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)

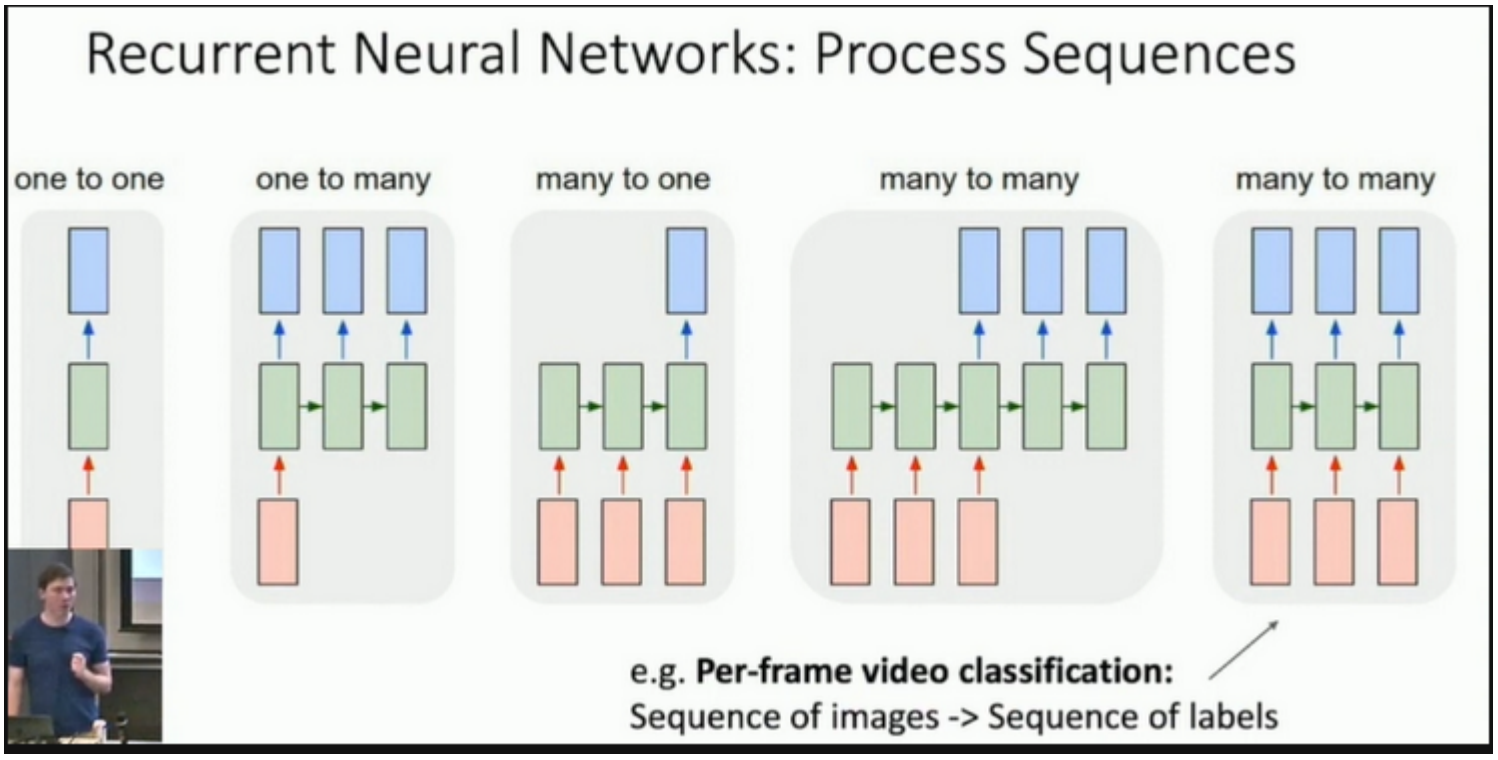
7. Intro

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.

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

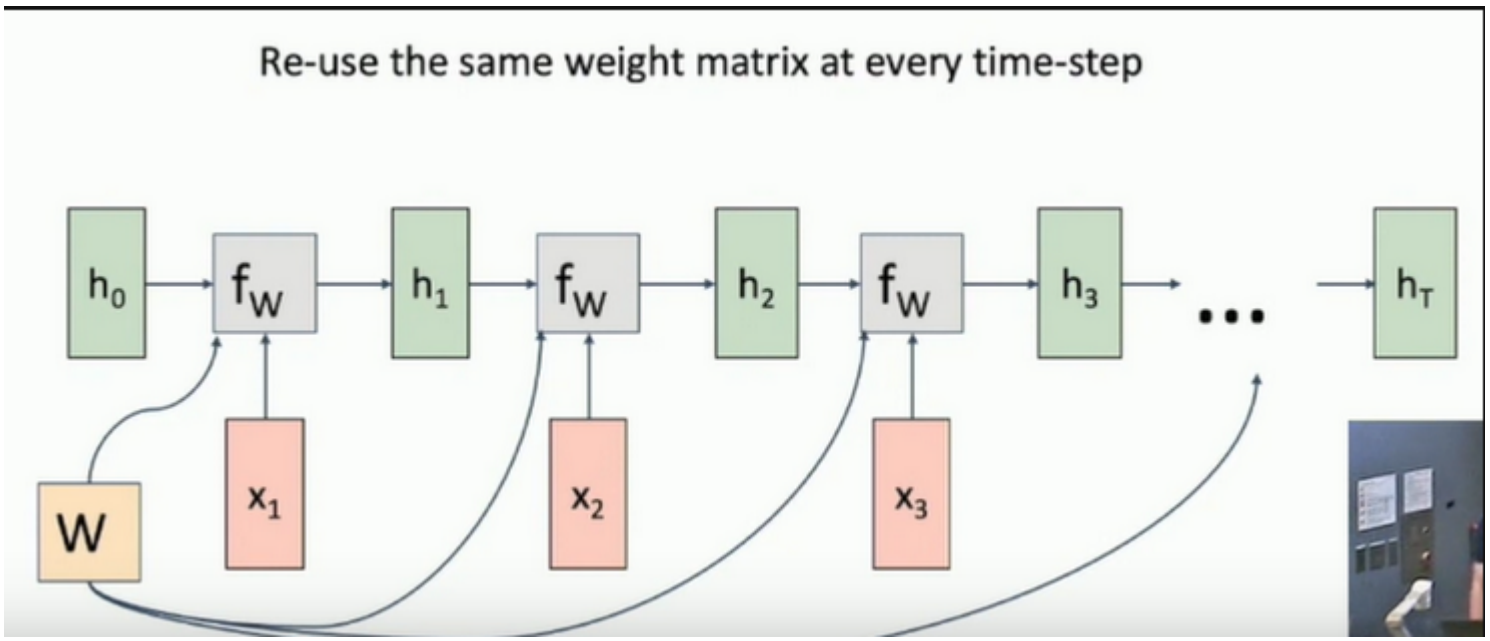
Process x by applying recurrence formula. W - learnable weights. h state. x input. Single weight matrix at every step of a sequence. And the same function f at every step.

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.

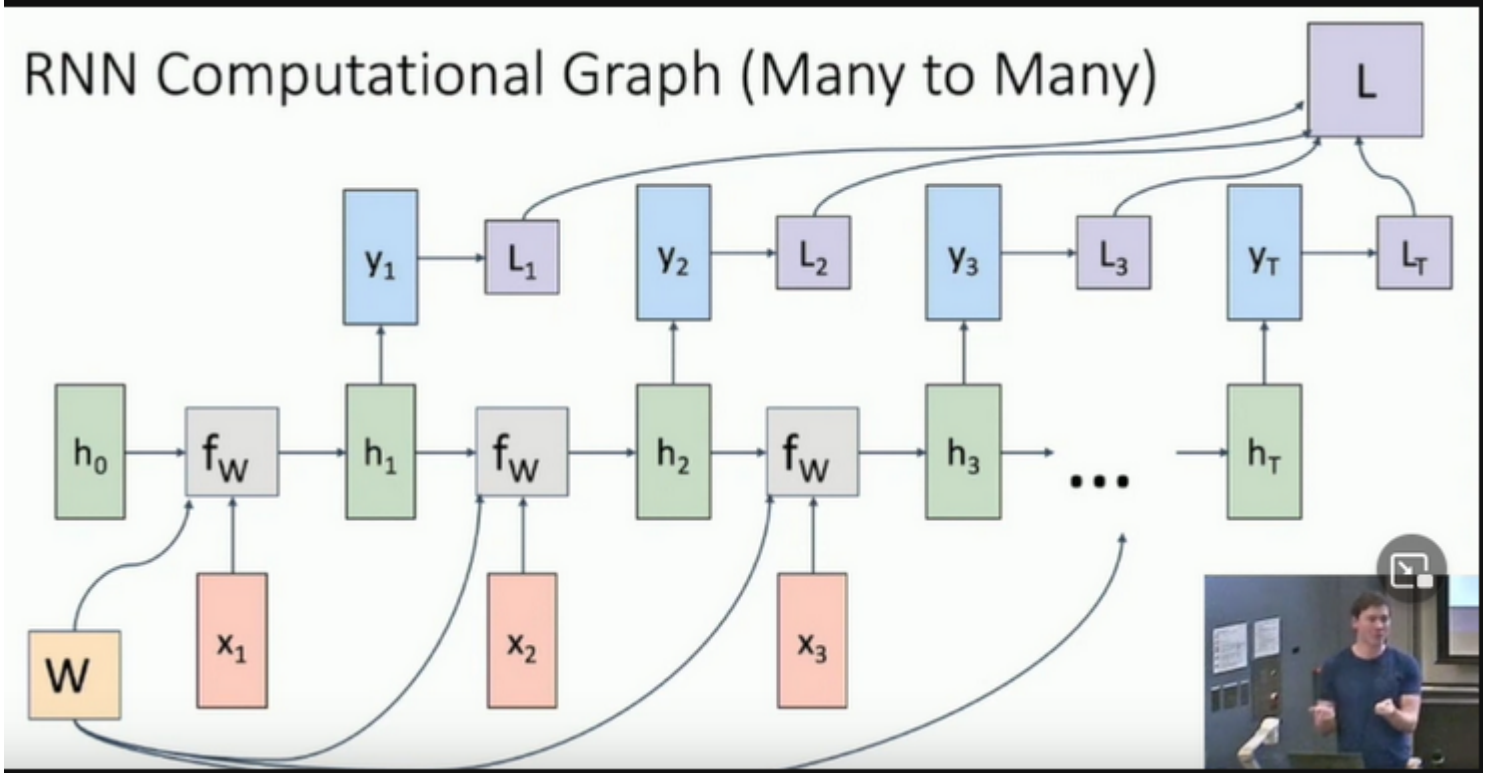
- xix. Example.



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

And we can use it for our types problems:

For many to many:

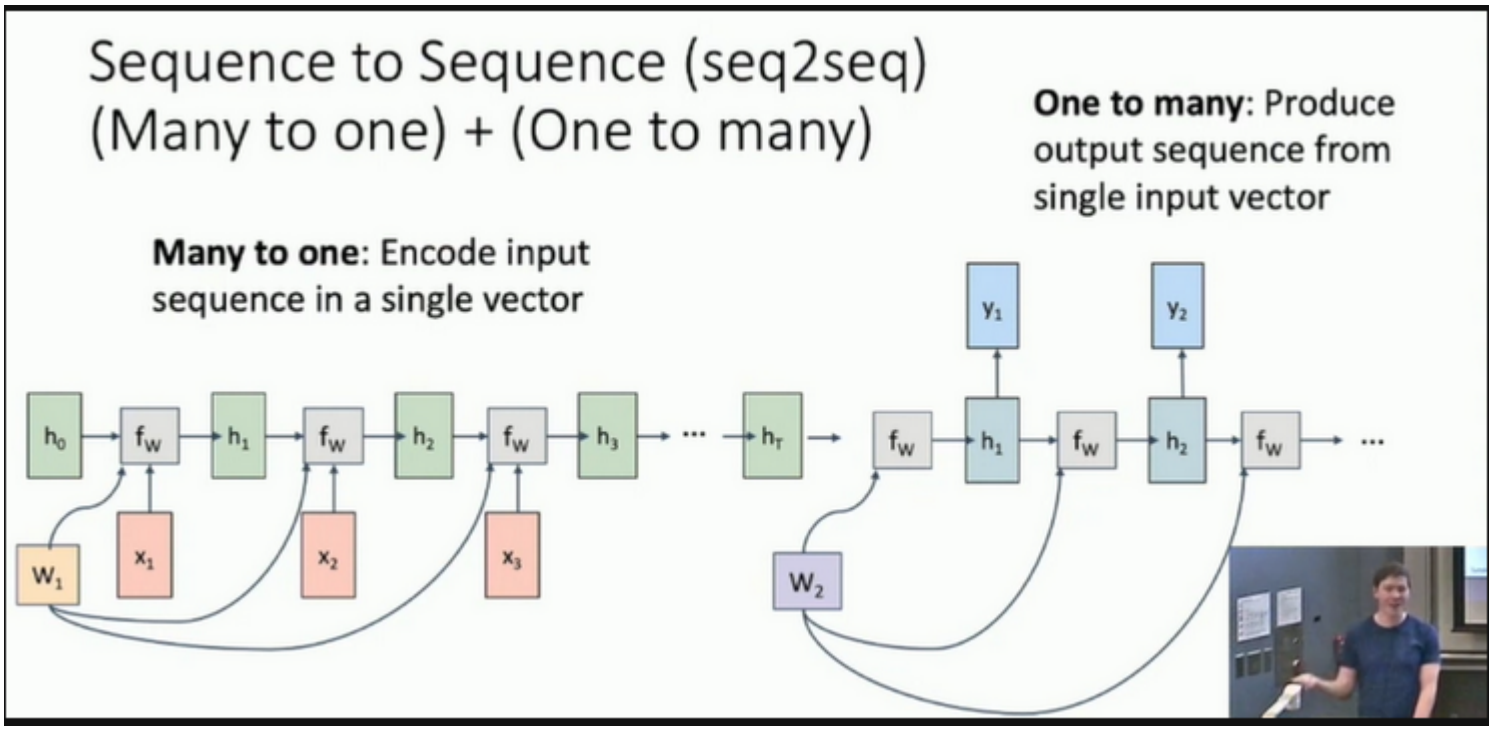


E.g. video classification per-frame. And it produces y_i and then we can apply L_i if we have labels (supervised). And the final L loss will sum those L_i .

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.

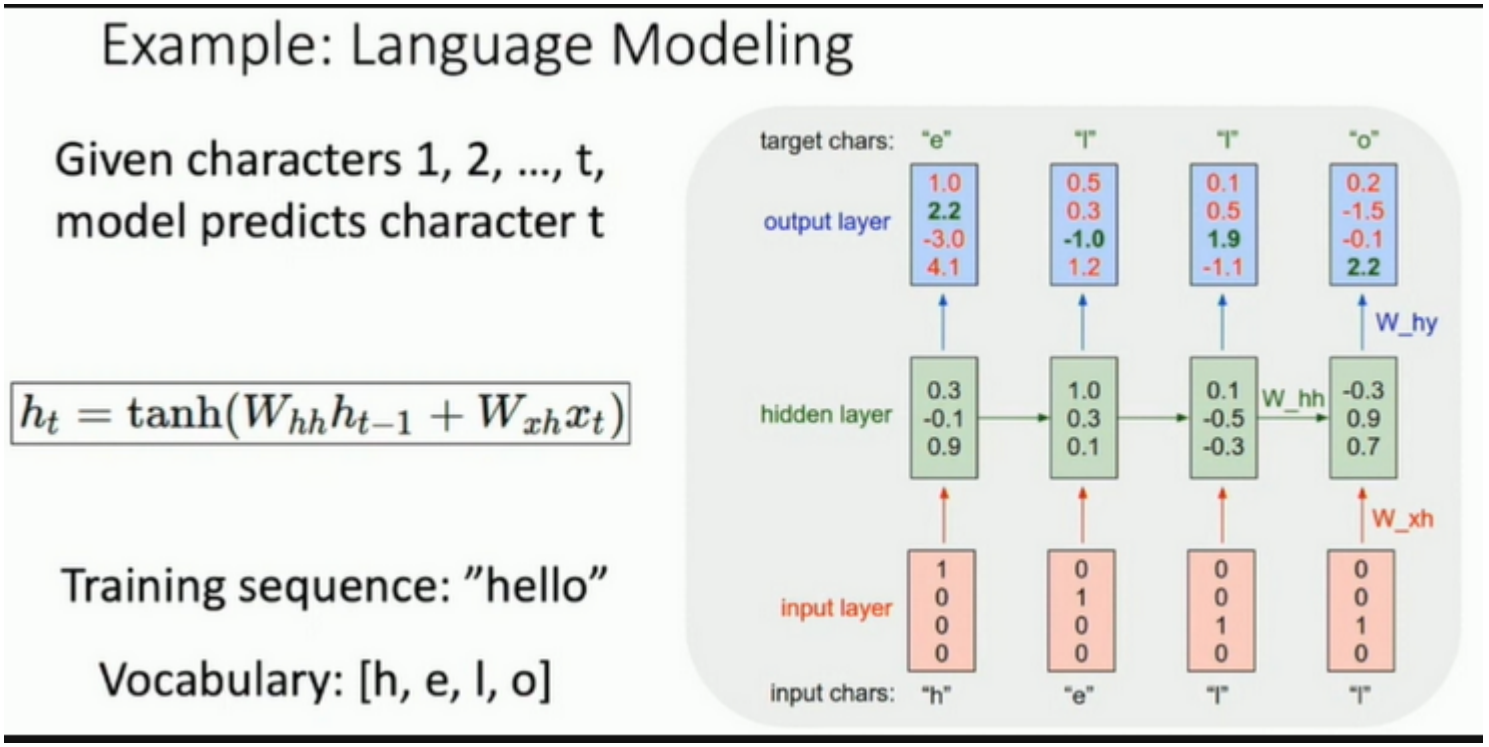
28. Sequence to sequnce (seq2seq). (Many to one) + (one to many). Another type of problem. E.g. to translate Eng to French.
    How to implement? From one NN (many to one) (encoder) and then into another NN (one to many) (decoder).



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

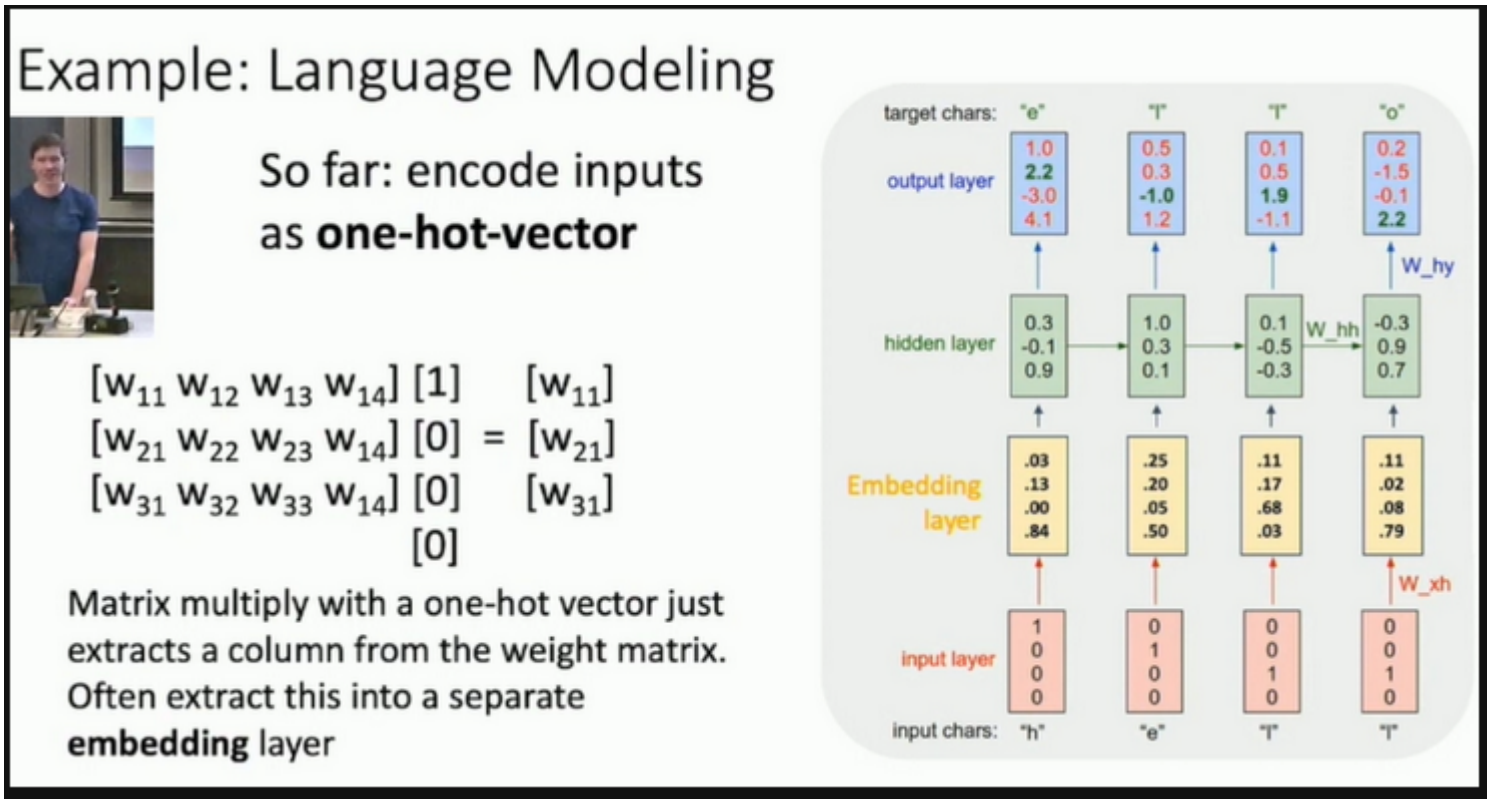
Example. Language Modeling. Given chars what's the next char? Infinite seq of chars and every time it tries

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).

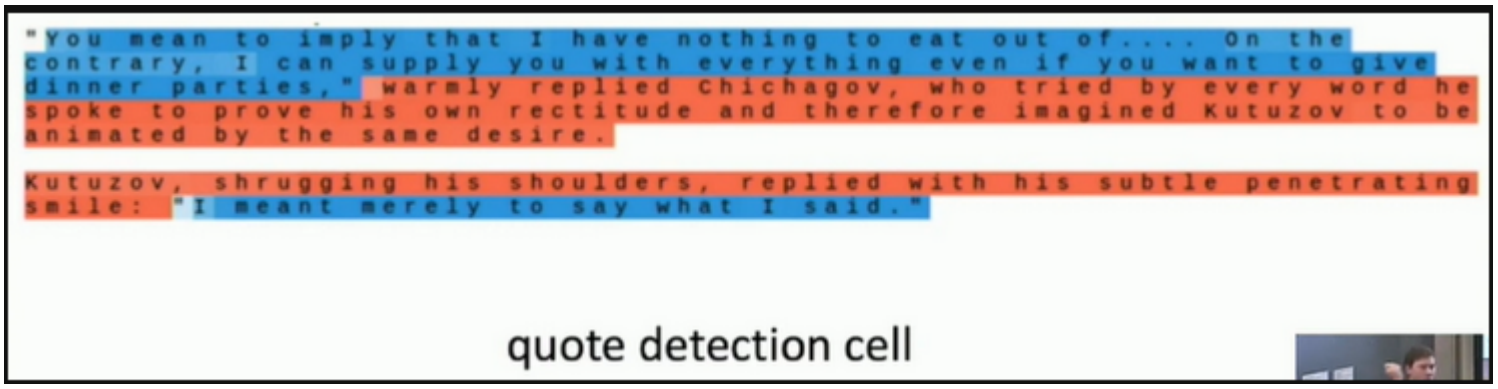
Examples.

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.

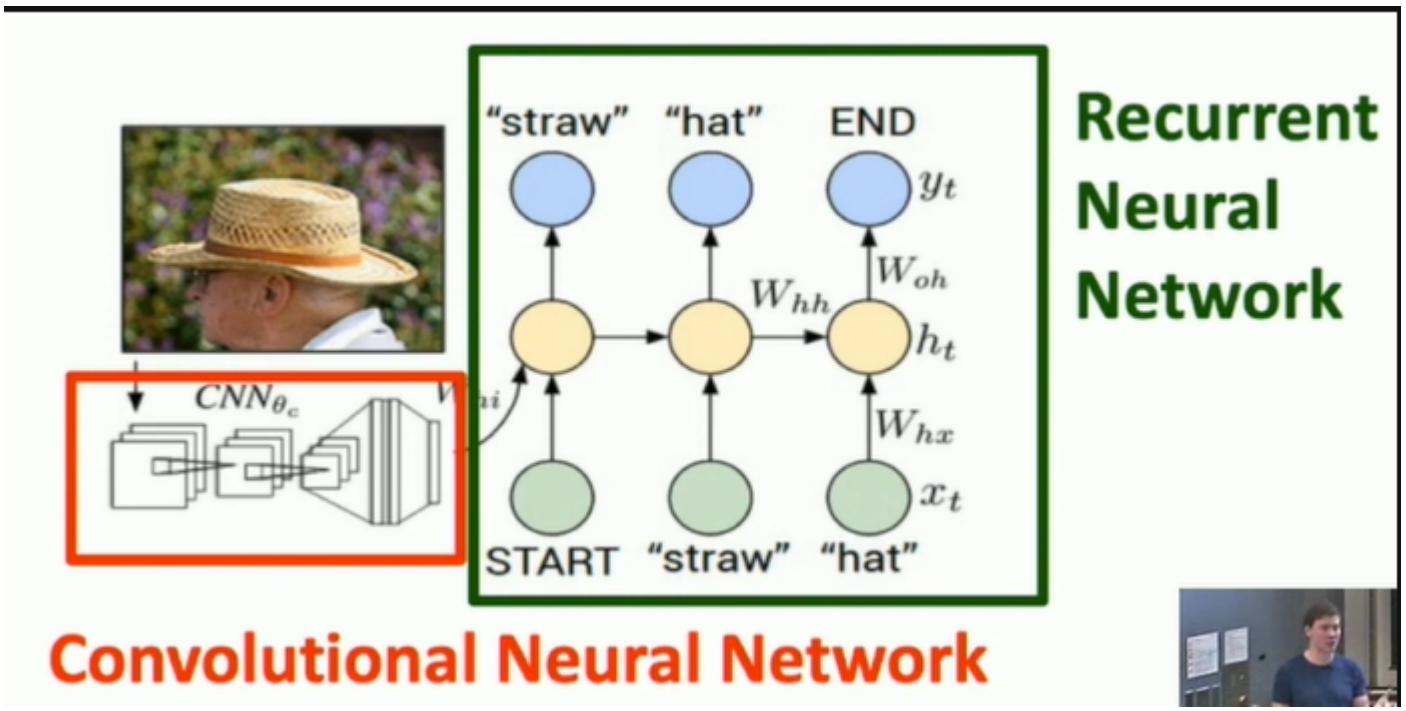
Example. Quote selection cell:



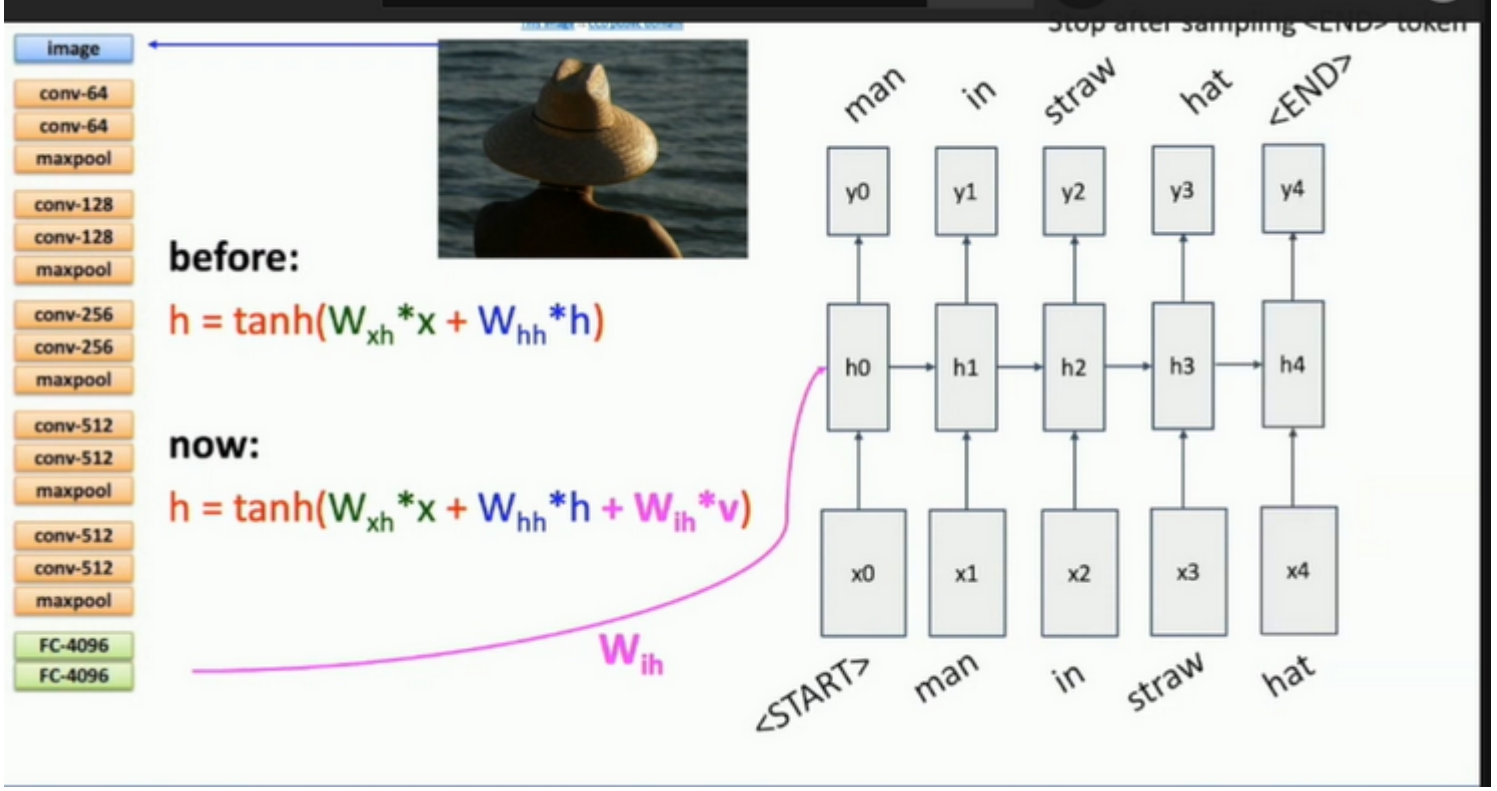
Other examples: inside a comment, length of line, indentation, etc.

70. Example: Image Captioning

That is use CNN with RNN.

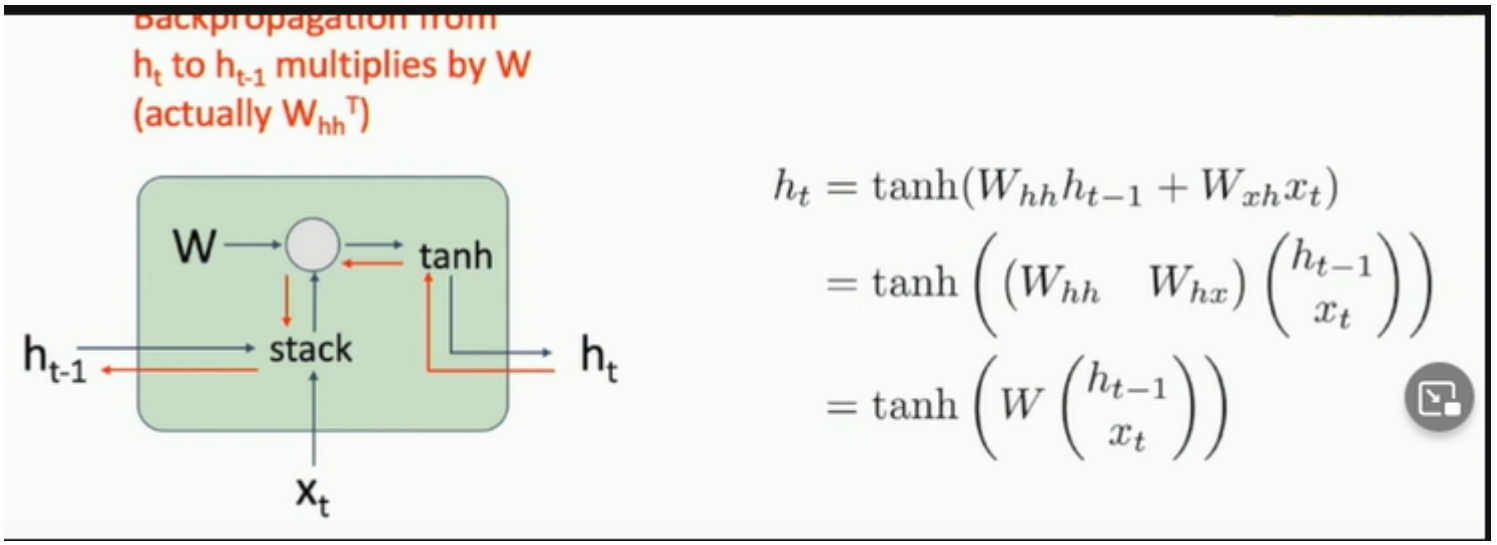


Uses transfer learning. It modifies recurrence formula.



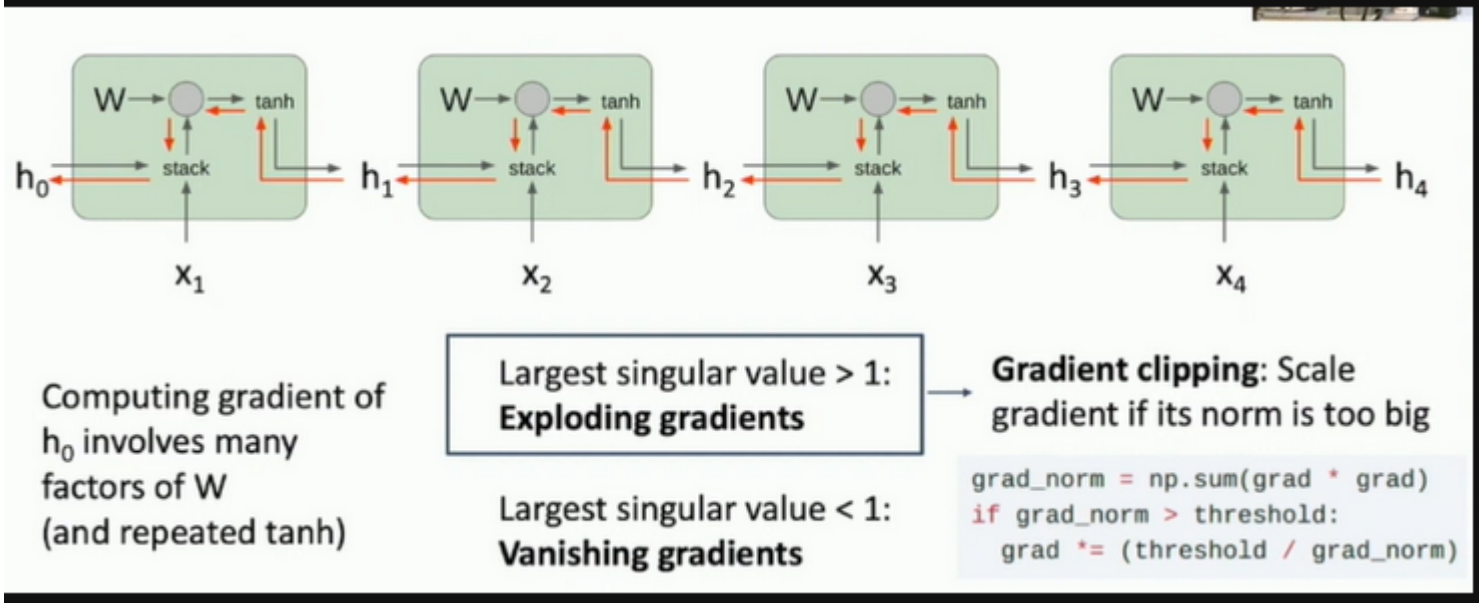
Gives good and garbarish results.

82. Vanilla RNN Gradient Flow.



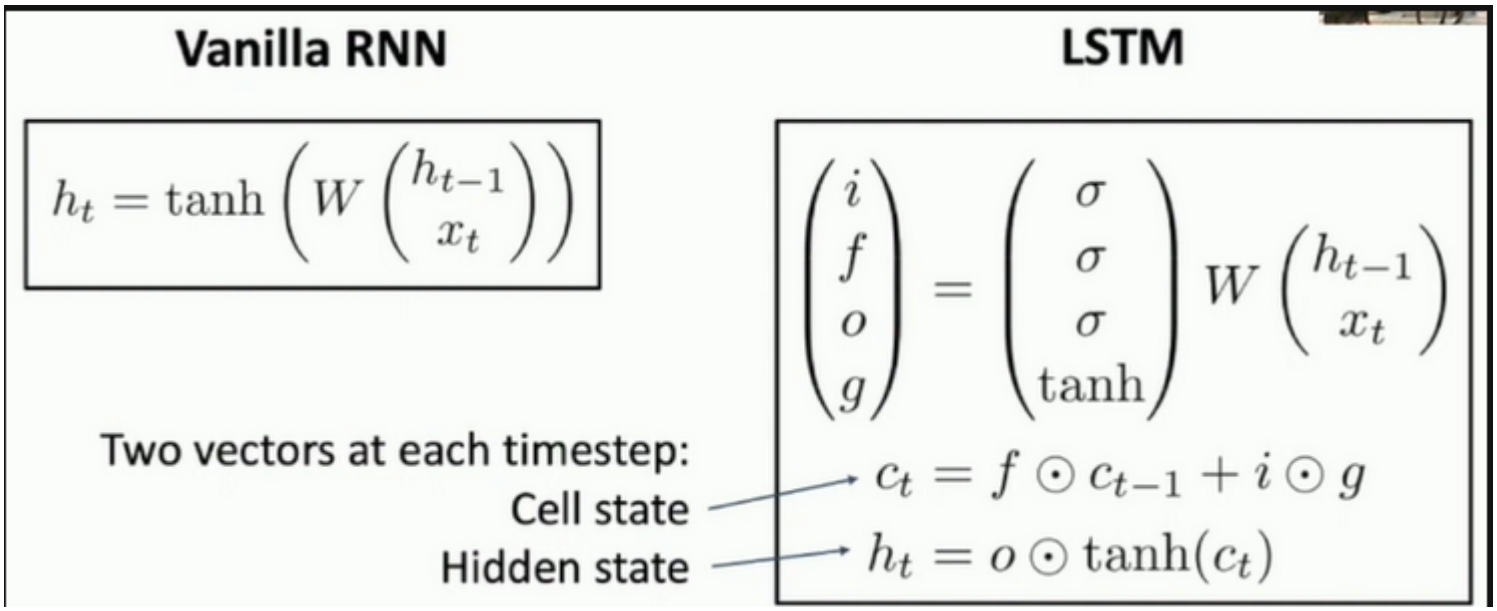
Problems

- $tanh$ 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.

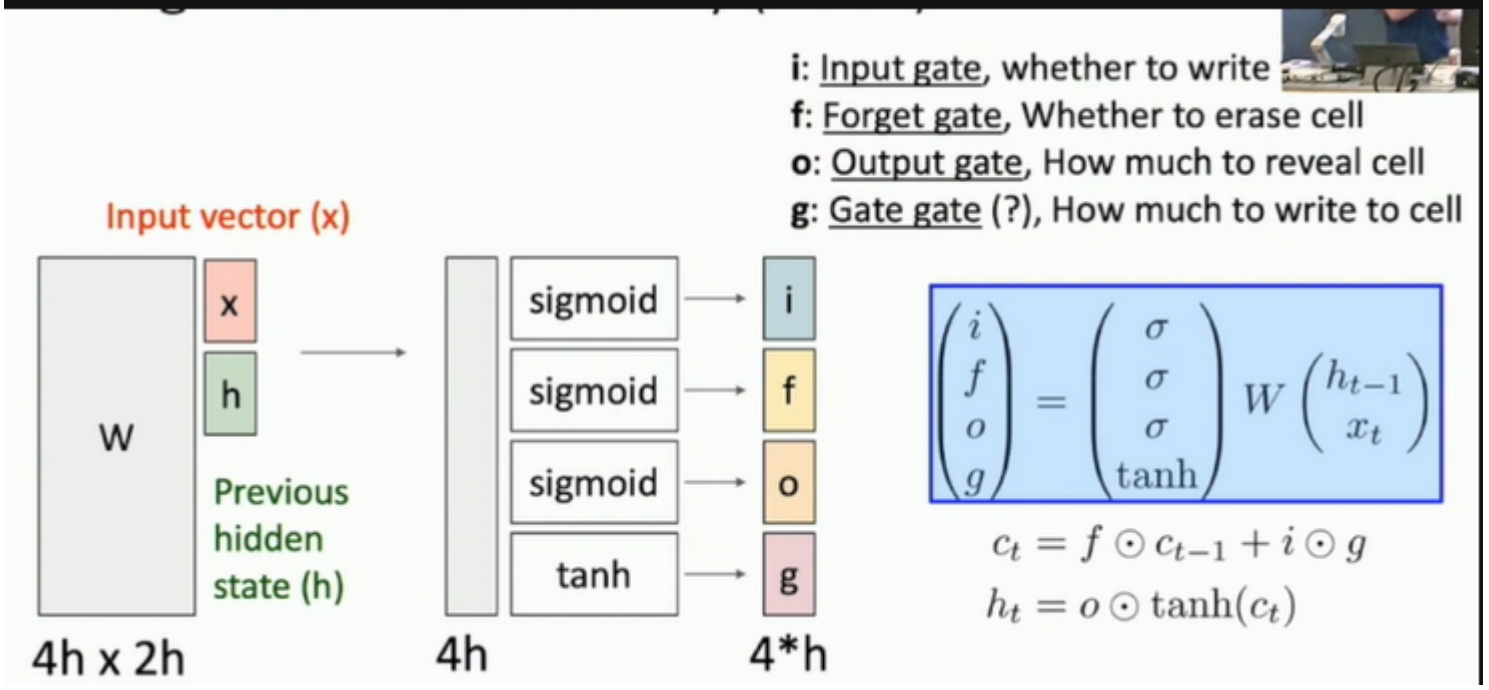


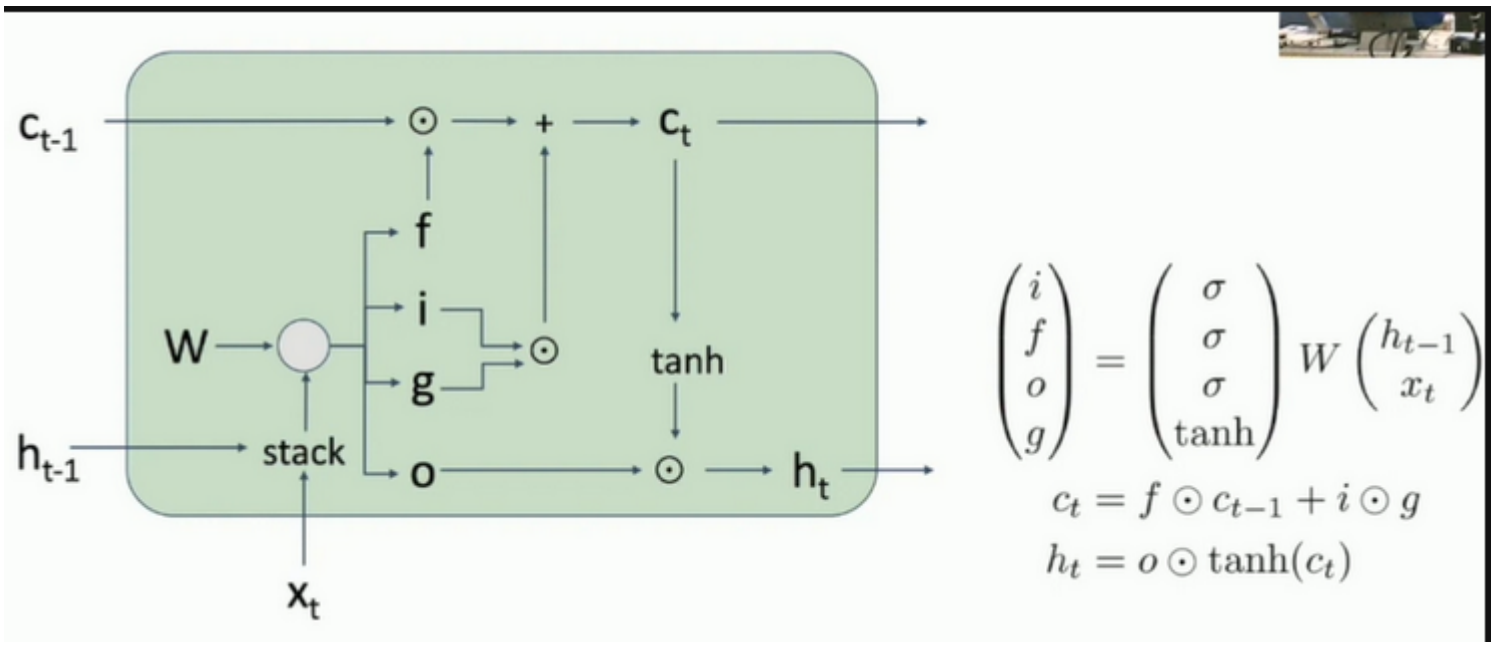
Gradient clipping.
For exploding gradients. So we use gradient clipping. We multiply gradient by a coef to clip it if it reaches some threshold (see picture above).
For vanishing gradients. Throw away this arch and use diff architecture for RNN (LSTM?)

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.

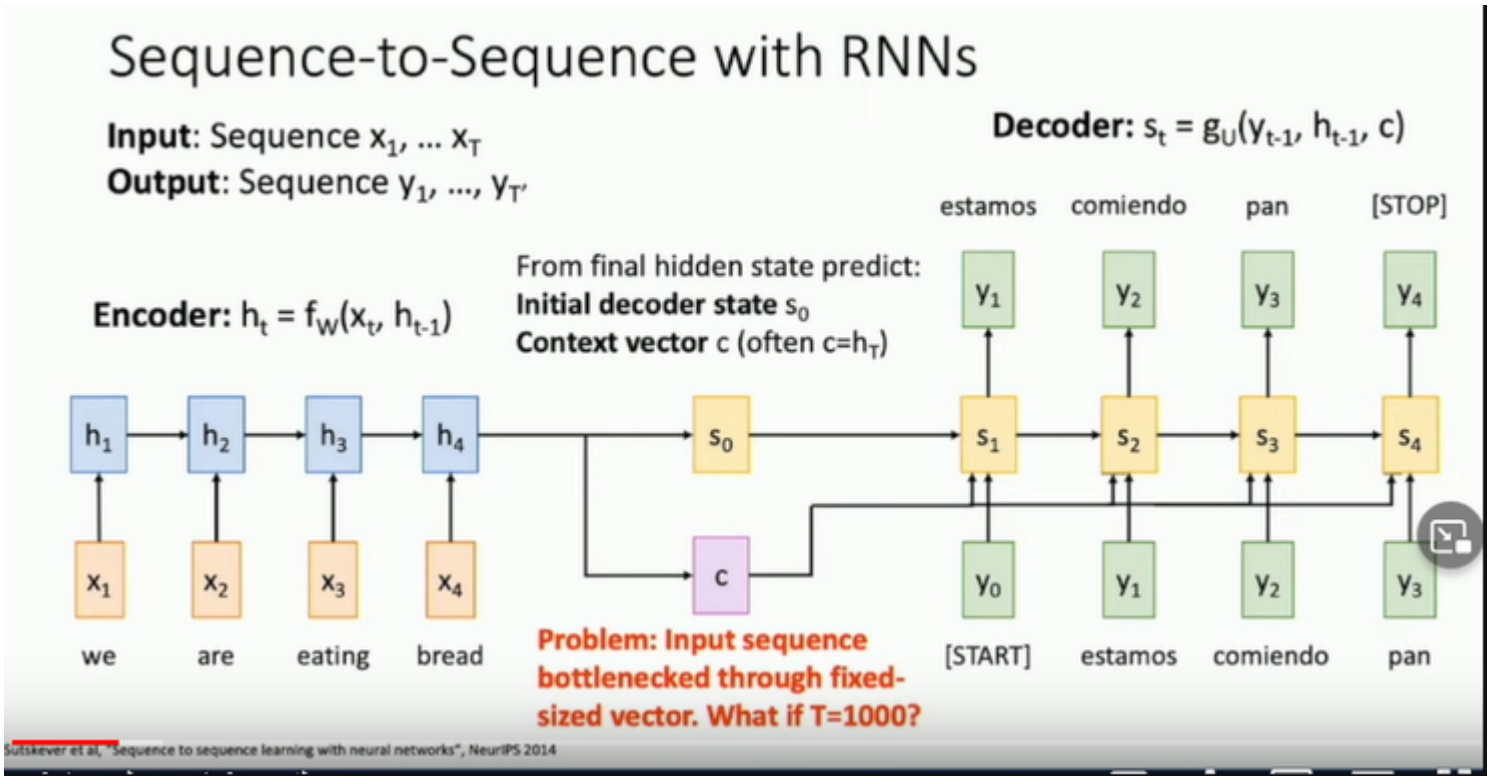
The seq2seq with RNN is:

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.

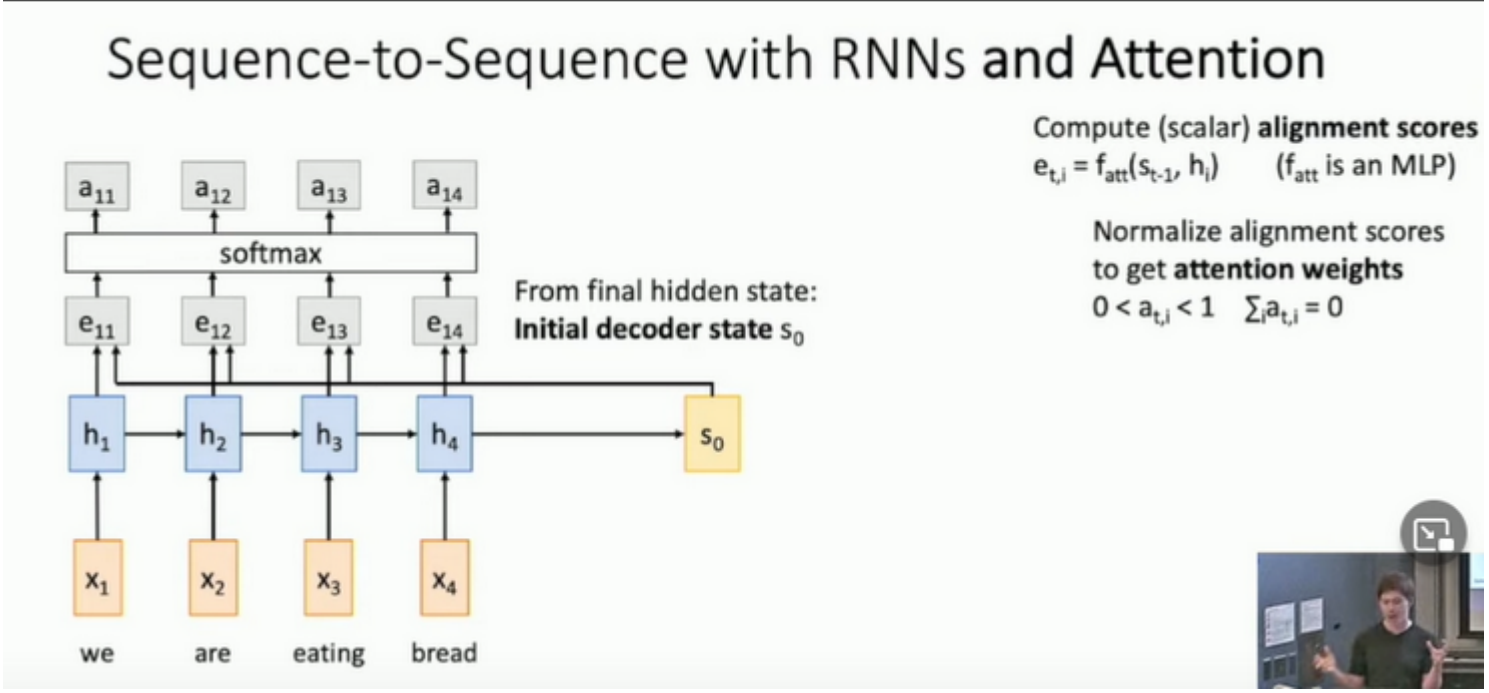
12/

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.

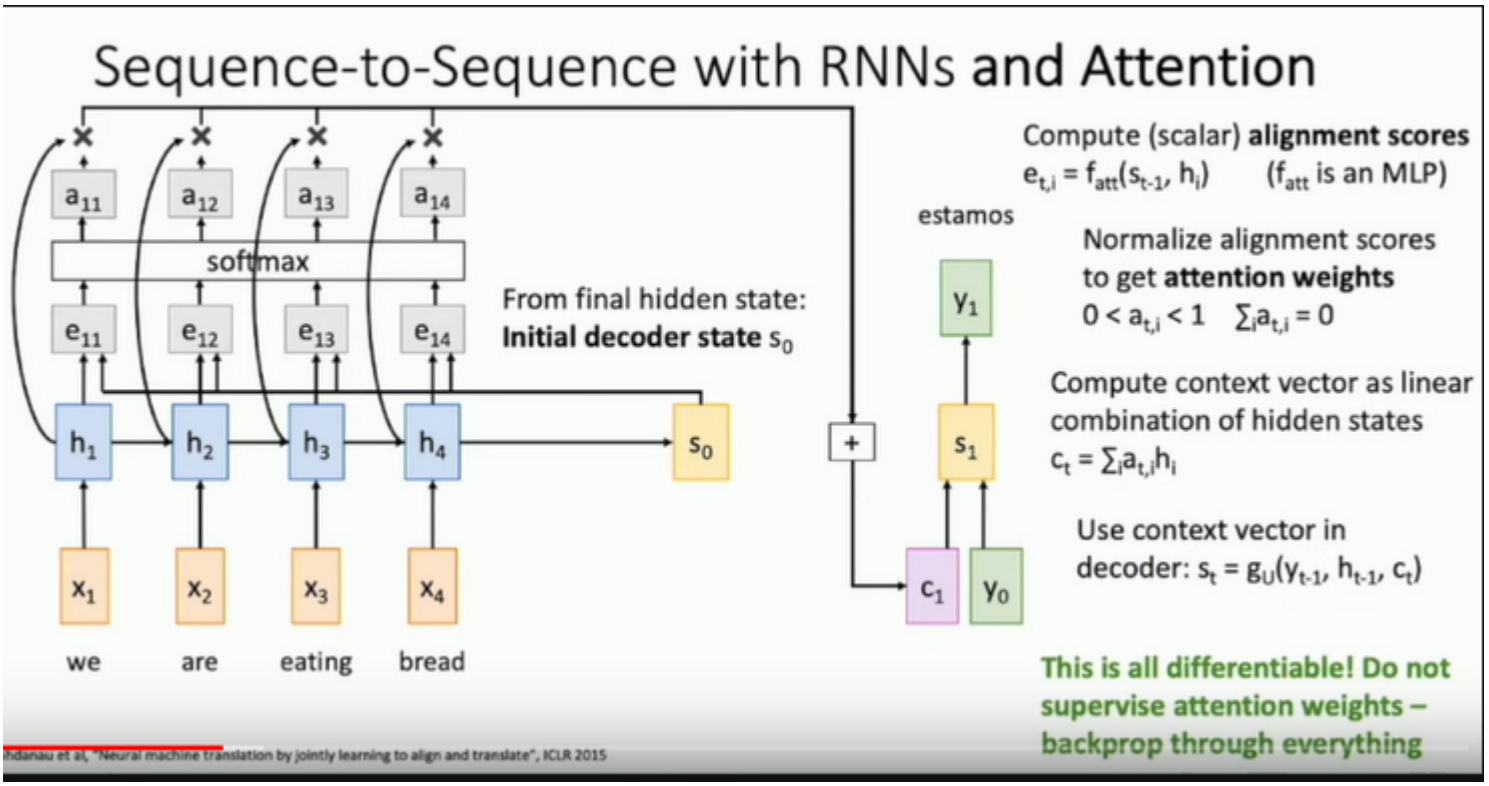
So for the below picture:



$e_{1,1}$ (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:

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

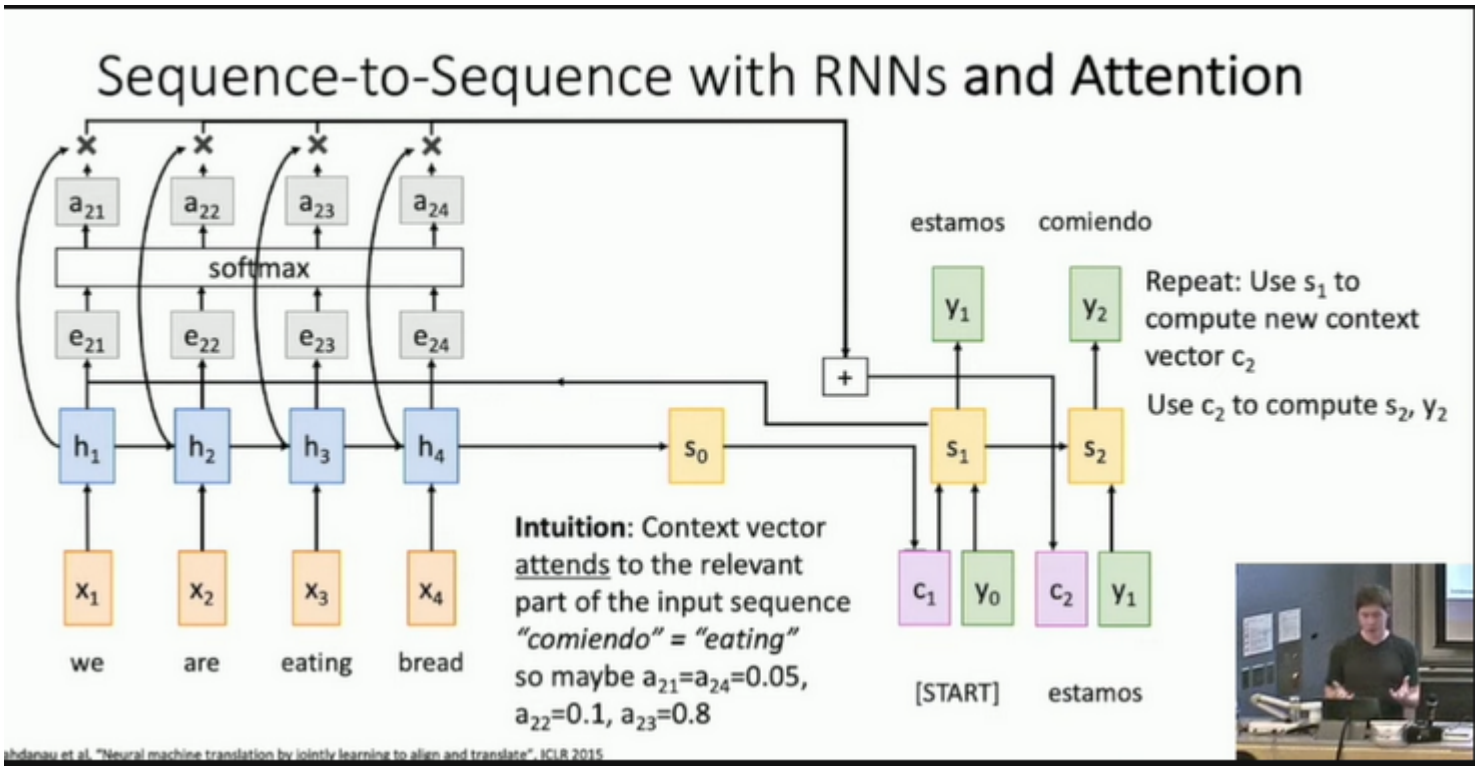


Then the decoder use this c_1 vector to output first token (or word) of the output sequence.

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.

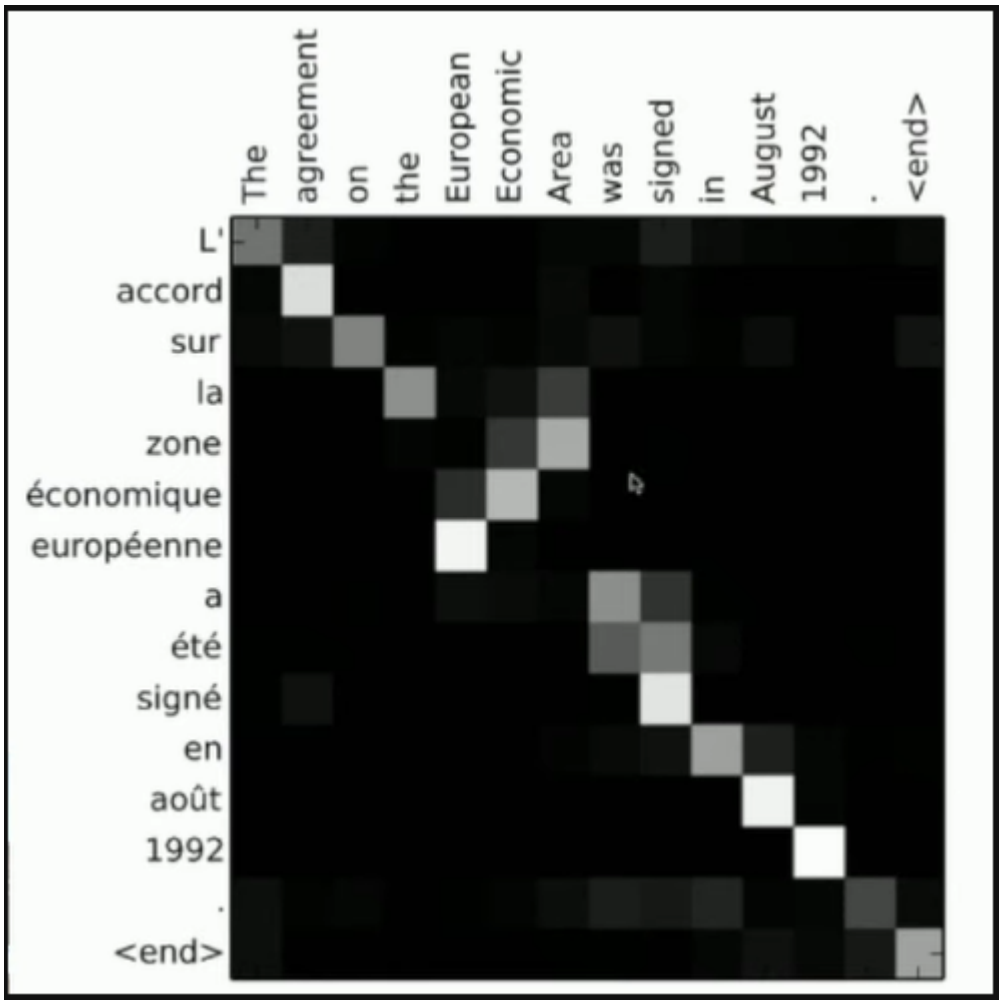
At step 2 we repeat the process to produce second output y_2 via computing c_2 that is computed by using s_1 (encoder uses decoder s_1), from those attention weights. Then it repeats.



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:



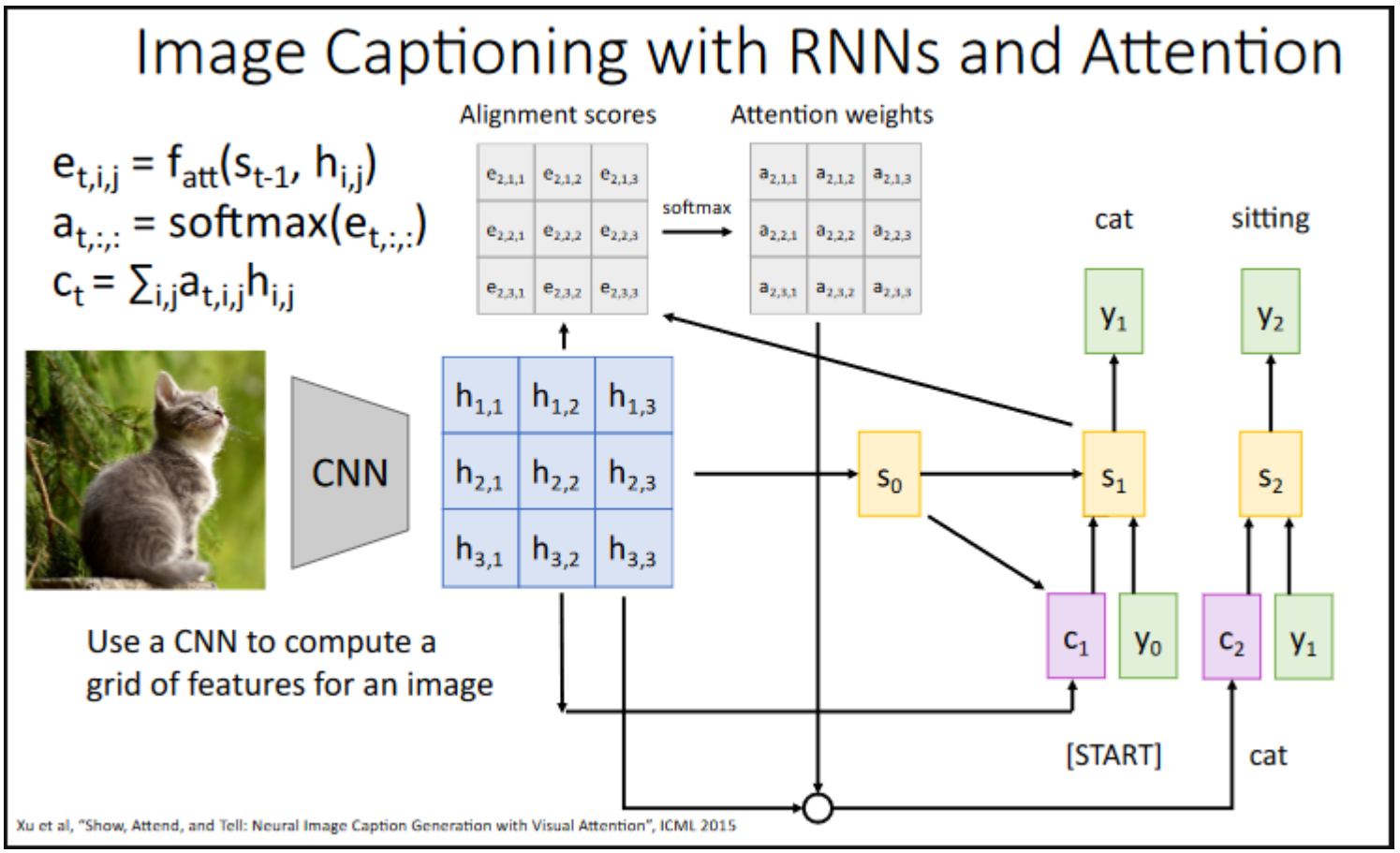
The above shows what weights were at each step for each word it produced. So what words correspond to what. It learns those itself! Model figures out it itself. It tells us how model makes decisions.

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:



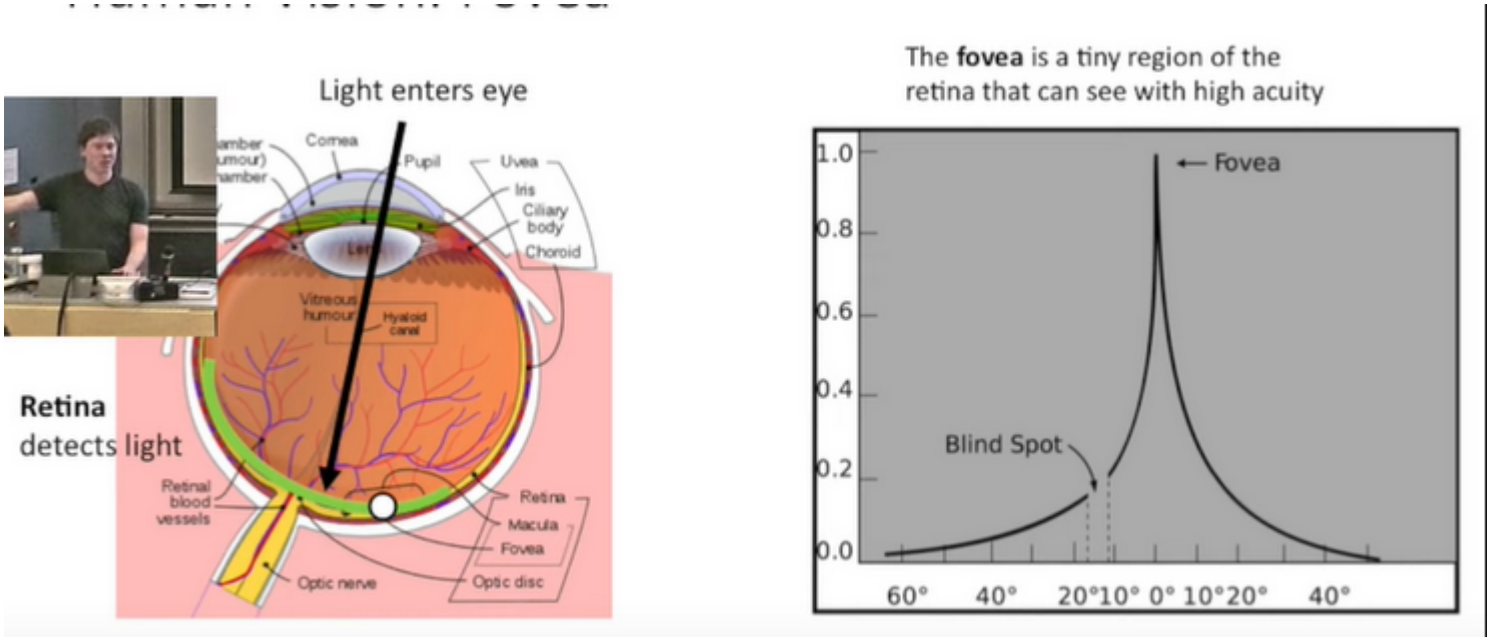
So we have attention weights for each part of the input. We then produce c_1 to get first word. Then repeat. Each time it looks at diff part of image.

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

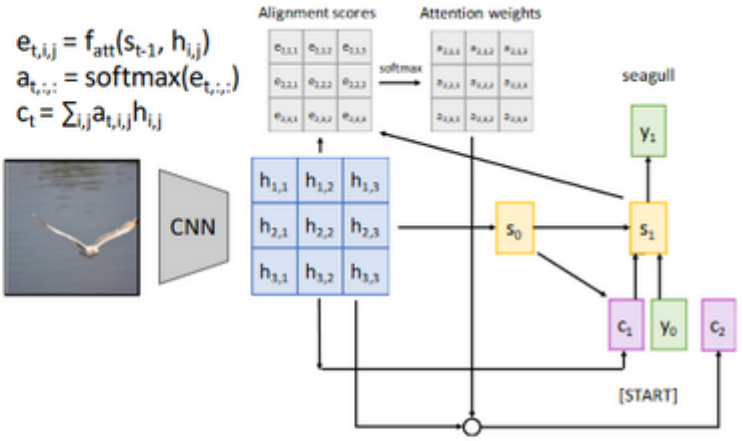
Generilizing this mechinism. Let's reframe this mechanizm.

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)

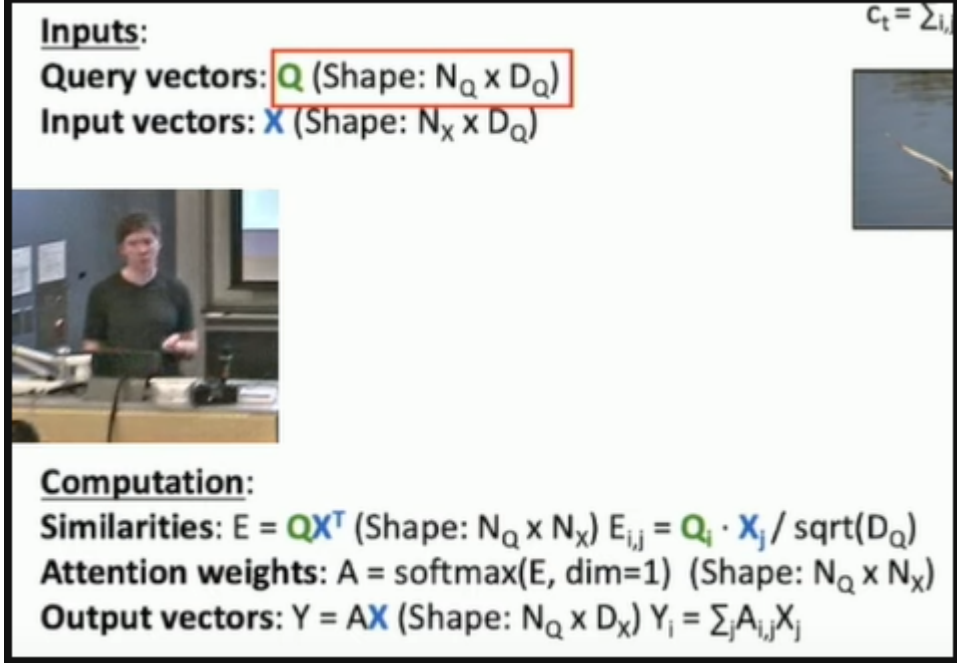
And it outputs:

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)

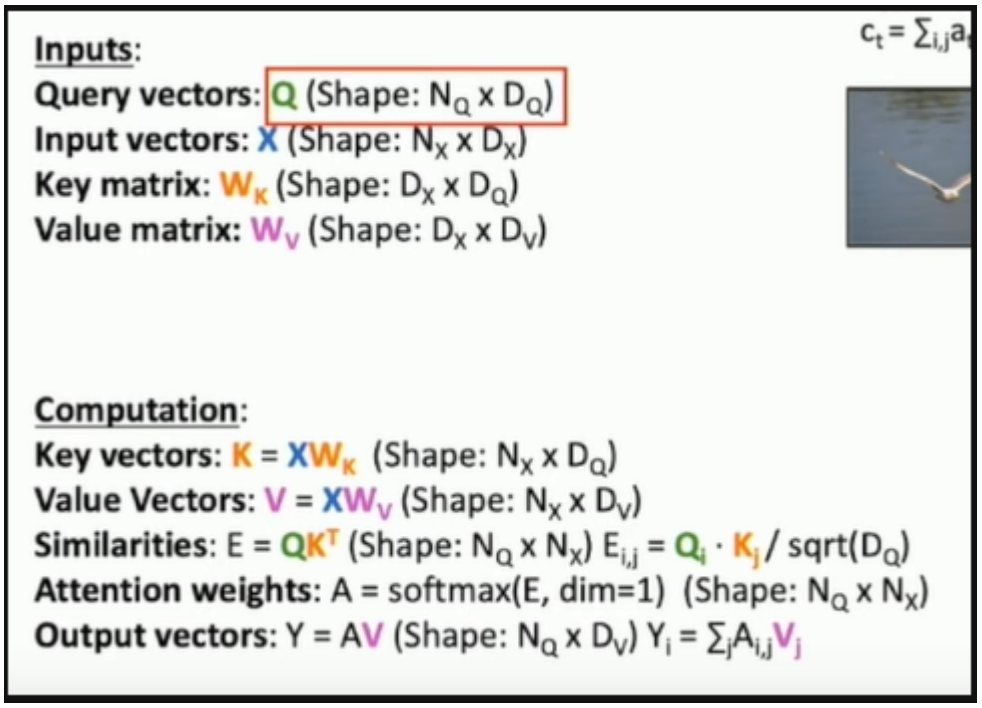
e - vector of simularities.
a - probabilities (as before) aka attention weights
y - output

How to generilize? Generalizations:

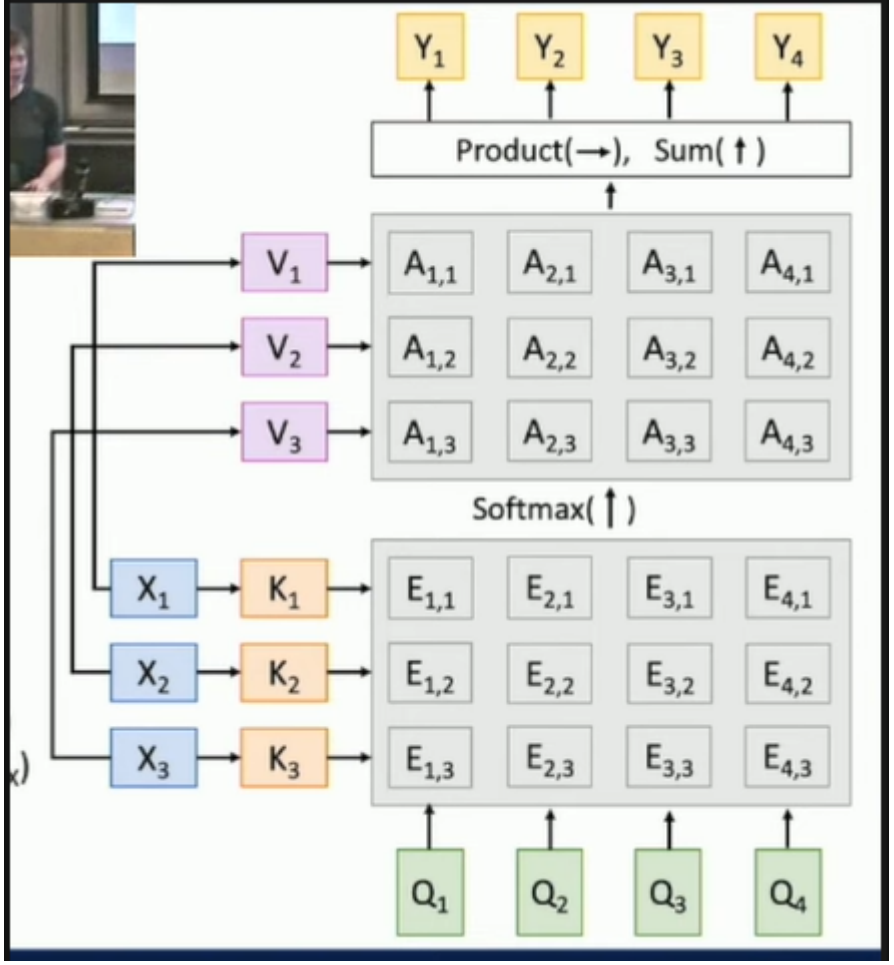
1. Generalization 1. Use dot product.
2. Scaled dot product: $qX_i/sqrt(D_Q)$. Why? Because of the vanishing gradients problem. So learning is challenging. Remember that dot product uses norm of vectors.
3. Multiple query vectors. Generate attentions for a set of queries. And we can get similarities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

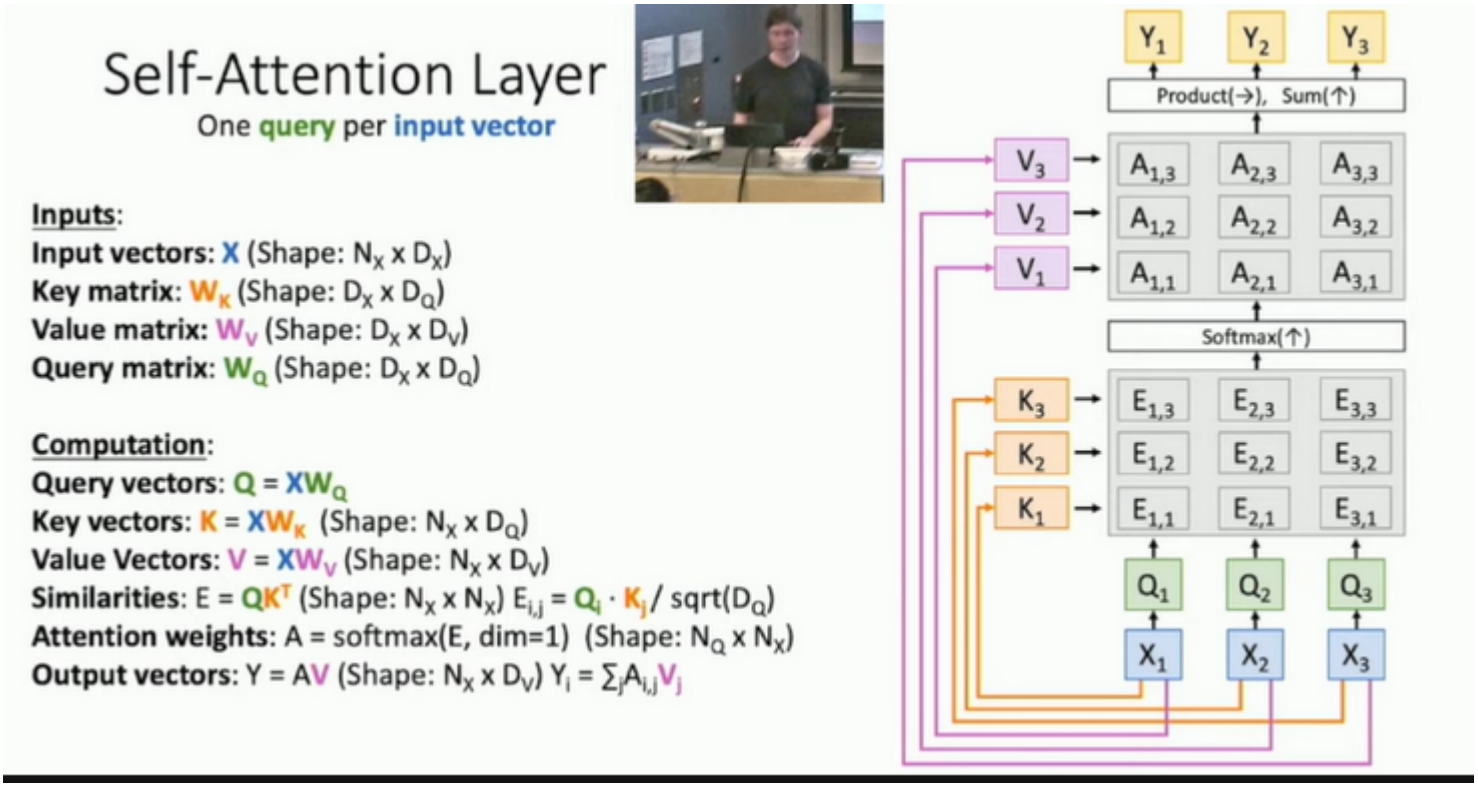


So we got very general attention layer.

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.

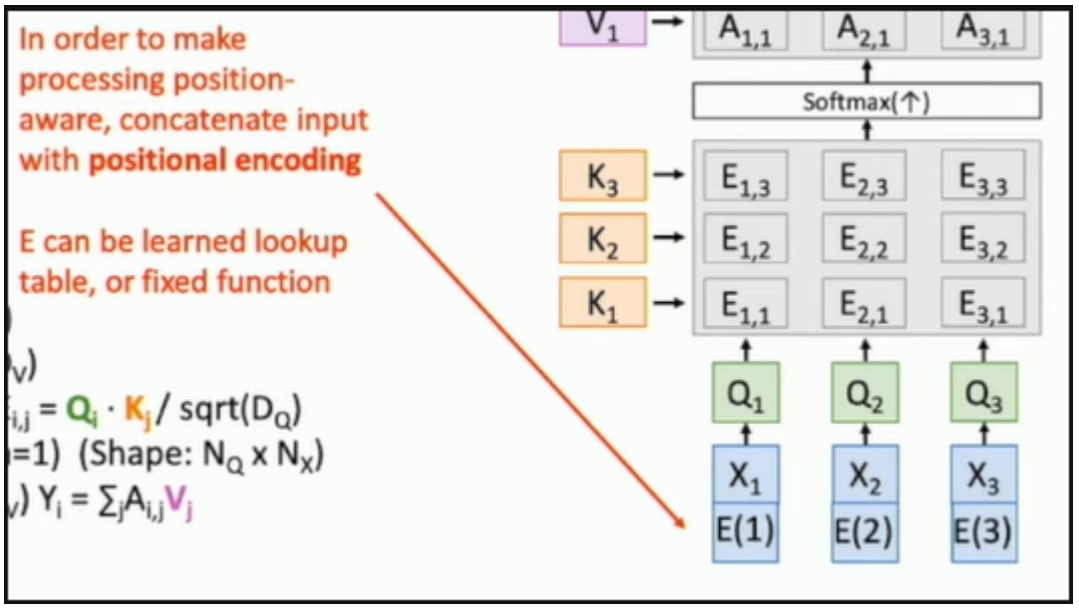
First we convert input vector to query vector. Then ... See below:



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).

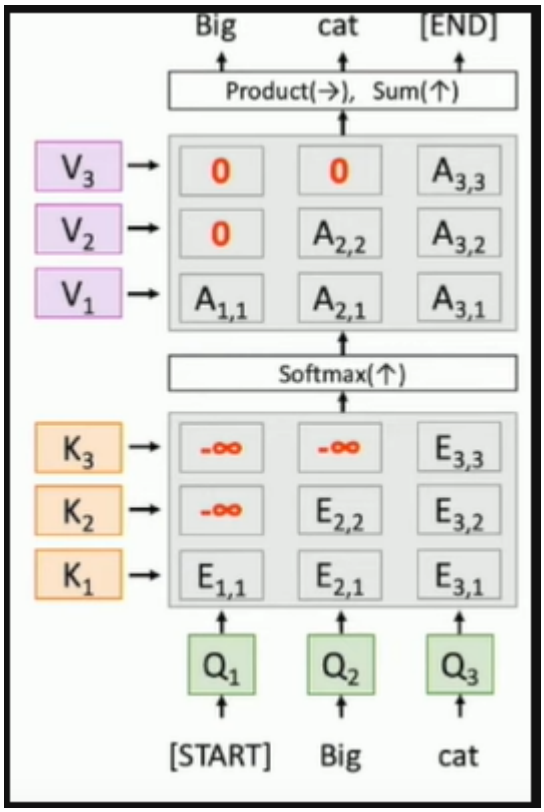
Problem is it doesn't know the order. How to make it know the order? So make a hack: concatenate with positional encoding:



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

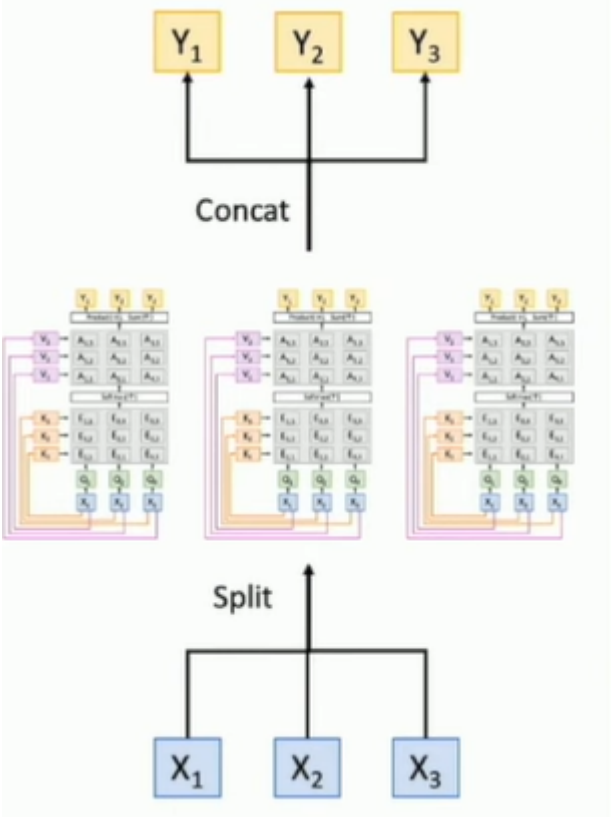
75\ Masked Self-Attention Layer.

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.



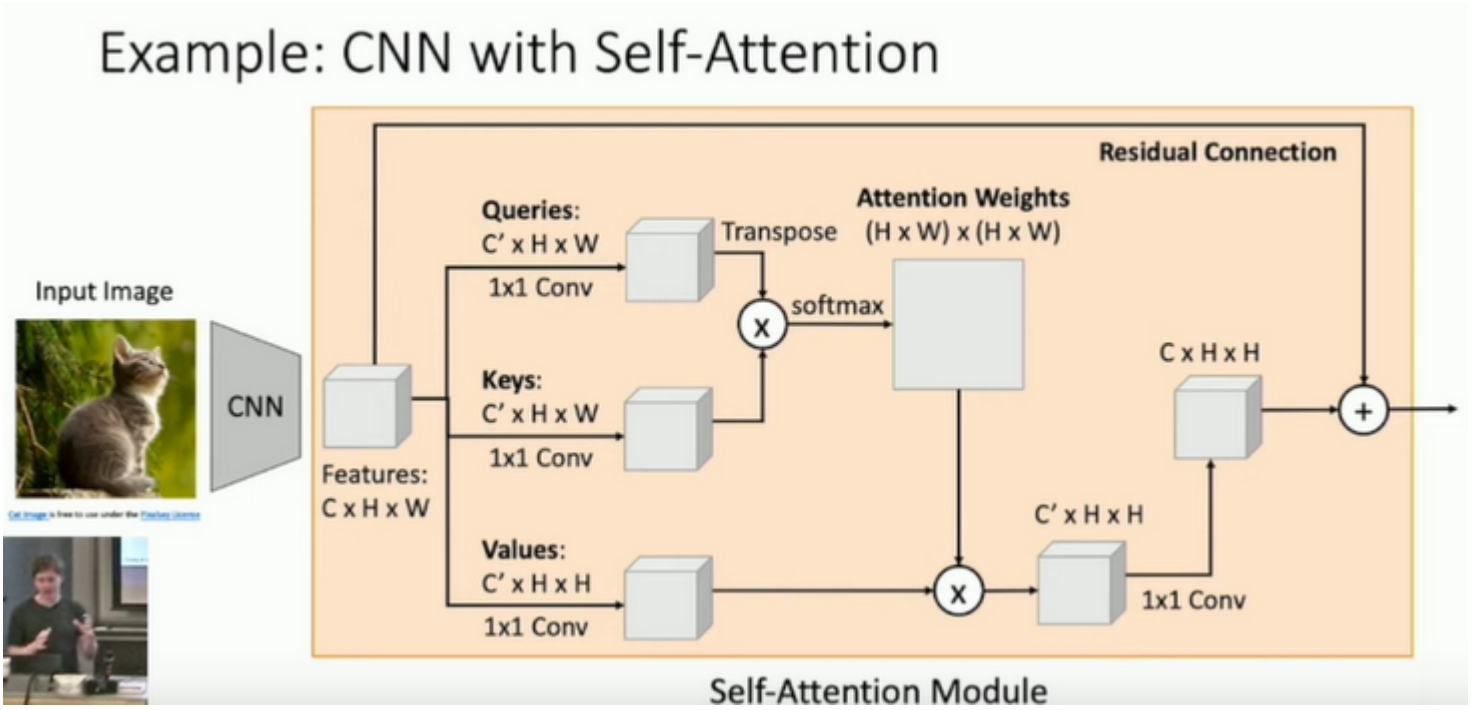
77\ Multihead Self-Attention Layer

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.



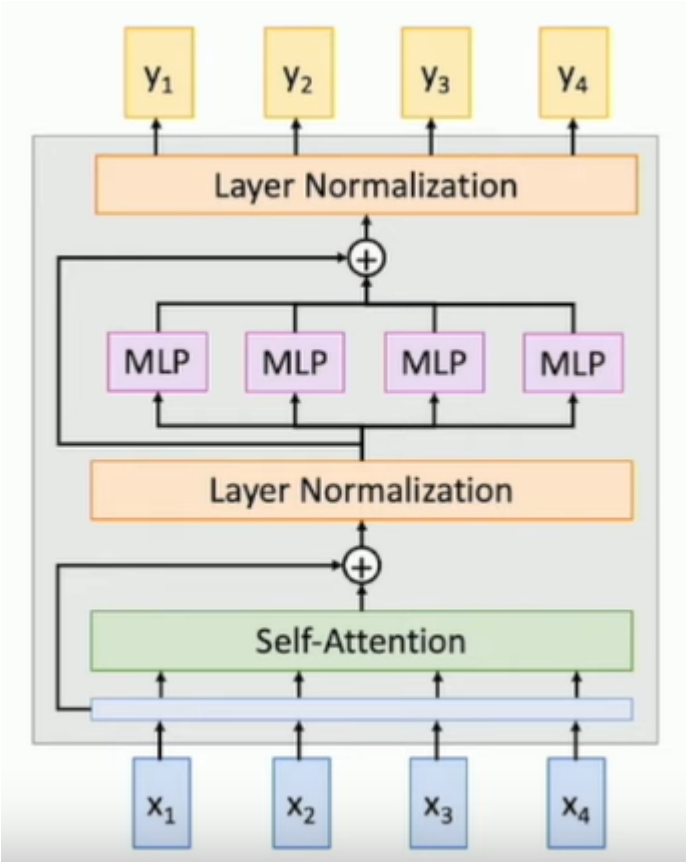
84\ How to process sequences?

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.

==========

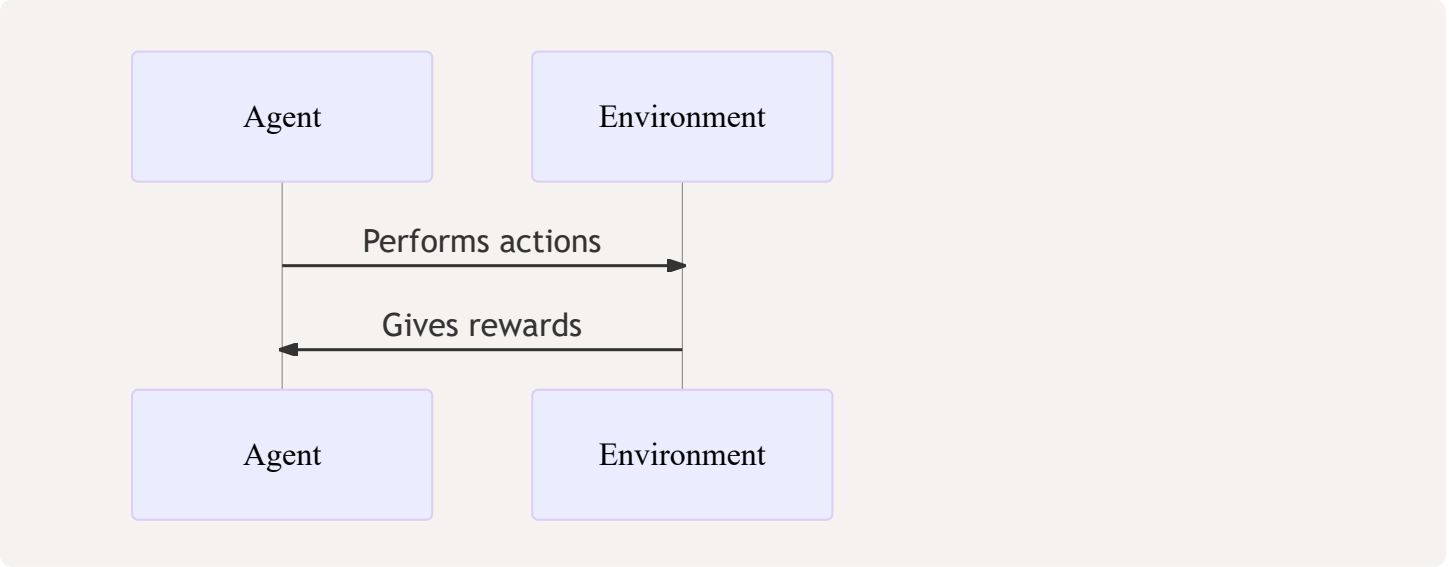
Lecture 21. Reinforcement Learning

4\ Recap.

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

6\ RL

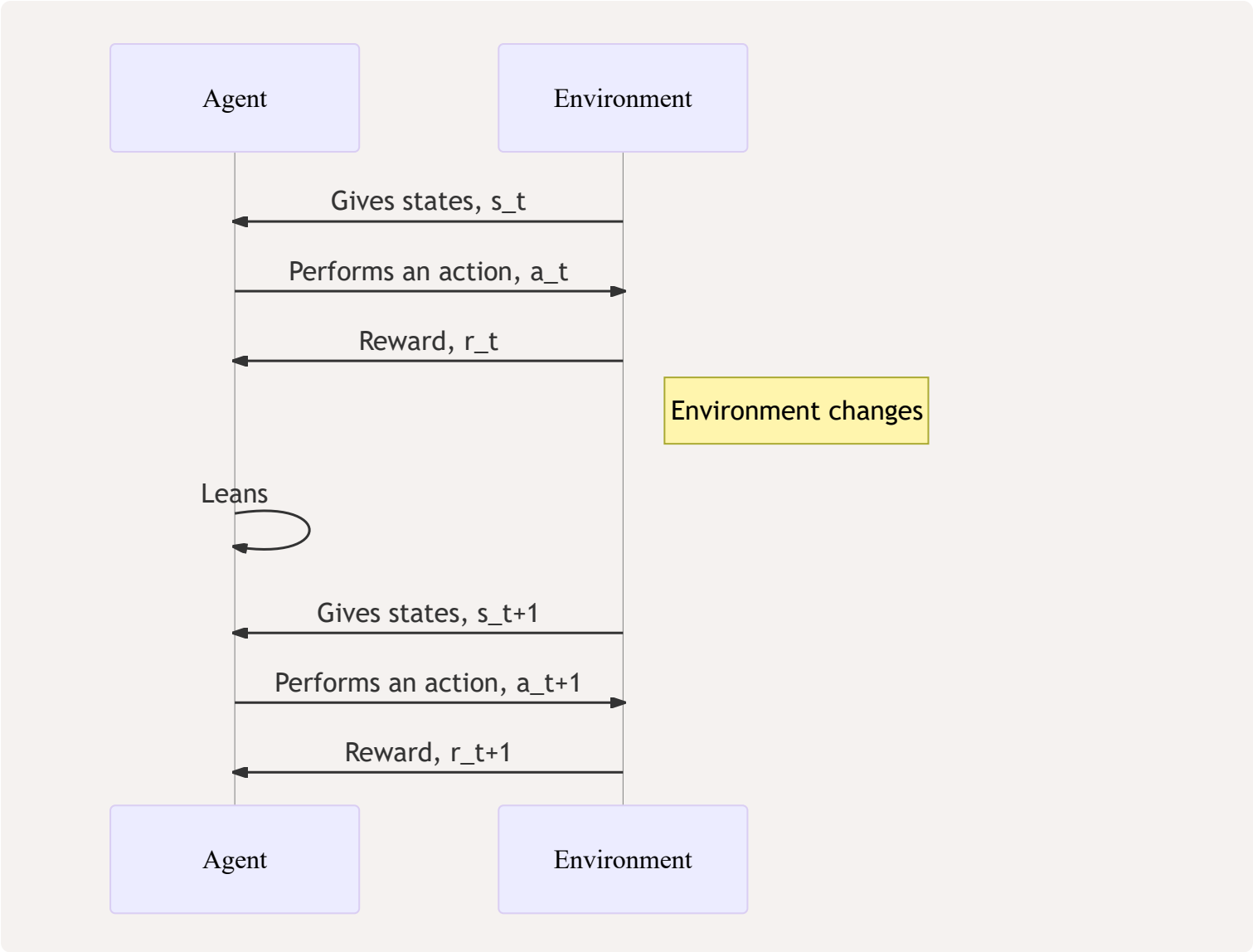
- An *agent* performs *actions* in an *environment* and gets *rewards*. Then it repeats.



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

10\ What's RL?

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

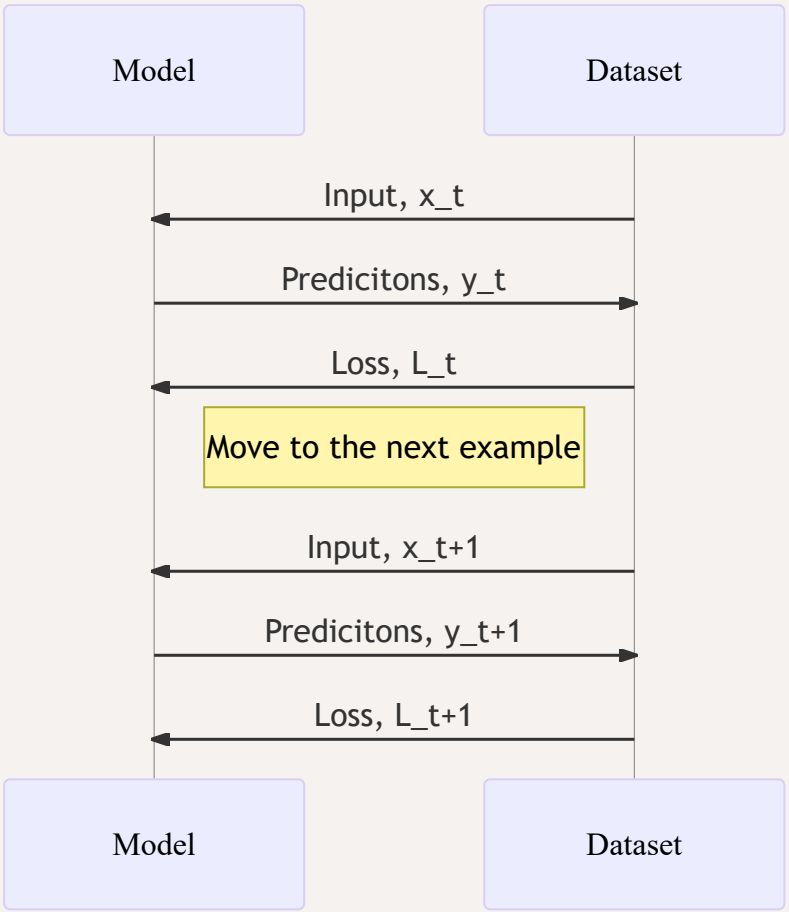


Examples for RL:

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.

====

...

...

...

Deep learning for NLP by NYU

DS-GA 1008 · SPRING 2020 · NYU CENTER FOR DATA SCIENCE

https://www.youtube.com/watch?v=6D4EWKJgNn0
https://atcold.github.io/pytorch-Deep-Learning/

00:00:00

High view on how DL used in NLP.

Great progress over last years. This wasn't expected.

Language Models

Example from GPT-2 about unicorns. This was shoking at the time: details, grammar, etc. There are flaws but looks amazing.

- Language model - density estimation of a text. Prob on every string. (They we can predict next sentence or word.) How model density? Factorize distribution using chain rule:

$$p(x_0, \dots, x_n) = p(x_0)p(x_1|x_0) \dots p(x_n|x_{n-1})$$

- So we want to turn factorization problem into classification problem: given words predict the next word. Then there are a lot of those classification problems! For each position.

Natural language models

- At high level this is always like we are given many words, we move them into a vector. We output probabilities of next words that the model knows of. And we train model to max likelihood. So this requires an encoder.

Context encoder. How to build this?

ConvNets

- First approach - convnets. It maps into a vector. Then applies 1d conv at each timestep (for each word). It had fixed length context (a flaw).
- This worked fast. Downside was the output was dependent on receiptive field (example 'unicord' depends on 5 words only).
- But large text has long dependencies, i.e. books (title there is hundreds of words distant).

RNN

- Concept: at every timestep we have state which is a combined words we've read so far. We combine with current word and output probabilities. Natural model - we do reading like this, i.e. we read from left to right and maintain some state as we read.
- In principle we could bound here unbounded context. In practice issues were no free lunches , we

needed to to have huge amount of info into single vectore.

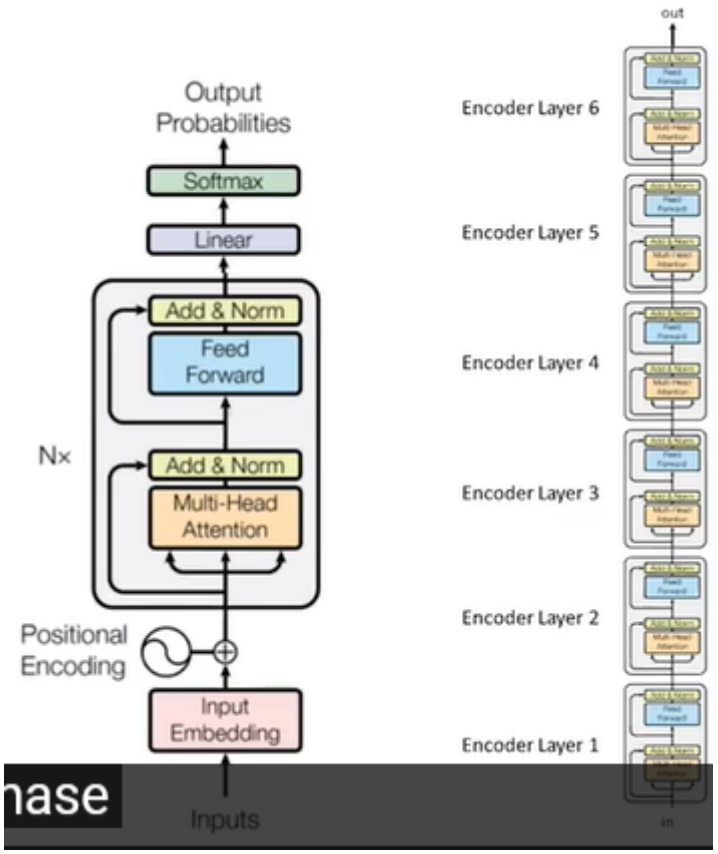
- And also there were practical issues: vanishing gradient, exponensially smaller.
- RNNs are slow, reason is they get output you need to build state for every previous word. So no parallel but only sync.

So ConvNets could do parallel and they are quick but this receiptive field limittation. And RNN were slow.

0:13:48

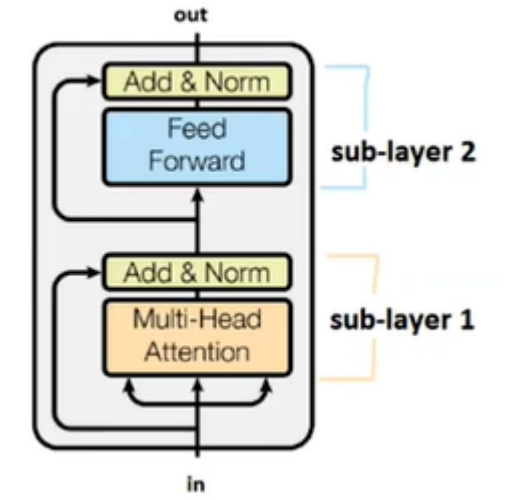
Transformer language model

Transformer introduced 2017 in All You need is Attention. The paper had this figure of transformer:



At the bottom - input. In center it is 6 transformer blocks ('Nx'). Original paper had 6 of those. Now there are a lot of layers now.

In detail, the transformer block:

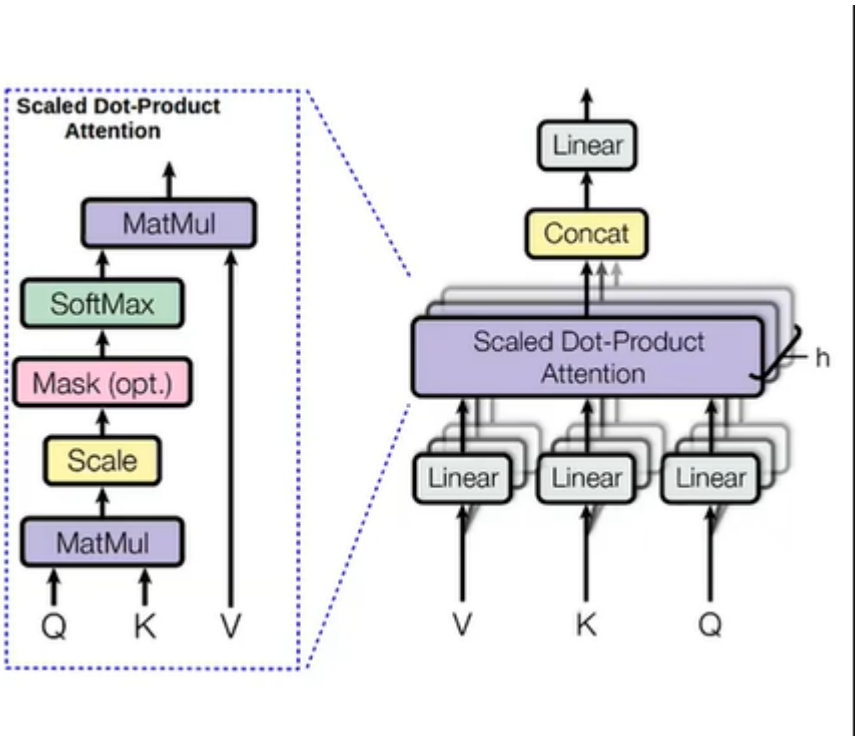


It has 2 sublayers. Both important. 'Multi-head attention' is the key. FF - is straightforward. 'Add & Norm ' - residual connection and layer normalization.

Q: why layer normalization? Rather than group or batch norm. A: it depends on practice, empirecal. In practice it is unclear why but it works.

Q: Do they share weights at everytime step? A: Yes, like in convnets.

Multi-head attention:



Q - query
K - key
V - values

So we compute keys, queries and values from words. Consider some position in text like in example 'These horned silver-white ..', where '..' is for unicorns, the word we are trying to predict. Now keys k are computed from context words. Query q is computed for the unicorn word. Keys and values computed for each word. A key can be some info about a word: is it adjective or verb, etc. A query also contains such info.

With q computed we can calculate distribution of previous words: $p_i = softmax(q, k_i)$. (It is attention weights.) It is attention distribution. Query can be: 'tell me what distribution of previous adjectives is.

Values v_i are concrete values of what a word is. And hidden state will be $h_i = \sum_i p_i v_i$.

This is a sclaed Dot-Product Attention (see the diagram above). We do this many times in parallel.

What's intuition behind this? In the example above, about unicorns. This multihead attention is a way to see all those words that relate to the current query (unicorns) so that the model sees that it is plural, unicorn, etc.

Q: Why softmax? A: Normalization is good here as we get long sequences so we avoid large numbers. In practice it works best.

Q: Why in gray those? A: It is extremely parallel. We can compute this all at once. But we want to avoid 'cheeting' words that looks ahead, in future words. So solution is to add attention masks where every word at the right has negative infinity or zero. Later we will use representation models of languages where we can omit this mask but so far for this task the mask is crusial.

Order agnostic. So far we talked about model that works for anything not particularly for a text. But for text the order is important. How we encode it? We encode the ordering. Technique is 'positional embedding', it is added into input.

Why this good? It gives you connections for every word in context. Contrast to convnets, it can access the previous words states but not far. And in RNN there is this bottleneck. But the great thing is that it is parallizable. It is quadratic thought, n^2 as you do every word with every other word, but you do parallel.

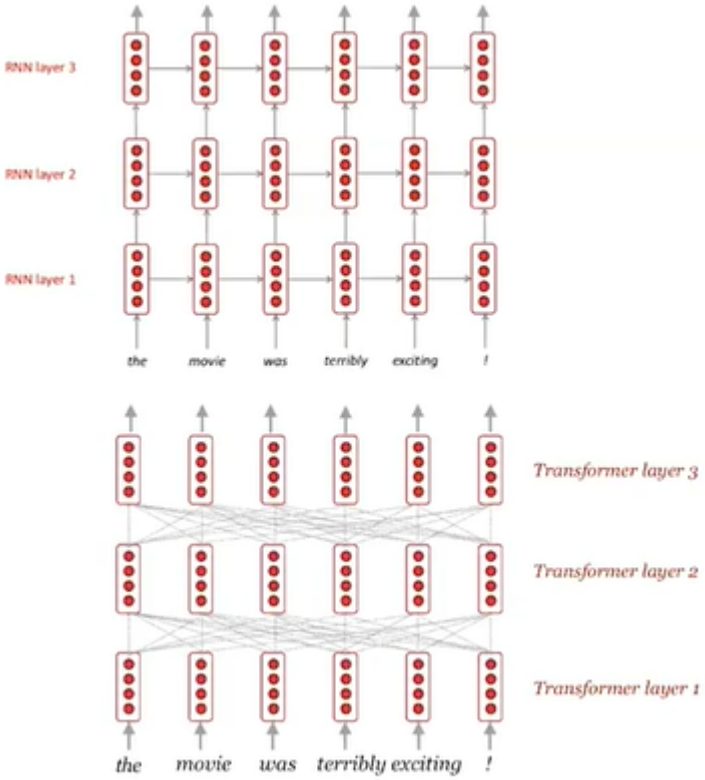
Tricks and facts

00:32:55

- Extensive use of layer normalization to stabilize training.
- Also adjust learning rate - with inverse-square root.
- Careful initialization.
- Label smoothing at the output (?)

History in transfermers in papers. At 2016 at 48. At 2018 with transformers it is 24.

Compare transformer with RNN:



Every word is connected with every other word.

Transformers scale up well. Feed unlimitted training data.

40:00

Q: Can hardware handle the quadratic load? When we have these long connections?
A: Yes, that's a problem that it is quadratic, so we can't model very large text. It is active area of research. Consider k-nearset neighbor. So it's limitting max sequence length. Contrast with RNN, it can do longer but at training time it will backprop many time steps and it is expensive. Side note: But RNN can be effective for other tasks such as number of zeros and ones. In that can we can put into a context vector all needed info. And transformer will have it harder.

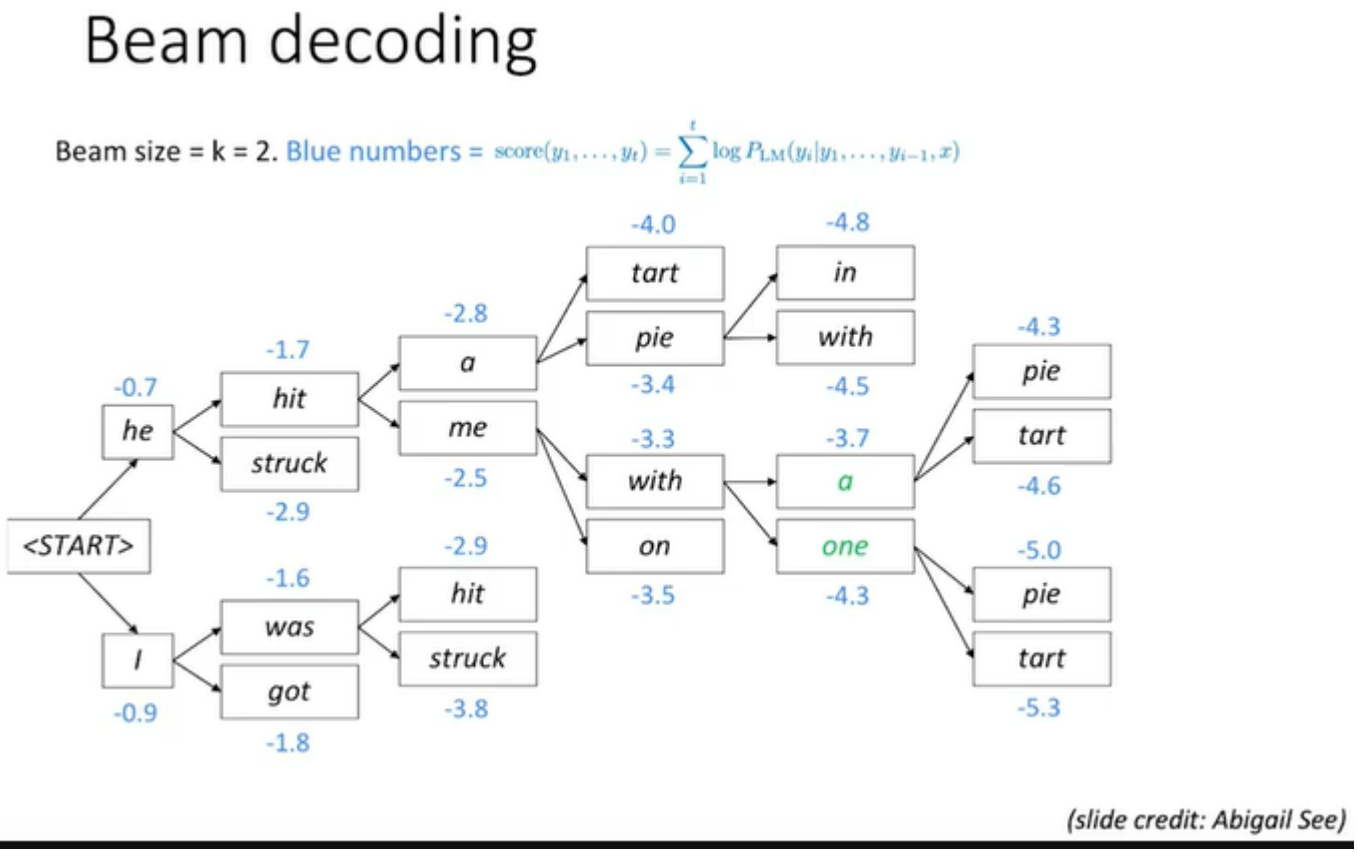
Decoding Language Model

Now we can generate the probabilities but how we generate text? So we have many possible outputs, so we can't just take max. A lot of of context we can't just find max. Can't use dynamic programming. So a greedy decoding was introduced.

Greedy: take most likely word each time. But can't undo the decisions. No guarantee.

Beam search. Middle ground. Key idea: tracking k most probable.

Example:



(slide credit: Abigail See)

We're searching for highest. How far we search? Search to the end token.

Q: Why empty translations? A: (?)

Q: about why 'a' and 'one' in same color? A: don't know.

0:45:32

Beam decoding

- Searches the tree, traverses it till the end till k hypothesis found or till other limit (?).
- Select top one based on normalized probabilities.

(TODO: review again)

55.30 Problem with hypothesis repeating the same output.

Other problem is that sometimes we don't want the most probable but a sample from distribution. Consider a dialog modellliing; we don't want in it to get the most generic answer like 'oh, that's interesting, thanks' but more specific.

Other problem is if we sample bad word we can go into a state we didn't train and it increases the chances of next bad word.

So we do 'Top-k sampling'. From huggingface.co. This truncates distribution to top-k, normalize and sample.

59.00

Evaluating text generation. It is easy to evaluate text model - we have text density. For generated text it is harder.

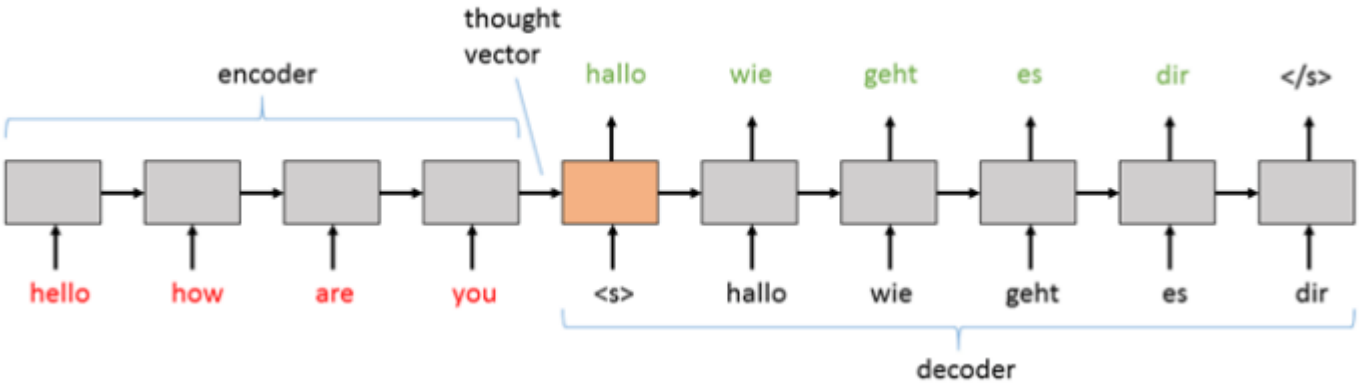
DL for NLP, continued, time: 1.00.00

Seq-to-seq models

Before that it was unconditional langauge models, i.e. there were no task. Uncoditioned lang. models can be used to generate random samples of language.

And conditional lang models are used to generate text given some input. They are usefull. Example: given French translate, given doc generate summary, etc. This is called seq-2-seq models.

Encoder module reads inputs. Decoder module writes output word by word.



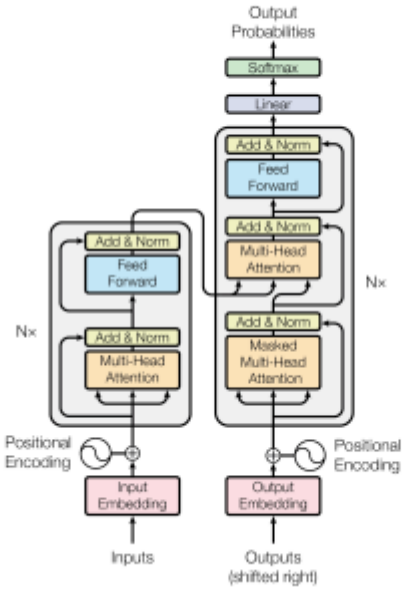
The above is bottleneck with this thought vector.

For Transformer:

Encoder Stack

Each input word attends to:

- All other input words



Decoder Stack

Each input word attends to:

- all input words
- All *previous* output words

Encoder stack was described before. Every input word attends every other input words.
Decoder stack. Every input word attends all input words and every previous words.

This improved substationally previous benchmark scores.

Back-translation, word2vec and BERT's

1:03:31

They train the translation models on European parliament procceedings. It is parallel text of input and output. But can we use monolingual text for translation? Can use use non-labelled text? Then we can do without this labelled text, without input and output pairs.

We do it with back-translation. First is do opposite - reverse translation. Given English output German. Then translate on billions of words. Finally, we do German to English model. ('Bitext' is a pair of sentences in Eng and in German.) This will be noisy translation as the text found in wild, in internet. Can't combine translation model with language model.

Then do again. It is iterated backtranslation.

Massively multilingual MT. This giant NN that translate from all languages to all languages. THis gives improvements.

Self-Supervised Learning

How good if unlabeled text only? This looks like cake - unsupervised learning a lot and a bit of supervised learning - this is the best.

How unsupervised learning?

word2vec

- They trained for word embeding, this is space representations for words. Intuition is that if words are close to each other then they relate to each other somehow. Like 'unicorn' in the example if missing. So they removed it and did encoding, and the label was the missing word.

- The difference now is that they used all surrounding words not just those leading to the current one.
- They found interensting structures in text. Like: 'King - Man + Woman = Queen'.
- It was fast and scalable.
- But those were independent of context. But in real lang they interconnected.

How we add context? For that we would do unconditional lang model, i.e. it produces the next word. And then we would do supervised learning (icing on a cake) to predict other propertices - fine-tuning. This can be all kinds of tasks.

It was GPT like trained. Nice: can do any task. Before that people did various architectures for tasks but now just one model, fine-tune for a task and it works.
Cons: it didn't consider right words. Want representations to depend on any future words.

ELMo. To fix that leftward only context. It did left-to-right and then right-to-left, then contacat. It was shallow representation.

BERT. It overcomes the limittation of ELMo. This made very difference in NLP. It masks some words and then fills in those blanks.

Q: how to maintain for a particular task, when fine-tuning. A: can lost most info.

BERT. (continued). It did very well. Better than humans. Implementation:

- 15% masked
- Scale up batch sizes
- Train on 1000 GPUs

So that made a model that was superhuman on question answering task. This is RoBERTa.

Other variations from BERT:

- XLNet.
- SpanBERT. Masking spans of words.
- ELECTRA. Substitude some and then predict what were substituded.
- ALBERT.
- XLM. This run on many languages not only English.

Limittations:

- They do only classification text. But problems are not classification of text. Modelling seq 2 seq e.g..

BART and T5. It was to train for seq to seq tasks. So it masked the text like BERT but didn't make representation instead feeding this corrupted text into decoder and make it restore the sentence.
- So corruption can be anything.
- Results great: for translation, other.

Open questions. NLP not solved.

- How to integrate world knowledge.

- How to model long docs? BERT-based used ~512 tokens.
- How to get one model that solves everything without fine-tuning?
- Fine-tune with less data?
- Are those models really understand lang?

Take aways:

- Big models beats explicitly modelling linguistic structure.
- Lang model then fine-tuning
- Bidirectional is important.

Q&A

Q: how we quantify understanding language? A: In practice: those models don't understand lang but in next week someone trains model and solves it on superhuman level. .. The classic example: 'the trophy doesn't suit the case because it is too small; the trophy doesn't suit the briefcase because it is too large' (too small / large refers to trophy or a briefcase). Years ago computes did about 60% and humans about 90-95% but now computers do ~90%. But it is more just learning language not common sense! Example : it can give unicorn four horns. This is because no one explicitely writes common sense knowledge. So there is limittations of how far you can learn using only language.

Q: Work on grounded language? A: it is based on a dialog between two people trying to come up with agreement. What's good for you, for me, etc. Using language on purpose is better just seeing text in the wild.

Q: If we fine-tunie those large model. How we quantify what's forgoten. Interesting paper on this, T5 system with 15 bln params. It wasn't trained to answer questions but could do it scarily good.

My questions

- Language model. Context encoders.
  - What does natural language model consist of? What is input for such a model and output?
  - What's context encoder?
  - ConvNets approach. What are advantages and disadvantages?
  - RNN approach. Pros and cons?
- Transformer
  - When Transformer was introduced and what's the paper name?
  - What is the general view of the encoder from the paper?
  - Transformer block diagram. What are sublayers?
  - Do Transformer share weights for every step?
  - Multi-head attention
    - What's keys, values, queries? How they are used.
    - How it is parallelized?

        - How order is introduced?
    - Why Transformer is better compared to RNN or ConvNets?
    - Tricks and facts
        - What are important hyperparameters and methods? Why?
        - What is the time complexitiy for Transformer? What is made parallel?
        - How many parameters are used in recent models (2020)?
- Decoding Language Model.
    - How? We have those probabilities but how to generate text?
    - Greedy approach
    - Exhaustive search
    - Beam search.
        - Describe the algorithm. What does it optimize? What does it output?
- Sequence to sequence models. What are they? Unconditional and conditional models.
    - What encoder / decoder do?
    - Encoder and decoder in the Transformer.
- Back-translation
    - How it was done?
    - Massively multilingual MT.
- Self-supervised learning.
    - word2vec. Unsupervised leaning.
        - How they do it?
        - Pros and cons? Context.
    - Supervised learning as an icing on a cake.
    - GPT. How?
    - ELMo. It's pros / cons.
    - BERT. Pros / cons.
    - Variations.
    - BART and T5.
- Open questions in NLP
- (Q&A secion)

==========

Introduction to STAMP (Part 1 and 2)

STAMP - System-Theoretic Accident Model and Processes.

https://www.youtube.com/watch?v=_ptmjAbacMk
http://psas.scripts.mit.edu/home/wp-content/uploads/2020/07/STAMP-Tutorial.pdf

Part 1

Prof. Nancy G. Leveson

She started in 1980s with torpedos safety (it won't return and hit a launcher). She found that reliability and safety wasn't connected, they often not connected. Spent on it 40 years, worked with many comparity.

STAMP is relatively new. But used in hundreds and hundredds places, petrochemicals, robotics, mining, aviation, nuclear, etc. etc.

5/

What's safety?

- Mishap, accident = loss. Includes injurecies, finince, human lifes, etc. Also inadvertant and intentional.
- Constraints vs goals. Most times safety not in goals but in constraints.
- Safety? Absent of losses.
  This applies to many fields

6/
Cartoon: "we've been stuffing him with worms all day .. and it's still hungry". Means we put efforts into wrong directions. People work hard but it's not effective.

7/
About agenda.

8/ Causality model

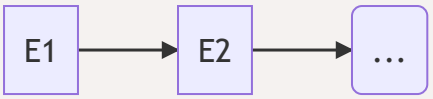
It predicts how will system work in future. No wrong or right. Models just to filter information in messy world.

We want simple question to complex problems.

Example of model:

COE. Chain of events model

Most old. It presents linear chain of events: E2 happens if and only if F1 happens. But it was simplification, it doesn't work now in complex world. And there is one root event, it is somewhere in event chain.
Problem: is it usefull? Will it filter out some important factors?



15/

Exercise. Bhopal accient

1984. Thousands dead, injured. Indian state.

- UC (union carbide?) produced pesticides and other (plastics).
- Uded MIC (methyle isocaynade), chemical when in contact with water causes large amount of heat. Hence its effect on lungs, eyes, other.

Union Carbide had safety features. They had specifications. Used "defense in depth": several backup systems that were expected to protect from disaster. Hence probability of diseaster is multiplication of probabilities.
Those were basically several systems that were supporced to cover each other. The MIC chemicals contained in tanks (should be half full), then there were a scrubber that utilized the chemicals, the flare tower to spread extra in atmosphere high above the ground, the water curtain (to pour out if previous ones didn't work), and the refrigirator that kept temp around 0 deg. The alarm was there if temp reached 11 deg.

'Chain' of events. My notes:

- Worker was not taught well so he/she didn't insert those safety disks.
- No body checked no safety disks.
- When gauges showed pressure and temp rise no relevant procedures taken. They ignored them. So the operators not taught well enough. Hence the directors or those in control is responsible for not teaching those operators.
- Operators didn't react quickly to the leak when noticied.
- Superviser ignored the urgent report.
- Backup systems didn't work. The chemical escaped in a fountain high above the ground. So the designers of those backup systems did bad also.
- In emergency. Superviser and ooperator didn't risk to help the situation. So they hired wrong people?
- The gauge for soda circulation.
- The detector for spare tank not working. Technicians' bad.
- And on and on. Police, siren.

> Mine view: Everybody did bad. No root cause? Owners of this is the root cause. They didn't make sure all procedures done.

Root cause selection is arbitrary because we want to feel better like we are in control. Overall seeking root cause is bad.

UC made the worker guilty - wrong.

But many questions raised: why all backup system failed, why they didn't attend before tea break, etc., etc. So this is not a chain of events. We want to look at conditions of events. This is level 2. Level 3 is systemic factors.

Hierarchical models:

- Level 3: systemic factors
- Level 2: factors
- Chain of events.

Addional info:

In theory failure probabilities are low but in practice opposite! Flare town, scrubber, curtain, etc. were not operating as expecting, couldn't handle those conditions, etc. This is not uncommon: gauges turned off, procedures not followed, instructions were not full (insert slip disk), worker very low trained, alarm siren fired too often so turned off, etc. etc.

My view now:

> So all factors are bad, whole plant should have been closed by some state inspection. Agree, no root cause.

And additional questions raised. Why no supervision for the worker, why maintainance so poor, where is operation safety group? etc. etc.

Level 3. Systemic factors.

- Systemic factors link to conditions and even to events.
- Often dropped from accident reports.
- They are 'safety culture'. They are widely applied.

Systemic factors at Bhopal:

- Low demand for MIC. Hence reduction in expenses on safety.
- Training at US stopped for workers.
- Skilled workers live the plant for better places.
- Indian gov. requires only indians workers. Last US superviser visit 3 years ago.
- Safety engineer resigned 1 year ago and replaced with unskilled worker.
- Low moral
- Refrigirator shut down.
- There was the audit 1 year before but ignored!
- There were several warnings and accidents which were ignored again.

This is **common**. Usually a root cause is some operator who was at the beginning. This might deflects attention from powerful parties, often, esp. if a pilot dies (in case of aircraft accident). There are various names for this COE. We need new causality model.

LESSONS:

- We need to look beyond events, i.e. at conditions, systemic factors.

1.05.00

59\ Dealing with complexity

Complexity in modern world rises: introduction of software, internet, complex system. Starndards for safety created 50-75 years old.

Examples:

- Missle launch by a navy aircraft. Causes, factors? Requirements for software were flawed.
- Mars Polar Lander. Landing crash. Requirements? No. Early start of software.
- A320. Reverse the thurst. Software to protect from turning on thrust if not landed. System required flawed.
- Ferry problem. No way to move cars because they were blocked by car owner company (theft assumed). Flaw? Unkonwn unknowns.

Two types of errors:

- A component(s) failure. Like a landing on Mars.
- Components *interaction* failure, interaction failure between them. Unexpected interaction of components. Like reverse thrust disabled. Software exacerbated this.

Confusing starndard in Safety and Reliability: 1. Unrelable but safe. 2. Unreliable and unsafe. 3 Reliable but unsafe.

Software.

- Software doesn't 'fail'. General Purpose Machine + Software = Special Pupose Machine. Software doesn't fail because it is a pure design.
- Allows unlimited system complexity:
  - Can no longer: plan, anticipate etc. Test thouroughly.
  - **Context** derterminines if safe. COmponents don't fail. But integraion them in other context fails, Like butter knife. Not possible to look at software alone and determine 'safety'. If alone you CAN'T determine if it is safe only in context.
- Role of software in accidents is almost always because of requirements.
  - Incomple or wrong assumptions.
  - New states.
  - Level of rigor nothing to do with safety.

76\ **LESSONS**:

- Need to consider design errors, not just component(s) failures.

- Software - doesn't fail but can lead to unsafe behavior, impossible to test fully.

77\ Software changes the role of humans in systems

78\ Software changes roles.

- Traditional approach seeks a cause(s) in an operator error(s). Hence more training or making their work rigit, etc.
- But accidents show that a cause is complexity. Systems were reliable.
  - Accidents: 1) B757 in Cali; 2) Columbia, Tu-204, Moscow, 2012
- We can't isolate humans and systems in critical situations. But no training in this integration!

85\ Operator error is a symptom, not a cause!

- Because all behavior is affected by the context, role of operators changes.
- Wrong to blame operators when systems are designed so that they lead to errors.

87\ **LESSON**:

- Need to integrate human behavior, software and hardware engineerings together.

87\ Part 2: STAMP

How do we solve those lessons?

We need models that handle 'unknown unknowns', human factor, policy, interaction between components, etc. AND interactions between these.

New tools. Paradigm change. It is not that previous all bad, now is good. Not. It is integrating previous ones and extends them, includes old. STAMP doesn't invalidate CoE but it says that it is not enough.

Wrong approaches:

- Pretend no problems.
- New tech and levels into old methodologies.

94\ STAMP is

- New causality model
- Allows to build more powerful tool.

How to cope with complexity? 1) Analytic decomposition, statistics, systems theory.

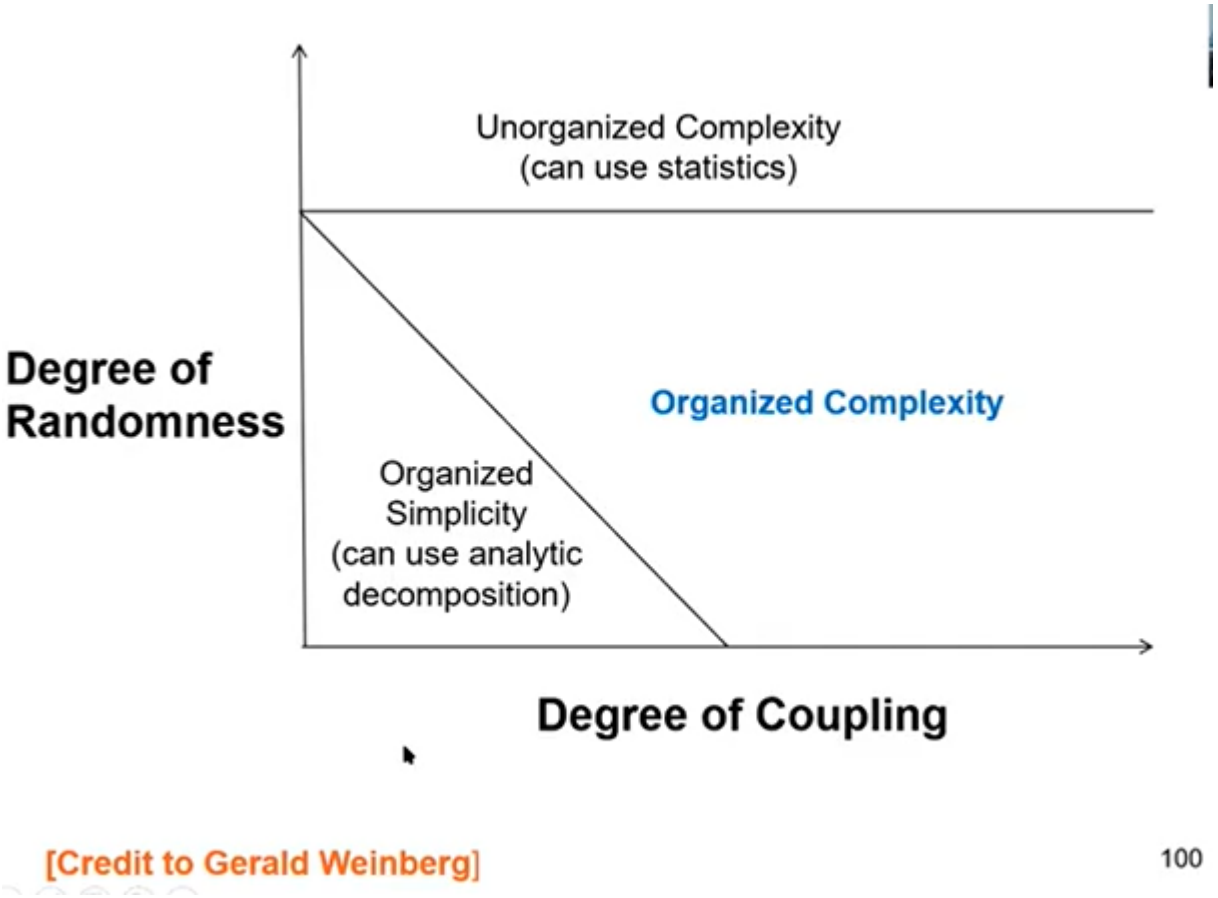
96\ Anal. decomposition.

Divide and conquer

1. Divide.
2. Examine pieces separately and combine.
   - Remember it is wrong to assume that components behave separately, etc. So people tried to

decompose accidents and what went wrong there. But that also not good, still bottom up as all systems tightly coupled, software, connected, etc.
    - Need theoretical basis.

At figure below we see that we can't use analytic decomposition.

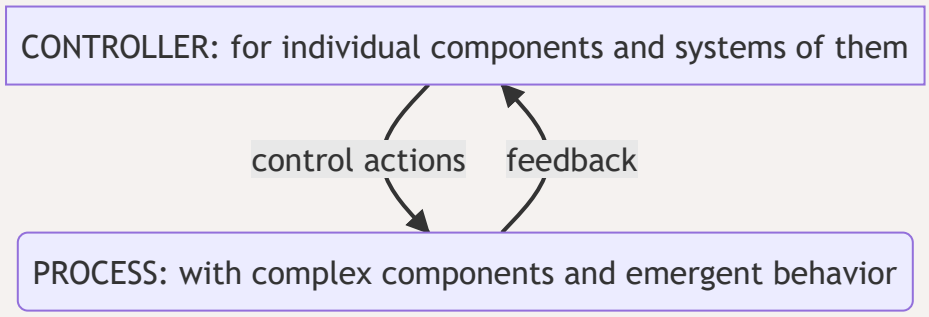


Systems Theory is a way to deal with complexity. First used in 1950/60s.

Systems theory:

- Focus on a system as whole
- Emergent properties from a whole system. The whole is greater than the sum of its parts. Arise from comp interaction.
    - Safety and security are emergent properties.
    - Example: only taking a valve we can't tell if whole system is safe.

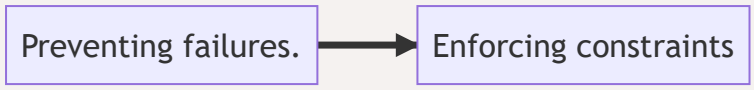
Solution is to have controller for those emergent properties.



Safety contraints. These are given requirements, constraints we should address like preventing hazards

from leakage, virus contamination, etc. Then we break those into componenets, then to design.

Paradigm shift:

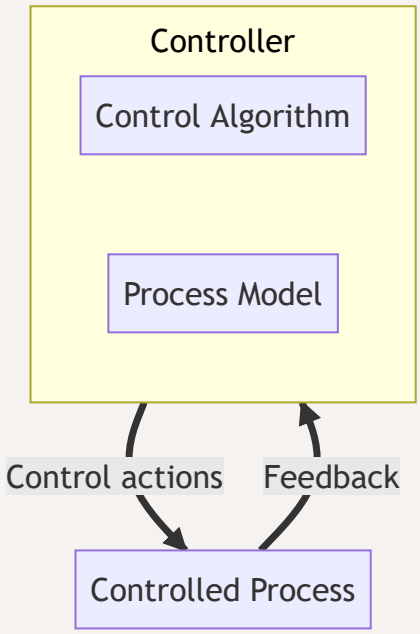


From a reliability problem (a component(s) failure) to a control problem (interaction, emergent behavior).

What's 'Control'?

- Through design to 'control' components interactions.
- Through processes of manufacturing, maintenance, other.
- Through social controls: governmental, culture, insurance, individual motivation, etc.

Safety as control problem:



Examples:

1. Mars Polar Lander. Hazard: too much force on landing. So the control problem is that software (controller) receives signal from a leg (complex process) and treats it as if the space craft landed. And wrong decision: turn off descend engines.
2. Aircraft and reverse thrusters in Warsaw. The control problem is that software controller ignores pilot command to turn on reverse thrusters.
3. Aircraft and reverse thrusters in Moscow. The same as in 2 but additionally some.
4. Missile Release Mishap. Hazard: Friendly Fire. Control problem is Software optimizes missle launch success by launching non-dummy missile.

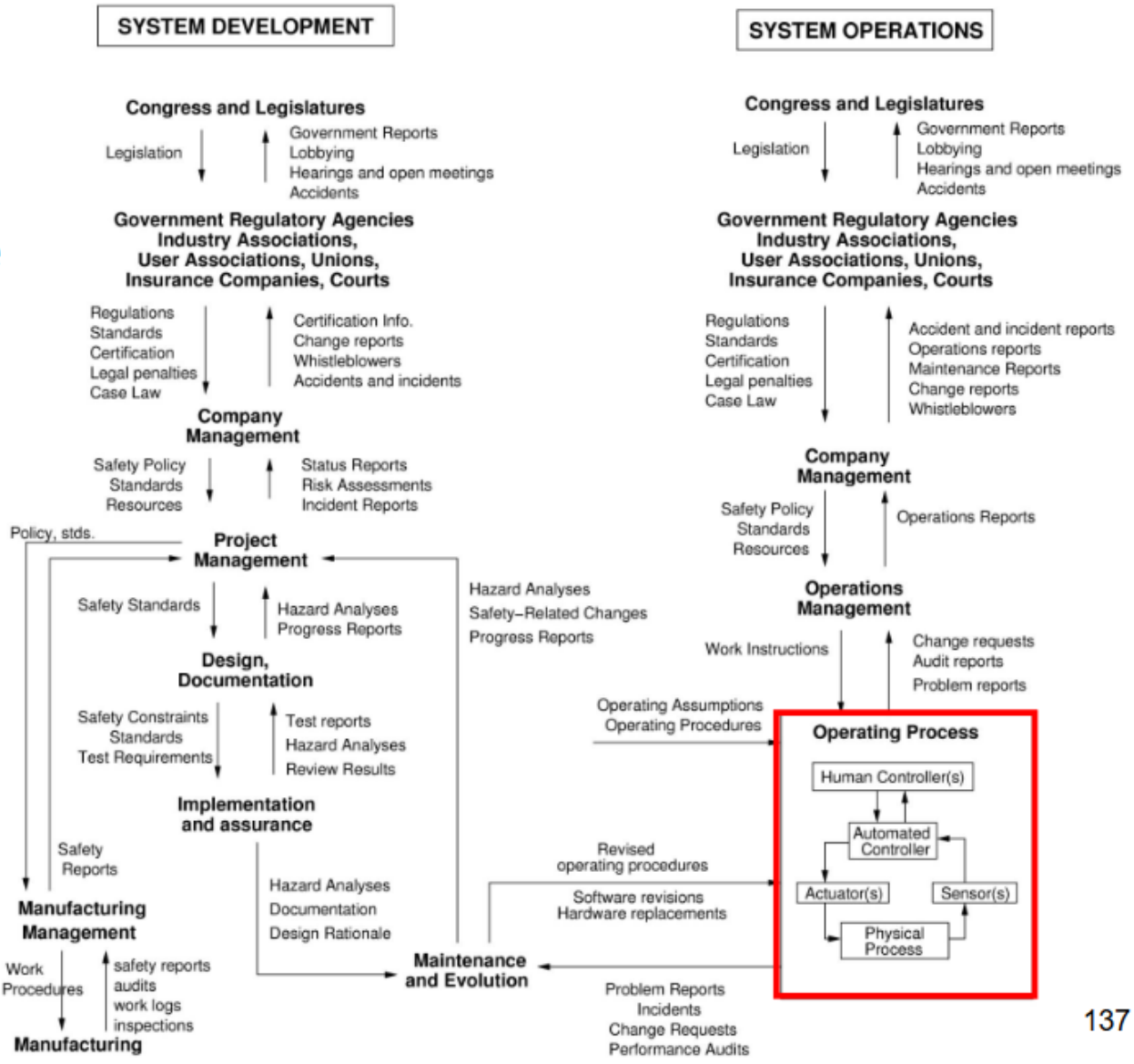
So analyze hazards and accident with STAMP. We can treat components, systems in this fashion: controller and processes. So that we treat it as control problem.

135\ We can use STAMP for Safety and Security.

Every complex system can be viewed at diff levels as a control problem: software vs hardward, software vs human, human vs human. afety and security are connected to losses. So the same paradigm shift for security.

- Example - Stuxnet (computer worm), it made centrifuges to run fast while got on plant computers (Windows); so how to control? same thing - add a controller: mechanical limiters; so preventing hazards not keeping those worms out.
- Example. Safety Contorl Structure. From System Development to System Operations. Large diagram, see below. We can treat it *all* as control problem. The red rectangle is an example of treating this Operating Process as control problem.

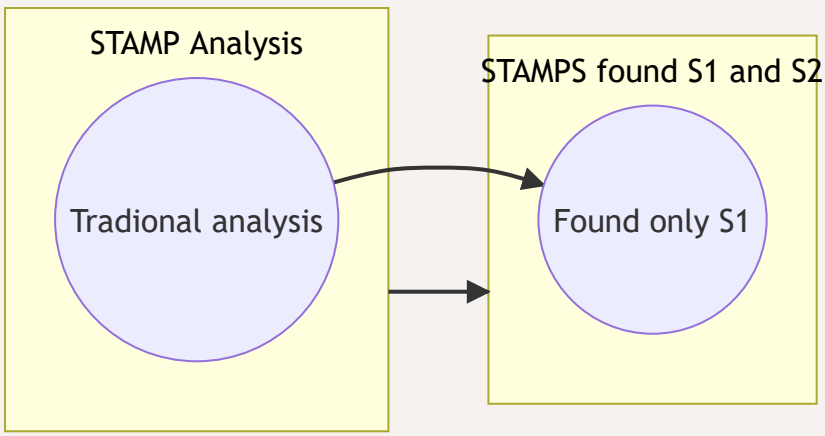
Example Safety Control Structure (SMS)



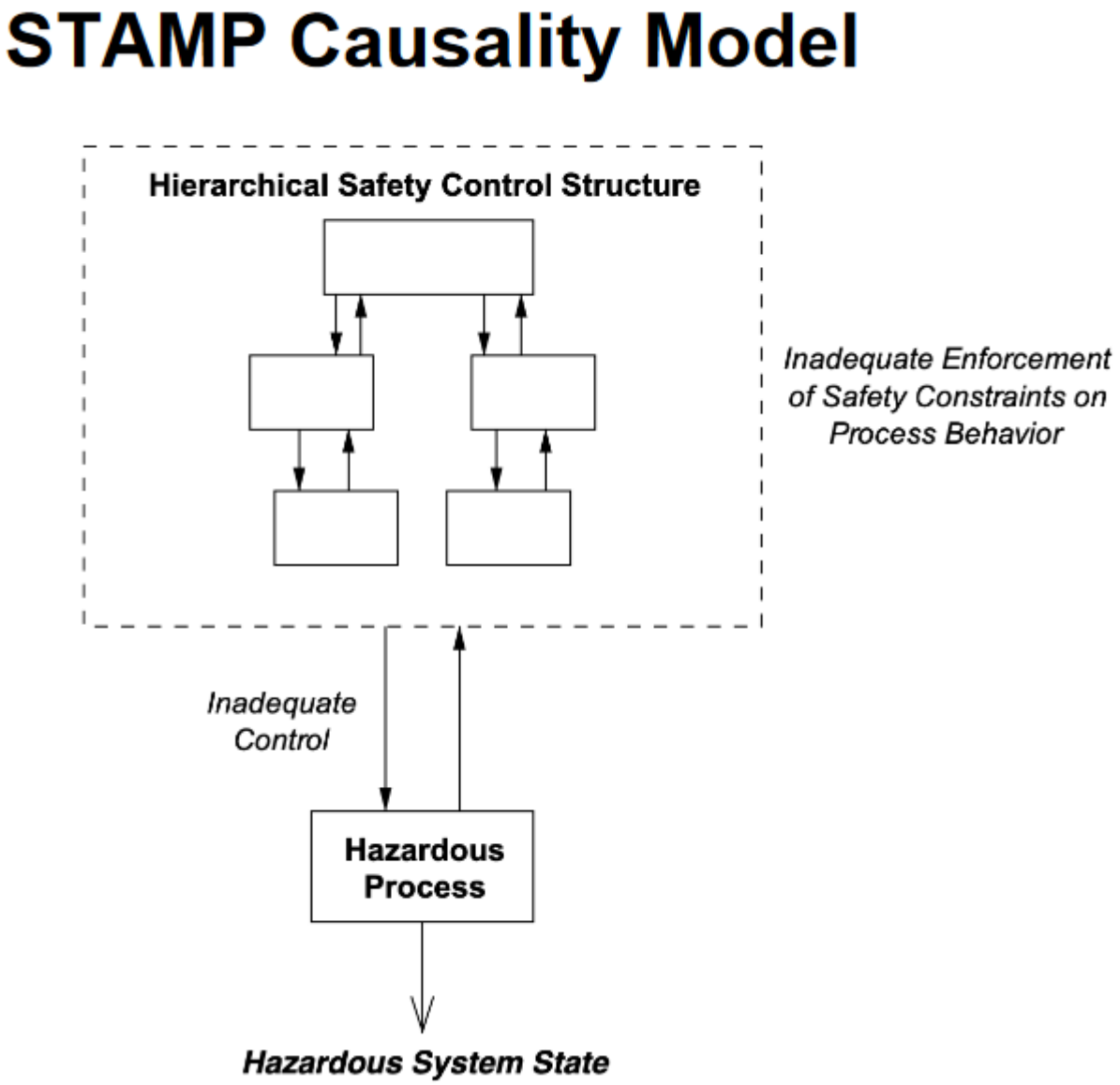
137

139\ STAMP - System-Theoretic Accident Model and Proceses)

- New, causality model
- On systems theory not reliability theory
- As cotrol problem
- To very complex systems
- Includes all previous methods: human errors, software erorrs, etc.



- STAMP causality model:



- STAMP include complex interactios. Example is 'Herald of Free Enterprise' accident.
- Systems thinking: not thinking in lines but in circles or more complex.
- Example: Columnbia Shuttle Loss. There are complex connections (one increases /decreases other thing increases / decreases, etc.)
- Safety as Control Problem:
  Goal: design an effective control structure that eliminates or reduces adverse events (losses)

147\ What tools are available?

- Processes: system engineering, risk management, etc
- Tools: Accident Analysis (CAST), Hazard Analysis (STPA), etc.

Does STAMP work? Evidence? Evaluations and estimates of ROI (return of investment):

- Hundreds example. All show STPA is better, requires fewer resources.
- 15-20% for ROI when using STPA.
- She mentions 900 mln USD project.
- Example. Ballistic Missile Defense System (MDA). 2 people in 5 month found many flaws and they fixed those early. Hence saved mlns of usd.
- Example. Blackhawk hylicompter. Found additional hazards.
- Example. Navy Escort Vessels. Found scenarious not found by prev analysis. Nave ignored their analysys. Later accident stamp predicted.
- Example Nuclear power plant. 2 grad sudents spent 2 weeks. Found new 16 accidents that prev analysts didn't know.
- (Other) Found many more causes. Medical. Automotive electric poower steering system. hospital. Organizational. Supply chain. other.

Approach, sumary:

- Emphasizes building in safety rather than measuring it and adding later.
- System as a whole (not components)
- Larger view of causes not just failures.
- Goal is to design and operate NOT to predict the likelihood of a loss.

System Eng Benefits:

- Finds underlying assumptions.
- Finds incomplete info
- Handling intended and unintended func
- Includes all software, operators,etc
- For very complexity systems
- Can do early
- Traceability from requirements to artifacts.
- Models show high level view.
- Augments sys engineering process.

Refs:

- http://psas.scripts.mit.edu
- http://mitpress.mit.edu/books/engineering-safer-world

My questions

- Part 1.
  - Give definition of safety. (Mishap, accident, loss.)
  - 'Chain of events' (CoE) causality model. What's it?
  - Lessons learnt from Bhopal accident? Levels of causality?
  - Dealing with complexity in modern world:
    - Two types of accidents? (When components.)
    - Reliability vs Safety. Name three types of scenarios.
    - Does software fail? Why?
    - For complex systems with software, where is the most likely a flaw? Why?
  - Lessons learnt from model complex system accidents?
- Part 2. STAMP
  - What's the paradigm change? And what's wrong approaches?
  - 96\ Divide and conquer approach
  - Systems theory.
    - What's it? When the theory developed?
    - What it states about a complex system? How to handle it?
  - What's paradigm shift in terms of systems theory? Safety as control problem.
    - Give examples of where there was control problems.
  - Can we use STAMP for security?
  - STAMP abbreviation?
  - Stamp causality model diagram
  - Goal of STAMP?
  - Evidance that STAMP works?

==========

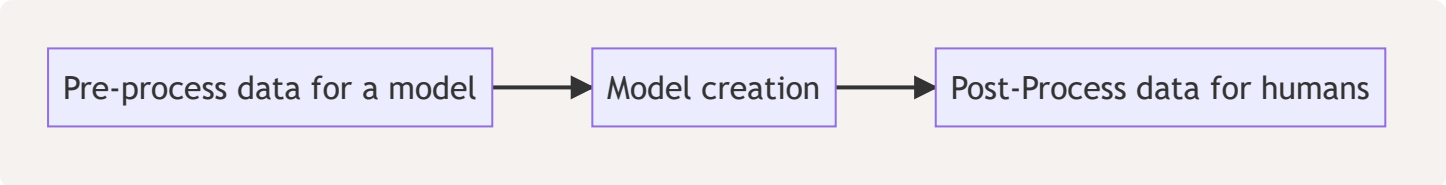
END

Hugging Face NLP course.

https://huggingface.co/course/chapter1/2?fw=pt

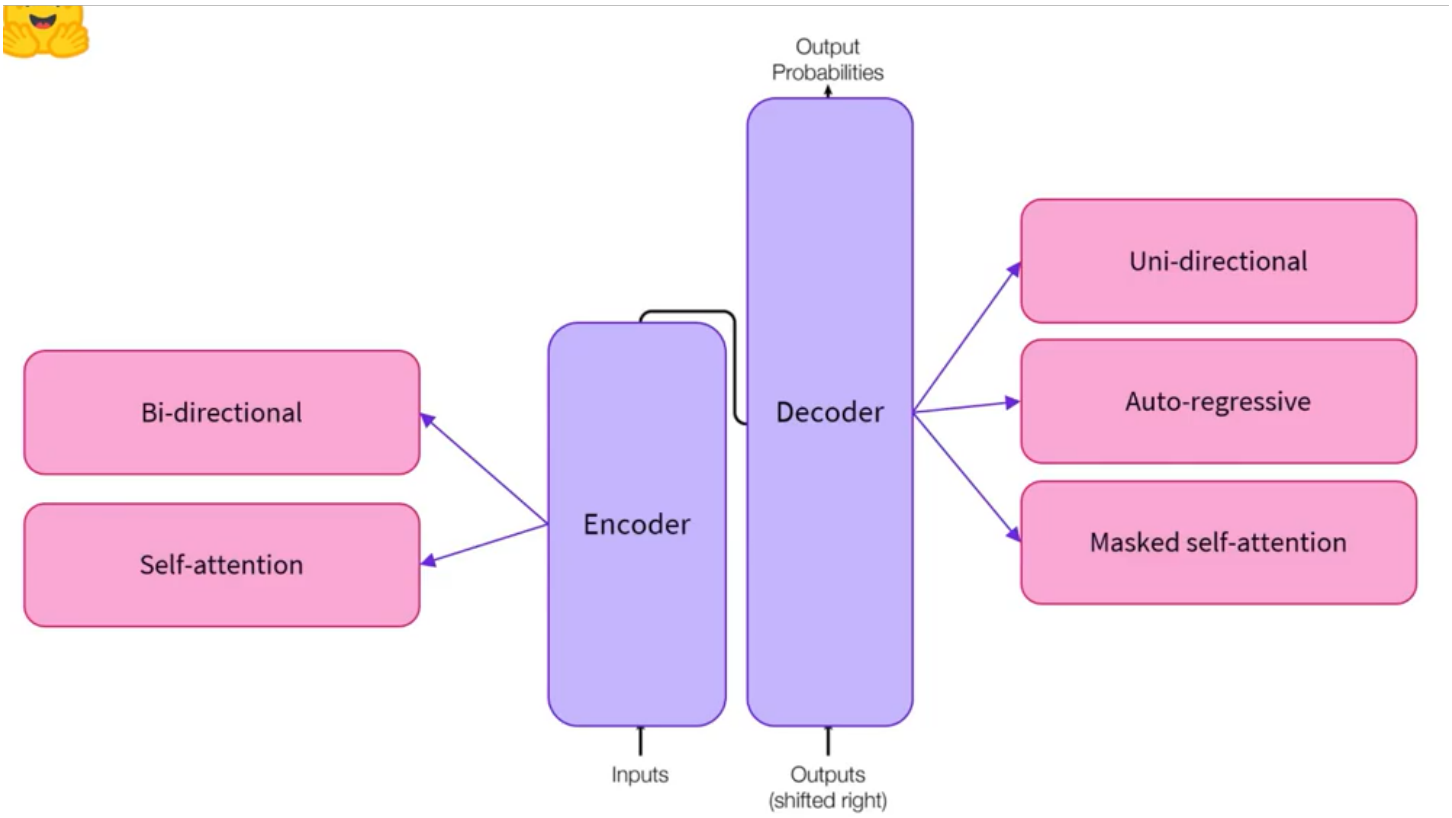
1. Transformer models

- NLP - Natural language processing
  - NLP = ML ∩ linguistics.
  - Tasks: language translation, summarizing text, filling removed words, generating new text, etc.
  - Challanging because we need to model text somehow.
- Possibilities of Transformer models:
  - Summary
  - Translation from one lang to another
  - Classify sentiments
  - Classify area
  - Classify words
  - Continuing, generating text
  - Fill in masked
  - other
- Pipeline function: final object for a task (sentiment, zero-shot)

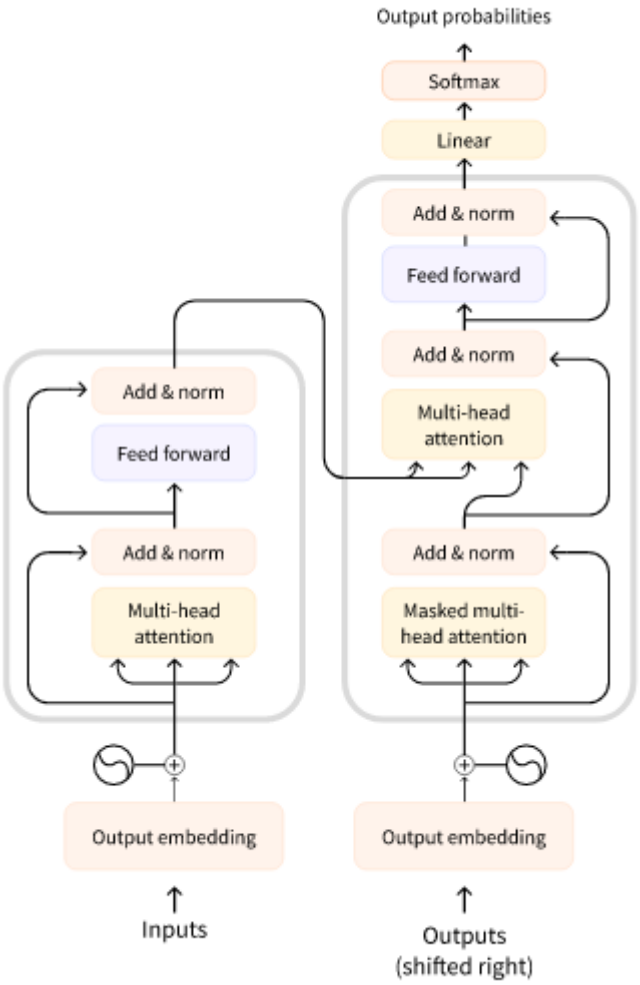


  - Uses diff models (gpt2, openai-gpt, etc.)
- Tasks: 'question-answering', 'summarization', 'translation', 'ner' (name entry recognition, classify words), 'zero-shot' (classify when no labels at train)
  - 'Zero-shot' - no need for fine-tuning for tasks.
- How Transformers work?
  - Categories of NLP models:
    - GPT-like. Auto-regressive.
    - BERT-like. Auto-encoding.
    - BART/T5 like. Seq-to-seq.
  - First they're trained on language model (not useful in practice) then they're fined tued for a specific task (masked text, causual language model(next word)) by using labled data.

- They are big. Billions of params.
    - A lot of CO2 emmisions like a car in its lifetime but depends on carbon footprint. That's why use fine-tuning, transfer learning, also use random hyperparams search. To measure - https://codecarbon.io/, https://mlco2.github.io/impact/.
- Transfer learning
    - Using other trained mode for other task - fine tuning. Why? 1) Better performance, 2) less data and 3) less resources (time, etc) for new task.
- Transformer Architecture



    - Parts:
        - Encoder - gets text input and generates 'understanding' of it.
        - Decoder - transforms input into another output.
    - 3 architecture types: encoder only, decoder only, encoder and decoder (seq 2 seq).
    - Attention layer. The main layer as it attends to specific context to build lanugage model, i.e. each word has not only its meaning but also depends on the surrounding context.
    - Original architecture:



    - Used for translation.
    - Encoder attends all text (left and right) so can translater.
    - Decoder attends only left output text till current i-th word and all input text.
  - *Checkpoint* (like BERT_base) means a particular set of weights used to train *architecture* (BERT) while *model* can mean both.

- Encoder models
  - AKA *bi-directional*, *auto-encoding* models as they access all the input. Examples: BERT, ALBERT, etc.
  - Good to extract meaningful info, classify, etc
    - MLM - guessing a randomly masked word.
    - Sentiment analysis

- Decoder models. AKA *auto-regressive models* (use previous output as input).
  - Attends only the words before current one. They mask the input, masking.
  - To generate text. NLG - natural lang generation. Causal Language Model.
  - Examples: GPT-2, etc.

- *Sequence-to-sequence* models. AKA encoder-decoder models.
  - Trains decoder and encoder.

    - To generate text based on other text
        - summary,
        - translation (encoder in English, and decoder in French, e.g.)
    - They don't share weights.
    - Examples: BART, T5.

- Bias and limitations. Can generate sexist, homophobic, etc. text as trained on data from all over the internet.

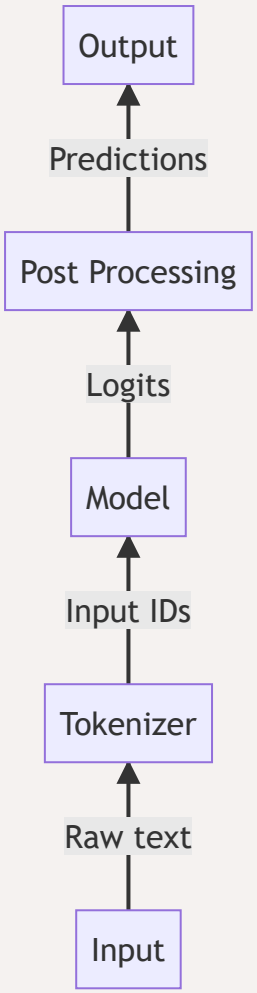
See quiz at https://huggingface.co/course/chapter1/10?fw=pt

Questions:

- What's NLP?
- What Transformer models can do?
- Three types of Transformers.
- What's transfer learning? Fine-tuning?
- Original architecture from All you need is attention paper.
- Encoder models. Their main props?
- Decoder models. Their main props?
- Seq2seq modesl. Their main props?

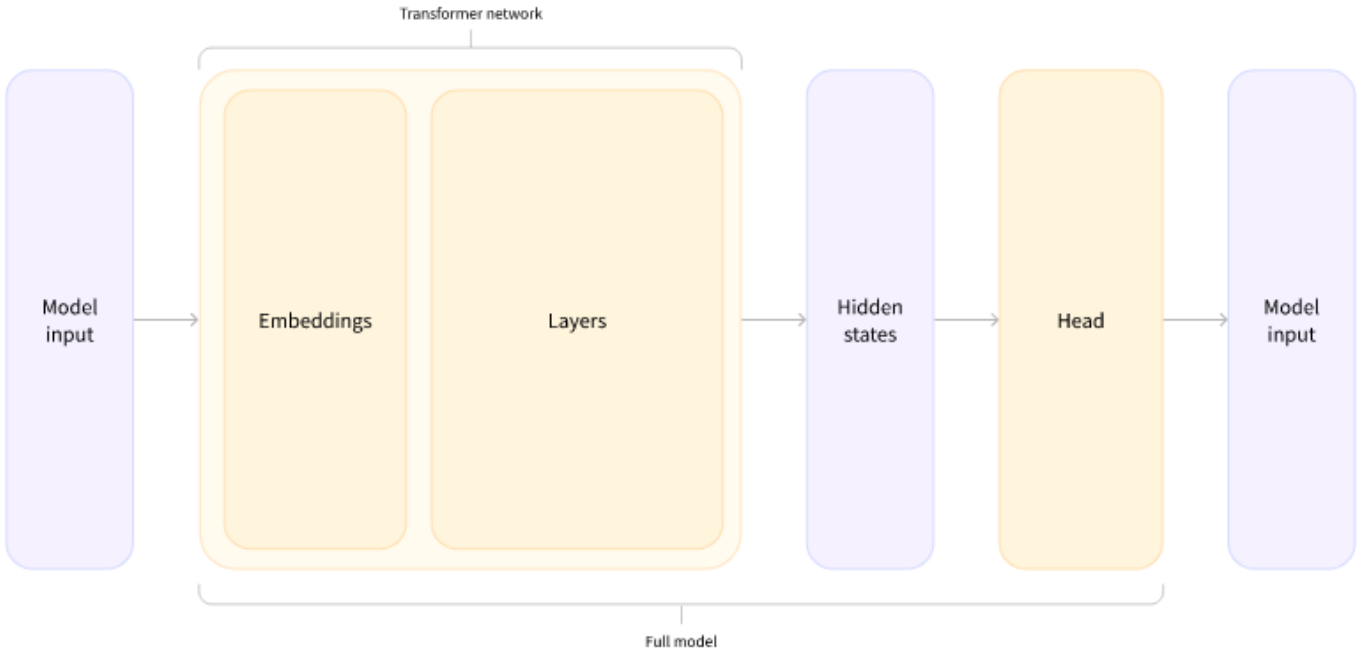
2. Using Transformers

- The Tranformers lib to handle the zoo of many NLP models.
- Pipeline func



`transformers.AutoTokenizer` and its `from_pretrained(checkpoint)` to load tokens of a model.
Logits: it is from a model. Those are not predictions (it is not normalized).
Predicions: after softmax, those are probabilities.

- Tokenizer: splits words, maps those words (now tokens) to integers.
- Embeddings - to convert token numbers into vectors
- Head - receives hidden layers input, to specialize in tasks. Hidden state - how model understands input; then hidden state passed to the head to perform a task.
- logits - un-normalized predictions (before softmax)



- Models.
  - How to create a model? Use `transformers.<Model>.from_pretrained(..)`
  - How to configure model? Use `transformers.<Model>Config`
  - How to save model? Use `save_pretrained`
  - How to use model? Use `transformers.<Model>(<model config>)(<model_inputs>)`
- Tokenizers
  - Models needs number not words. So translate those to numbers. How? Depends on models.k
  - How:
    - Word-based. By spaces or by punctuation. Use a vocab to map to tokens. Limitted by vocab (10K or 500K), unknown words.
    - Character based.
      - Less size of vocab
      - Fewer unknown chars then.
      - How punctuation?
    - Subword tokenizer.
      - To fix issues in word-based and char based tokenizers.
      - Break into meaningful subwords if larger word.
    - Other: byte-level, wordpiece, other.
  - Loading / saveing
    - `AutoModel.from_pretrained(<checkpoing>)`
    - `transformers.<Model>Tokenizer` , etc.
  - Encoding.
    - How we make numbers from tokens: text to numbers.
    - It is `ids = tokenizer.convert_tokens_to_ids(tokens)`
  - Decoder.

        - Reverse process. Use `decoded_string = tokenizer:decode(<Vector here>)`

- Multiple sentences.
    - Batching for parallel proc.
        - Models expect batche.
    - If diff length? Padding (with '0') or truncate.
    - Attention layer should get attention mask to avoid this padding.
    - Long sentences? Truncates. Typically - 512 or 1024.
- All together.

3. Fine-Tuning a pretraine model.

- This section is about how to load, preprocess a dataset used for our specific task. Then how to fine-tune our model for this dataset, how to train it.
- Processing the data. This is from a row text to batches. So that we can fine-tune our model for a task.
    - Loading datasets fom the Hub: Use `DatasetDict` from `datasets.load_dataset()` to download datasets from th eHub.
    - Preprocess: use `transformers.AutoTokenizer`. Check `attention_mask`. Use tokenizer of a checkpoint.
    - Dynamic padding. This makes the same size batches.
- Fine-tuning a model with the Trainer API or Keras.k
    - Use `transformers.Trainer` when have tokenizer, datasets (valid, train). Then `.train()`. Also give it an evaluation function as below.
    - Evaluation and metrics (to give it into `Trainer()`)
        - Use `trainer.predict()` to get predictions.
        - Use `evaluate` lib to evaluate accuracy andother.
- Full training
    - (This repeats the steps above in one run)
    - Then we run training loop: batch, backward loss, learning rate, etc.
        - optimizations: AdamW, learning rate decay, etc.
    - Evaluation loop: on batches, then accumulate.
    - Speed up with Accelerate: trains multiple batches simultanuously on GPU.

...

...

...

...

...

...

...

...

...

...

...

...

...