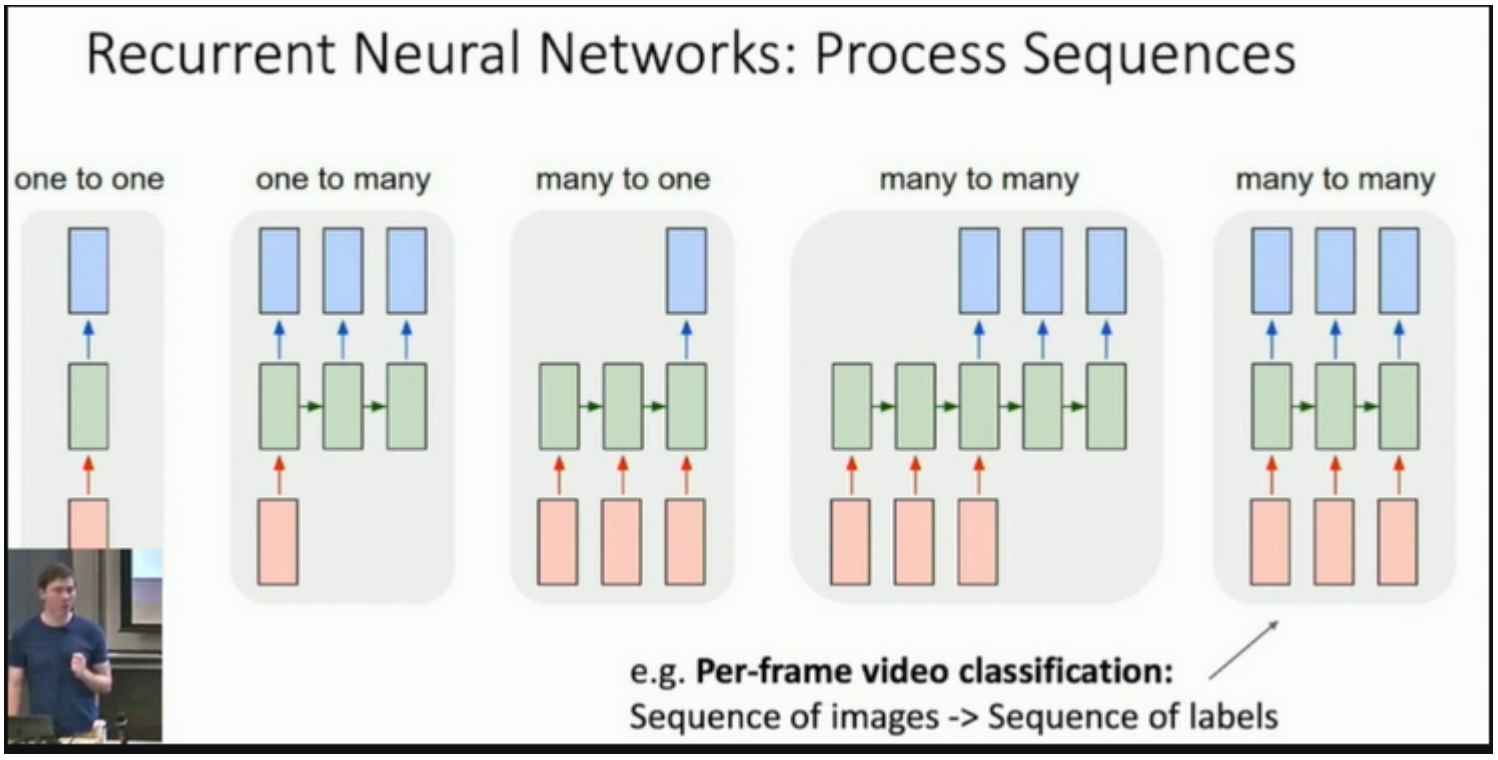
7. Intro

So far it was "feedforward" NN. One to one.

Other:

- "One to many". Image into image description.
- "Many to one". E.g. video to a label.
- Many to many. E.g. machine translation, Eng to French. Seq to sequence problem.
    - Per-frame video classification: sequance of images -> lables. Commentator on a video. For each.

To work with sequences as input or output we use some kind of RNN. We want to process seq of arbirtrary length.



12. Seq proc of non-sequential data.
    Take multiple glimpses of image. And then classify an image. And after some glimses it makes prediciton.
    Another: generating images. Generate an image one piece at a time. Examples: generated digits; painting images of faces.

13. What's RNN?

Key idea: is processing a sequence and also RNNs has internal state that is updated as seq processed.

$h_t = f_W(h_{t-1}, x_t)$

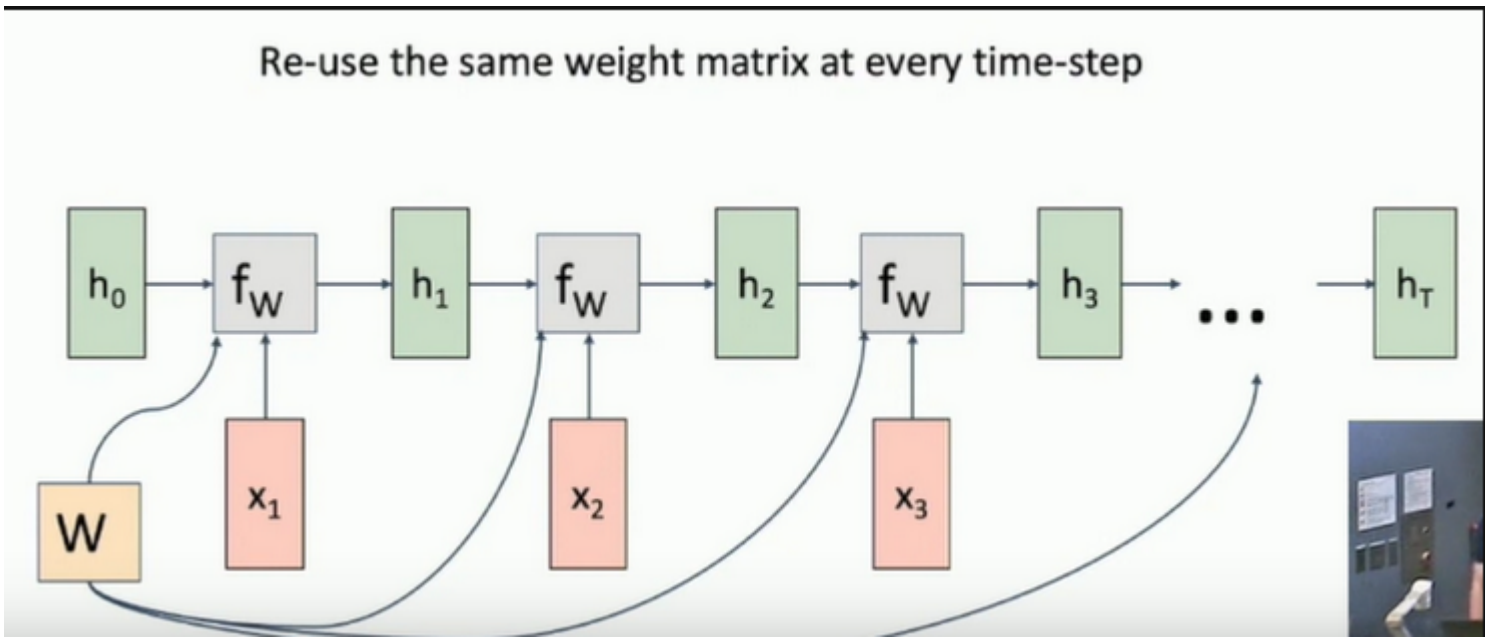
Process x by applying recurrence formula. W - learnable weights. h state. x input. Single weight matrix at every step of a sequence. And the same function f at every step.

18. Vanilla RNN (aka Elman RNN in owner of Prof. Jeffrey Elman)

$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$ - gives current state.
To get output: $y_t = W_{hy}h_t$

Where h single vector, and three W matrices for h_{t-1}, for x_t and for h_t.

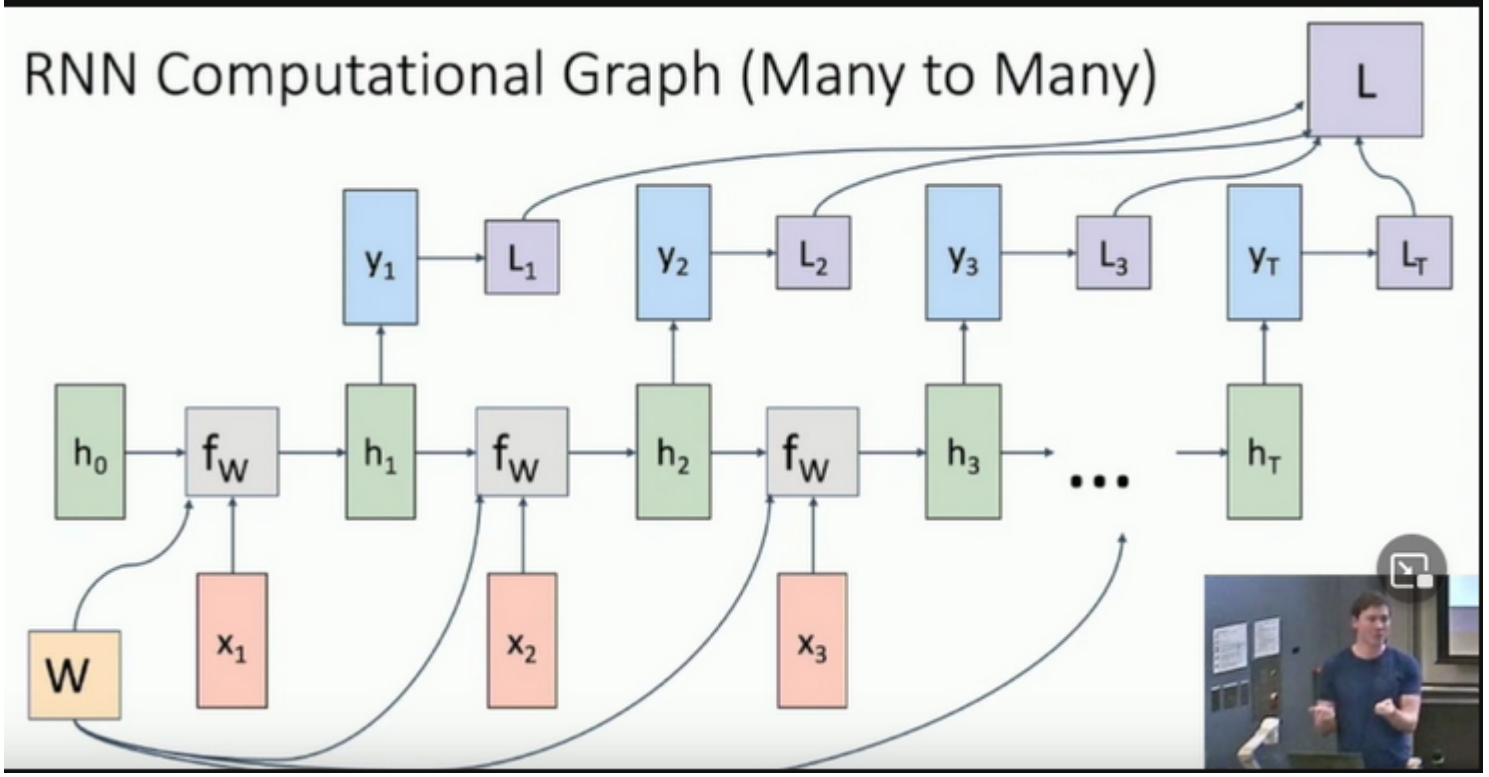
- xix. Example.



No matter how many T there, how long the sequence.

And we can use it for our types problems:

For many to many:

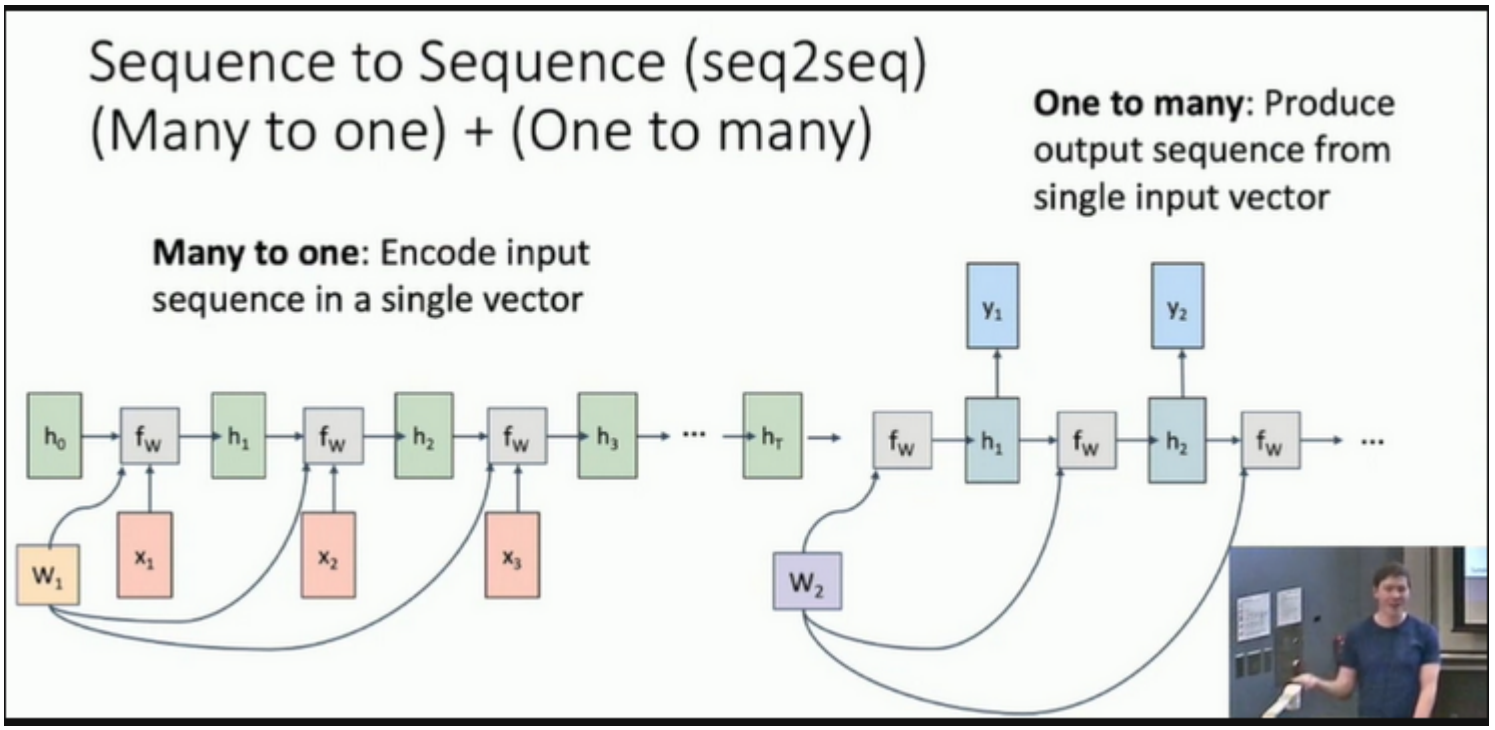


E.g. video classification per-frame. And it produces y_i and then we can apply L_i if we have labels (supervised). And the final L loss will sum those L_i .

Many to one then only one y at the end. E.g. single label for a video. Note it depends on entire sequence.

One to many. Also can use RNN. At beginning a single x.

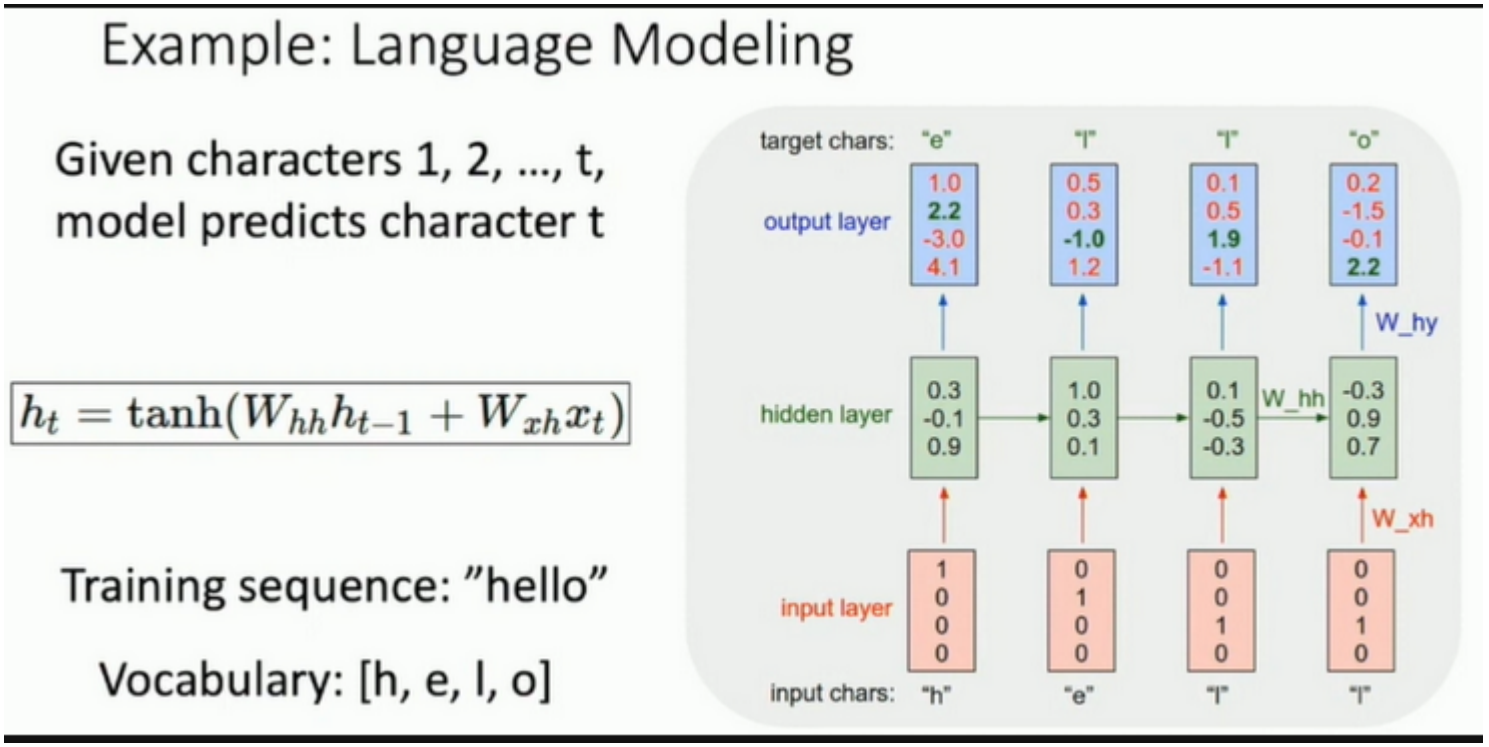
28. Sequence to sequnce (seq2seq). (Many to one) + (one to many). Another type of problem. E.g. to translate Eng to French.
    How to implement? From one NN (many to one) (encoder) and then into another NN (one to many) (decoder).



Why? Because we don't know how long sequence.

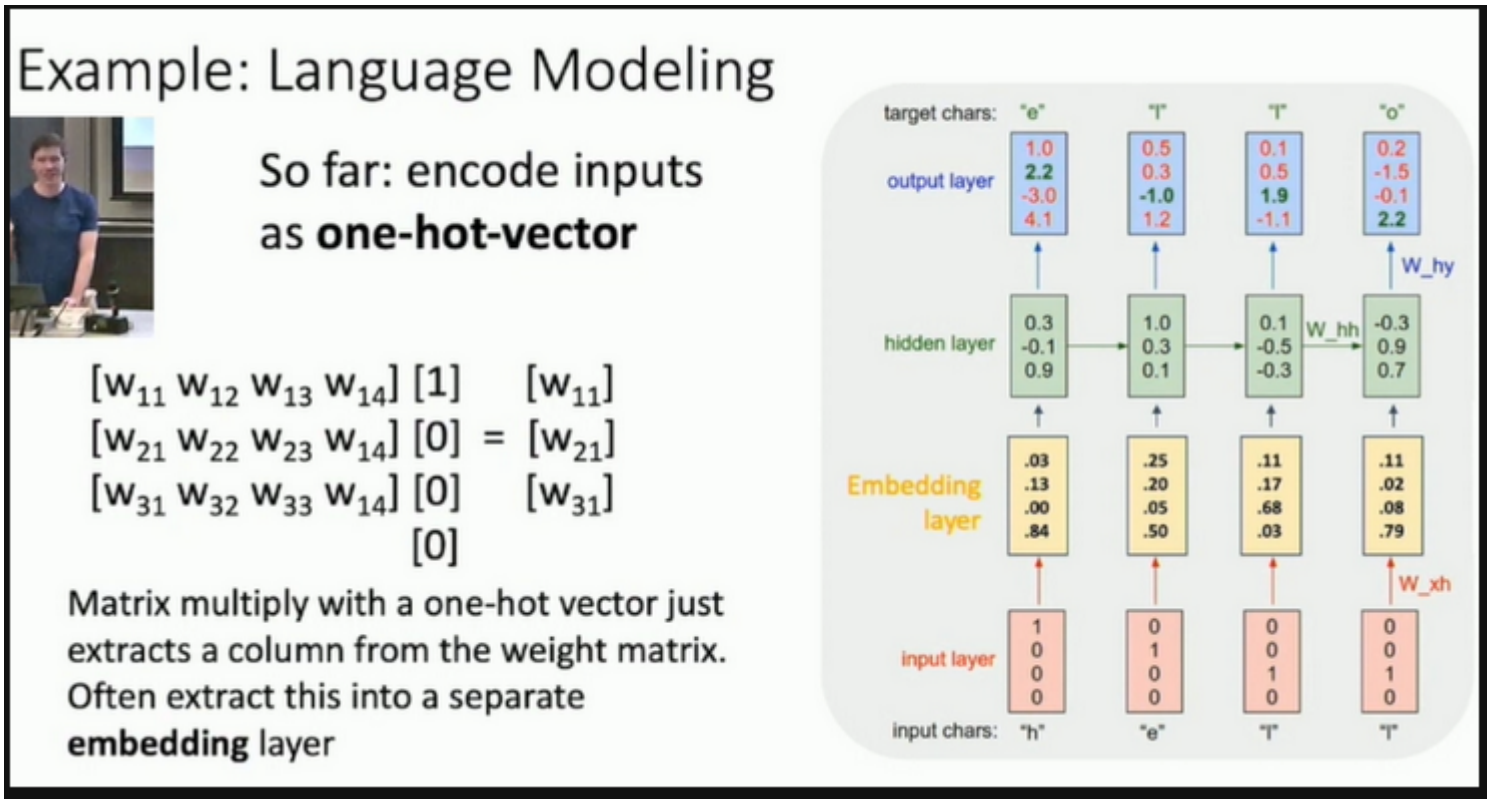
Example. Language Modeling. Given chars what's the next char? Infinite seq of chars and every time it tries

to predict next char.



Generating text. But we can generate text from this after we trained the NN. Example: given 'h' it outputs 'e'. Then we feed this 'e' into NN again, it outputs 'l' and so on. We keep internal weights ($W_h h$) and internal state h_i.

Optimization: that is one-hot-vector at beginning so extract this into a separate layer (embedding layer).



44. How to train. Backpropagation through time.

Problem: need a lot of memory if large graph for those sequences.
In practice we truncate those sequences: take subset of sequence. Compute loss then backprop. For the next chunk of seq. Backprop only to the beginning of the chunk. Then forward does infinite seq but backprop does only for chunk. Can be done in 112 lines on Python (not pytorch).

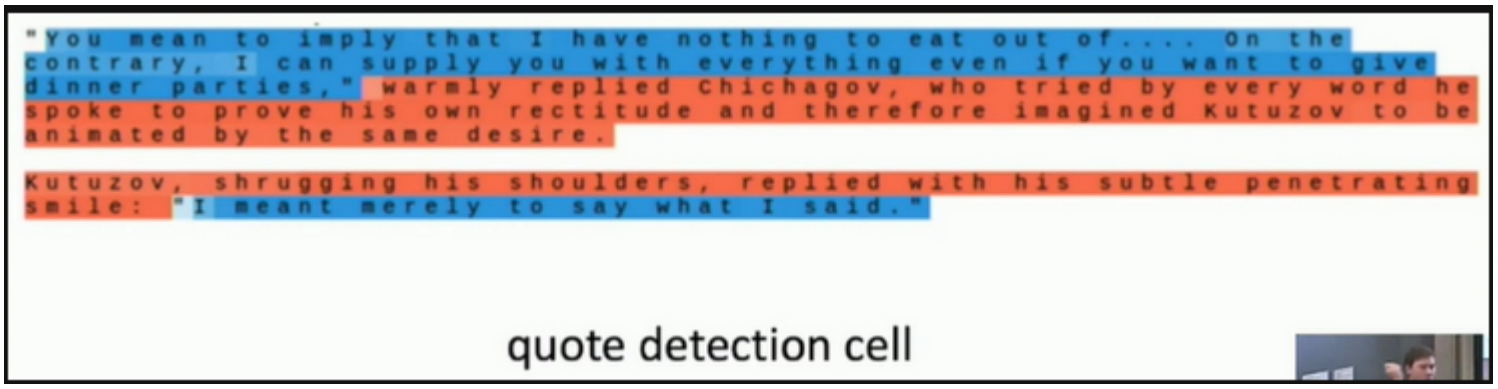
Examples.

1. William Shakespeare. The sonnets. No sense.
2. Latex algebra geometry. No sense.
3. Linux kernel source code.

63. Visualization. Why it succeeds in learning structure for those so well? What does these language model RNN learn?

Methodology. In the process of training it colors a next char using prediction from one cell, that prediction is from $tanh \in [-1, 1]$ where blue is close to -1 and red close to 1. Hence we can get idea what this cell looking for.

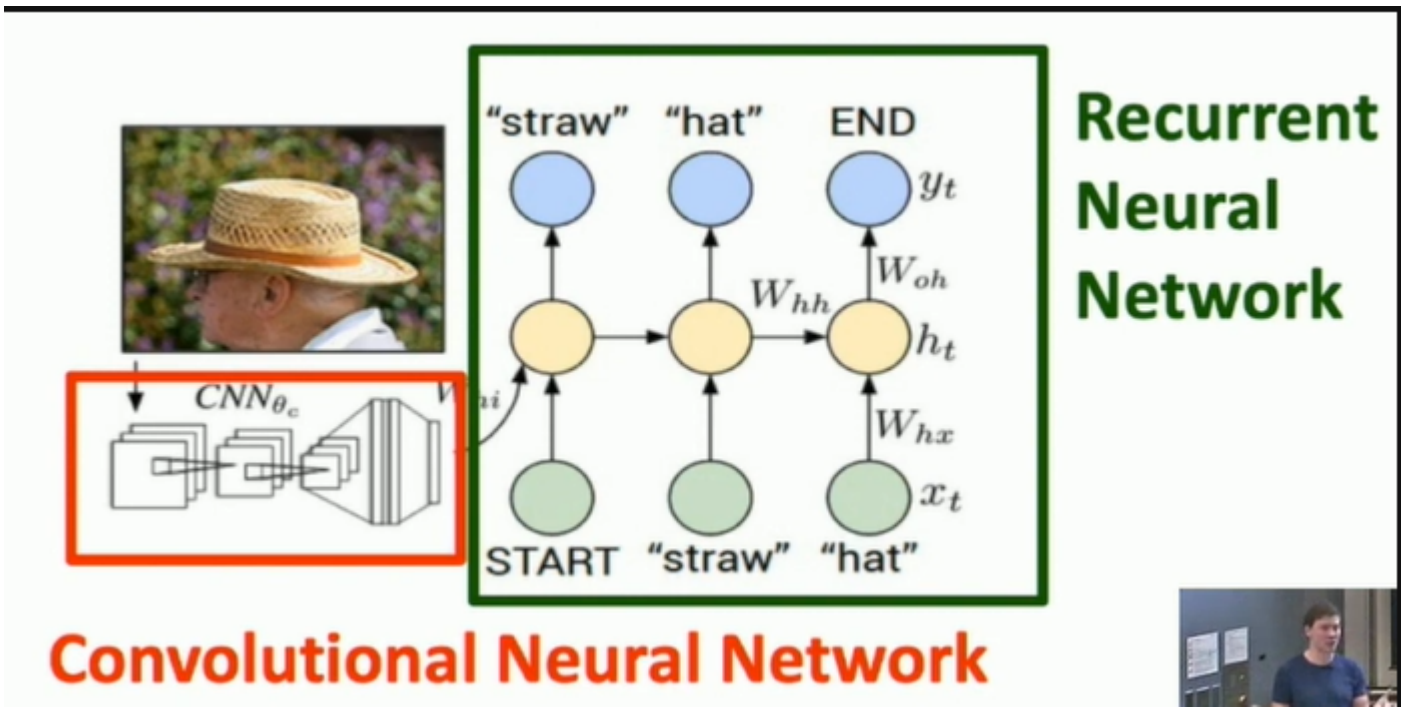
Example. Quote selection cell:



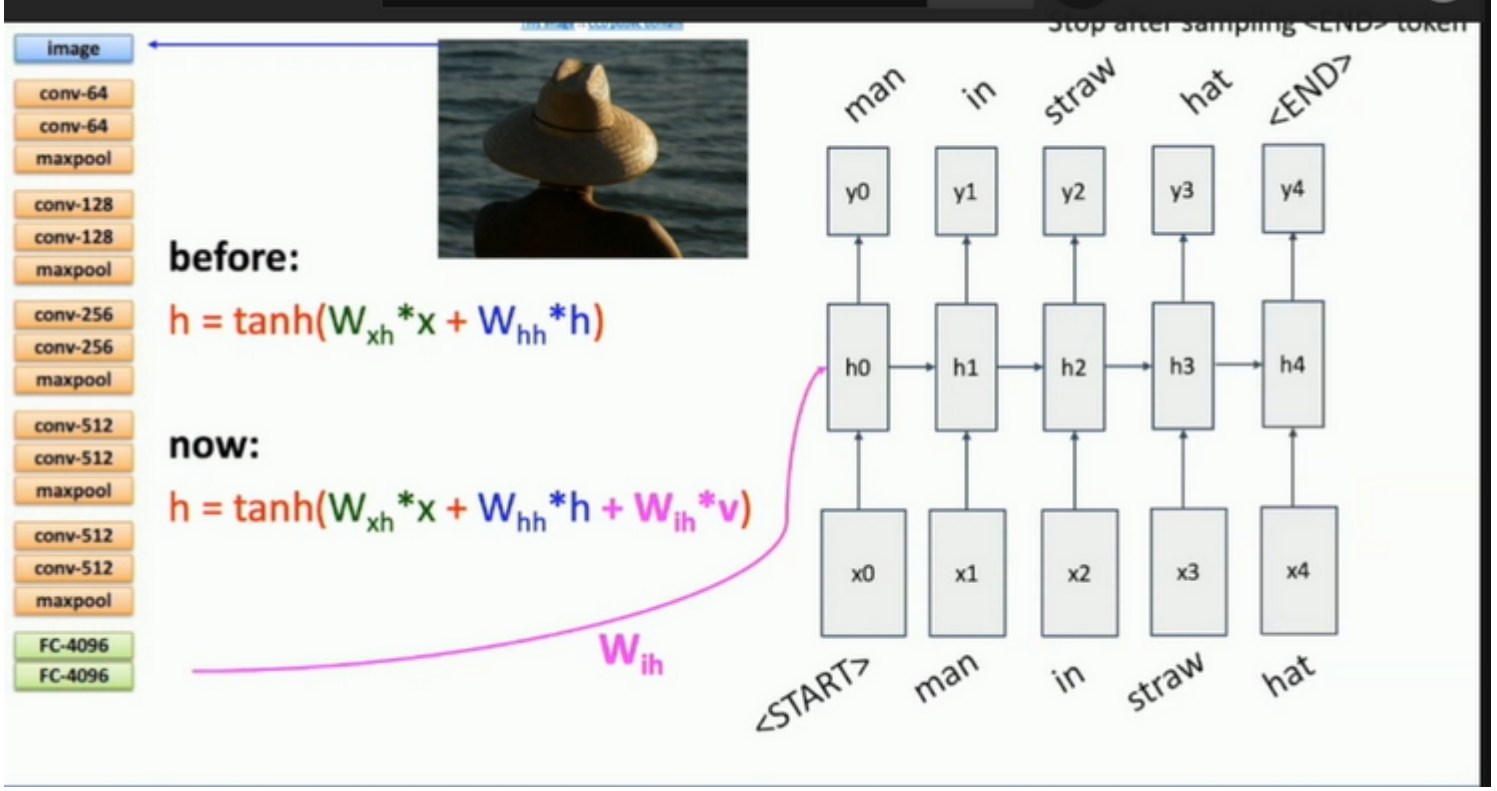
Other examples: inside a comment, length of line, indentation, etc.

70. Example: Image Captioning

That is use CNN with RNN.

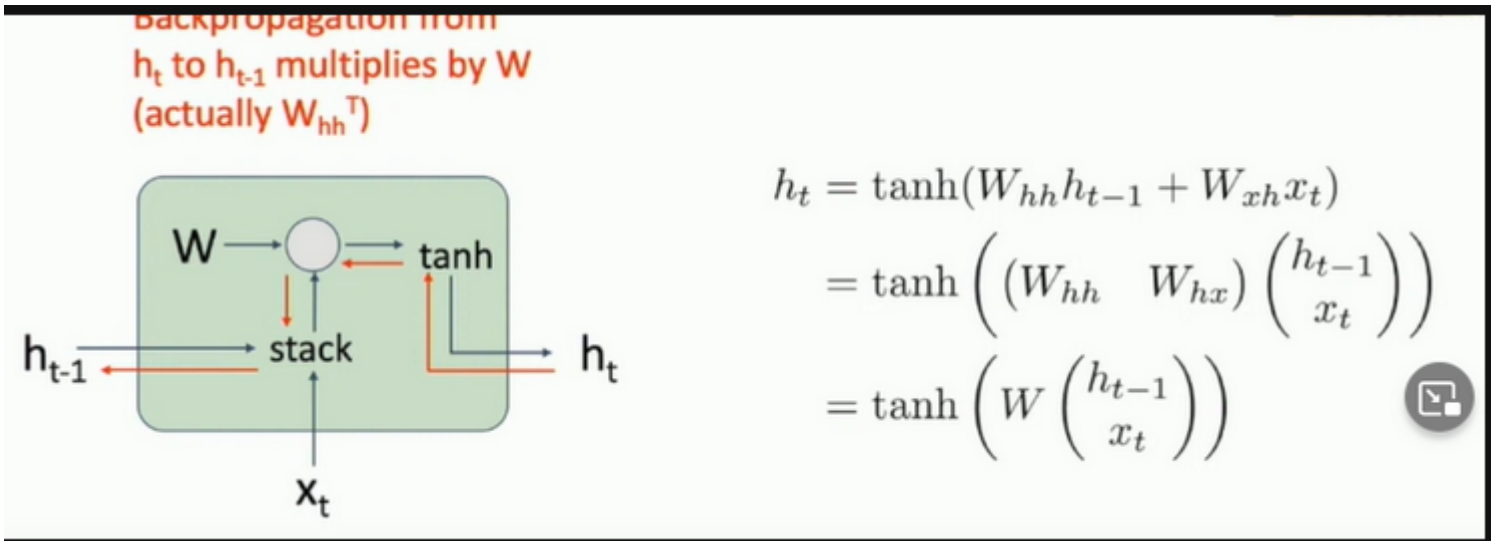


Uses transfer learning. It modifies recurrence formula.



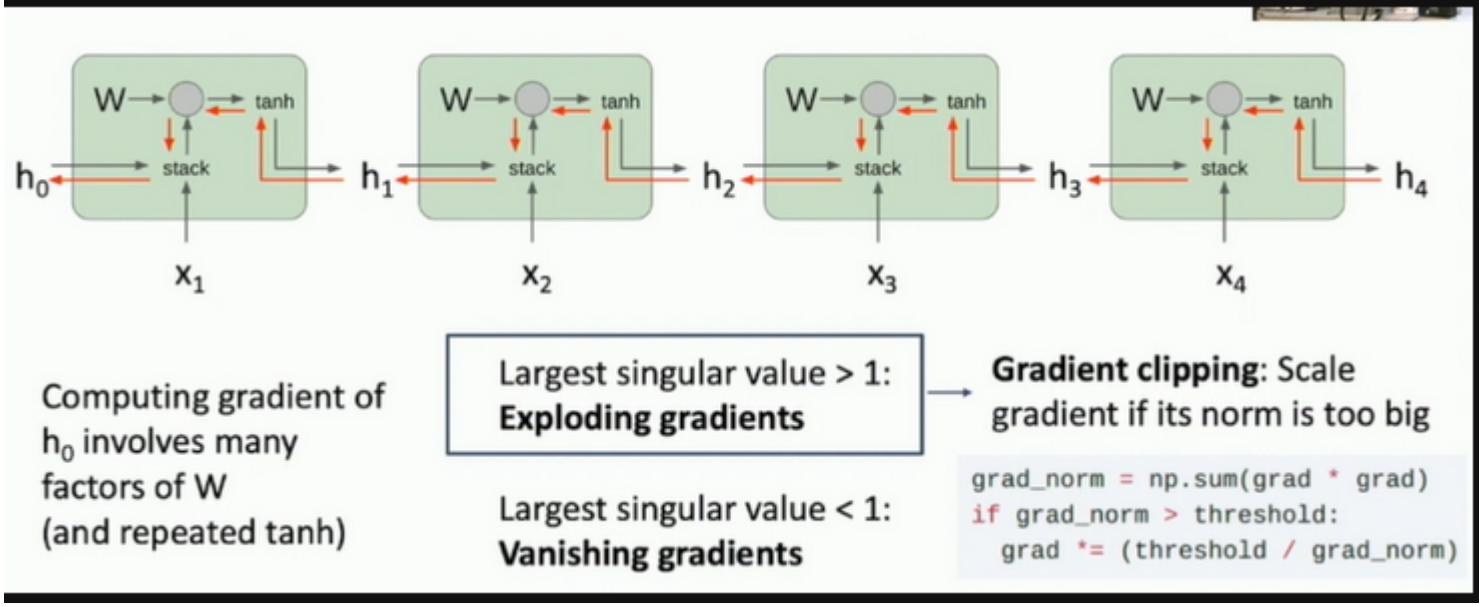
Gives good and garbarish results.

82. Vanilla RNN Gradient Flow.



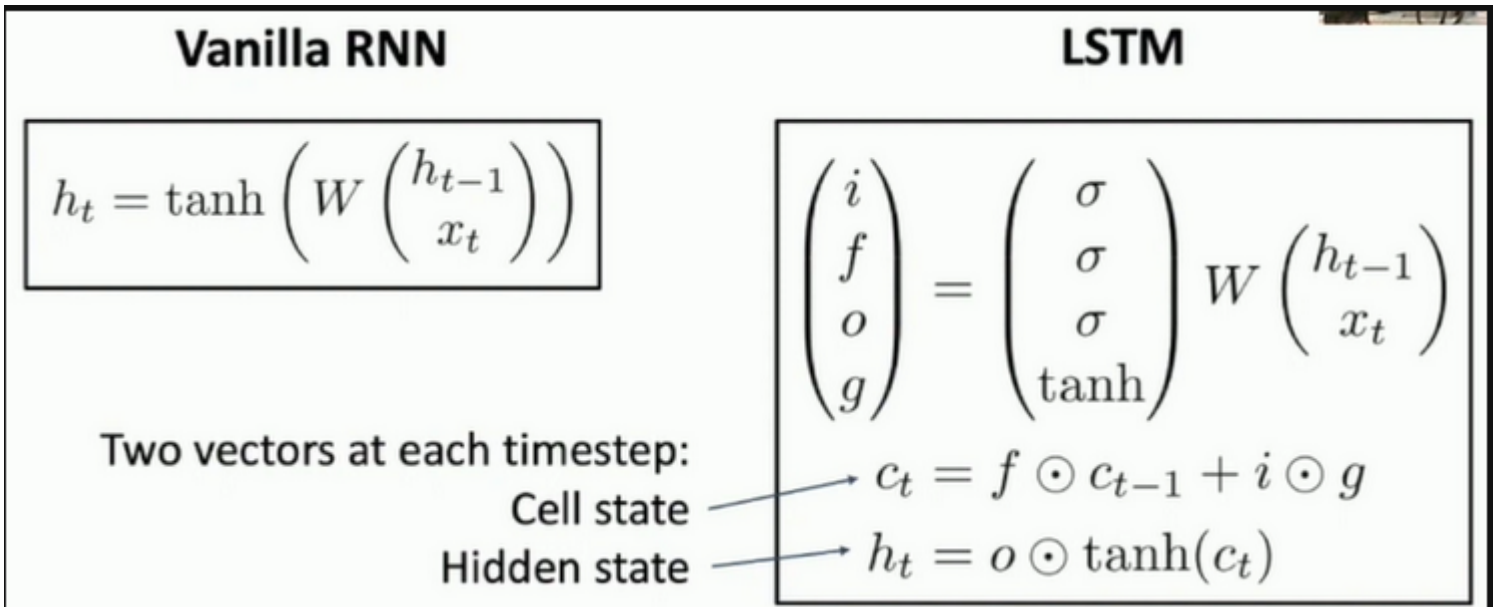
Problems

- $tanh$ not good
- Backprop for W. This will transpose weight. So we will multiply the same matrix thousands or hundreds times. Also if singular value > 1 then exploding gradients. Otherwise vanishing gradients.

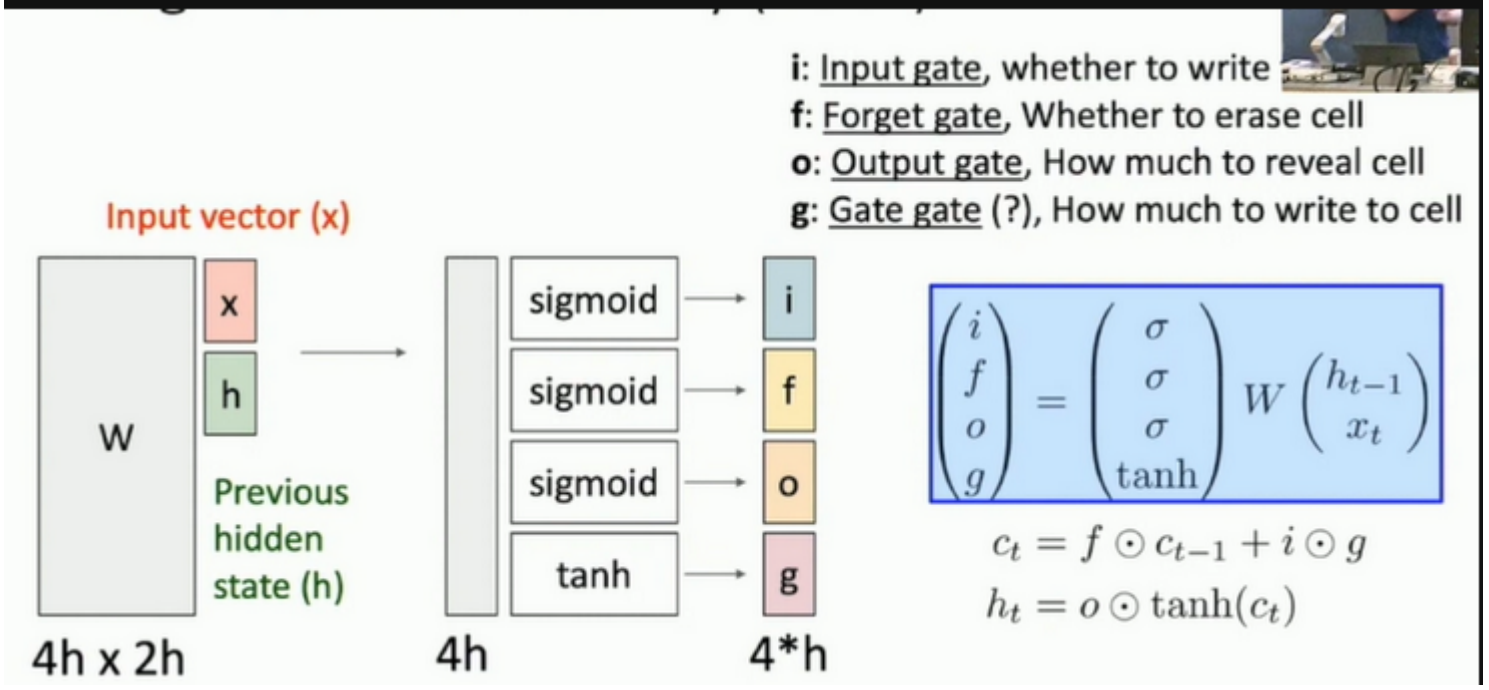


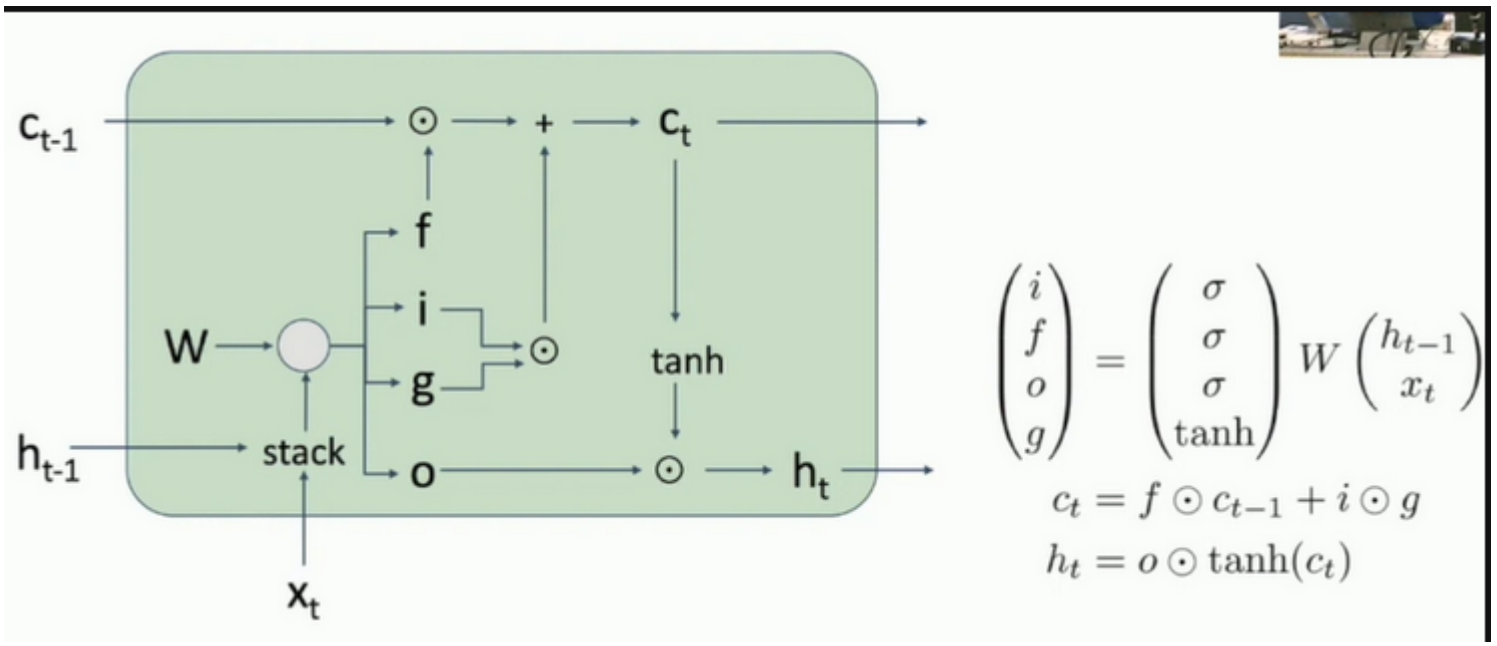
Gradient clipping.
For exploding gradients. So we use gradient clipping. We multiply gradient by a coef to clip it if it reaches some threshold (see picture above).
For vanishing gradients. Throw away this arch and use diff architecture for RNN (LSTM?)

88. Long Short Term Memory (LSTM)



This computes not one gateway but four.





LSTM is to archieve better backprop.

This reminds us about ResNet with those residual blocks. This is same idea with LSTM so it gets uninterrupted flow. Also there is Highway Networks.

98. Multilayer RNNs

It was so far Single layer RNNs - one layer of those h_i. So let's add many layers of those h_i^j. With diff weight matrices for each layer.

101. Other RNN variants.

- GRU - improved LSTM.
- Also tried a brute force a formula for RNN (from 10K formulas).
- Also they tried to search for architecture for RNN.

Questions

7. Types of NN by number of input and output. Their examples.

8. Applying RNN to non-sequential data. Examples.
9. RNN

- What's key idea?
- Formula for a step.

18. Vanilla RNN

- Formula. How many weight matrices we use? How many weights per steps?
- How we compute loss?
- Example for many-to-many.

28. Sequence to sequence.

- Translation. What's encoder and decoder?
- Language Modelling. Predict next char.
    - What's input layer? Output layer?

44. How to train (Language model)

- Backprop through time.
- Examples of for lang models.

63. Visualization (Language model)

- Coloring a text. What color means?

70. Example of RNN + CNN. Image Captioning.
71. Vanilla RNN gradient Flow. What problems?
72. Long Short Term Memory - LSTM

- How many states? Their names.

98. Multilayer RNN

Lecture 13. Attention

5/ Repeating RNN

Recaping to start with "attention" NN. Let's review previously discussed RNN.

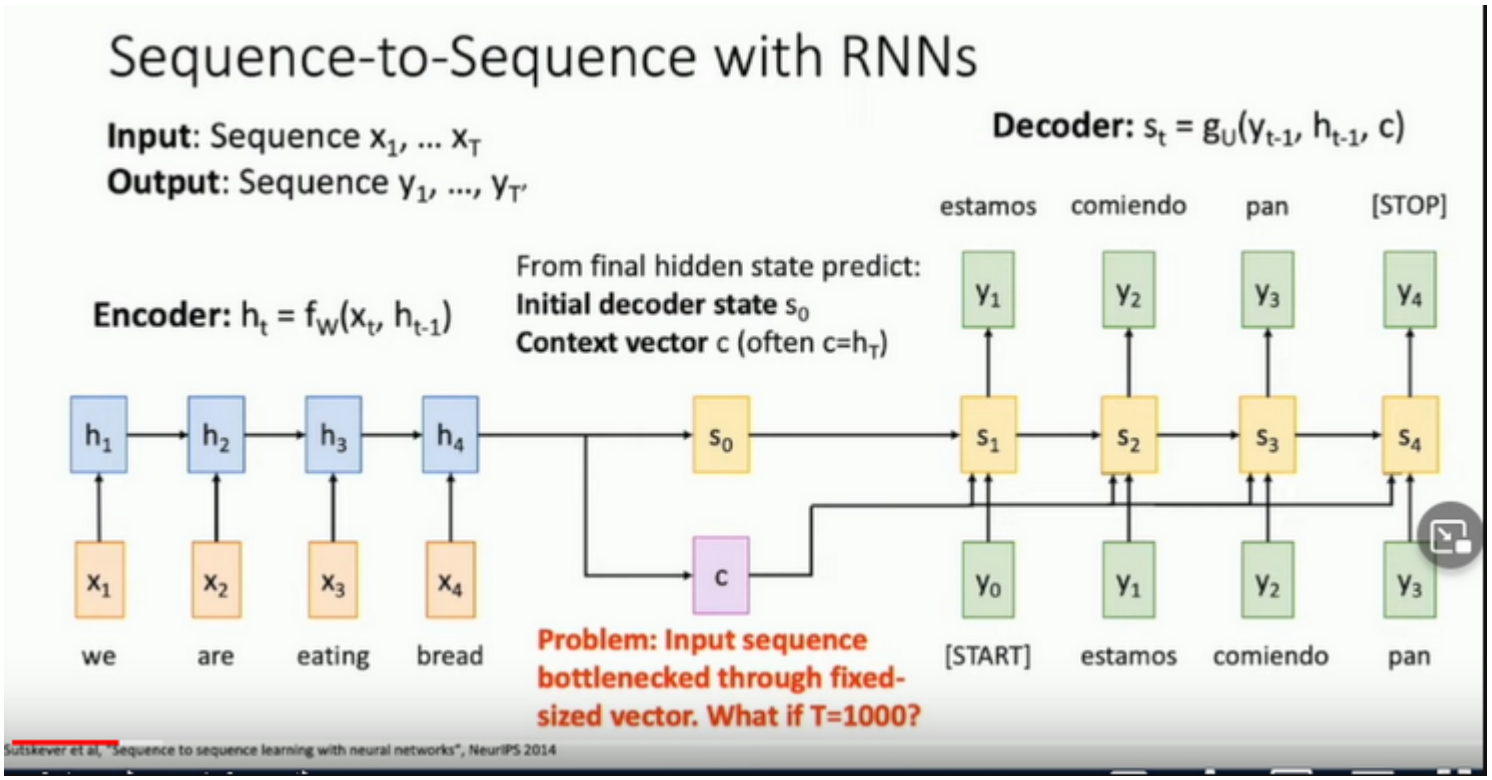
The seq2seq with RNN is:

Input: sequence x_1, .., x_t
Output: sequence y_1, .., y_t

Example. Input might be text in one language and output would be text in other language.

This was done with two NN: encoder and decoder.

Encoder - one NN. It will produce vector of hidden states h_i, when given a sequence of input vectors x_i. When we produced output we want to summarize all output into two vectors: s_0 (initial state for the decoder), c (context vector). Commonly $c = h_t$. Decoder then takes initial state, context vector and initial input y_0 and produces the first word of output y_1. Then it repeats with the y_1. c serves important aim to summarize all info that decorder needs to decode.



Problem is it works only when seq are short. But we want it to translate entire paragraphs or books. And it doesn't work with this small c context vector, this is a bottleneck. So let decoder recompute c vector at every step and it will 'focus' on diff parts of input each step. THis is formalized in 'attention' mechanizm.

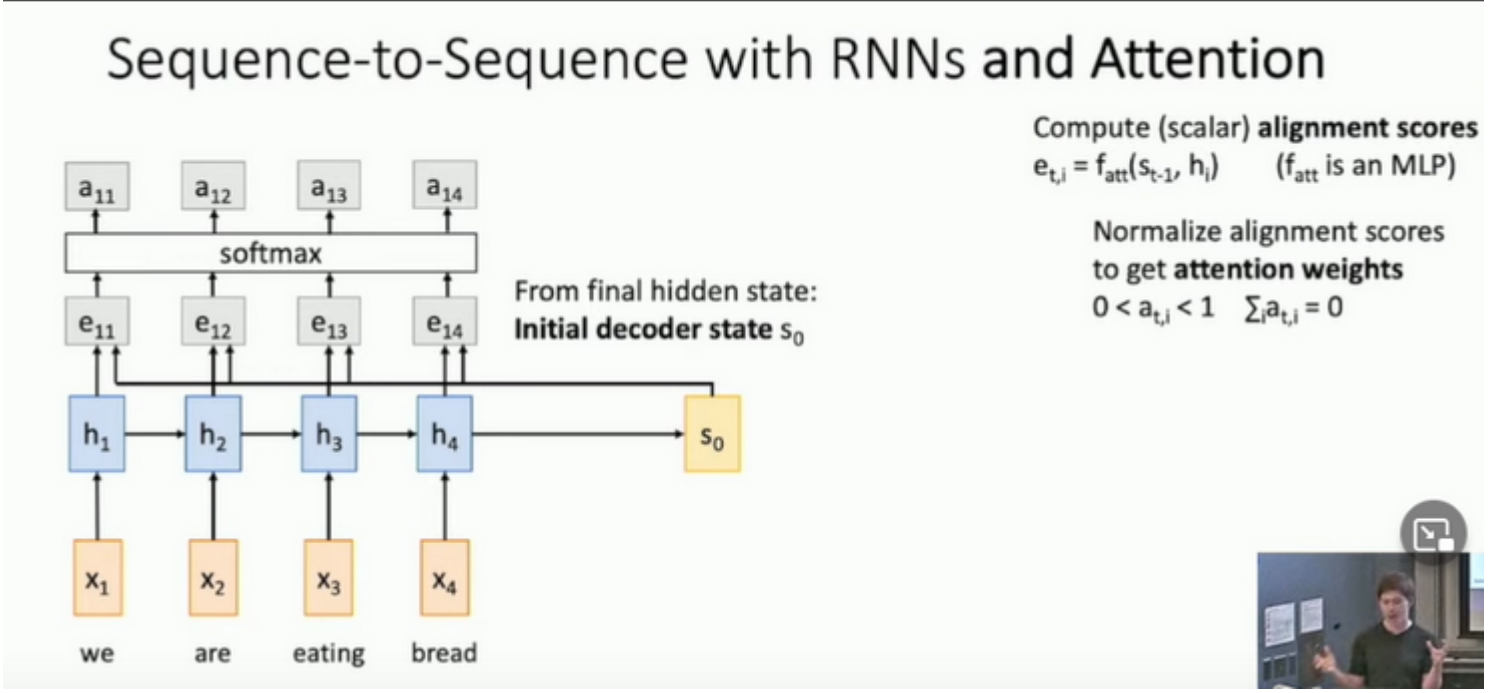
12/

Attention. Still seq 2 seq. This allows to recompute context vector. Let's add alignemetn functions, i.e. small NN, that outputs a score that how much should we pay attention:

$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

It says how much should we pay attention given current state of decoder, s_{t-1} and hidden state of encoder h_i.

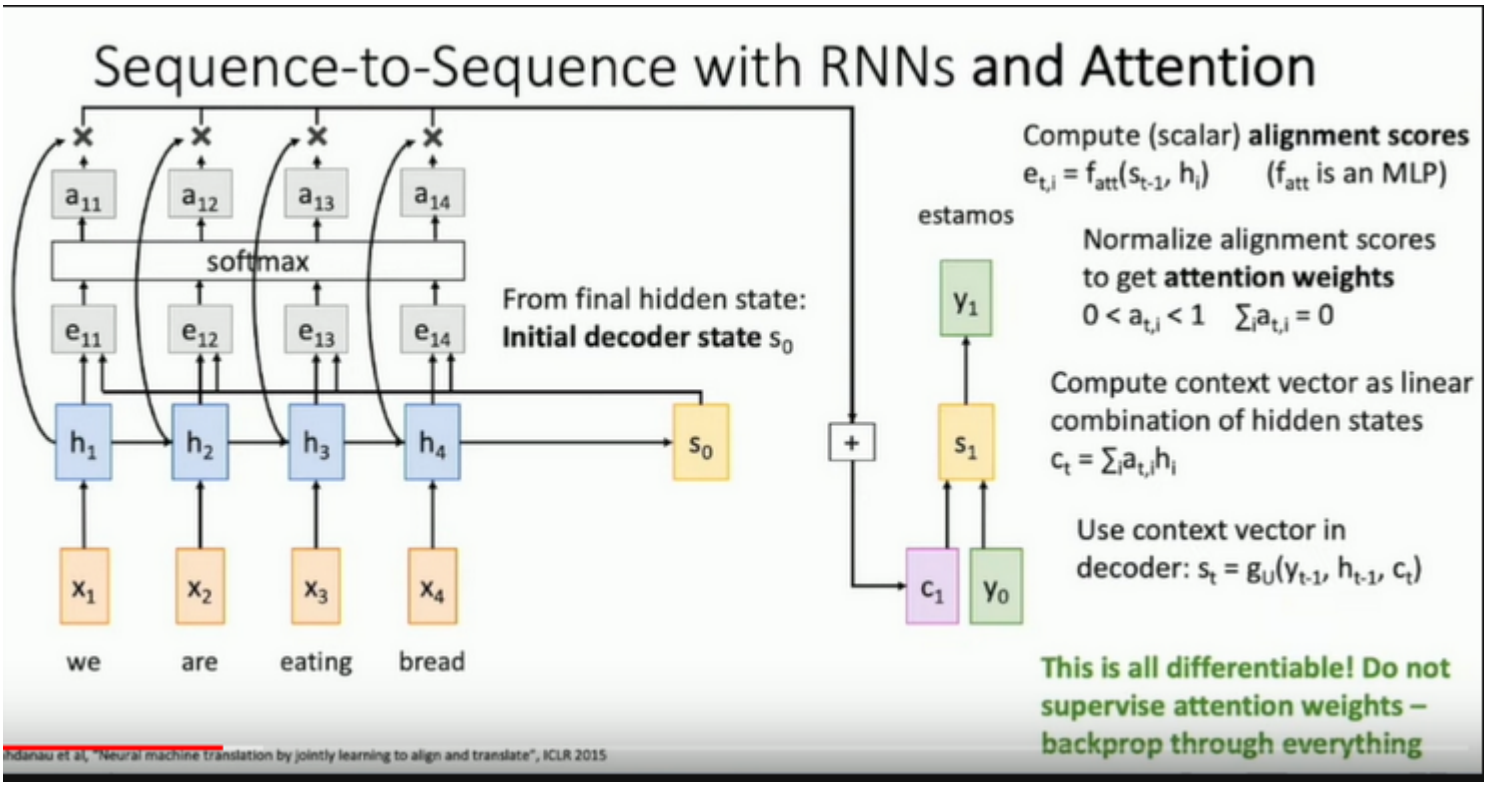
So for the below picture:



$e_{1,1}$ (scalar) is this score of how we should pay attentions at step 1 given hidden state h_1 and initial state for the decoder s_0. Then we convert them to probabilities using softmax func (sum to 1). This distribution is 'attention weights' that says how much weight we should put on each hidden state of encoder given this state of decoder.

Then we compute c_1 vector, context vector for decoder at step 1. As follows:

$c_t = \sum_i a_{t,i} h_i$

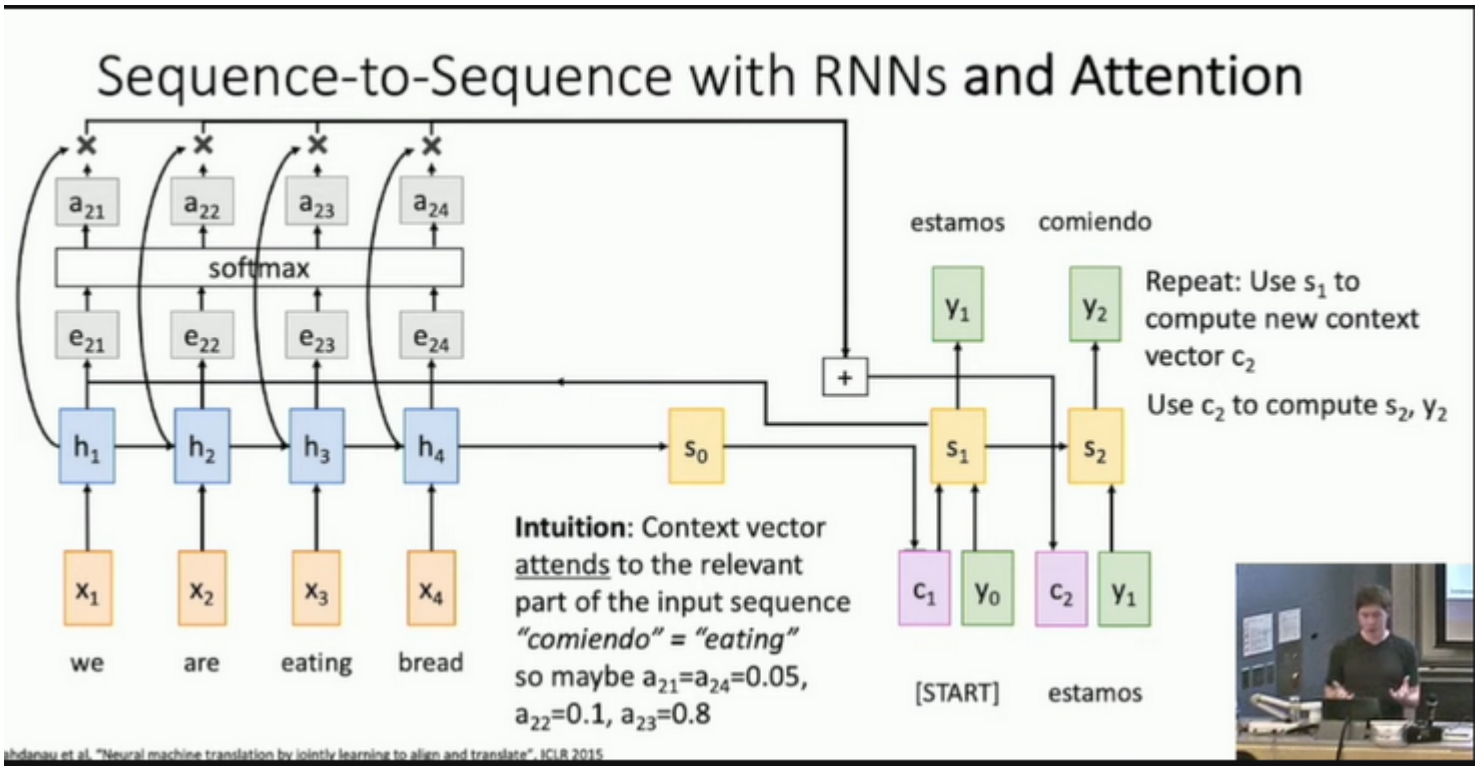


Then the decoder use this c_1 vector to output first token (or word) of the output sequence.

Intuition is that context vector c_i is composed from $a_{t,i}$ (scalar), and it tells if decoder should put more weight to this or that classes (types / words / tokens etc).

This is all differentiable. We let NN decide by itself. We can backprop and compute gradients to let it decide for itself.

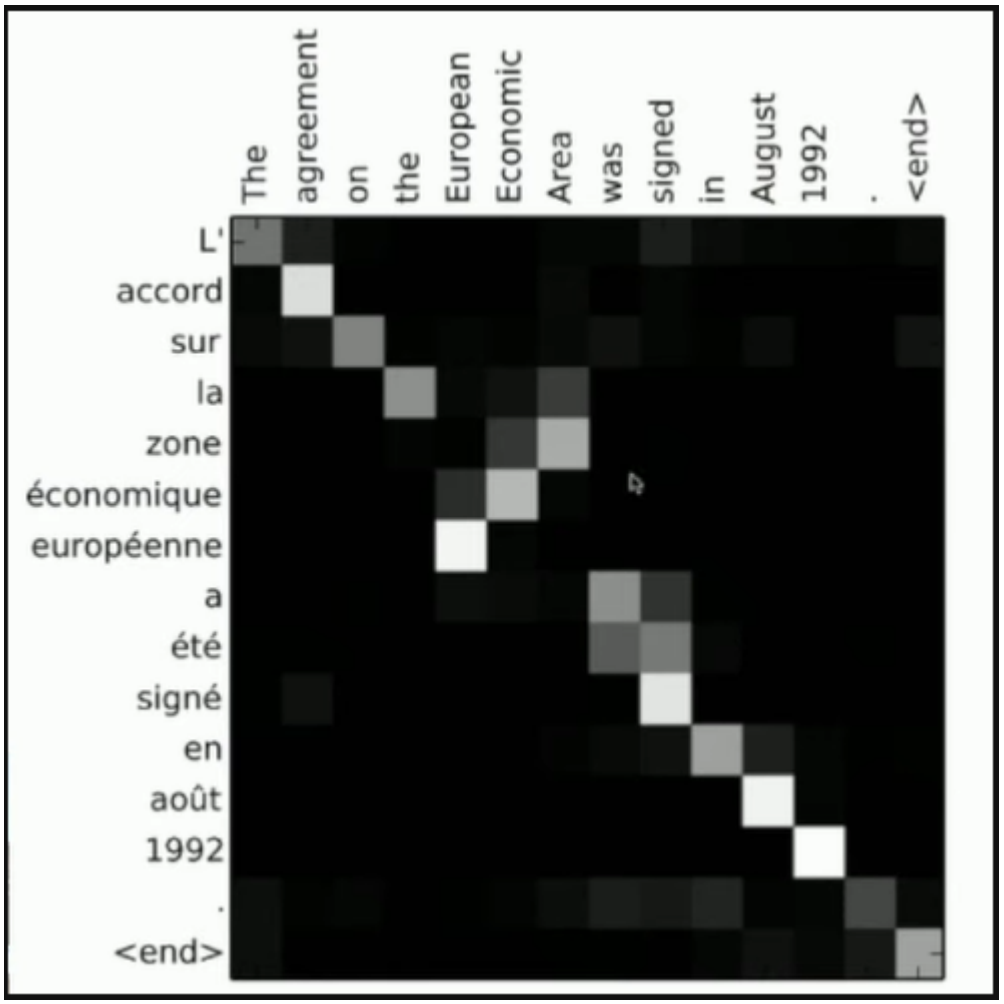
At step 2 we repeat the process to produce second output y_2 via computing c_2 that is computed by using s_1 (encoder uses decoder s_1), from those attention weights. Then it repeats.



Pros:

- Overcome the problem with one 'bottleneck' vector c.
- At each timestep it focuses at diff parts.

Example. From Eng to words in French. Trained seq 2 seq. The table of attention weights:



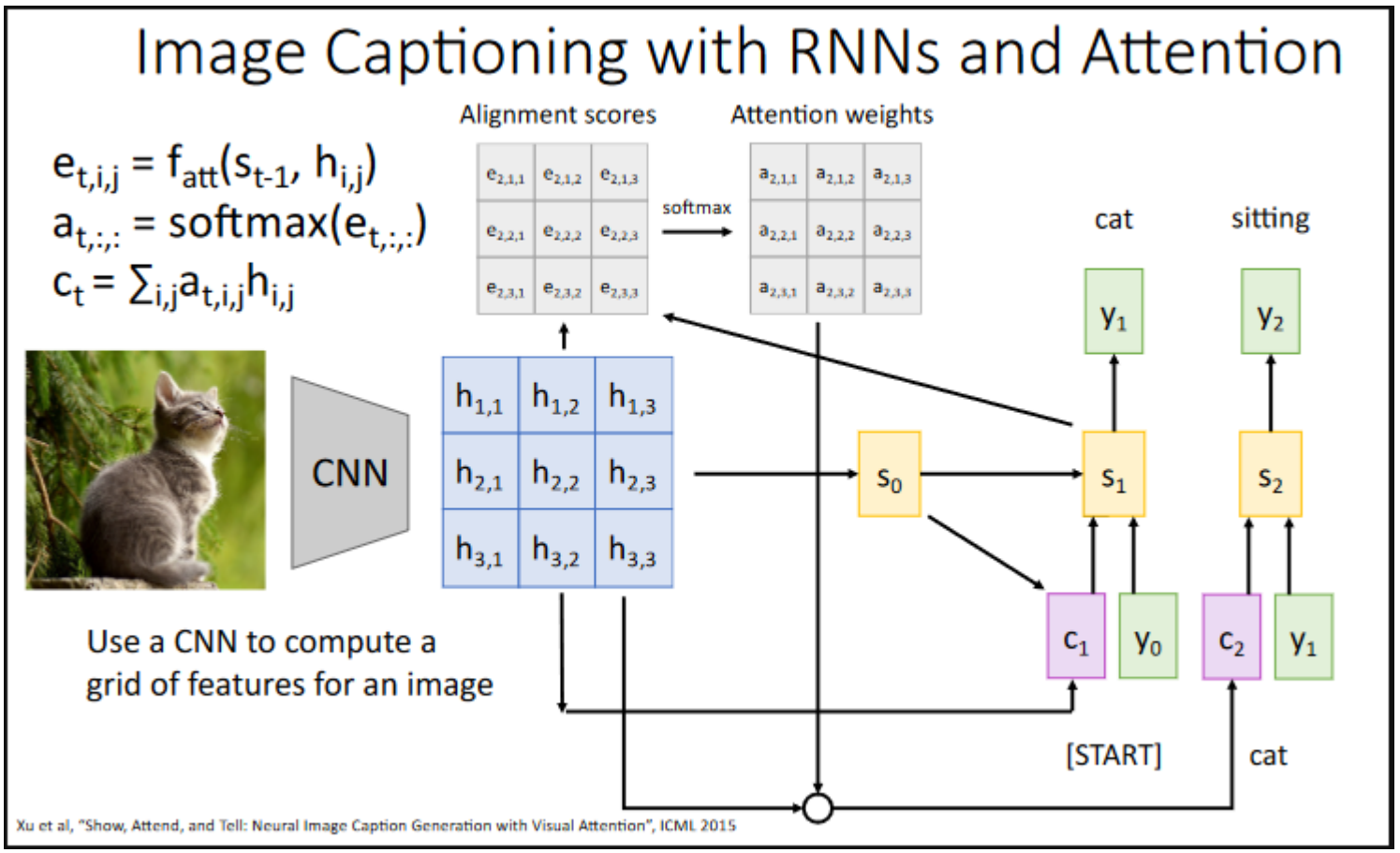
The above shows what weights were at each step for each word it produced. So what words correspond to what. It learns those itself! Model figures out it itself. It tells us how model makes decisions.

Cons: Attention mechanize doesn't know the fact that input is a sequence. So we can do it for non-seq input.

27/

Attention for non-seq input. Image Captioning with RNNs and Attention.

First we generate those hidden states: $h_{i,j}$ using CNN, the final grid of feature vectors. On top of it we use attention mechanizm to compute $e_{t,i,j}$, i.e. $f_{att}(s_{t-1}, h_{i,j}) = e_{t,i,j}$. And then convert to probabilities using softmax. See image below:



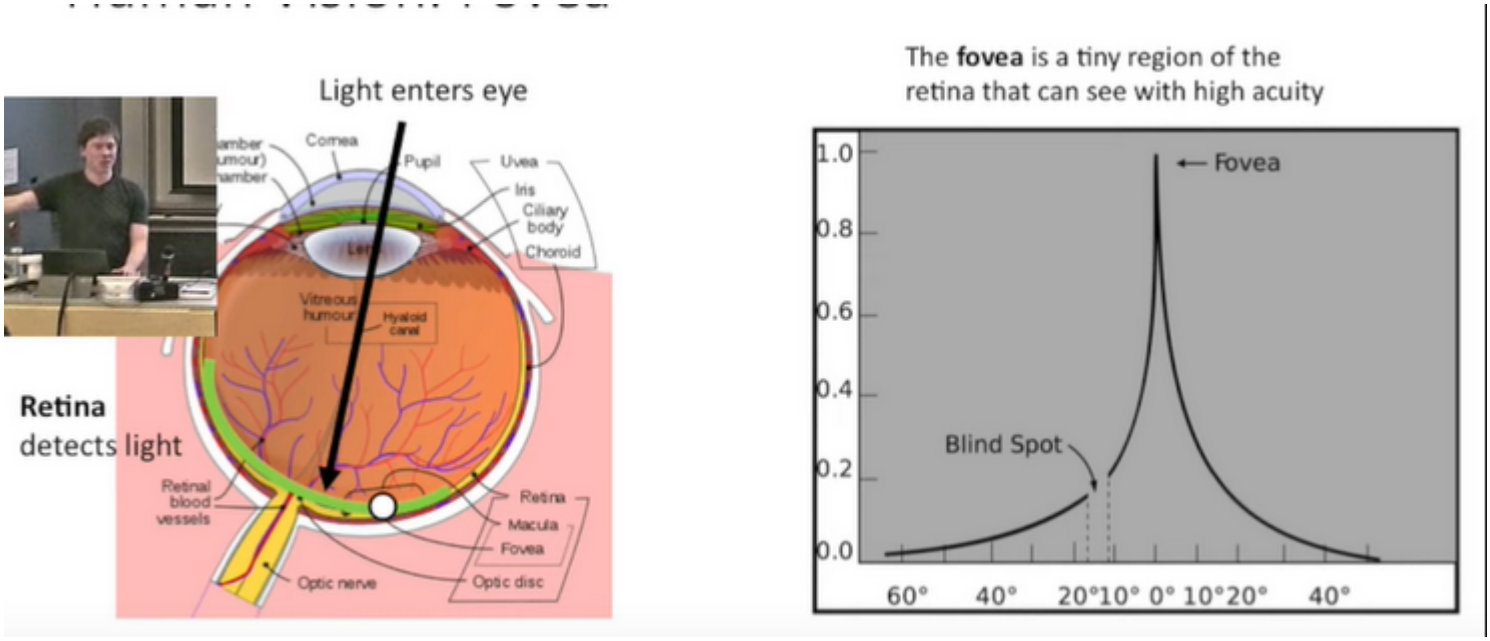
So we have attention weights for each part of the input. We then produce c_1 to get first word. Then repeat. Each time it looks at diff part of image.

Q: Can model select a subset of features from image? A: Can but it differs from current topic.

Examples of attentions:

40\ Intuition behind attention

Human Vision: Fovea. Projected at retina. Retina has very small region that have high sensitivity (fovea) but others don't. Hence eyes constantly moves around to fix it - saccades mechanizm. So 'attention mechanizm' is simular that it focuses very rapidly at diff areas.



45\ X, Attend, and Y.

First paper: "Show, attend, and tell". Other would catch up this form: "X, Attend, and Y". "Listen, attend, and spell", 2016. Etc.

So often when we convert some info to other data we might use this attention.

46\ Attention Layer

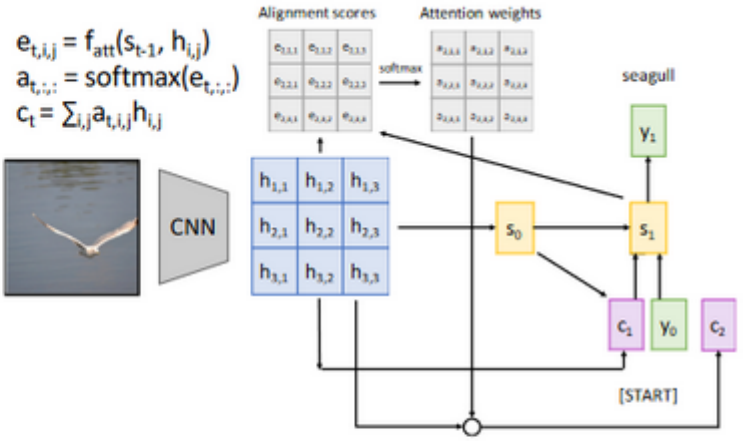
Generilizing this mechinism. Let's reframe this mechanizm.

Inputs (see below):

- Query vector q, like $h_{i,j}$ as before, i.e. intermmediate hidden states we computed recursively from input in RNN.
- Input vecotrs: X, corresponds to a set of vectors we want to attend (hidden vectors when it was on top of ConvNet).
- Similarity func, f_{att} is used to compare q with each of X.

Attention Layer

Inputs:
Query vector: q (Shape: D_Q)
Input vectors: X (Shape: N_X x D_X)
Similarity function: f_{att}



Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(q, X_i)$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i X_i$ (Shape: D_X)

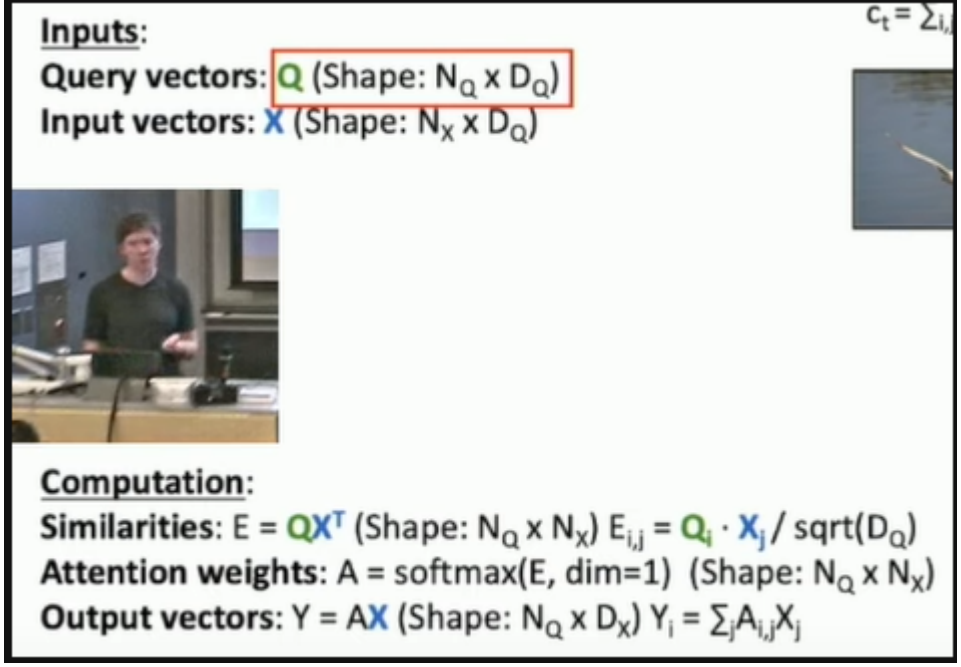
And it outputs:

Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(q, X_i)$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i X_i$ (Shape: D_X)

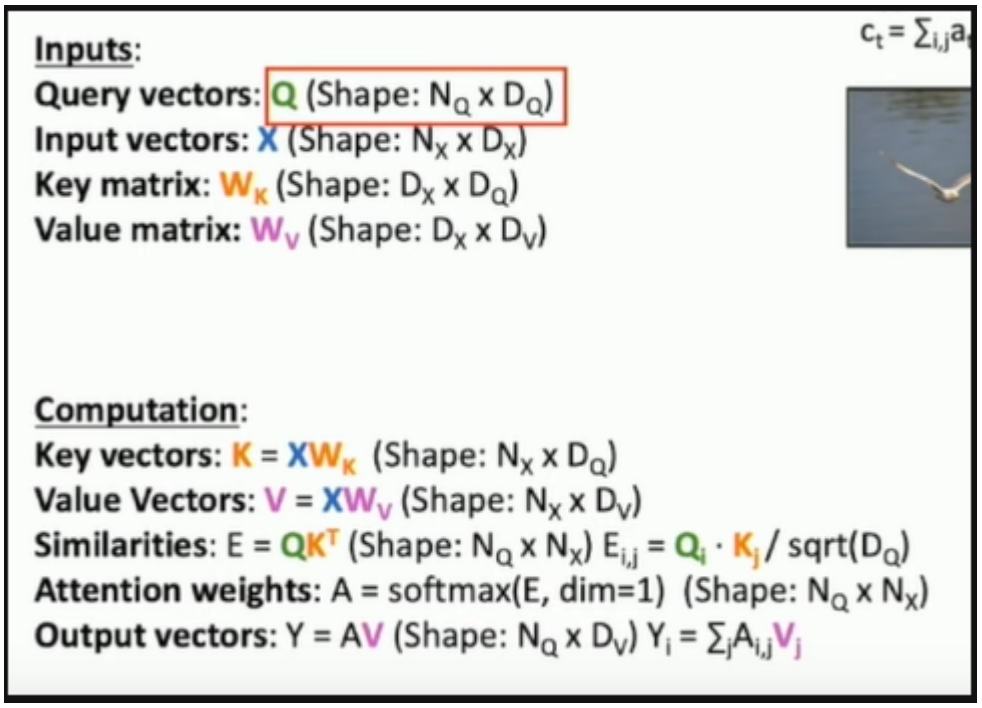
e - vector of simularities.
a - probabilities (as before) aka attention weights
y - output

How to generilize? Generalizations:

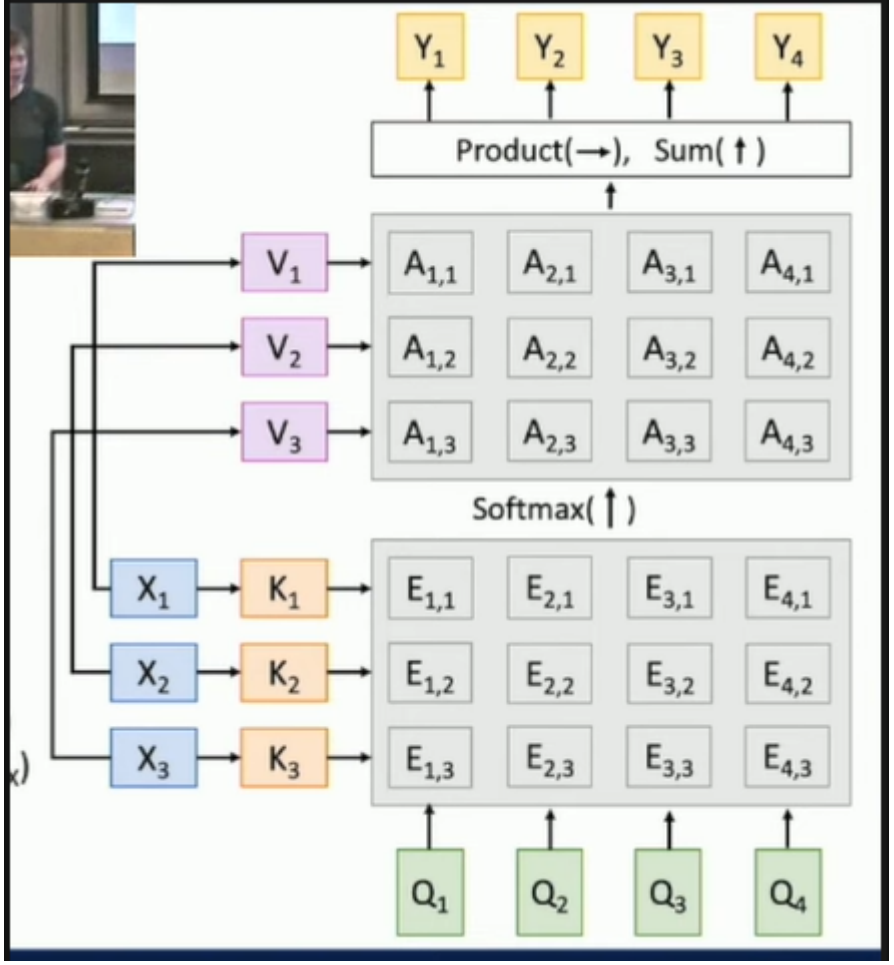
1. Generalization 1. Use dot product.
2. Scaled dot product: $qX_i/sqrt(D_Q)$. Why? Because of the vanishing gradients problem. So learning is challenging. Remember that dot product uses norm of vectors.
3. Multiple query vectors. Generate attentions for a set of queries. And we can get similarities for all those queries over all input vectors. And so for softmax and output vectors:



4. Separating input vector into key and value vectors because we used X for two diff operations (to get attention weights and to get output). And make them learnable. See image below. Why? More flexibility.



Example of this last generalization:

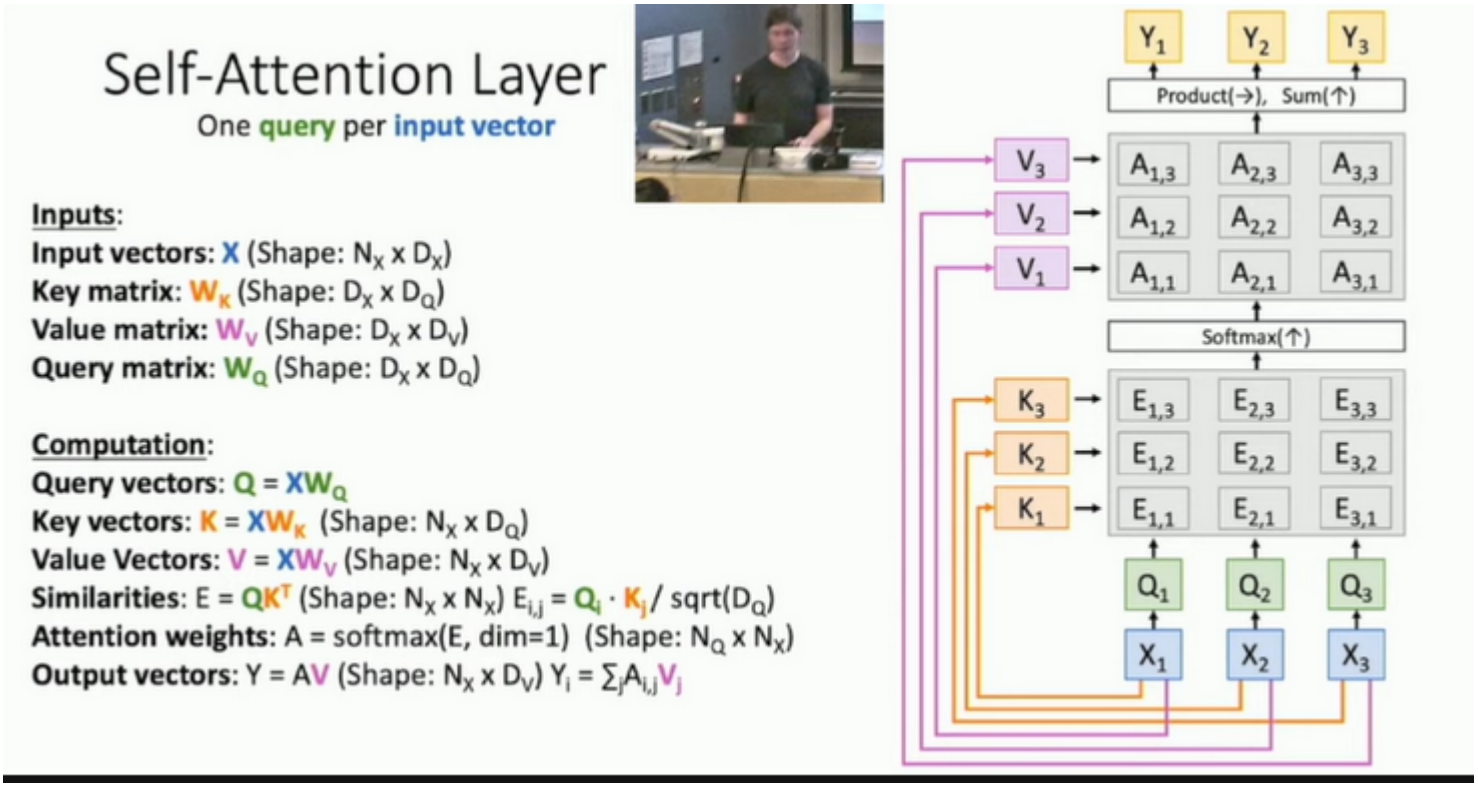


So we got very general attention layer.

59\ Self-attention Layer

This is a special case of attention layer. One query per input vector. Now we have only one input vector without query vector. So we compare each input vector with every other input vector.

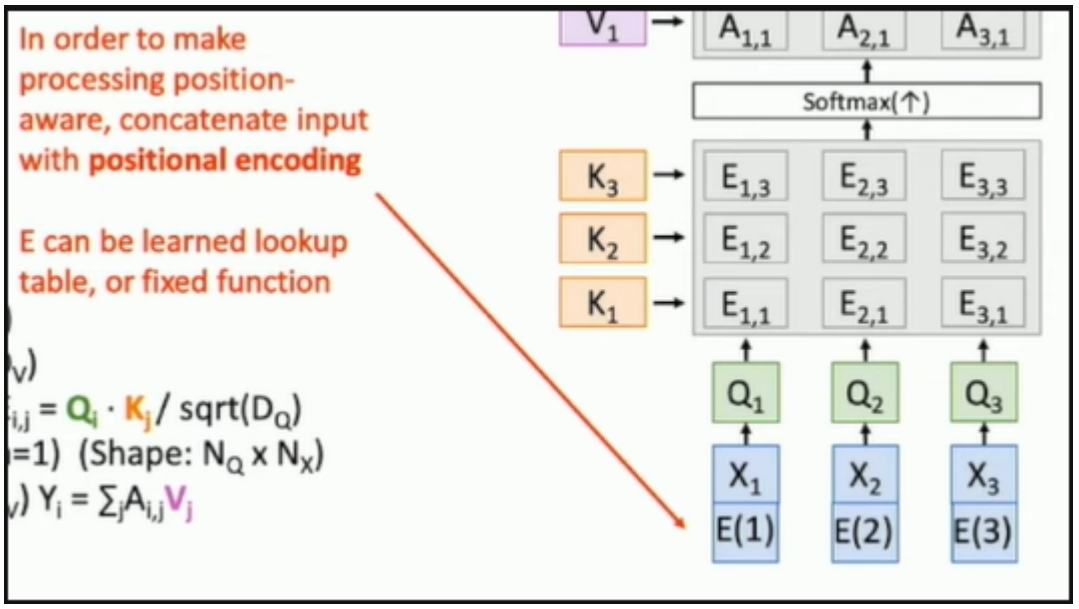
First we convert input vector to query vector. Then ... See below:



This is a very general mechanize.

What if we change order of input vectors? It will give the same values but reordered, for query vector, for attention layer, etc. That means our layer is permutation equivariant (indifferent to permutations).

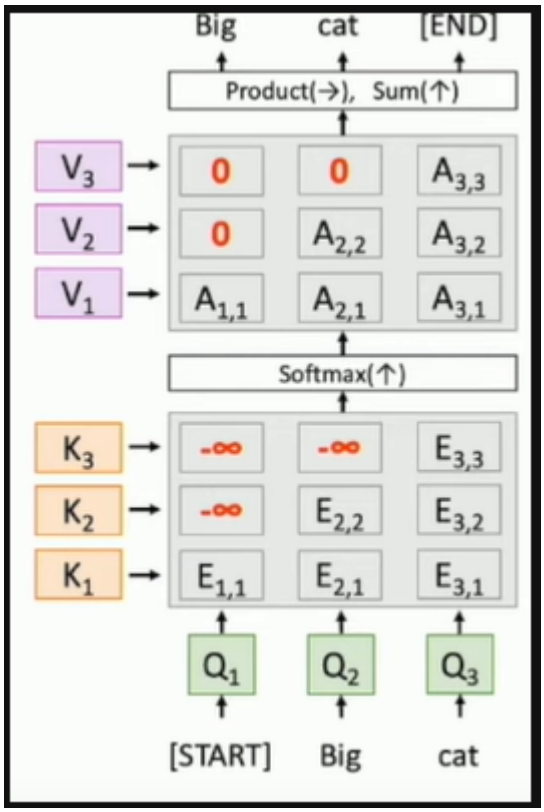
Problem is it doesn't know the order. How to make it know the order? So make a hack: concatenate with positional encoding:



Now it knows what's beginiing, left / right, etc.

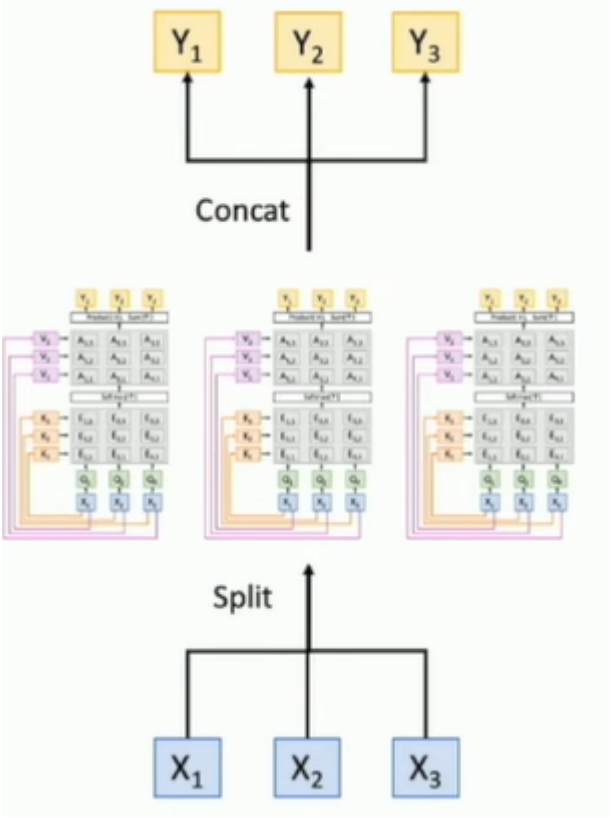
75\ Masked Self-Attention Layer.

To force a model not to use all the model. Use only information from a 'mask'. This is for language model, when we want a model to use only info from masked, only depend on some subset. So the model doesn't look ahead.



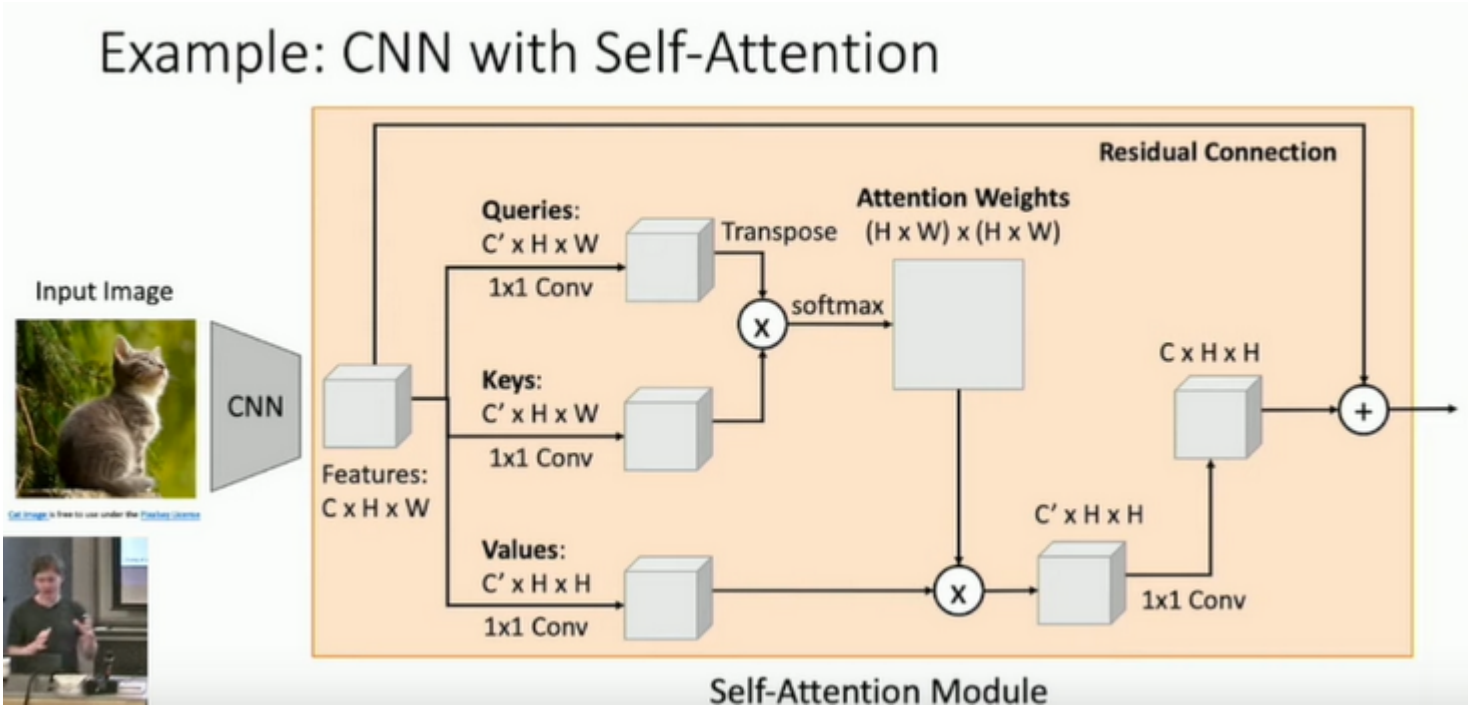
77\ Multihead Self-Attention Layer

Using independent 'attention heads' in parallel, i.e. separate attention layers.



78\ Example of self-attention layer

First use CNN to create three sets: queries, keys and values. Then transpose and get attention weights. Then we use linear combination of attention weights and values to produced weighted values, how much we weight each position in input. Also it is commmon to add residual connection and conv layer. This is a new type of layer.



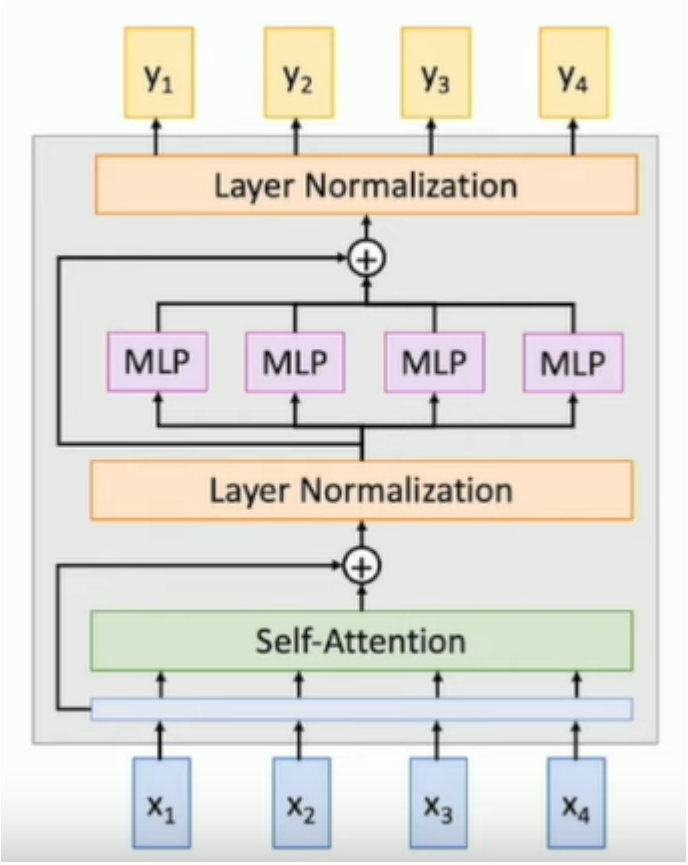
84\ How to process sequences?

1. Recurrent NN: (+) Good for long seqences. (-) Not parallelizable.
2. ConvNets. 1D Convolution to process seq. And we slide it. (+) highly parallel. (-) Bad on long sequences, need to stack those conv layers.
3. Self-attention. it overcomes prev minuses. (+) Good for long seq. (+) Highly parallel. (-) Takes a lot of memory.

87\ How can we combine those? "Attention is all you need". Transformers.

To process seq, we can use only self-attention mechanizm.

How transformer block works. Receive X, then run through self-attention. Then redisidual connection. See below:



where

- Self-attention layer is the only place where interaction between vectors.
- MLP - feed forward layer, multi-layer perceptron. per each vector independently.
- Layer Normalization per each vector independently.

(+) Highly parallizable, scallable.

And transformer model is a sequence of transformer blocks.

Significance of it? It is like ImageNet for NLP.

Can be applied in many tasks in NLP.

Labs compete with bigger and bigger models. Scalling up and up. GPT-2, RoBERTa, BERT-Base, BERT-Large, Megatron-LM, etc.

As they bigger they are better.

We can generate text from transformers. Given text it produces other text.

Questions

5\ How RNN works for language translation, many-to-many, decoder, encoder.

- What's bottleneck?

12\ Attention mechanism.

- What it computes? How computed? How context vector computed? How output computed?
- What's intuition behind this attention weights?
- What are pros?
- Example with language translation. Visualizing attention weights.

27\ Attention for non-sequencial data.

- On top of CNN. How?

40\ Intuition behind attention. Human vision. Retina and fovea.

46\ Attention Layer. Generalization.

- What we have as input, as output.
- Ways to generilize:
  - Dot product
  - Scaled dot product
  - Multiple query vectors
  - Separation of input vector into key an value vectors.

59\ Self-Attention Layer

- What's difference from attention layer?
- Order of input vector. How to learn order?

75\ Masked Self-Attention Layer

- Idea? Used where?

77\ Multi-head Self-Attention Layer

- Idea? Why?

78\ CNN with Self-Attention

- Self-Attention Module.
- Queries, keys, values.
- Attention weights
- Residual connection

84\ In general how to process sequences?

- Using RNN. Upsides / downsides?
- Using CNN. Upsides / downsides?
- Self-attention. Upsides / downsides?

87\ Transformer. Attention is all you need.

- Transformer block constituents.
- How significant is it?
- The bigger the better.

==========

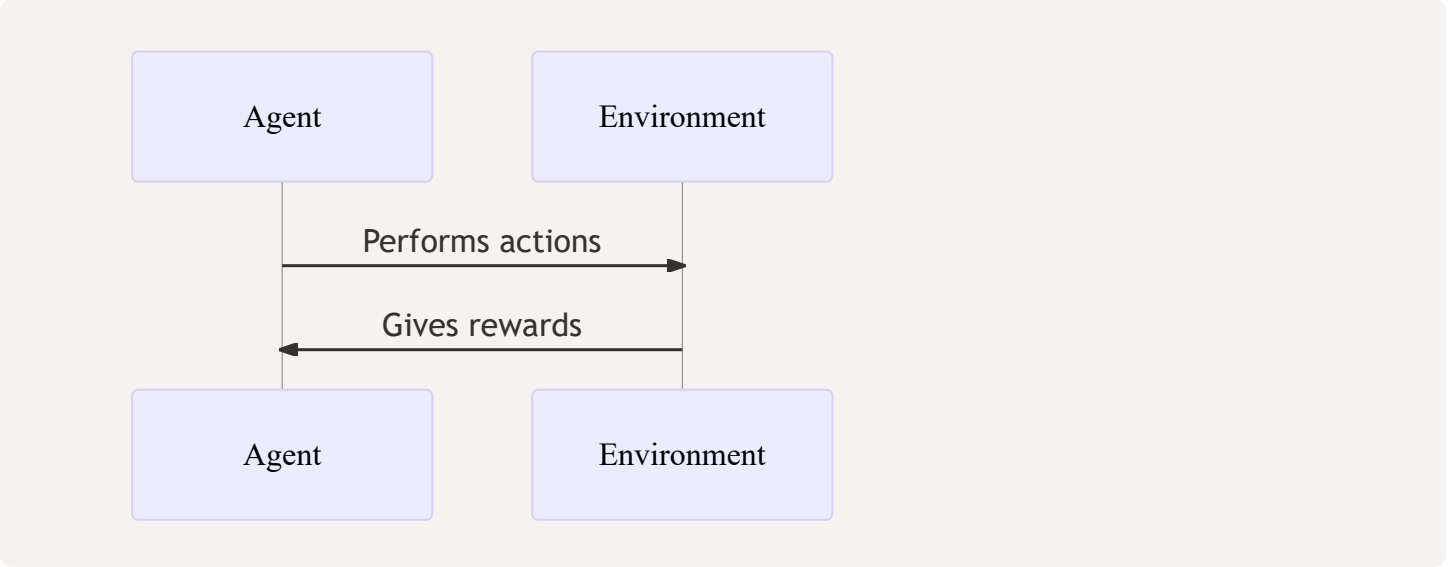
Lecture 21. Reinforcement Learning

4\ Recap.

- Supervised learning. With labels.
- Unsupervised learning. No labels.
- Third paradiam: Reinforcement Learning

6\ RL

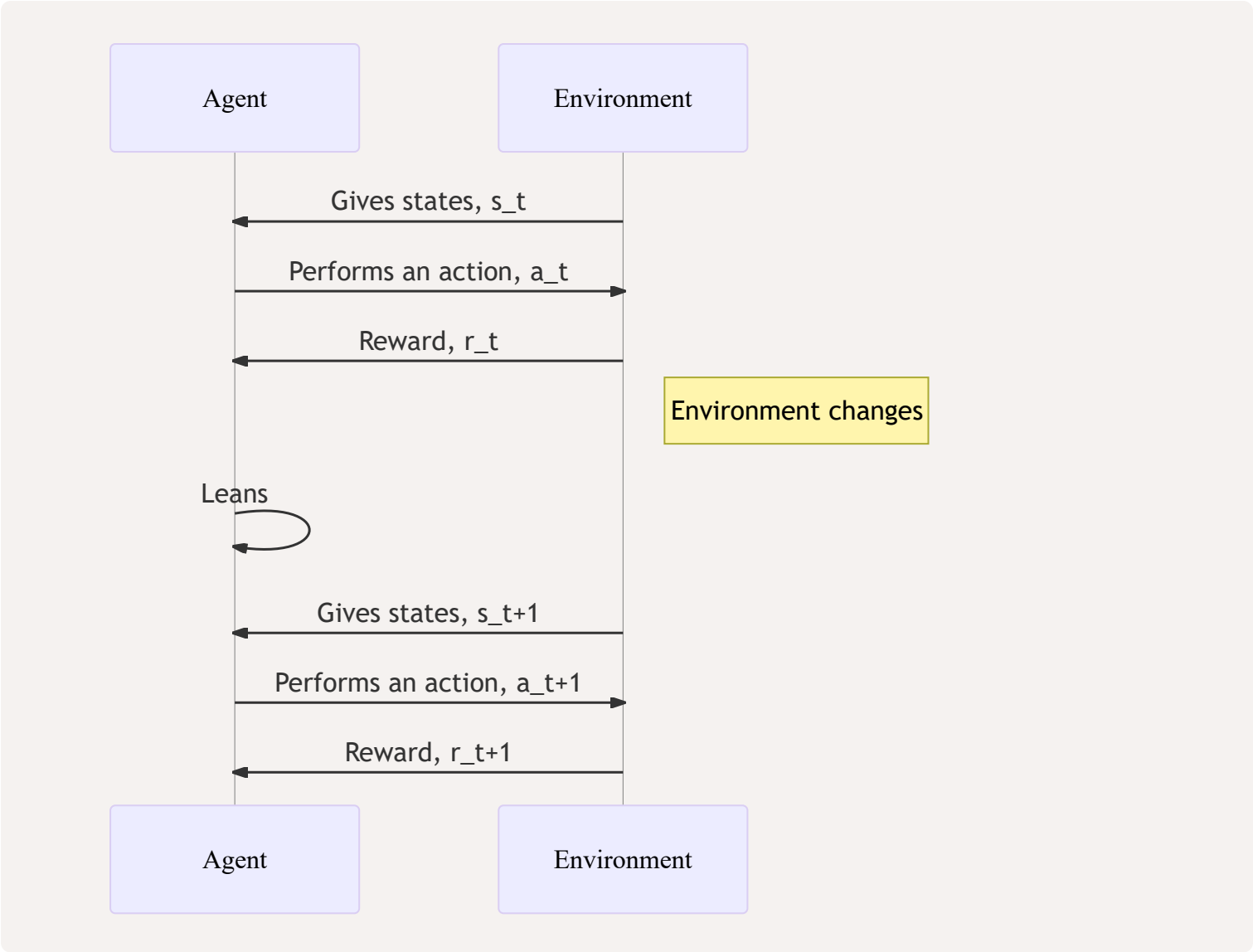
- An *agent* performs *actions* in an *environment* and gets *rewards*. Then it repeats.



- Goal: maximize reward.
- Lecture topics: Q-Learning, Policy Gradients.

10\ What's RL?

It is a model for interaction of an agent with environemnt.

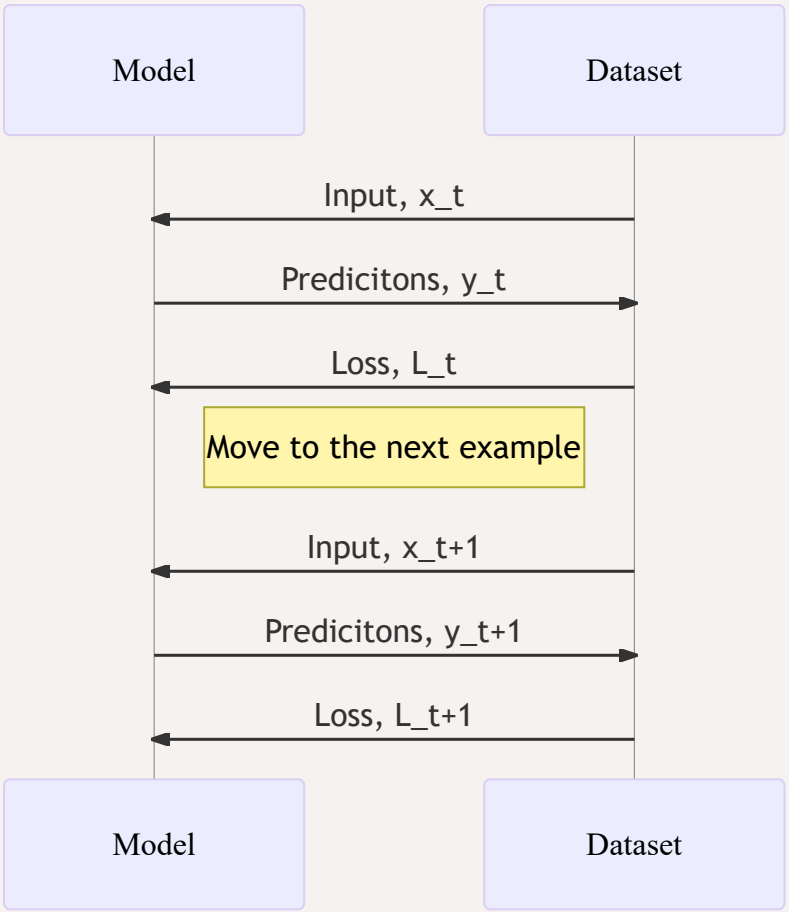


Examples for RL:

1. 14\ Cart-Pole Problem.
    - Objective is to balance a pole on top of a cart.
    - State: angle
    - Action: moving left or right
    - Reward: 1 each time if on top.
2. 15\ Robot Locomotion
    - Object: make it move forward
    - State: joints positions
    - Action: apply force to joints
    - Reward: not fall over, how far learns to move.
3. 16\ Atari games
    - Highest score
    - State: pixels on screen
    - Actions: pressing buttons
    - Reward: scores
4. 17\ GO. Two player game now.
    - Object: win

- State: pieces positions
- Action: where it plays a piece
- Reward: Can be at the end, 1 if wins, 0 if not.

18\ RL vs Supervised Learning:



Differences. RL is fundamentally different!

1. Stochasticity. Rewards and state transitions may be random. Underterministic if the same action and same state.
2. Credit assignment. Reward r_t is not connected to a_t. Reward can be dependant on very long actions chain and states. But in Supervised it is determenistic.
3. Non-differentiable. We can't backprop through world.
4. Nonstationary. Agent experiences depends on actions. Example: Coffee robot delivers coffee from other place, etc. Nonstationary because state changes! In Supervised learning state didn't change.

RL is much more challenging. So try to reframe problem not as RL then it is easier.

Math for RL

24\ Markov Decision Process (MDP)

Model for RL.

$$(S, A, R, P, \gamma)$$

- S - set of possible states

- A - set of actions
- R - distribution of rewards for (S_i, A_j)
- P - distribution of transitions for (S_i, A_j)
- γ - discount factor (future vs present). How far we prefer reward now vs in future. $\gamma \in [0, 1]$
- *Markovian Property*: s_t describes the whole world. r_{t+1} and s_{t+1} depends only on the current state, s_t.

MDP: Agent executes a policy π. Goal is to find $\hat{\pi}$ that is $max \sum_t \gamma^t r_t$, where γ^t is a discount, reward in future are less valuable, inflation for environment.

General algo:

- t=0 environment samples init state $s_0 \sim P(s_0)$
- Until objective met, or done:
  - Agent chooses an action using its policy: $a_t \sim \pi(a|s_t)$
  - Environment produces reward: $r_t \sim R(r|s_t, a_t)$
  - Environment produces next state: $s_{t+1} \sim P(s|s_t, a_t)$
  - Agent receives r_t and s_{t+1}

Example of MDP: Grid World. Grid. Moves in cell. Goal reach 'start' in as few moves as possible. Policy A: move up or down randomly; it's bad. Policy B: optimal policy, shortest dist.

Finding *Optimal Policies*. It is to find such policy that maximizes sum of rewards. We max *expected* value of rewards becuase of randomness world:

$$\hat{\pi} = argmax_{\pi} E\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$$

Value Function and Q Funciton

RL goal is to find such a policy. How?

Let us have some (not optimal) π and it produces sample trajectories: $s_0, a_0, r_0, \ldots$. Now, how good is a state? *Value function* tells it. It is at s - exp cumulative reward from following the policy, it is how good is it if we execute policy π beginning with s_0.

$$V^{\pi}(s) = E\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Slightly diff version of V is Q function. To make it math better. It adds a, to tell how good s if we follow π with this a:

$$Q^{\pi}(s, a) = E\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Goal is to find $\hat{Q}$. Optimal Q-func, it is func for $\hat{\pi}$, best possible policy for s_0. It is $Q = max_{\pi} E$.

Relation betwee policy and Q: $\hat{\pi}(s) = argmax_{a'} Q(s, a')$. So we can use only Q func.

Belman Equation

$$\hat{Q}(s,a) = E_{r,s'}\left[r + \gamma max_{a'}\hat{Q}(s',a')\right]$$

where $r \sim R(s,a)$, $s' \sim P(s,a)$

It is a way to present best policy. It tells that the best Q func is recurssively defined via expected value function of steps taken: we take action a at state s then get r reward, then again we take best action and expected value of reward plus next Q func.

Idea for algo. So we can turn that inf sum into someth tracktable. We try to find such Q that satisifies Belman Equation, then it must be an optimal one. So we search for some Q that satisfies Bellman Eq. Algo:

1. Start with random Q.
2. Use Bellman Eq as an update rule.

In practice, $Q_i \to \hat{Q}$ with $i \to \infty$. Why?

Problem: Need to keep track of $Q(s,a)$ for all (state, action) pairs. If states and actions are large it is hard. Now we use ML! **Solution using NN**: Approx $Q(s,a)$ with NN, loss func is Bellman Equation.

Deep Q-Learning

Train NN (weights Θ) to approx $\hat{Q} \approx Q(s,a;\Theta)$. And targets:

$$y_{s,a,\Theta} = E_{r,s'}\left[r + \gamma max_{a'}\hat{Q}(s',a';\Theta)\right]$$

Hence the loss: $L(s,a) = (Q(s,a;\Theta) - y_{s,a,\Theta})^2$. Can do gradient descent, find Θ. And we will have those weights to make actions.

Problems:

1. Nonstationary problem. The weights it predict depend on the NN itself as it takes actions from those weights.
2. How to sample batch? How to form mini-batches.

Deep Q-Learning was effective for Atari games. Architecture:

- Input: 4x84x84 stack of last 4 frames. Output: Q-values for all actions.
- Layers: `Input → Conv → Conv → FC-256 → FC(Q-values) → Output`

Problems: For some problem it is better to learn from states not from mapping of states to actions (actions conditioned on states). Like drink from a bottle.

Policy Gradients

\55

Idea: now get distribution of actions (policy) to take using states. For this train NN. So it is $\pi_\Theta = p(a|s)$. Then objective fun will be expected value of those acions (policy):

$$J(\Theta) = E_{r \sim p_\Theta} \left[\sum_{t \geq 0} \gamma^t r_t \right]$$

And use gradient ascent. But it is not differentiable. How to compute gradient over environemnt. How then?

Trick:

Policy Gradients: REINFORCE Algorithm

General formulation: $J(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)]$ Want to compute $\frac{\partial J}{\partial \theta}$

$$\frac{\partial J}{\partial \theta} = \frac{\partial}{\partial \theta} \mathbb{E}_{x \sim p_\theta}[f(x)] = \frac{\partial}{\partial \theta} \int_X p_\theta(x) f(x) dx = \int_X f(x) \frac{\partial}{\partial \theta} p_\theta(x) dx$$

$$\frac{\partial}{\partial \theta} \log p_\theta(x) = \frac{1}{p_\theta(x)} \frac{\partial}{\partial \theta} p_\theta(x) \Rightarrow \frac{\partial}{\partial \theta} p_\theta(x) = p_\theta(x) \frac{\partial}{\partial \theta} \log p_\theta(x)$$

$$\frac{\partial J}{\partial \theta} = \int_X f(x) p_\theta(x) \frac{\partial}{\partial \theta} \log p_\theta(x) \, dx = \mathbb{E}_{x \sim p_\theta} \left[f(x) \frac{\partial}{\partial \theta} \log p_\theta(x) \right]$$

Approximate the expectation via sampling!

So this can be approximated.

In the end (after more tricks):

Reinforce Rule

Expected reward under π_θ:

$$J(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)]$$

$$\frac{\partial J}{\partial \theta} = \mathbb{E}_{x \sim p_\theta} \left[f(x) \sum_{t \geq 0} \frac{\partial}{\partial \theta} log \, \pi_\theta(a_t | s_t) \right]$$

where

- $x = (s_0, a0, s_1, a_1, \ldots)$ seq of states and actions when π_Θ.
- $x \sim p_\Theta$ sequence of states and actions when π
- $f(x)$ reward from x
- $\frac{\partial}{\partial \Theta} log \pi_\Theta(a_t | s_t)$ is gradient of predicted action scores.

Algo: 1. init weights. 2. Run π_Θ and get x with $f(x)$ rewards. 3. Calc gradient. 4. Ascent using the grad. 5.

Repeat with 2 step.

Intuition: If $f(x)$ is high probability of actions are higher, otherwise decrease probabilities. Or if rewards of actions are high then actions treated as good, otherwise bad.

Q-Learning and Policy Gradients are just beginning.

Other approaches

85\

- **Actor-Critic**. Actor predicts actions. Critic predicts rewards.
- **Model-Based**. Tries to learn world's state transition function (model of world).
- **Imitation Learning**. Learn from experts.
- **Inverse RL**. Infer reward func that experts optimized by observing their actions.
- **Adversarial Learning**.

Case Study. Playing Games

- AlphaGo. 2016. Beat 18-time world champion.
- AlphaGo Zero, 2017. Beat #1.
- Alpha Zero, 2018. Other games
- MuZero, 2019. Plans through learned model of the game.
- StarCraft II. AlphaStar, 2019
- Dota 2. 2019

Stochastic Comp Graphs

\97

Train NN with nondifferentiable components.

- For image classification. Making classification decision: choose from a set of CNNs then update policy.
- Attension. Hard Attention (vs Soft Attention as it was for attention in CV). At timestep we seelct features, then train with policy gradient.

My questions

- What's RL?
  - Idea.
  - Examples of RL.
  - Compare RL with Supervised Learning: simularities, differentcies.
- Q-Learning
  - Markov Decision Process.
    - Model. Discount. Markovian Property.
    - Algo.

        - How to find optimal policy?
    - Value Function. What value func adds to the previous?
    - Q-Function. What q-fun adds to the value func?
    - Bellman Equation.
        - Formula. Describe. Idea.
        - Algo. What's problem with this algo?
    - Deep Q-Learning.
        - What are now the targets, y?
        - What is now the loss func, L?
        - Problems with that?
- Policy Gradients
    - Idea.
    - Formula for loss, for gradient (is it computable?)
    - Algo now.
- Other approaches. Name few
- Cases.

====

...

...

...