Recurrent Neural Networks

- Some of the more recent results in RL and ML more broadly use more sophisticated network architectures. Here, we'll try to build up some more recent architectures from their constituent parts.
- What if our input comes as a sequence

  $x_1, x_2, x_3, \ldots$
- Motivating problem: natural language processing.
- One version: input is a sequence and output is also a sequence.
- For example, in translation the input might be "the brown cat jumped over the black fence" while the output is "коричневый кот перепрыгнул через черный забор."
  This presumes we can have some representation of each word [more on this later].
- Alternatively: input is a sequence and output is a scalar. This happens in sentiment classification.
  "BUY BITCOIN!!!" —-> +1
  "Guys, I just lost all my savings" —-> -1
- https://venturebeat.com/2020/12/27/how-machines-are-changing-the-way-companies-talk/

- So the input is a sequence and the output is a sequence.
  One possibility: do nothing differently.
  Take $x_1, x_2, \ldots, x_T$ and concatenate them.
  Enter them into the NN.
  Read off the output.
- Issue #1: requires you to know the length of the output.
- Issue #2: let's recall our discussion of CNNs.
  Performance improved when we built an architecture that was very well adapted to the structure of the input.
- Let's try to do the same here.
- We'll generally discuss this in the context that it was created, which is Natural Language Processing.
- We could discuss this more generally, since its useful in many other areas...but since a lot of this is motivated by NLP applications, it's easier to just describe it as it was created.

- The starting point of NLP: we are going to associate each word with a vector. How should we do this?
- Of course, we can just take $\mathbf{e}_i$ for a word which is the i'th in the dictionary.

  [recall this is the vector with a one in the i'th place and zeros elsewhere]

  But we want the vectors to somehow reflect the meanings of the words and be useful for downstream tasks (translation, writing novels :) , etc).
- For example, it is common to embed words into vectors in $\mathbb{R}^{300}$.
- Our main resource: there are a lot of sentences out there on the internet. This is great training data.
- Let's start with the earliest methods.

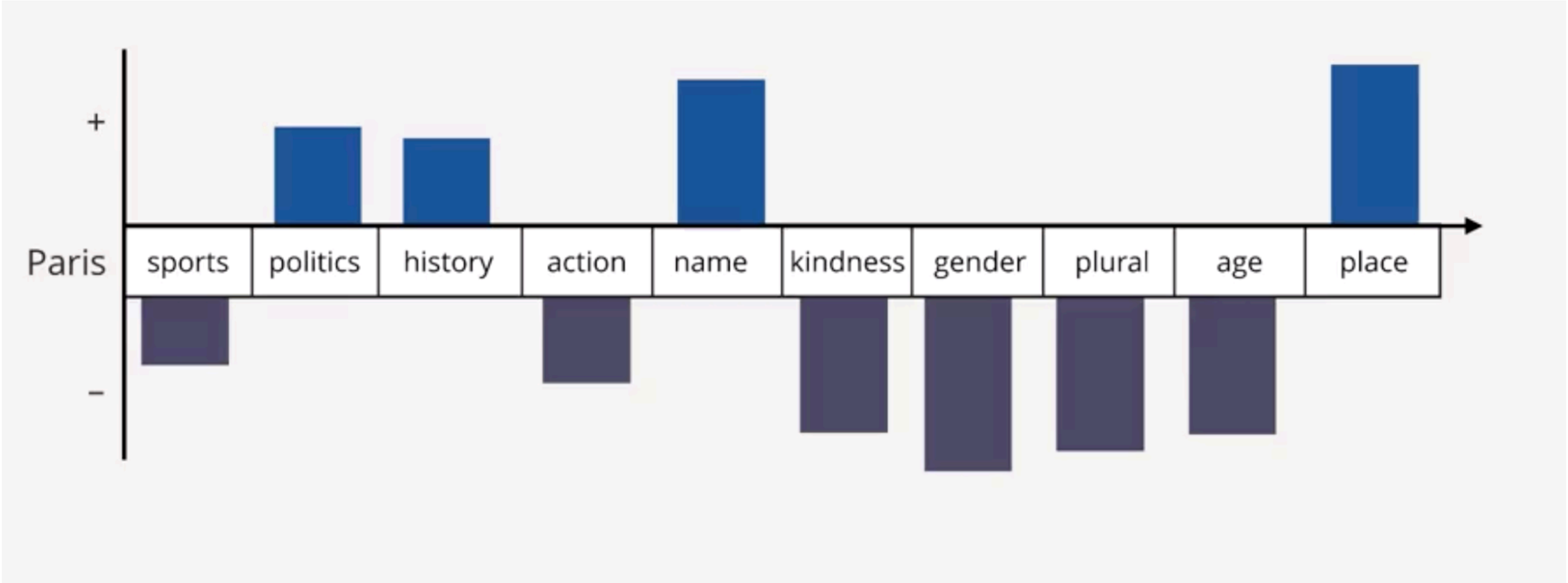
- We take a collection of sentences from the internet. We then remove one word from the middle:
  “If you have ____ that require further review, a Healthway medical professional will contact you. “
- I did this in an arbitrary way, but standard thing to do is to consider 4 words before the removed word and four words after the removed word.
- Then you build a neural network to predict the word you just removed.
- The input to the neural network is 8 words. The words are given as $\mathbf{e}_i$, according to their location in the dictionary. There are about 200,000 thousand words in the dictionary, so the input to the neural network has size 1.6 million.
- ...really not that large compared to what many people today are doing. Recall, we trained hundreds of thousand parameters on google cola with relative ase.
- Useful cheat: restrict yourself to a dictionary of 10,000 common words. These represent ~97% of the words being used in speech. Add a symbol for “unknown word.” This can decrease the size of the network.

- So to recap:

  — input to the neural network is a sequence of 8 vectors $\mathbf{e}_i$, concatenated

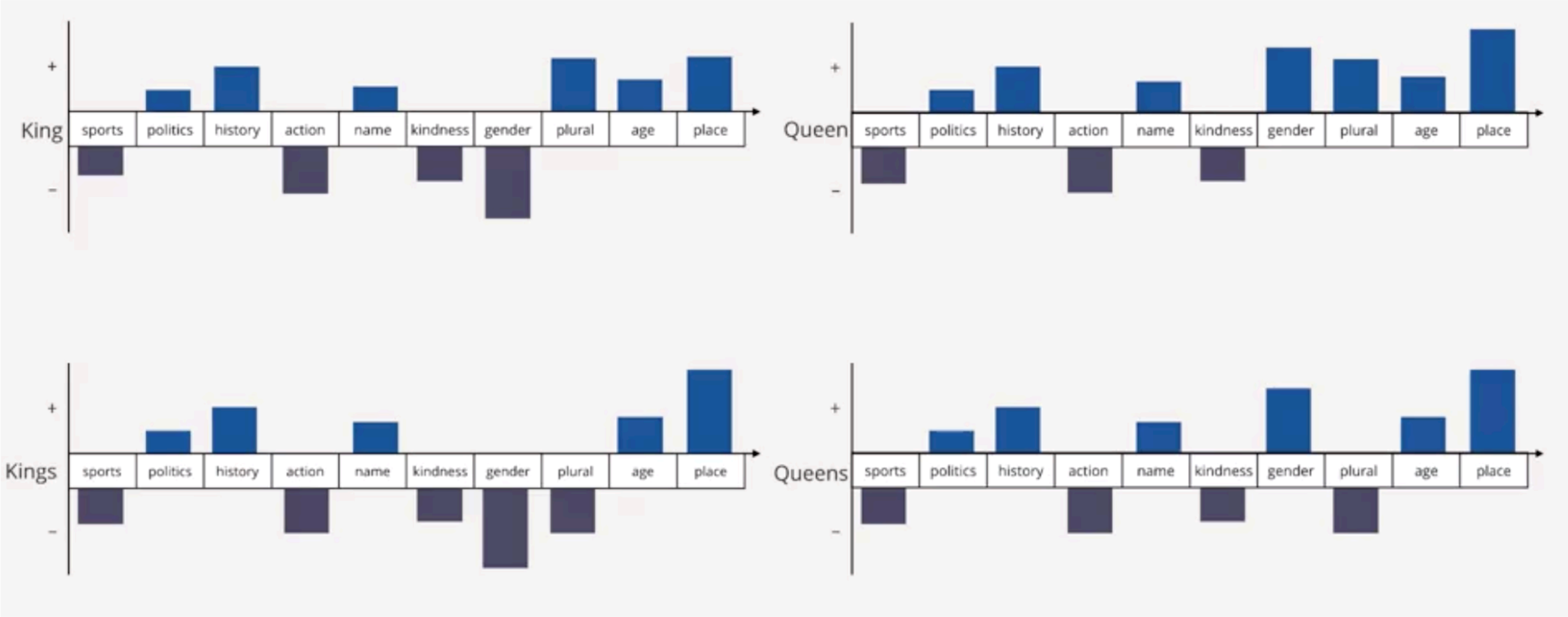
  — if we use the "cheat," then each $\mathbf{e}_i$ belongs to $\mathbb{R}^{10,000}$.
- Second layer of the neural network: reduce each word to size 300 (no activation functions, no bias).
  [300 is a popular choice. Another popular choice is 1,000].
- Use the same neural network to reduce each word.
- Preview: we will use the weights put on the first neuron as the embedding of word $\mathbf{e}_1$.

  We can use the weights put on the second neural as the embedding of $\mathbf{e}_2$.
- Now the last layer of the neural network has size 10,000. The activation function is softmax. As before, we predict the word and take a cross entropy loss.
- Use a third layer which **averages** all the size-300 outputs of the first layer. This is called CBOW (the continuous bag of words model).
- In practice: you can download word embeddings:
  https://nlp.stanford.edu/projects/glove/



A few entries from an embedding of “Paris”

King vs Queen



Plural vs Singular

- There are some interesting results that show that we can use word embeddings to solve analogies.
- Man:Woman as King is to ?
- We can take the vector

  $\tilde{w} = \left(w_{\text{woman}} - w_{\text{man}}\right) + w_{\text{king}} +$

  and then look for the word that has embedding closest to $\tilde{w}$.
- This works, in that it finds the correct answer, which is Queen.
- This can always solve simple analogies e.g.,

  boy:boys as girl:girls

  short:tall as slow:fast
- Interestingly, it will often correctly predict more complex relationship, e.g.,

  US:Hamburger as Canada:?

  will return Poutine as the top choice .
- On SAT analogies

  MEDICINE : ILLNESS as

  law : anarchy

  hunger : thirst

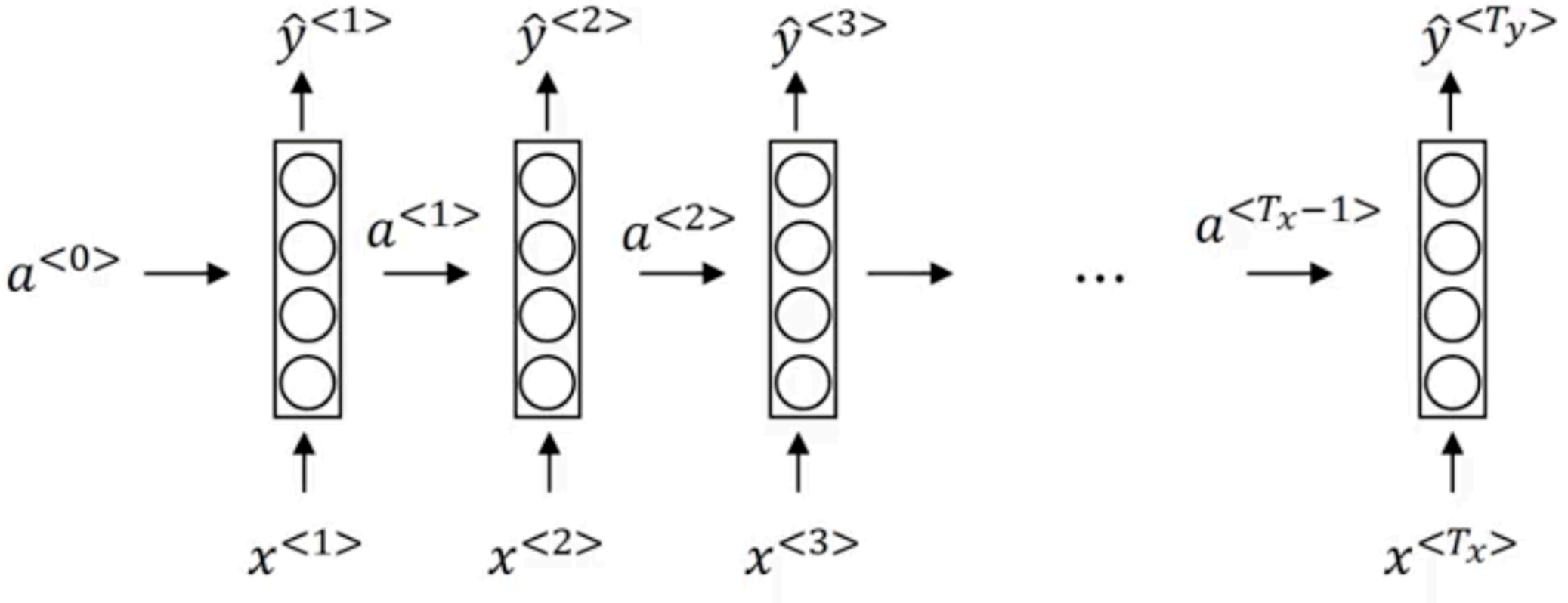
  etiquette : discipline

  love : treason

  stimulant : sensitivity

  this gives an accuracy rate of about 50%.

- There are now simpler and faster methods to produce word embeddings.
- For example, you can try to predict the presence of a word based on a **single** randomly chosen nearby word. Google “Negative Sampling” if you want details.
- Punchline is: words are vectors.
- A sequence of words is a sequence of vectors.
- We view problems like machine translation as mapping sequences of vectors to sequences of vectors.
  This brings us to the problem that we started with.
- We need a NN architecture for processing sequential data.



Each “traffic light” is a neural network
I personally like to use subscripts rather than superscripts
Note: y-hat^<I> is predicted from a-hat^<I>

- In equations:

  $a^{<n+1>} = f_{NN}(a^{<n>}, x^{<n>})$

  $\hat{y}^{<n>} = g_{NN}(a^{<n>})$
- Roughly corresponds to how we humans read a sentence.
- The neural networks f_{NN}, g_{NN} are the same for all n. Will usually refer to this as a generic RNN.
- Typically: g_{NN} is a NN whose last layer is a softmax with as many outputs as the size of the vocabulary (e.g., 10,000).
- We can take this architecture and train it.

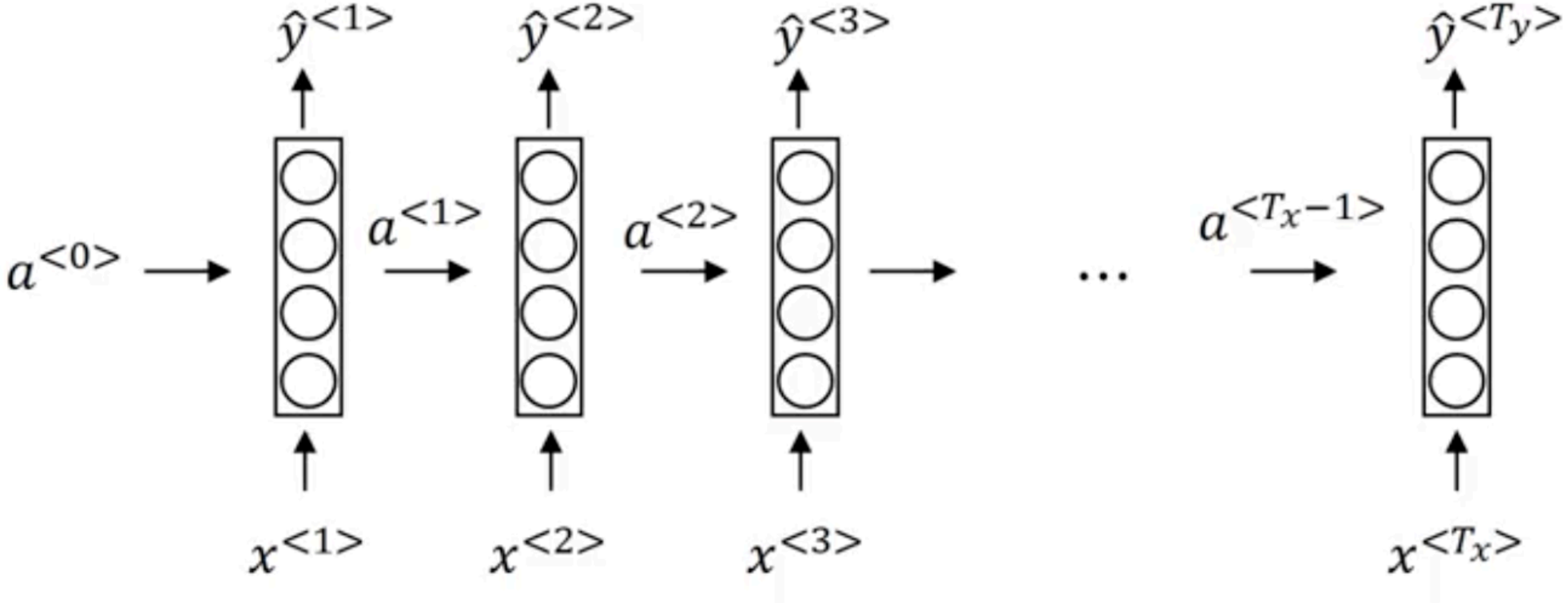
  If we have word embeddings for Russian and English, we take the input pair

  $(x^{<1>}, x^{<2>}, x^{<3>}, \ldots, x^{<8>})$ = "the brown cat jumped over the black fence"

  $(y^{<1>}, y^{<2>}, \ldots, y^{<6>})$ = "коричневый кот перепрыгнул через черный забор."

  and see if we can train it.

  It helps to add y_7 which is a special symbol denoting end of sentence, so that the model automatically learns when to stop.
- We run the network "forwards" and obtain $(\hat{y}^{<1>}, \ldots, \hat{y}^{<6>})$
- We have some kind of loss functions $L(y^{<i>}, \hat{y}^{<i>})$.



First step: backpropagation on the loss between y^<1> and y-hat^<1>
Second step: backpropagation on the loss between y^<2> and y-hat^<2>
Each weight appears in TWO places.
So add both derivatives.
Third step: backpropagation on the loss between y^<3> and y-hat^<3>.
Each weight appears in three places.
So add all three derivatives.
Etc

- Remark: RNNs suffer from vanishing/exploding gradients much more so than feedforward networks.
  Why?
- Solution: RNNs are typically shallow.
- One layer is common!
- We'll see in this set of slides some state of the art results that are quite amazing with one or two layers.
- Sometimes they are deeper, but they're never very deep.

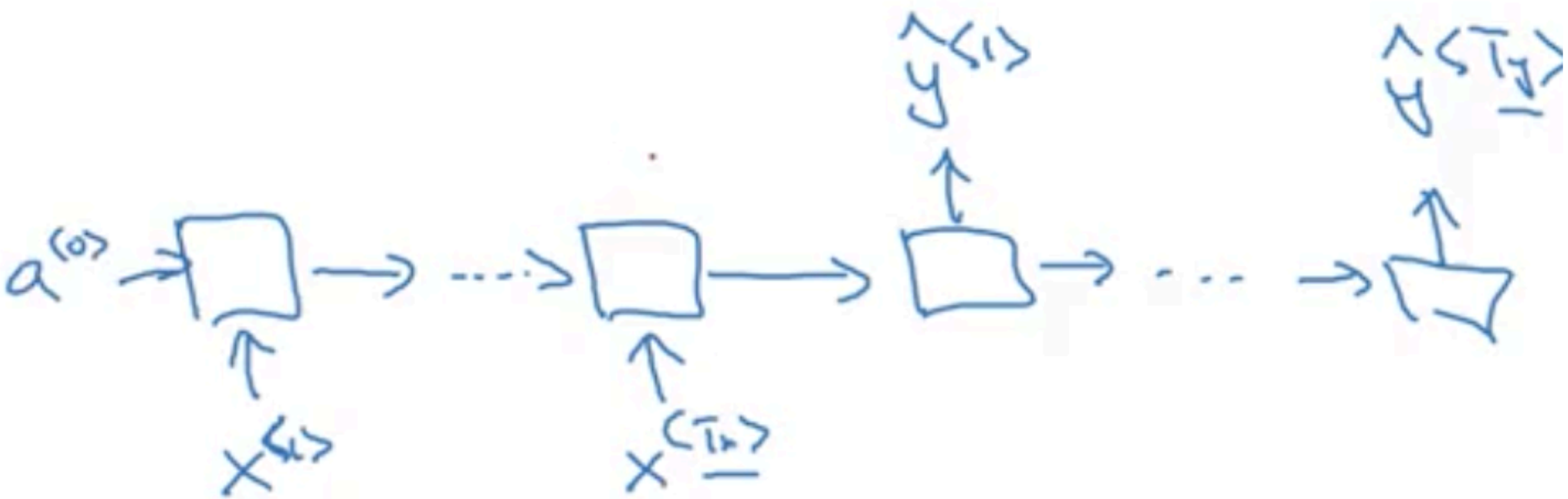
- This architecture with these two training examples does not quite make sense. Why?
- $a^{<n+1>} = f_{NN}(a^{<n>}, x^{<n>})$

  $\hat{y}^{<n>} = g_{NN}(a^{<n>})$

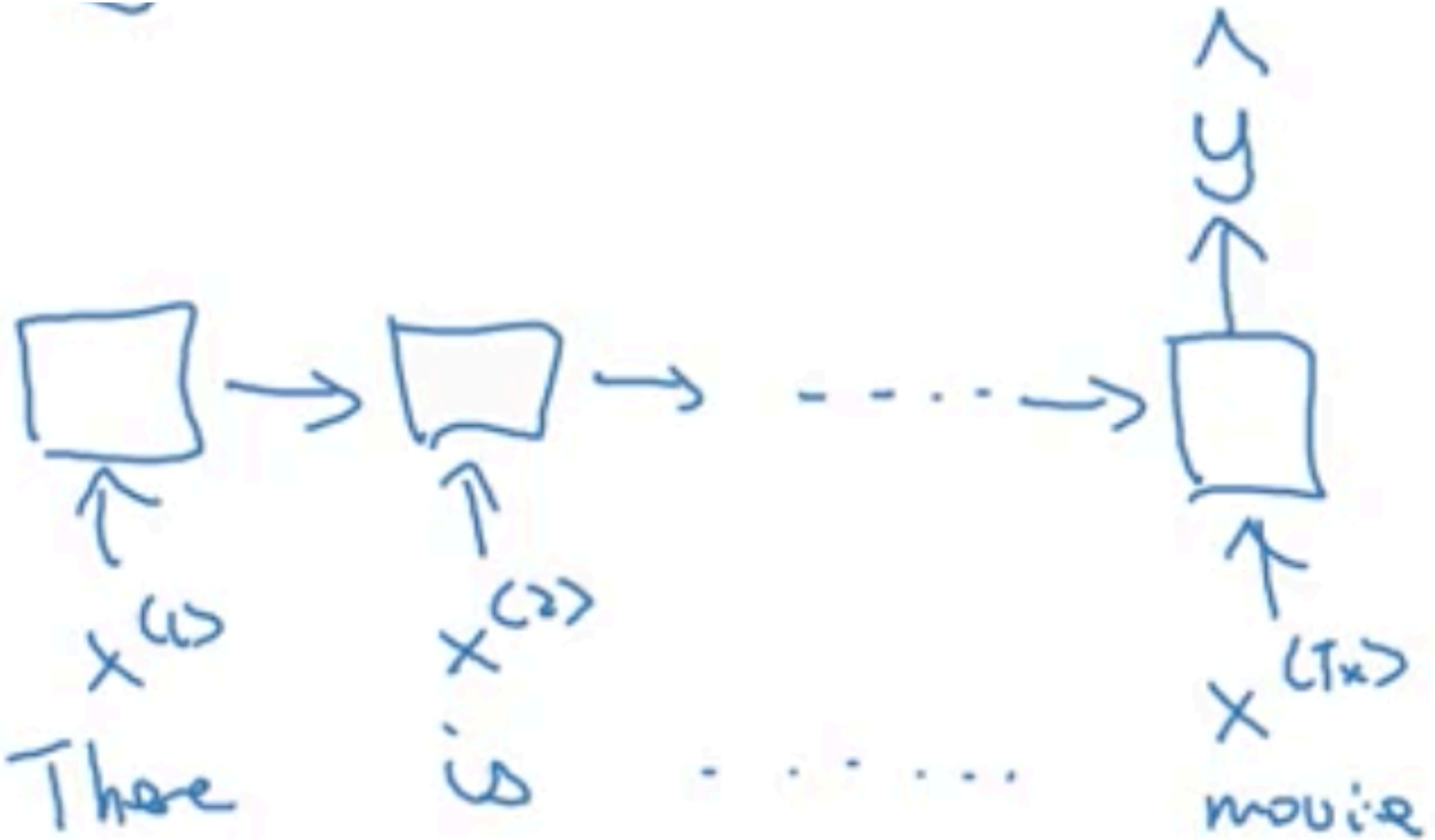
  $(x^{<1>}, x^{<2>}, x^{<3>}, \ldots, x^{<8>})$ = “the brown cat jumped over the black fence”

  $(y^{<1>}, y^{<2>}, \ldots, y^{<6>})$ = “коричневый кот перепрыгнул через черный забор.”
- The last word in the Russian translation, “забор” is a translation of the English word “fence.”

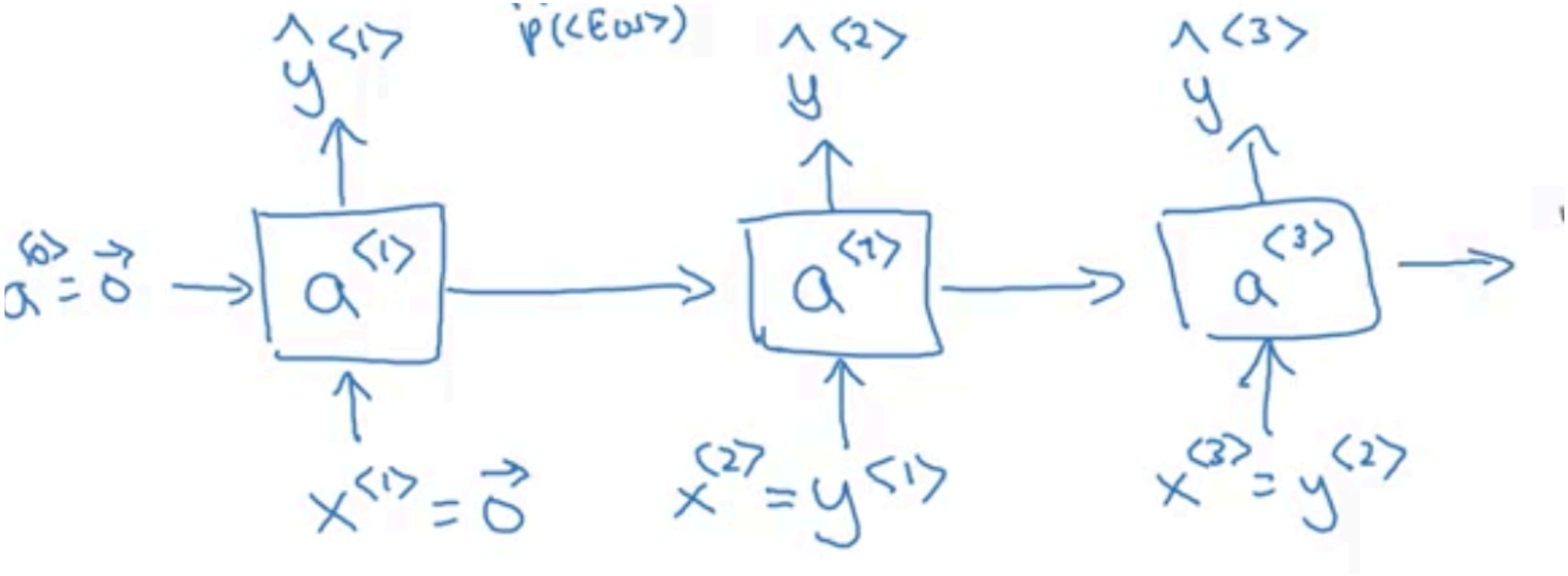
  But the system has to output it after only seeing six words!
- Input and output length are often not the same.



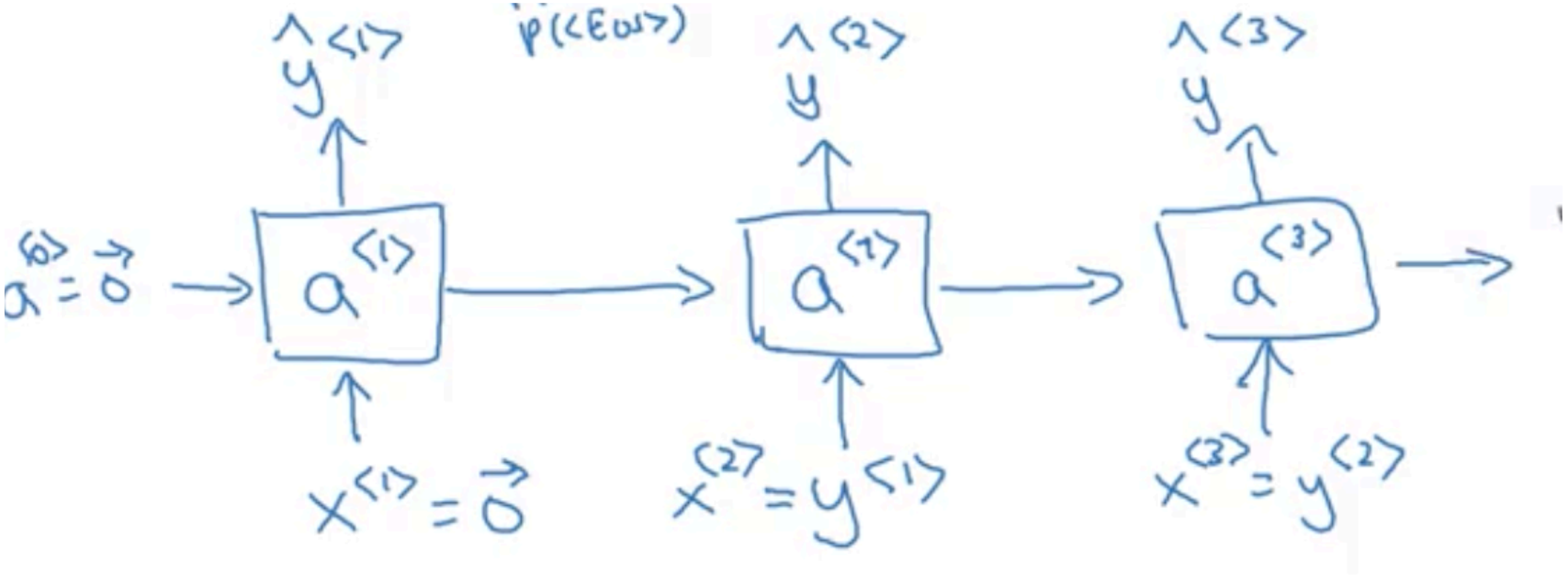
A potential fix.
The left side of the picture is called the ENCODER and the right side is called the DECODER.
Could create special symbol for “empty input”
Or could train two different RNNs (one for encoder and one for decoder).



When the problem only has one output (e.g., sentiment analysis)



Another thing you could do is build a language model.
Your training set could be the entire internet: that's a lot of sentences.
You always have it predict the next word.
For example, the sentence "the brown cat jumped..."
Results in: x^<2> = y^<1> = the,
x^<3> = y^<2> = brown
x^<4> = y^<3> = cat
etc
If you feed it the entire internet, it gets better at predicting the next word



Once its trained, you could use it to generate text!
Feed it x^<0> = 0.
Then sample a random entry from the last layer of the soft-max that predicts y-hat^<1>
Next, feed y-hat^<1> to the next layer, and sample a random entry from the softmax that predicts y-hat^<2>
You could try to use this to write a short story
The results are not terribly good....for now.

- One small tweak: instead of

  $a^{<n+1>} = f_{NN}(a^{<n>}, x^{<n>})$

  $\hat{y}^{<n>} = g_{NN}(a^{<n>})$

  sometimes people do

  $a^{<n+1>} = f_{NN}(a^{<n>}, x^{<n>}, \hat{y}^{n})$

  $\hat{y}^{<n>} = g_{NN}(a^{<n>}, \hat{y}^{n})$
- There are also many variations on these architectures, as we describe.
- One issue is that the network needs to learn what to remember and what to forget.
- Context is very natural to us, but less so to a network where everything is a string of symbols.
- Let's try to help it by giving it an architecture more suited for remembering "context."

- Solution:

  $$\tilde{c}^{<t>} = f_{NN}(c^{<t-1>}, x^{<t>})$$

  $$\Gamma_u = h_{NN}(c^{<t-1>}, x^{<t>})$$

  $$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + (1 - \Gamma_u) \odot c^{(t-1)}$$

  $$\hat{y}^{<t>} = g_{NN}(c^{<t>})$$

  The final activation function of h_{NN} should make sure that Γ_u is between zero and one. A standard choice is sigmoid or tanh.

- As before, sometimes we use the previous prediction $\hat{y}^{<t-1>}$ as an input argument.
- Interpretation:

  You update your context to get the new vector $\tilde{c}^{<t>}$.

  But you don't just update to it.

  Rather, Γ_u tells you what you should remember and what you should forget.

  This is called a **Gated Recurrent Unit (GRU)**.
- Endless NN architectures have been tried, and the community has converged on a few that are believed to be be effective. The GRU is one of them.

- In what is sometimes called the “Full GRU” people make one additional tweak.
- On the previous page we had:

$$\tilde{c}^{<t>} = f_{NN}(c^{<t-1>}, x^{<t>})$$
$$\Gamma_u = h_{NN}(c^{<t-1>}, x^{<t>})$$
$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + (1 - \Gamma_u) \odot c^{(t-1)}$$
$$\hat{y}^{<t>} = g_{NN}(c^{<t>})$$

- We add one more term, Γ_r. Intuitively, Γ_r stands for relevance:

$$\tilde{c}^{<t>} = f_{NN}(\Gamma_r \odot c^{<t-1>}, x^{<t>})$$
$$\Gamma_r = q_{NN}(c^{<t-1>}, x^{<t>})$$
$$\Gamma_u = h_{NN}(c^{<t-1>}, x^{<t>})$$
$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + (1 - \Gamma_u) \odot c^{(t-1)}$$
$$\hat{y}^{<t>} = g_{NN}(c^{<t>})$$

- Of course, the network could learn this quantity on its own, but, again, it helps learning to put it there explicitly.

Abstract

Objective

We sought to predict if patients with type 2 diabetes mellitus (DM2) would develop 10 selected complications. Accurate prediction of complications could help with more targeted measures that would prevent or slow down their development.

Materials and Methods

Experiments were conducted on the Healthcare Cost and Utilization Project State Inpatient Databases of California for the period of 2003 to 2011. Recurrent neural network (RNN) long short-term memory (LSTM) and RNN gated recurrent unit (GRU) deep learning methods were designed and compared with random forest and multilayer perceptron traditional models. Prediction accuracy of selected complications were compared on 3 settings corresponding to minimum number of hospitalizations between diabetes diagnosis and the diagnosis of complications.

From https://academic.oup.com/jamia/article/27/9/1343/5899846?casa_token=iSgmBu7lPjwAAAAA:cjahYqzjP5Dx_gtuggiyTXf2--1SlZ6i6hVGgibCkC1DNqyf5wYfro8OsACTjhC1siZbsEhY8Fm2

The diagnosis domain was used for experiments. The best results were achieved with RNN GRU model, followed by RNN LSTM model. The prediction accuracy achieved with RNN GRU model was between 73% (myocardial infarction) and 83% (chronic ischemic heart disease), while accuracy of traditional models was between 66% – 76%.

Discussion

The number of hospitalizations was an important factor for the prediction accuracy. Experiments with 4 hospitalizations achieved significantly better accuracy than with 2 hospitalizations. To achieve improved accuracy deep learning models required training on at least 1000 patients and accuracy significantly dropped if training datasets contained 500 patients. The prediction accuracy of complications decreases over time period. Considering individual complications, the best accuracy was achieved on depressive

Adaptive stock trading strategies with deep reinforcement learning methods

Xing Wu [a, b], Haolei Chen [a], Jianjia Wang [a, b], Luigi Troiano [c], Vincenzo Loia [d], Hamido Fujita [e, f, g] [1]

Show more

Share Cite

https://doi.org/10.1016/j.ins.2020.05.066

Highlights

- Gated Recurrent Unit is proposed to extract informative features from raw financial data.
- Reward function is designed with risk-adjusted ratio for trading strategies for stable returns in the volatile condition.
- Two adaptive stock trading strategies are proposed for quantitative

- One issue is that the network feels come conflict in the design the function f_{NN}:

  — the quantity $c^{<t>}$ needs to be really good for predicting $y^{<t>}$.

  — the quantity $c^{<t>}$ also needs to represent context that is passed into the future.

  — these two goals overlaps somewhat but may be in some conflict
- Also: it might help to make it easier for the context $c^{<t>}$ to grow.

  As in: instead of setting $c^{<t>}$ to be a linear combination of a new proposed context, allow us to add the new proposed context to the existing one.
- This leads to an architecture known as LSTM.
- Historically, LSTM came first. LSTM is also the most popular architecture for recurrent neural networks.

- LSTM:

$$\tilde{c}^{<t>} = f_{NN}(a^{<t-1>}, x^{<t>})$$

$$\Gamma_u = h_{NN}(a^{<t-1>}, x^{<t>})$$

$$\Gamma_f = q_{NN}(a^{<t-1>}, x^{<t>})$$

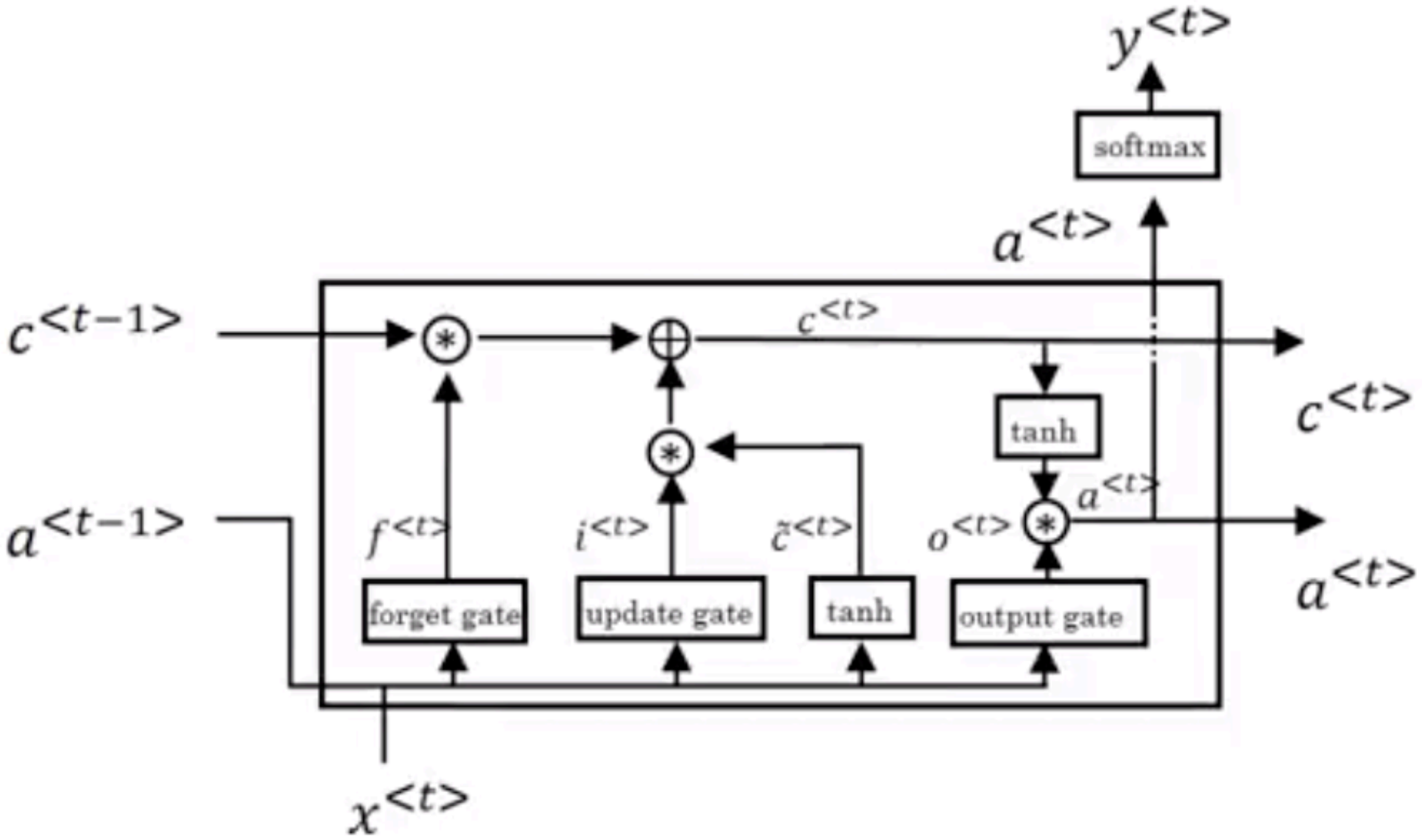
$$\Gamma_o = v_{NN}(a^{<t-1>}, x^{<t>})$$

$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$$

$$a^{<t>} = \Gamma_0 \odot \sigma(c^{<t>})$$

$$\tilde{y}^{<t>} = g_{NN}(a^{<t>})$$

- $\Gamma_u, \Gamma_f, \Gamma_0$ should be in [0,1]. The activation function σ should force $a^{<t>}$ to be in $[0,1]$. The function tanh is standard here.

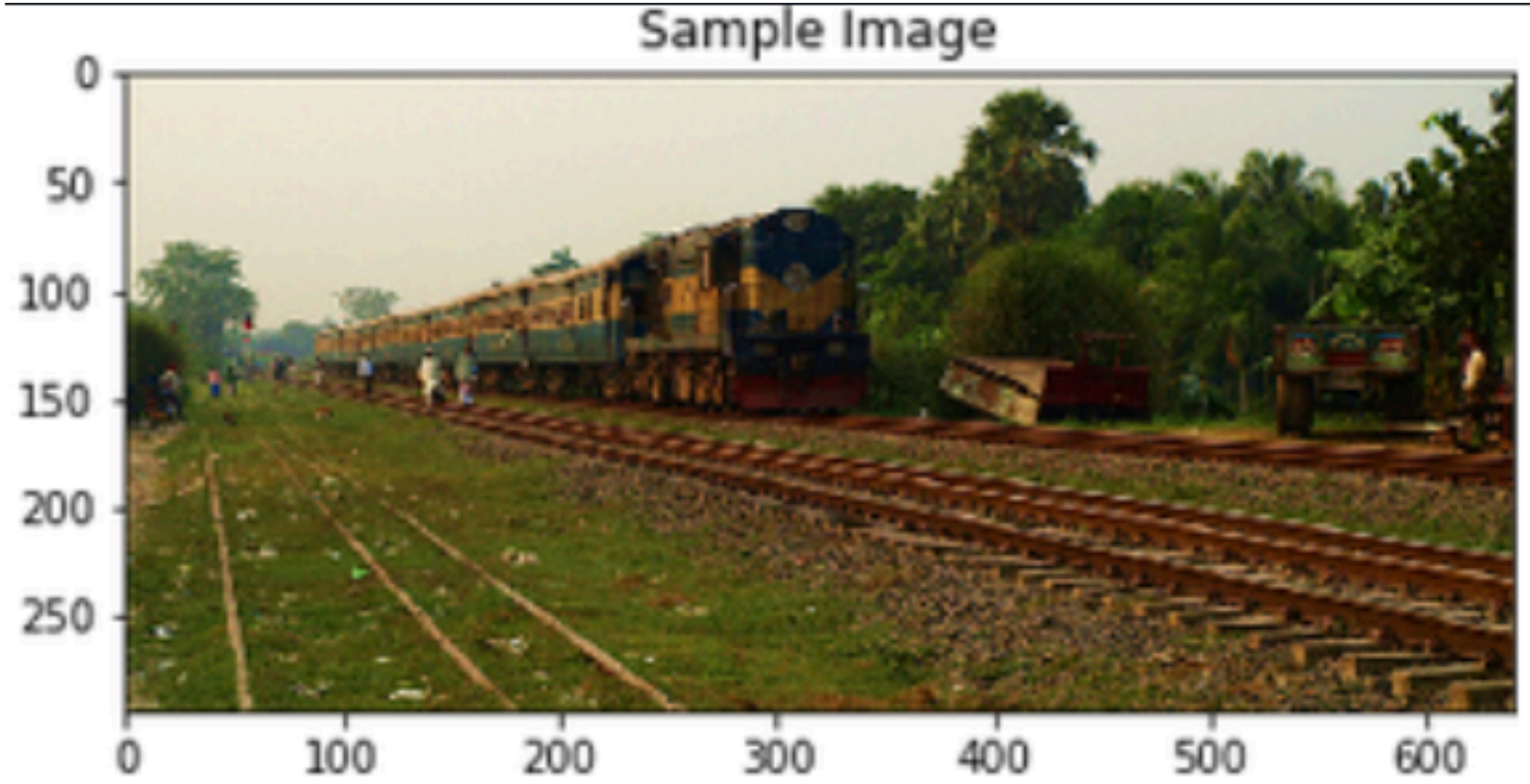


Usual illustration of the equations.
Personally, I find the equations easier to understand than the picture.

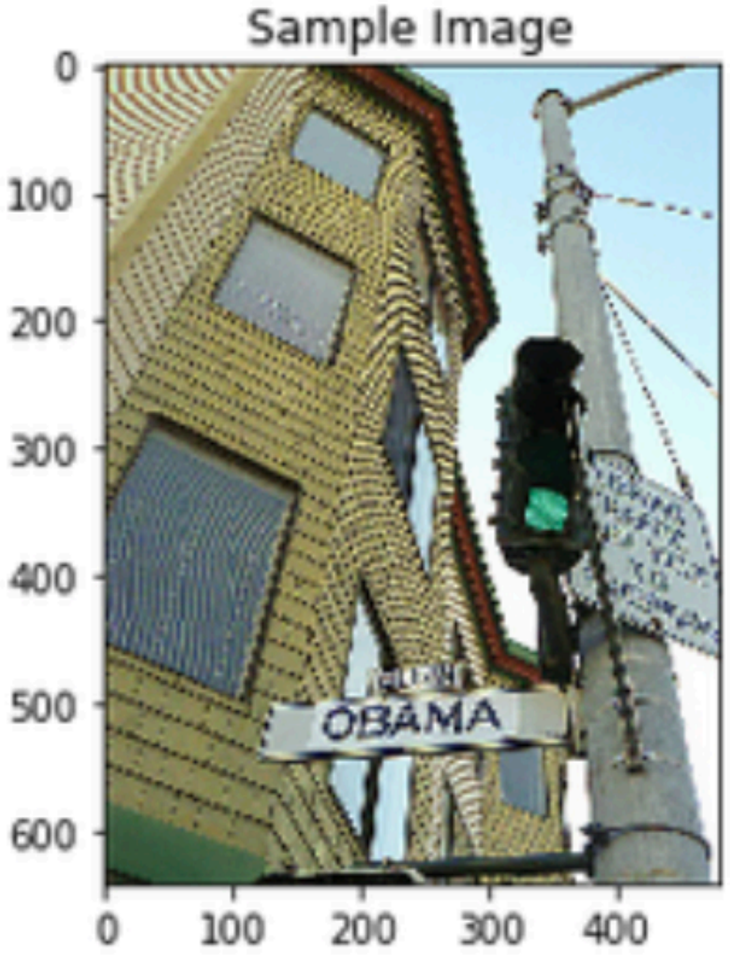
- The combination of the ideas we've discussed can do some remarkable things.
- Here is one: caption an image.
- Simple architecture:

  — take an image and feed into a few convolutional layers

  — feed to result to an LTCM as $c^{<0>}$.
- Train on MS-COCO, a data-set of ~80,000 images, with each image having five captions.
- Very simple idea. How well do you think it does?




```
a train on a train track with trees in the background
```




```
a street sign on a pole on a city street
```

a man is skiing down a hill in the snow .

a man in a darth vader shirt and tie

But there are also some failures

All these are from https://github.com/siddsrivastava/Image-captioning

- Robot locomotion is challenging. In 2015, DARPA ran a competition asking robots to do a number of simple tasks (go from one location to another, which involved opening doors, traversing stairs, etc). It did not go well: https://www.youtube.com/watch?v=g0TaYhjpOfo
- Well worth remembering that the entrants to this competition had large budgets.
- No ML was used for any of these.
- Recent years have shown a number of advances, both in terms of classical control approaches as well as RL. I'll discuss this video: https://www.youtube.com/watch?v=MPhEmC6b6XU which is associated with the paper https://arxiv.org/pdf/2105.08328.pdf

- State: all the positions of the robot (relative to the ground and velocities).
- We put two special sinusoids $\sin 2\pi t, \sin 2\pi(t + 0.5)$ to the policy. These signal can help it produce a "synchronized" policy like in walking. Different offsets intuitively correspond to different legs.
- Output: target angles and velocities for each joint, which are fed into a PD controller.
- The policy is an LSTM network with two hidden recurrent layers of dimension 128 each.
- Hand-crafted cost (with terms on foot velocities, forces experienced by the foot, cost on not matching a certain velocity and orientation).
  Also a "mirror loss term" to try to force the network to learn symmetric gaits.
- Domain randomization is the idea that you should randomize the physics somewhat in simulations: in this case, they changed joint masses, friction strength, some delays from actuation to execution, and some other parameters.
  Domain randomization has been shown to help sim2real transfer.
- Terrain is randomized (whether there is an incline) as well as stairs (length, width, height).
- Used a variation on policy gradient with thresholding: if your new parameter values are too far from your old parameter values, you do not update.
- Result: 80% success rate in ascending stairs in the real world.
- Takeway: it takes a lot of thinking + trying to make this stuff work.

- We discussed earlier how batch normalization can help NN performance.
- Not clear how to apply this to RNNs, where data typically does not come in batches.
- A popular solution: layer normalization.
- Recall the batch normalization equations:

  $$x_{i,\text{new}} = \gamma_i \frac{x_i - \mu_B}{\sigma_B} + \beta_i$$

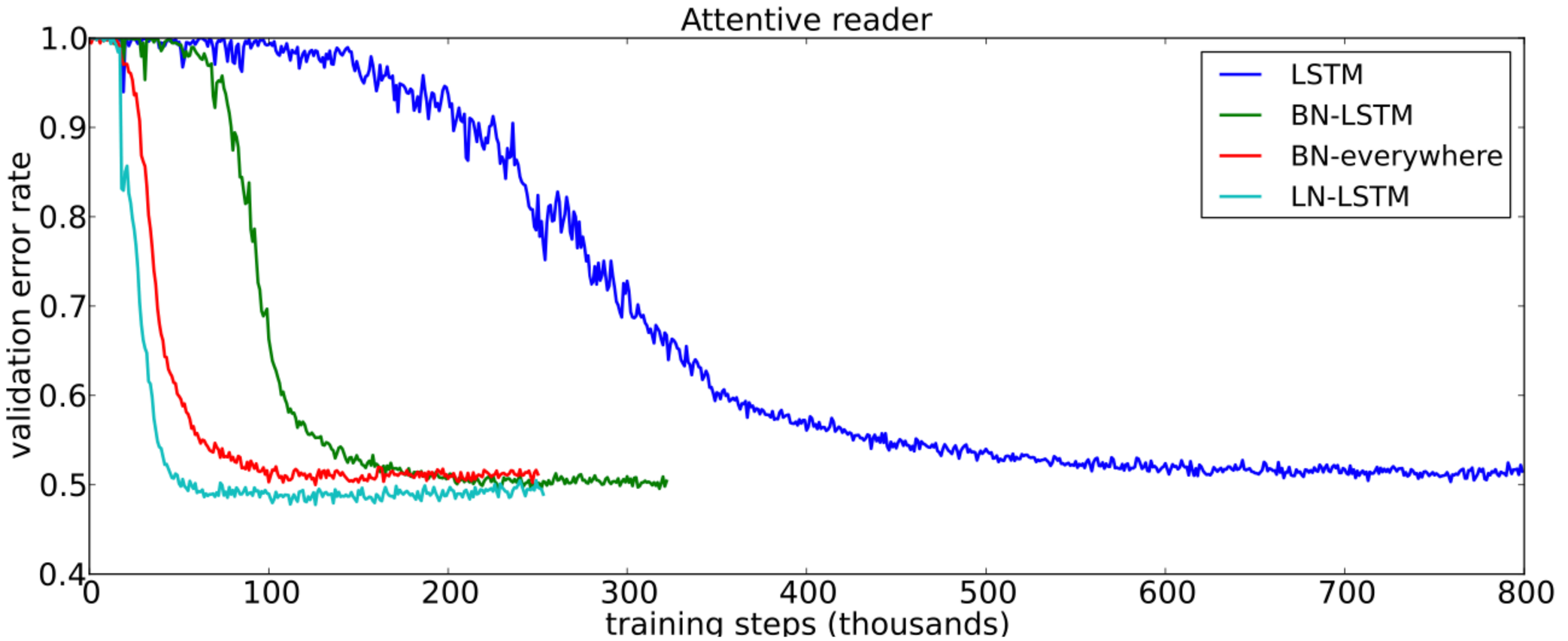
  where μ_B, σ_B are the mean of the activation x_i over the batch.

  We learned the parameters γ_i, β_i and then when we implemented the NN, we replaced μ_B, σ_B by the mean and STD over the entire data set.
- One slight disadvantage: we were doing something different at test time vs training time.
- Layer normalization:

  $$x_{i,\text{new}} = \gamma_i \frac{x_i - \mu_l}{\sigma_l} + \beta_i$$

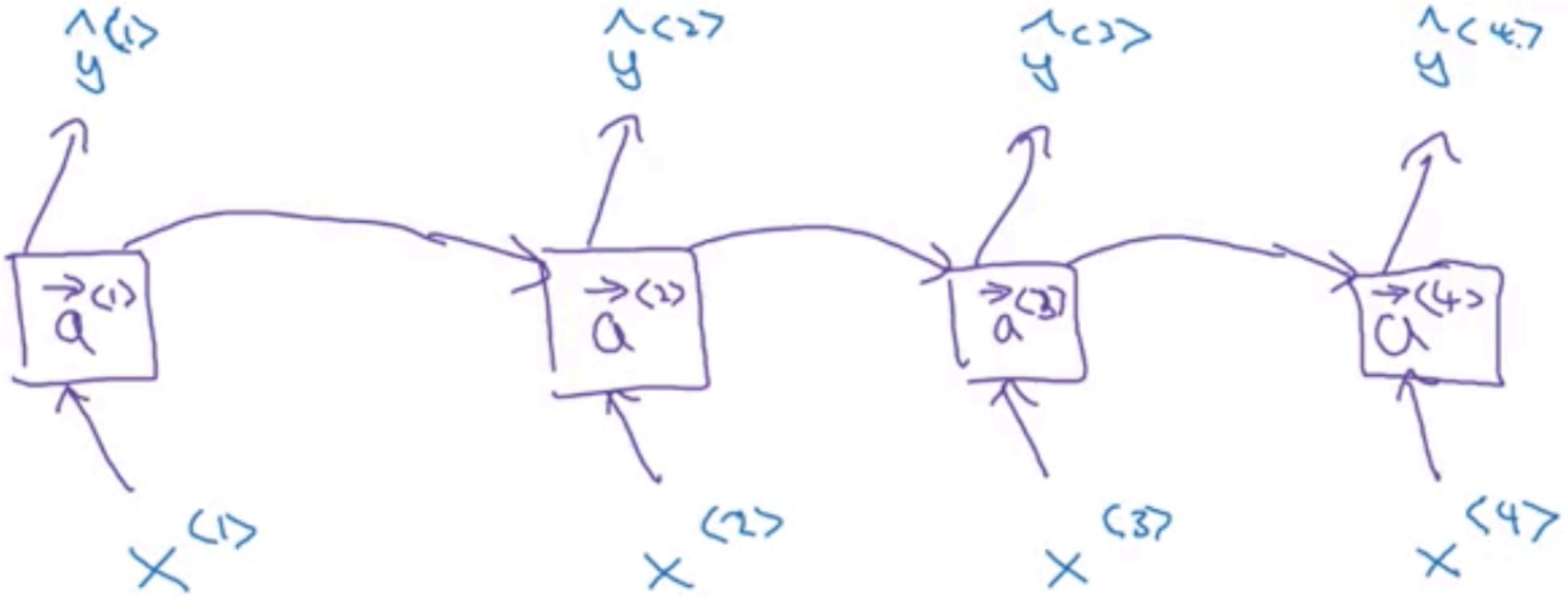
  where μ_l, σ_l are the average and std of the x_j in the same layer as x_i.

- Observe: μ_l, σ_l depend on the training example.
- At test time, can do the same thing as training time.
- Layer norm is popular! Below we give an image from the original LayerNorm paper (https://arxiv.org/pdf/1607.06450.pdf), on the task of answering SAT-style reading comprehension problems where the answer has to be filled in (rather than multiple choice).

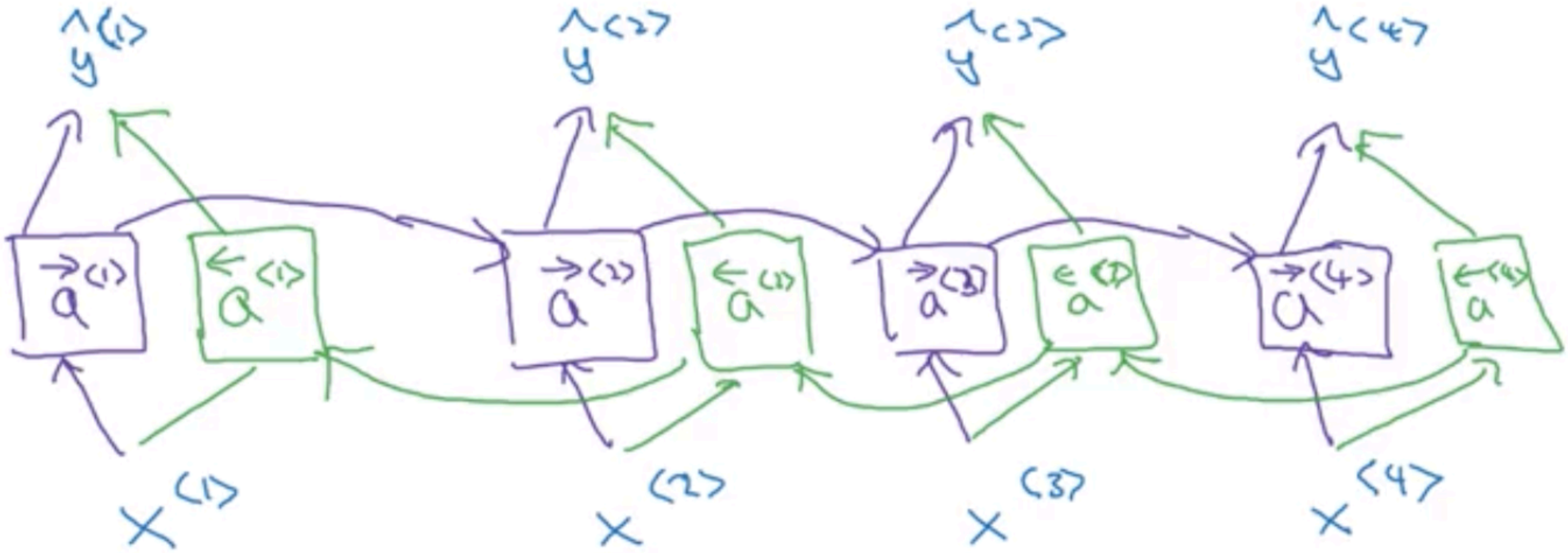


Validation performance as a function of training on a fill-in-the-blanc task

- OK, but: if you want to translate, sometimes you need information from the “future.”
- Consider:
  “Teddy bears are for sale.”
  “Teddy Roosevelt was a great president!”
  How you translate the word Teddy depends on what follows it.
- The models we have described have to fail at this, since they produce a translation without looking to the future.
- Solution: Two RNNs, one moving from beginning of a sentence to the end and the other from the end to the beginning.
- Predict based on the results generated by both RNNs.
  This is called a bidirectional RNN.



The usual RNN architecture



A Bidirectional RNN

We can, of course, do the same for GRUs, LSTMs, or other architectures

- Now let's come back to language models. We've described how to build one with LSTM.
  It isn't particular good.
- Next, we'll describe a language model that is surprisingly good: GPT-3. You can see https://en.wikipedia.org/wiki/GPT-3. GPT-3 is pretty recent (released summer of 2020).
- GPT stands for "Generative Pre-Trained Transformer." GPT-3 has 175 billion parameters.
- It is trained on 45 TB of text from wikipedia, books, and various internet websites.
- We'll spend some time on the core component of GPT-3, which is the transformer architecture (which has applications to Reinforcement Learning).
  But let's start with examples of what GPT-3 can do.
- GPT-3 takes a prompt (i.e., text) as an input. The more text you give it, the better the output is.

Instructions: There is a country in northern Asia that starts with "M". Go south from there, across a desert, and you will enter a very large country. Take the population of that larger country. Now divide that number by 10. Which country starts with "R" and has approximately that many people? That country used to be part of a much larger socialist state. One of the countries that was a part of that state starts with "G" and is also a female first name. Which American artist, born in 1887, has that first name? She died in which US state? Take the last of the two words of that state, which is the name of a country. What is the name of a peninsula in that country that starts with "Y"? At the tip of that peninsula is a popular vacation spot that starts with "C". If you were to sail from that location, what large island nation would you find? Who led that country until 2008? His first name is based on what Latin word, and what does it mean in English? What letter does that word start with? What profession starts with that letter and depends on the rain and sunshine? What is something from there that starts with "B" and is sold in strips? What is an American actor whos last name is that product?

Results: **Mongolia is a country in northern Asia that starts with "M". Going south from there, across a desert, you enter China, which is a very large country. China's population is approximately 1.3 billion. Dividing that by 10 gives 130 million. The country that starts with "R" and has approximately that many people is Russia. Russia was part of the Soviet Union, which was a socialist state. Georgia is a country that was a part of that state. The American artist with that first name is Georgia O'Keeffe. She died in New Mexico. The last of the two words of that state is "Mexico". The peninsula in Mexico that starts with "Y" is Yucatan. The popular vacation spot that starts with "C" is Cancun. If you were to sail from that location, you would find Cuba. Fidel Castro led Cuba until 2008. His first name is based on the Latin word "fidelis", which means "faithful". The letter that word starts with is "F". The profession that starts with that letter and depends on the rain and sunshine is farming. Something from there that starts with "B" and is sold in strips is bacon. An American actor whose last name is that product is Kevin Bacon.**

This is not something you can produce by merely replicating statistics

Question: A 10 year old boy presents with recurrent episodes of dyspnoea, cough and wheeze triggered by cold air and allergens. After performing tests he is diagnosed with an obstructive disease and given medication. Which receptor is the medication most likely to work on:

A) muscarinic receptor
B) G-protein coupled receptor
C) Era
D) Erb
E) a-1 receptor

Correct answer is **B**

Explanation: The patient is suffering from asthma, a disease of the airways. The drugs used to treat asthma are bronchodilators. The bronchodilators act on the beta-2 receptors. The beta-2 receptors are G-protein coupled receptors

At some point, prediction necessitated understanding?

- Jerry Seinfeld and Eddie Murphy comedy routine: https://arr.am/2020/07/17/jerry-seinfeld-and-eddie-murphy-talk-shit-about-san-francisco-by-gpt-3/
- Marcus Aurelius interview: https://arr.am/2020/08/17/ai-tim-ferriss-interviews-ai-marcus-aurelius-gpt-3/
- https://twitter.com/sharifshameem/status/1282676454690451457?s=20
- https://twitter.com/sh_reya/status/1284746918959239168
- Failure mode:https://twitter.com/eigenrobot/status/1284042570507542528?s=20

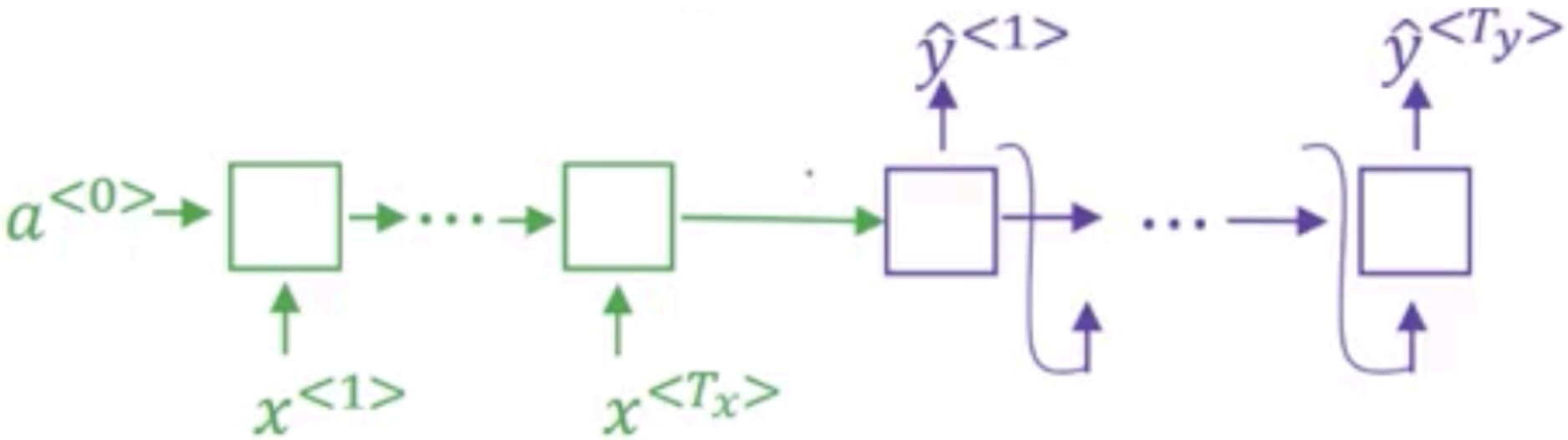
  ...but see response below.
- Clearly, some kind of magic is happening.

  An AI can get stuck just predicting word co-occurence statistics.

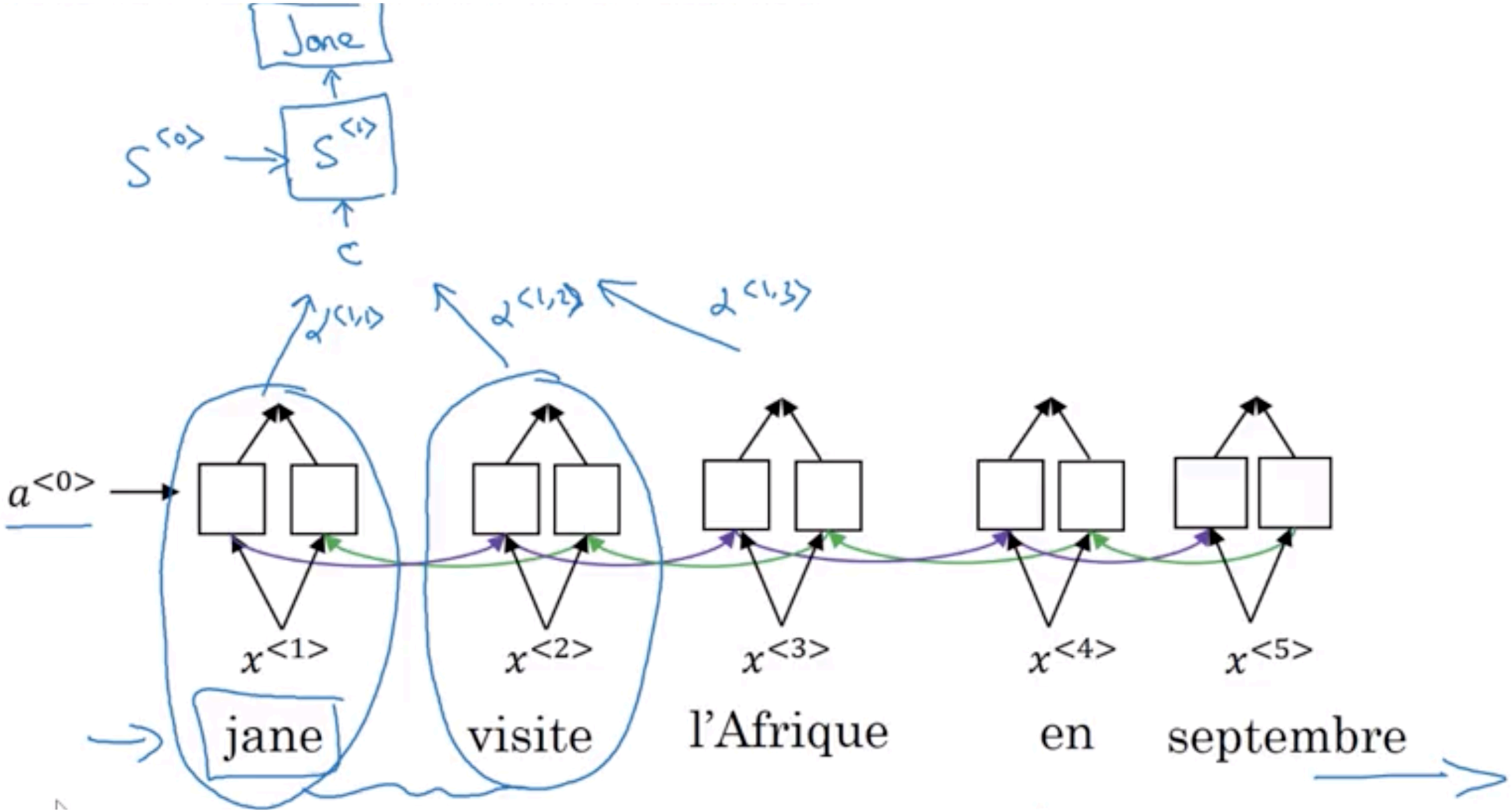
  Here it has clearly moved beyond that.

- Moreover, GPT-3 does reasonably well on tasks it did not train for: https://arxiv.org/abs/2005.14165
- For example, it is good at translation even though it hasn't been explicitly trained for that.
- It's good at multiple-answer questions (you've already seen that) although the text it has trained on contains few such examples.
- This is a big deal in ML.
- Hopefully, these examples have motivated you to understand the details of the Transformer, i.e., the GPT-3 architecture.
- Key hope: transformers can be used for RL.
- We will first describe the key component of transformers, so-called *attention*.

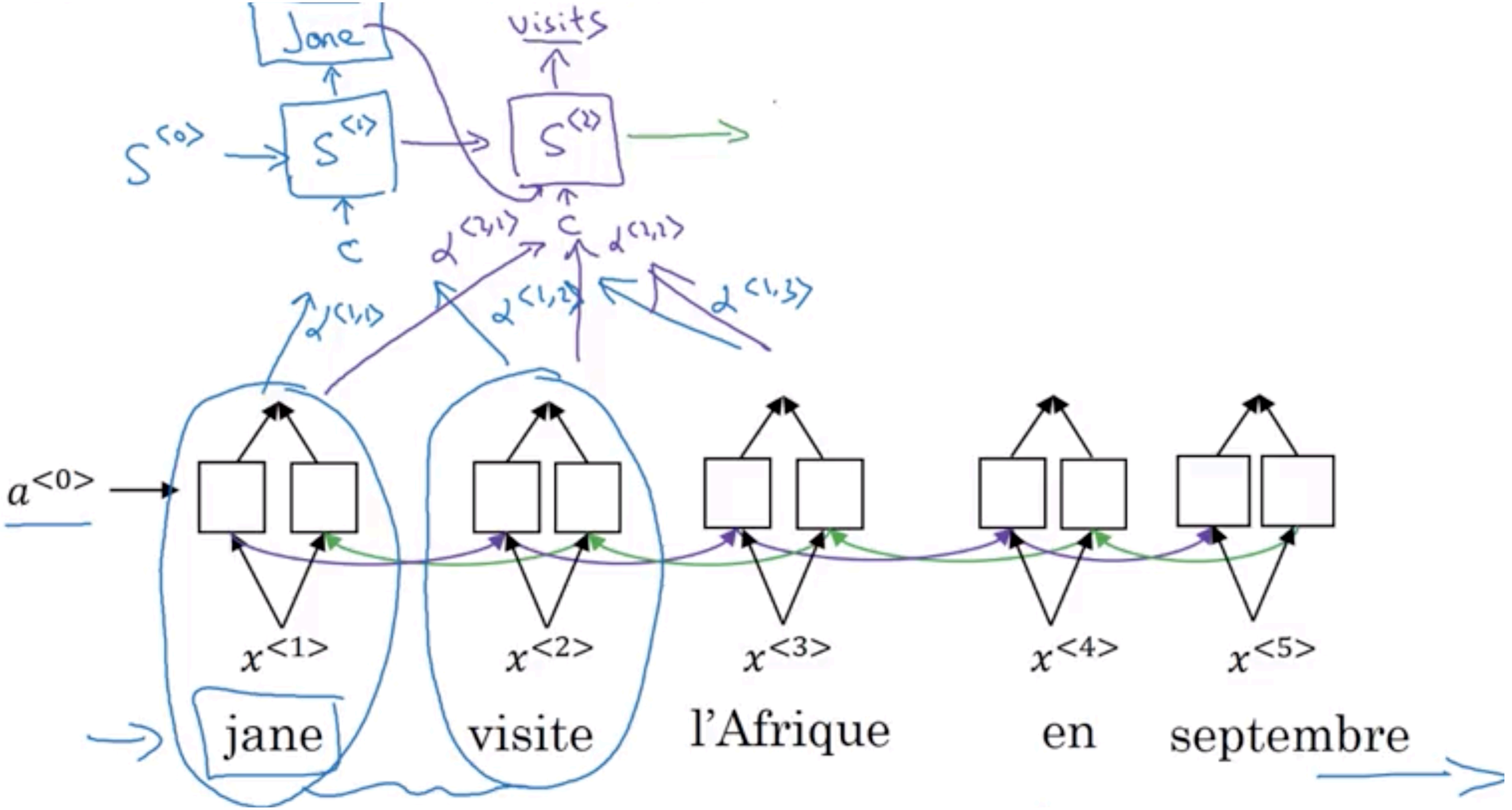
- Let’s motivate attention from the application where it first arose, which is translation.
- Recall our architecture for translation: an RNN/GRU/LSTM passes through the sentence, and then spits out the translation.
- The reason this is the case is that the sentence and its translation do not have the same number of words.
- But this is really different from the way we humans translate sentence, which is more word-for-word.
- We do not memorize the entire sentences, then spit it out.



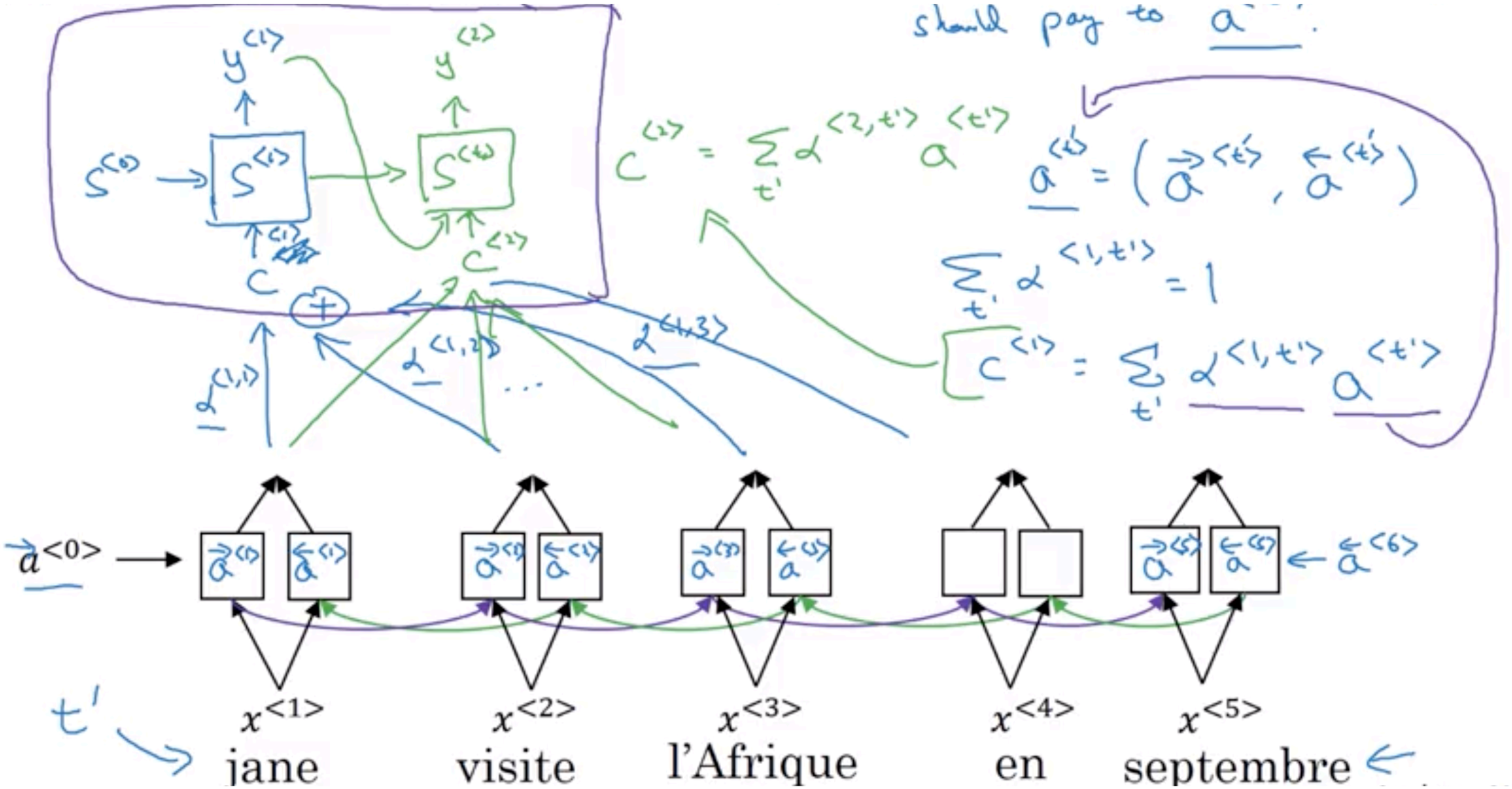
This used to be the standard architecture for language translation



The attention model for translation has 3 RNNs (or GRUs or LSTMs or whatever)
Need, however, coefficients alpha^<i,j> telling us how much attention to pay to the j'th output in the bottom
When producing the it's output at the top



At the next step



More details plus coefficient selection

- Except...how do we select the attention coefficients $\alpha^{<i,j>}$?
- Well, we have one trick up our sleeve. What is it?
- That's right, we use a NN.
- We generate $\tilde{\alpha}^{<i,j>}$ during time using a neural network!

  Input to the neural network is $s^{<i-1>}$ and $a^{<j>}$.
- Then we obtain $\alpha^{<i,j>}$ by doing a softmax:

$$\alpha^{<i,j>} = \frac{e^{\tilde{\alpha}^{<i,j>}}}{\sum_k e^{\tilde{\alpha}^{<i,k>}}}.$$

- Original paper that introduced this: https://arxiv.org/pdf/1409.0473.pdf

- So that paper introduced attention in 2015.
  Then a couple of years later Vaswani et al. in https://arxiv.org/abs/1706.03762 gave a variant of attention that performed really, really well.
- Let's discuss the components of the Vaswani et al. model.
- For every word, we will associate 3 different numbers: query, key, value.
- If $x^{<i>}$ is the word embedding of the i'th word, then

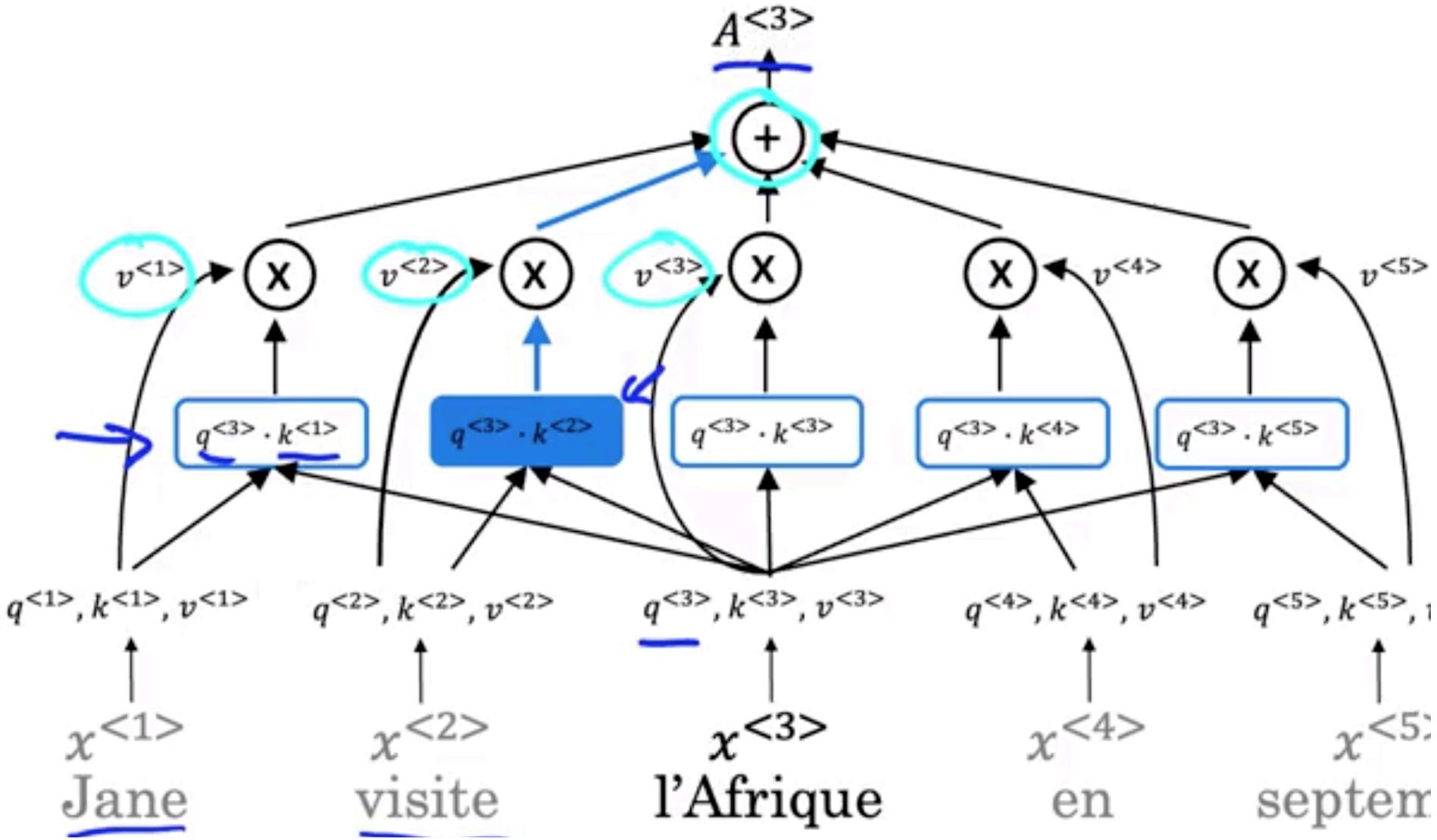
  $q^{<i>} = W^q x^{<i>}$

  $k^{<i>} = W^k x^{<i>}$

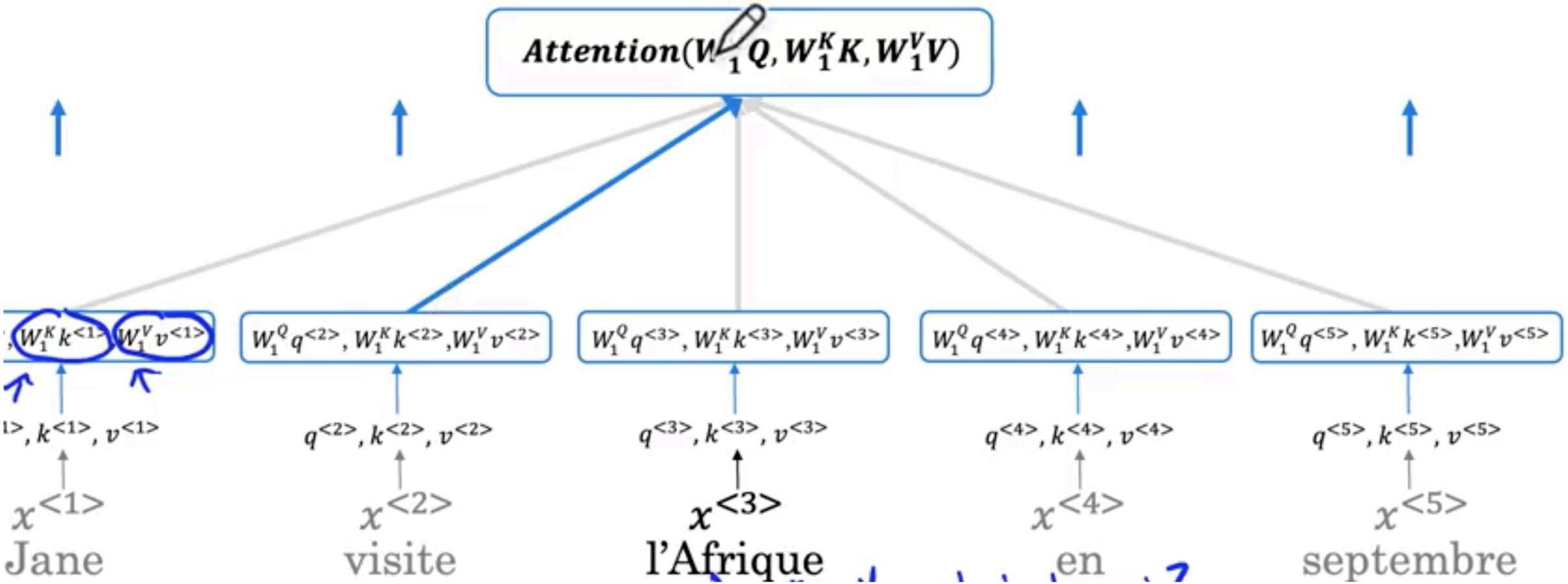
  $v^{<i>} = W^v x^{<i>}$
- The "attention network" maps $x^{<i>}$ to $\sum_j \frac{e^{(q^{<i>})^T k^{<j>}}}{\sum_k e^{(q^{<i>})^T k^{<j>}}} v^{<j>}$
- For example, if there are two words whose word embeddings are $x^{<1>}$ and $x^{<2>}$, then

  the attention network maps $x^{<1>}$ to

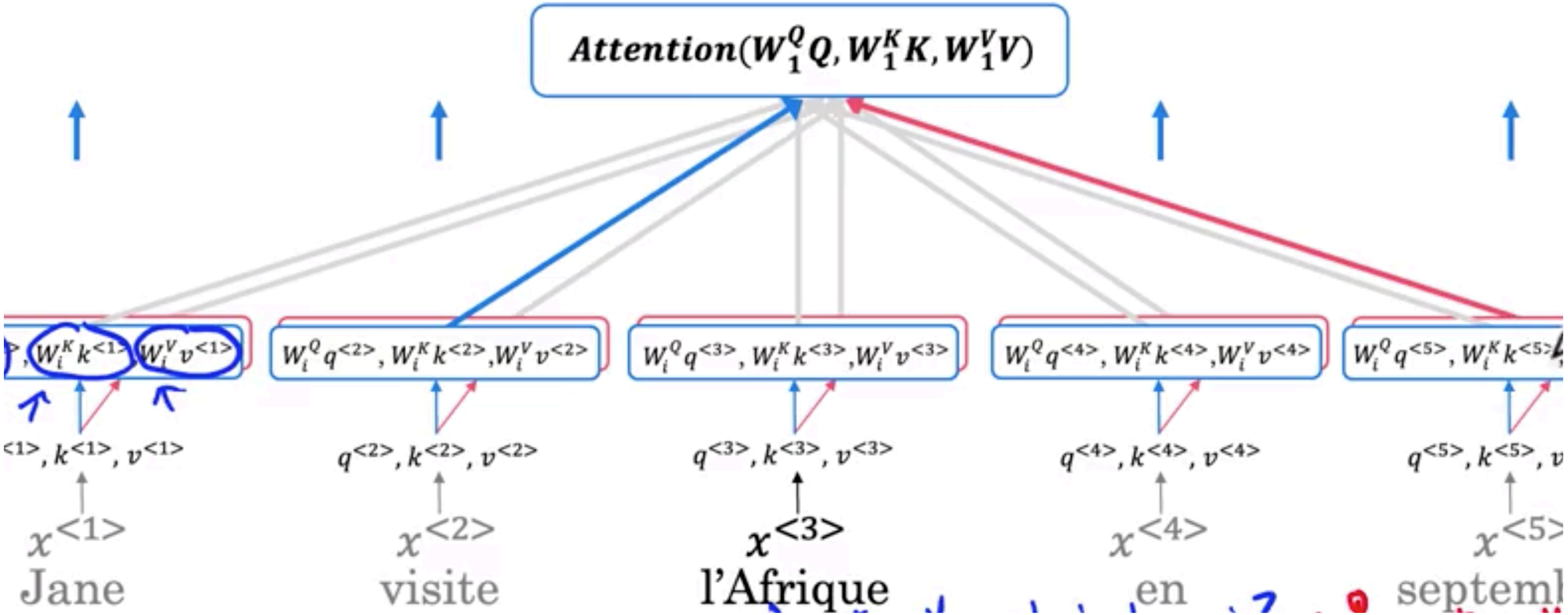
$$\frac{e^{(q^{<1>})^T k^{<1>}}}{e^{(q^{<1>})^T k^{<1>}} + e^{(q^{<1>})^T k^{<2>}}} v^{<1>} + \frac{e^{(q^{<1>})^T k^{<2>}}}{e^{(q^{<1>})^T k^{<1>}} + e^{(q^{<1>})^T k^{<2>}}} v^{<2>}$$



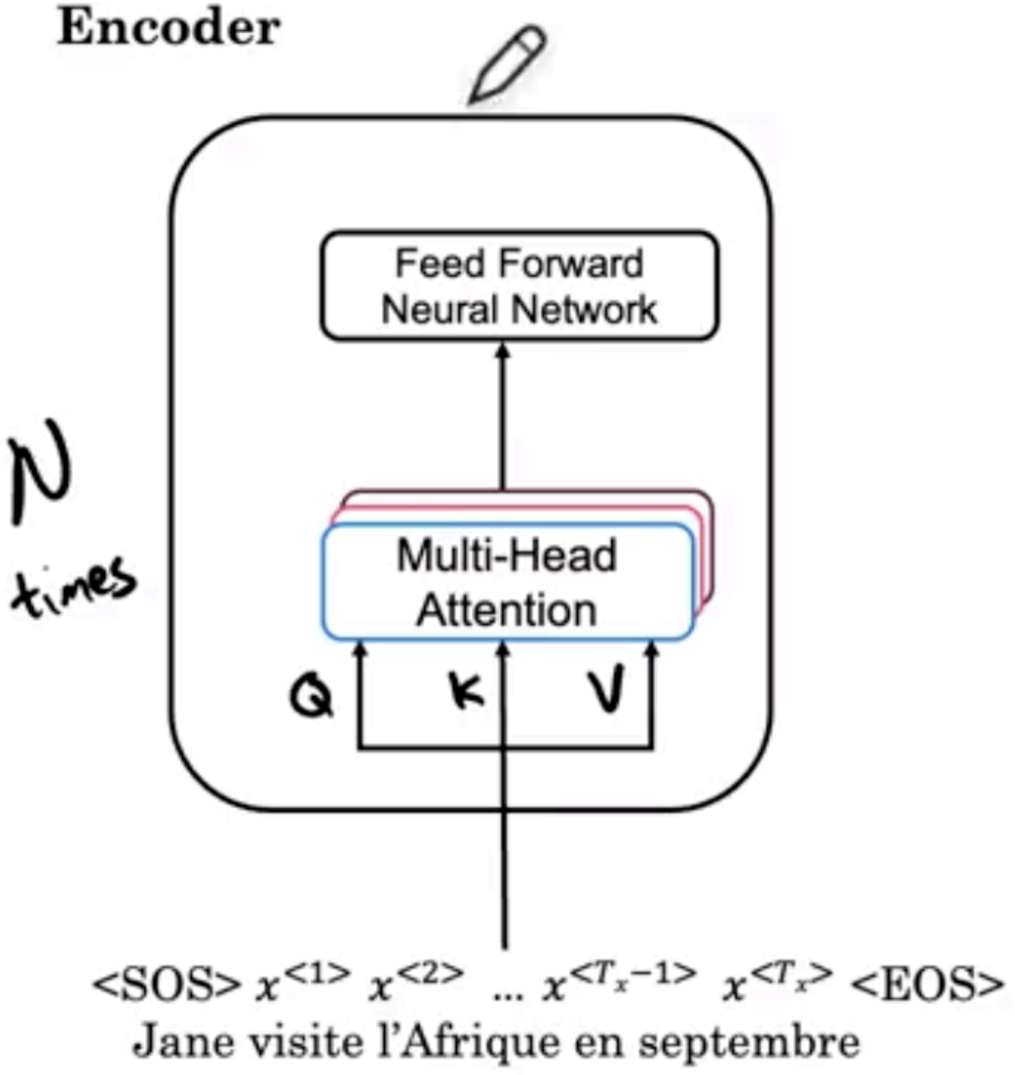
Intuition: the output is the answer to a question



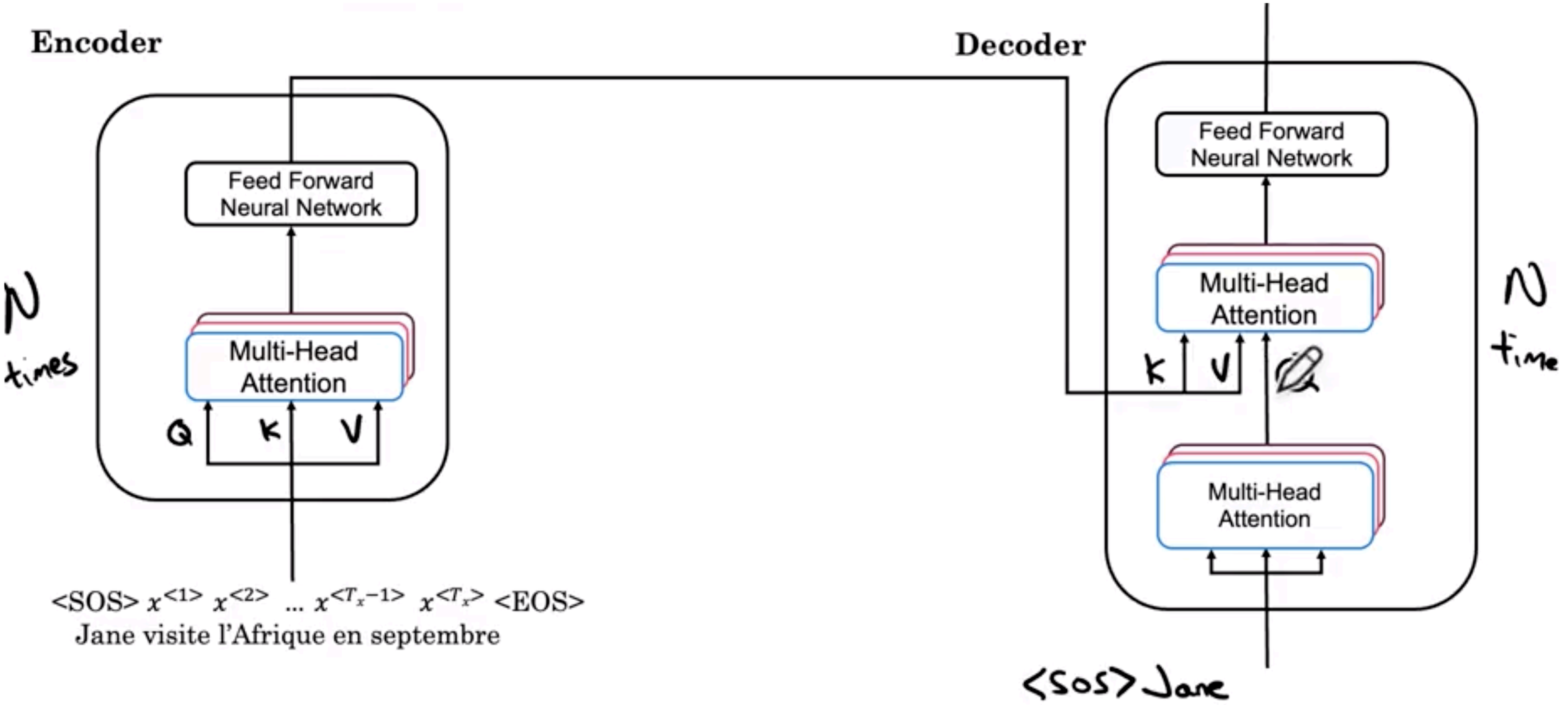
This is called single-head attention



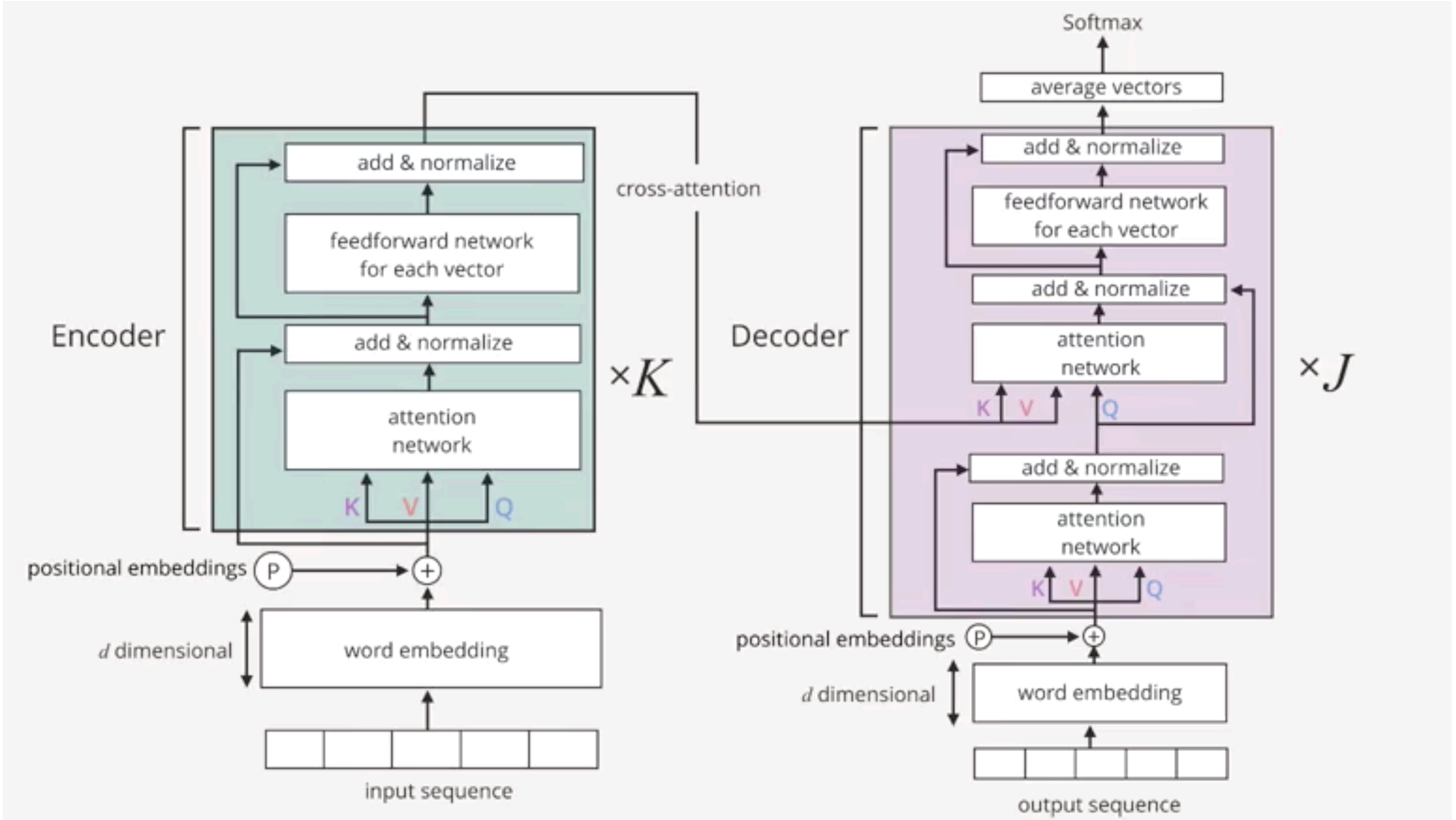
Multi-head attention: just do the same thing but with two different sets of matrices in parallel
Intuitively, corresponds to asking multiple question
In the end, concatenate the results (there are d vectors if there are d heads)



Transformer network: the encoder
The FFNN is applied to each vector at the output



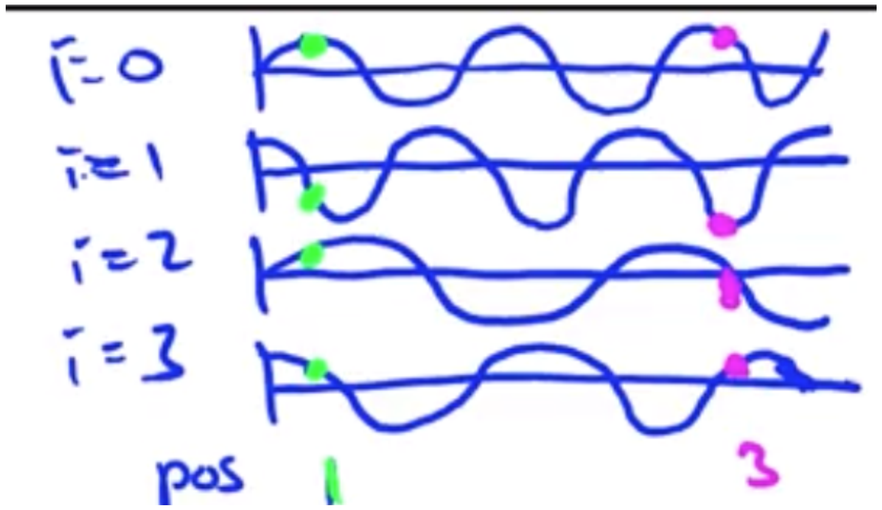
Encoder plus decoder



Another view.

Two more tricks: (i) skip connection (ii) positional encoding

- Positional encoding: we need something like this because when we feed in $x^{<i>}$ we are ignoring its position in the sentence.
- To put it another way: change the order of the words in the sentence. When the encoder reaches $x^{<i>}$ it will produce the same output.
- But how do we incorporate position into the word embedding $x^{<i>}$?
- Answer: we create the vector below, and add it to $x^{<i>}$.
- This is it. This is the “magic.”



Positional encoding. There are as many indices as the dimension of x^<i>

- So here is another “magical” result, from https://arxiv.org/pdf/2106.01345.pdf
- Take an RL problem. They consider a number of them, including open AI gym.
- Generate many sequence of states randomly. Annotate each state with a “reward to go” (i.e., the total reward from that point onwards in the path).
- Feed these into GPT.
- We now have a method of predicting reward to go.
- Now at each step choose the action that has the largest reward.

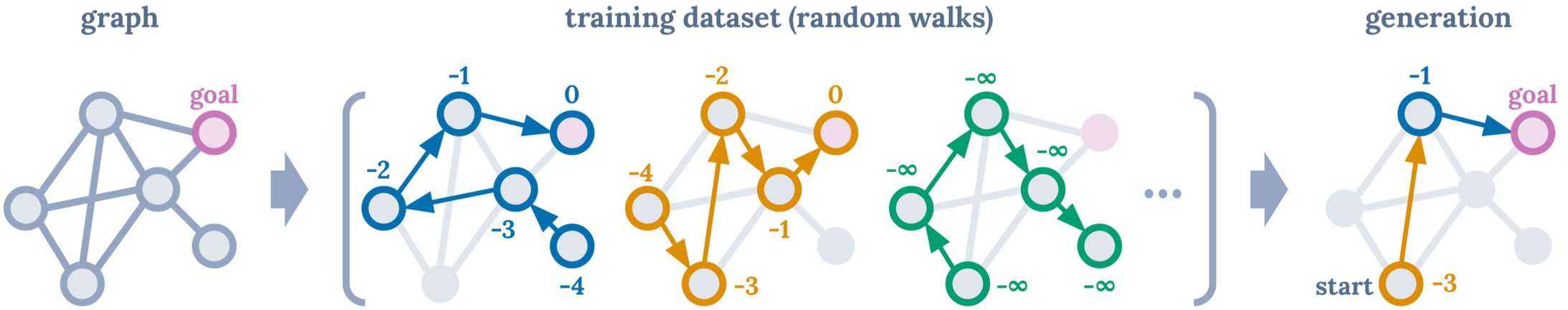


Figure 2: Illustrative example of finding shortest path for a fixed graph (left) posed as reinforcement learning. Training dataset consists of random walk trajectories and their per-node returns-to-go (middle). Conditioned on a starting state and generating largest possible return at each node, Decision Transformer sequences optimal paths.

An illustration.

Now the punchline: on many examples, this gives results comparable or better with the best offline RL methods!

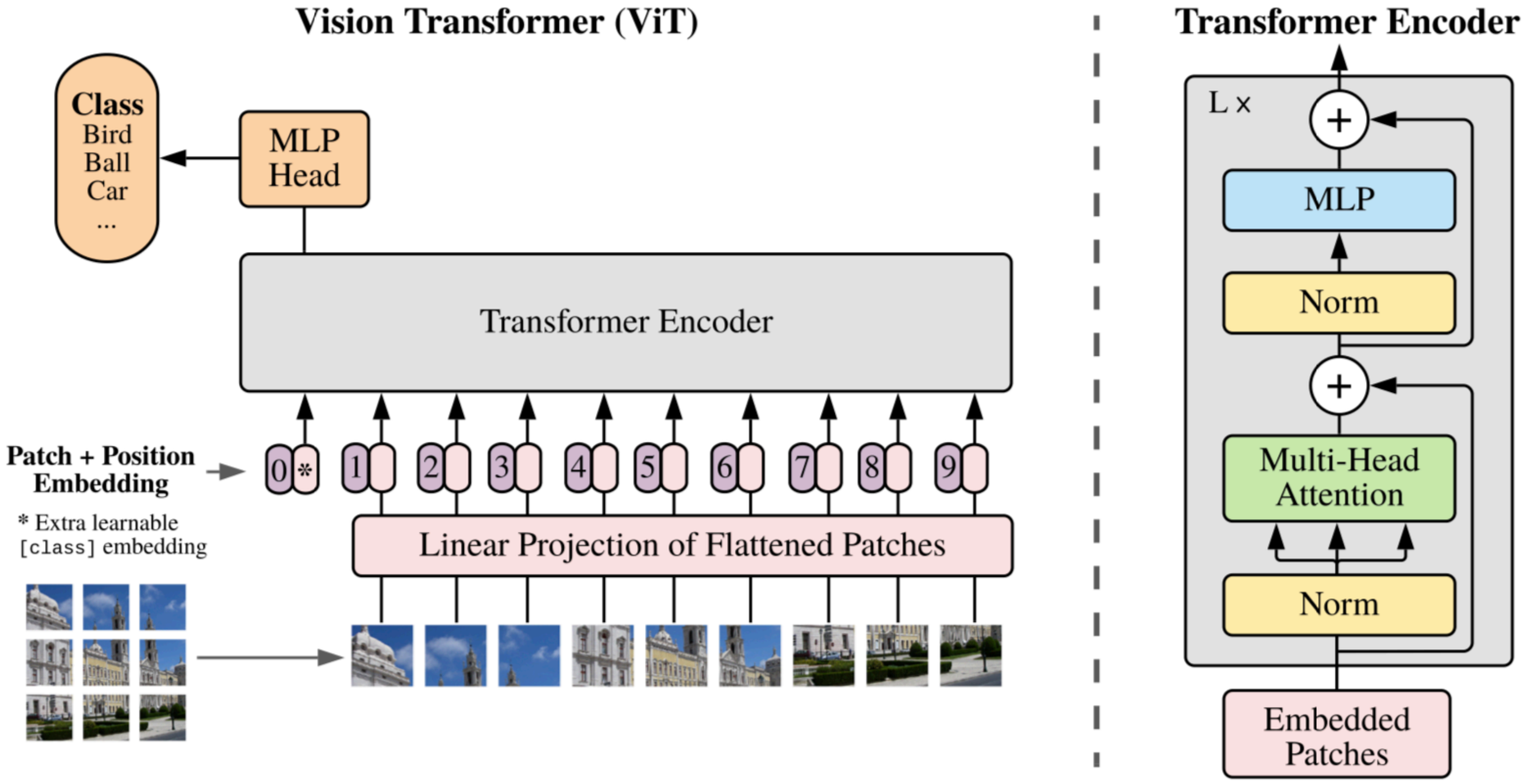
Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	$\mathbf{86.8 \pm 1.3}$	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	**111.0**	96.3	0.8	27.1	79.6
Medium-Expert	Walker	$\mathbf{108.1 \pm 0.2}$	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	$\mathbf{89.1 \pm 1.3}$	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	$\mathbf{67.6 \pm 1.0}$	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	$\mathbf{51.2 \pm 3.4}$	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	$\mathbf{82.7 \pm 7.0}$	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	$\mathbf{66.6 \pm 3.0}$	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	$\mathbf{18.0 \pm 2.4}$	**19.0**	-	-	-	5.4
Average (Without Reacher)		**74.7**	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		**69.2**	54.2	-	-	-	47.7

Comparison with some other popular offline RL methods.

Using transformers instead of LSTM generates much better captions
(skipping a lot of details here)
See https://arxiv.org/abs/2105.13290

- Here is another interesting result: Q-attention for robotic manipulation.
  See the videos at https://sites.google.com/view/q-attention
- Key idea: what does attention do? It allows us to zoom in on parts of the input.
- But isn't this how we manipulate things in the real world? Imagine taking a lid off your sauce pan. You are paying attention to a constantly shifting part of the input image.
- So their idea: take an RL method that has an image as input
  [their method takes in the image and outputs a goal configuration of the robot, which is then used as an input into a control method]
  and add an attention step at the beginning.
- They do a "hard attention" which effectively sets all the attention coefficients to zero outside a small range. In other words, they are effectively cropping a different part of the image at every step.
- They claim to solve robotic manipulation problems that other RL methods fail at.

- As of 20x20, transformers have even beaten cone-nets at image recognition: see https://arxiv.org/pdf/2010.11929.pdf
- See table 2 in the above paper: transformer model can reach similar performance to a resnet (convolution network with residual layers) with about 1/10 as many parameters.
- Note the x-axis “TPUv3 core days.” BTW, a single day on a TPUv3 costs about $8.
- An image is broken up into smaller “patches” and each smaller patch is treated identically to a word embedding in the previous discussion.
- There is also a position encoding, but it is just “1”, “2”, ..., “9” added toe each patch. Picture on next slide.



Norm here is "LayerNorm" which is similar to Batch Normalization