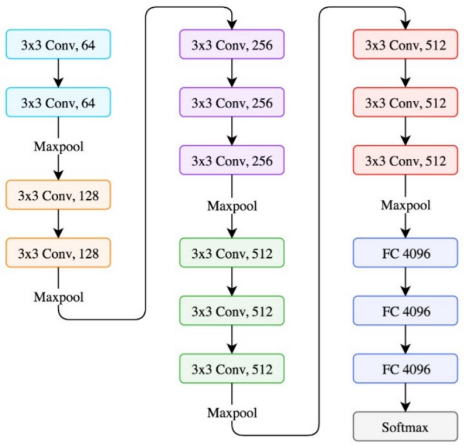
Foundations of Deep Learning

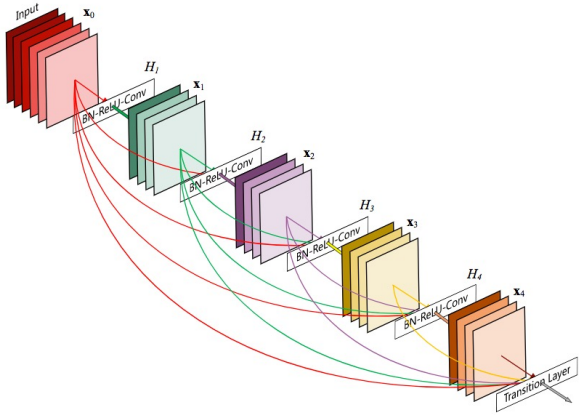
Sequence-to-Sequence Learning

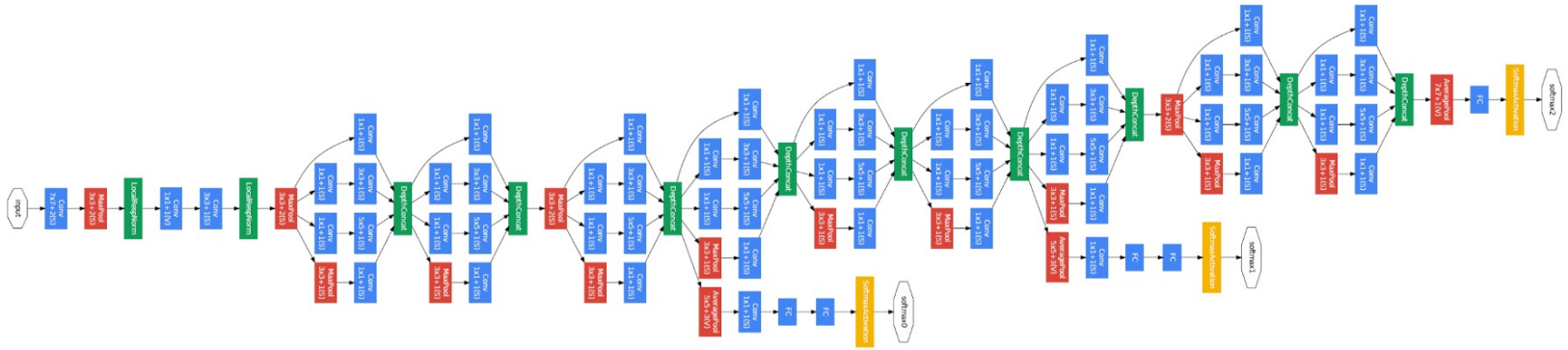
Stergios Christodoulidis
MICS Laboratory
CentraleSupélec
Université Paris-Saclay
https://stergioc.github.io/

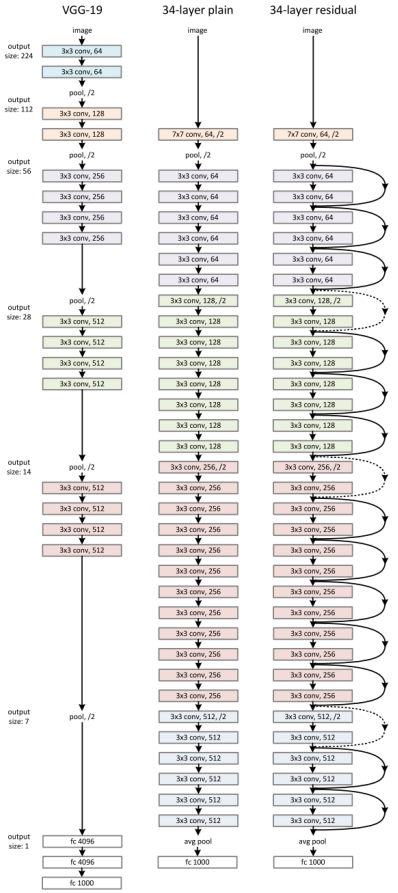
Last Lecture

Modern Classification Convolutional Neural Networks

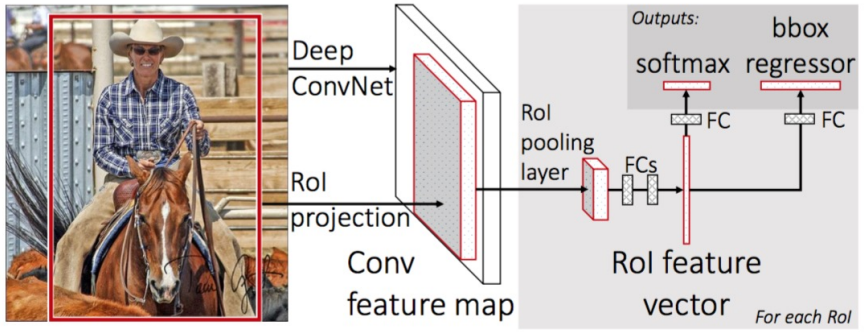




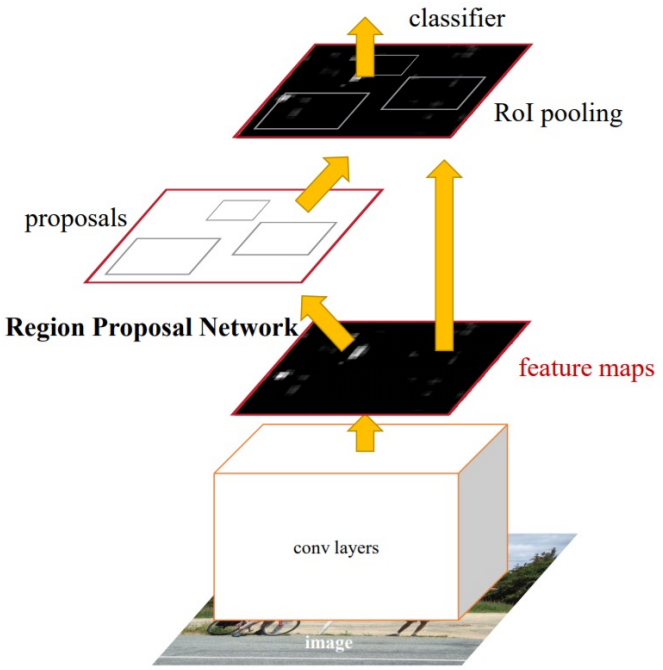




Modern Localization Convolutional Neural Networks

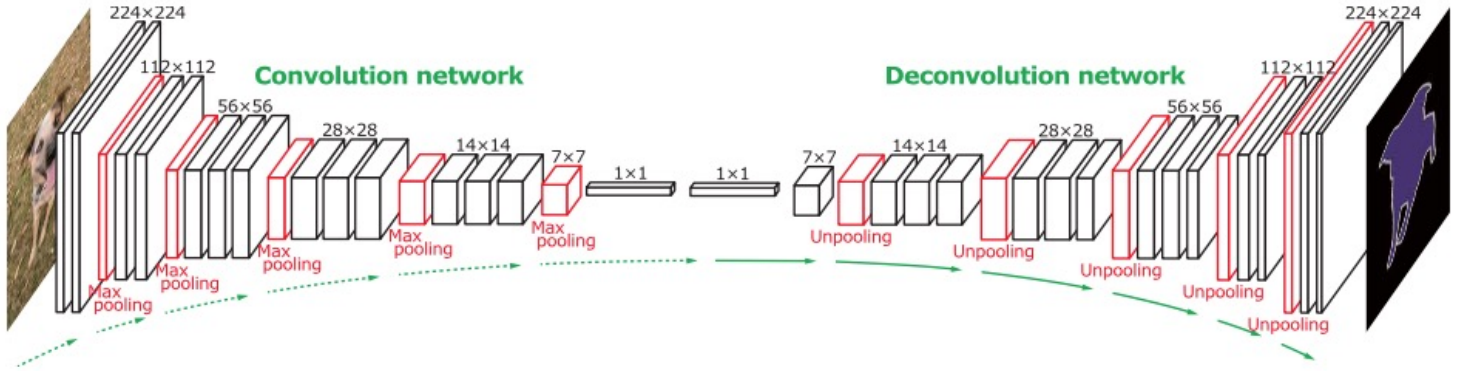


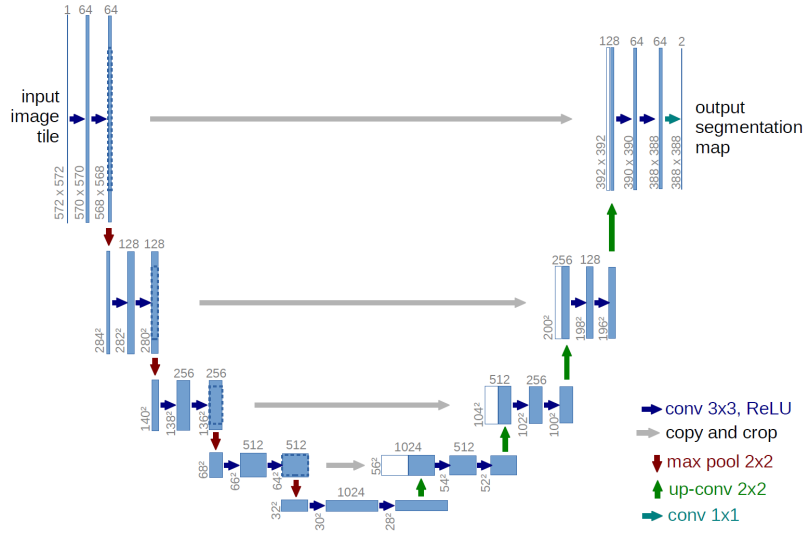
Fast R-CNN



Faster R-CNN

Fully Convolutional Neural Networks





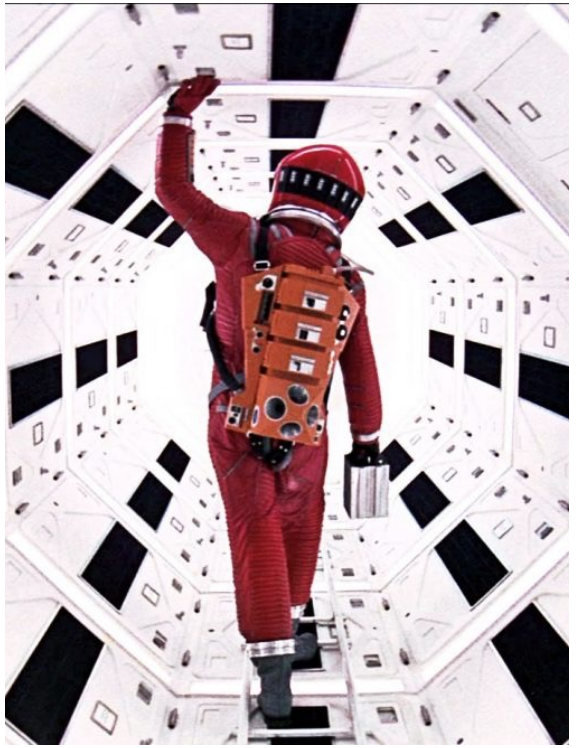
Today's Lecture

Today's Lecture

- Working with Sequential Data
- RNN models
- Backprop through time (BPTT)
- Vanishing/Exploding Gradients
- LSTMs
- Transformers

Motivation

- Up until now we have been looking at independent and identically distributed (i.i.d.) configurations.
- Imagine you want to predict what kind of activity is happening in a movie using a single frame.
- Or similarly, imagine you want to predict the topic of a sentence/paragraph using a single word/sentence.
  - E.g., “year”
  - E.g., “It was the previous year when I first realized it.”
- A large amount of the real-world tasks rely on sequential data (e.g., Human attention)



Sequential Data

- Text
- Time Series
  - Stock prices
  - Biological measurements (heart rate, EEG, etc.)
  - Climate measurements (temperature, humidity, etc.)
- Speech/Music
- Videos
- Human Behavior
- ...

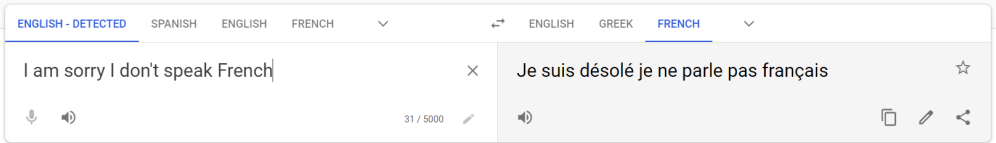
How are sequences different?

- Sequences are of arbitrary or even infinite length.
- Data in sequences are not i.i.d.
  - Next word depends on previous words
- In order to answer questions that rely on sequential data we need some notion of memory.
  - **Data:** "In the year 2019, the SARS-CoV-2 virus emerged and quickly spread around the world. The World Health Organization (WHO) declared the outbreak a Public Health Emergency of International Concern on 30 January 2020, and a pandemic on 11 March 2020."
  - **Question?**

⟶ Recurrent Neural Networks (RNNs) are often used for handling sequential data

Applications of RNNs?

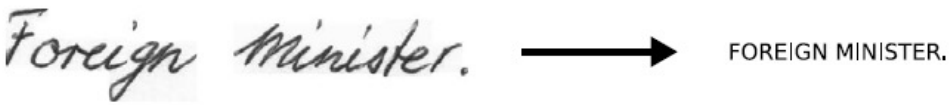
- Machine translation
- Video/Image captioning
- Question answering
- Video generation
- Speech synthesis
- Speech recognition
- ...



"man in black shirt is playing guitar."

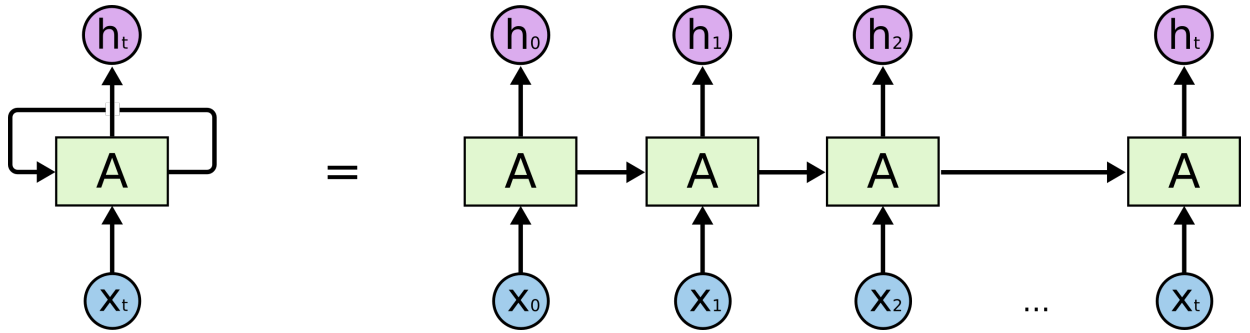
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."



Recurrent Neural Networks (RNNs)

- They were first introduced in 1986.
- They are neural networks with loops in order to allow information to persist.
- These loops represent the influence of previous value on the same value at the current step.
- For a simpler representation we could unroll this RNN in time.



(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Properties of Sequences

- We cannot, and ideally should not, have a constraint over the length of our sequence.
- Our model should have no problem with:
  - Varying
  - Unknown
  - Or even infinite

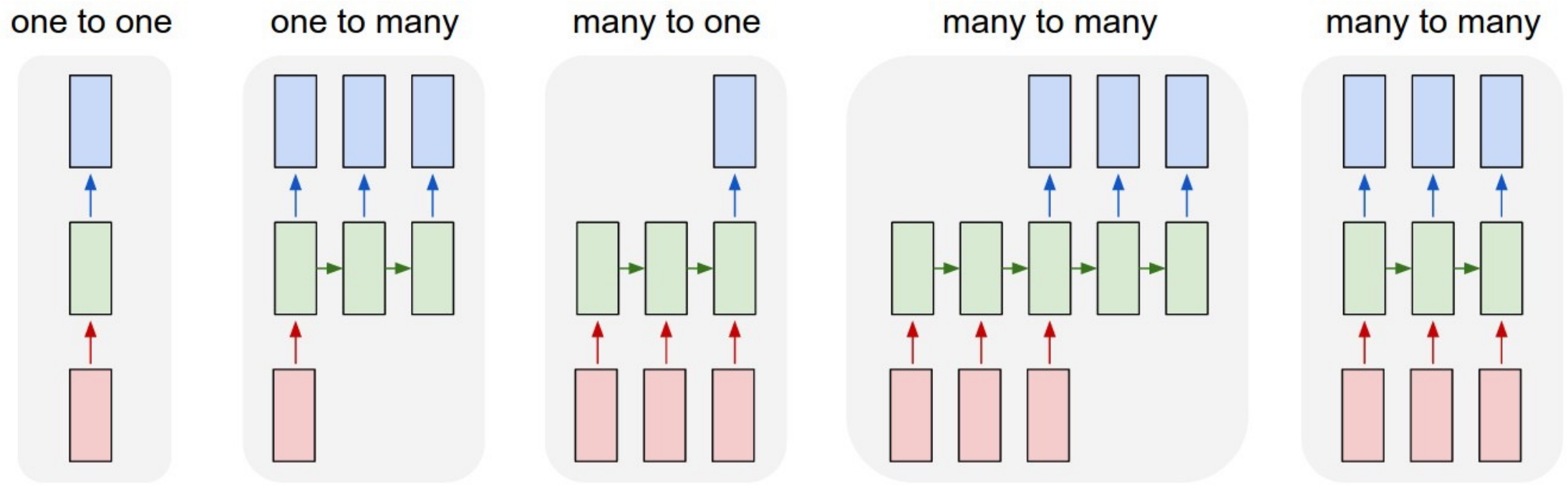
  Sequence lengths

⟶ Parameter sharing makes it possible to extend and apply the model to examples of different forms and generalize across them.

- It can be particularly important when a specific piece of information can occur multiple times or at different positions
  - "Answer to the Ultimate Question of Life, The Universe, and Everything"
  - "My name is Bond, James Bond"

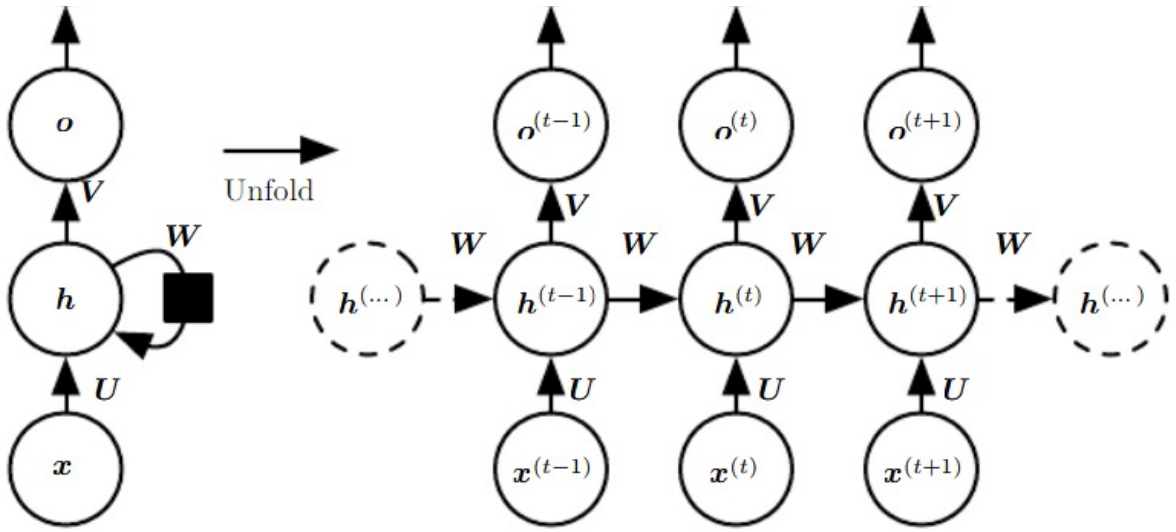
Input-Output Schemes of RNNs

e.g. Image Captioning | e.g. Sentiment Analysis | e.g. Machine Translation | e.g. Parts of Speech



(Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015)

Example Recurrent Neural Network

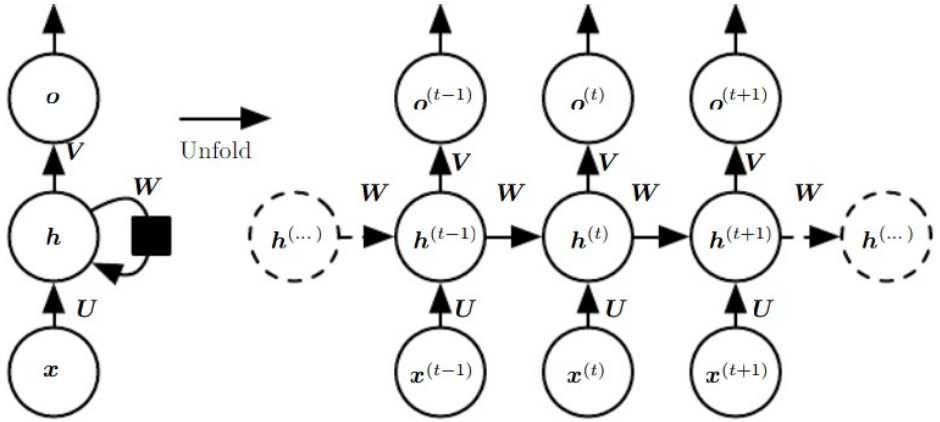


$$\boldsymbol{h}^{(\boldsymbol{t})} = \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(\boldsymbol{t-1})} + \boldsymbol{U}\boldsymbol{x}^{(\boldsymbol{t})}\right)$$

$$\boldsymbol{y}^{(\boldsymbol{t})} = softmax(\boldsymbol{V}\boldsymbol{h}^{(\boldsymbol{t})})$$

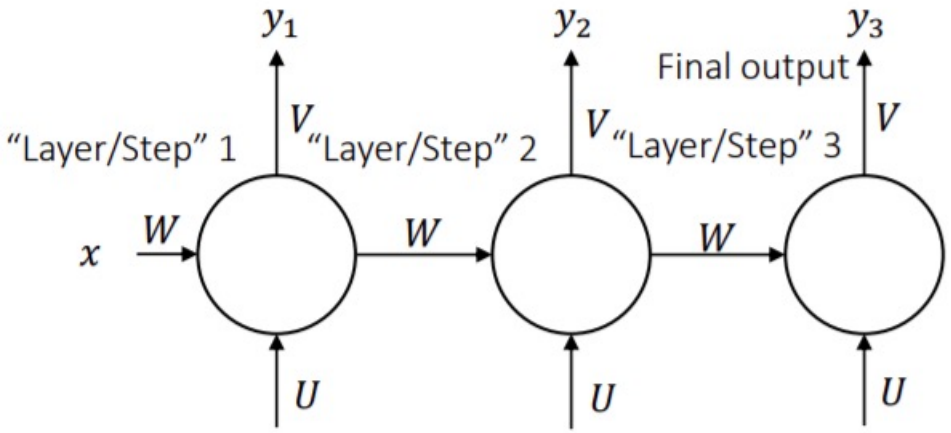
Example Recurrent Neural Network

- In the simplest case with fully connected layers $\theta = \{U, V, W\}$
- We map our inputs to the latent representation trough U
- This map our current latent representation to the new one trough W
- The output is computed through the set of parameters V
- At each timestep t all previous information $1, \dots, t$ is used through the recurrent connection.

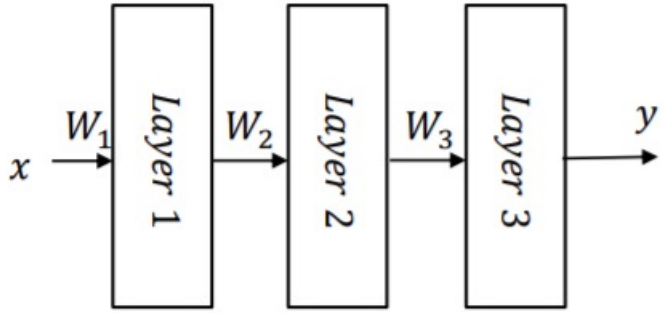


RNN vs MLP

- Instead of layers ⟶ Steps
- Output in every step ⟶ MLP outputs in every layer possible
- Main difference: Instead of layer specific parameters ⟶ Layer shared parameters



3-gram Unrolled Recurrent Network

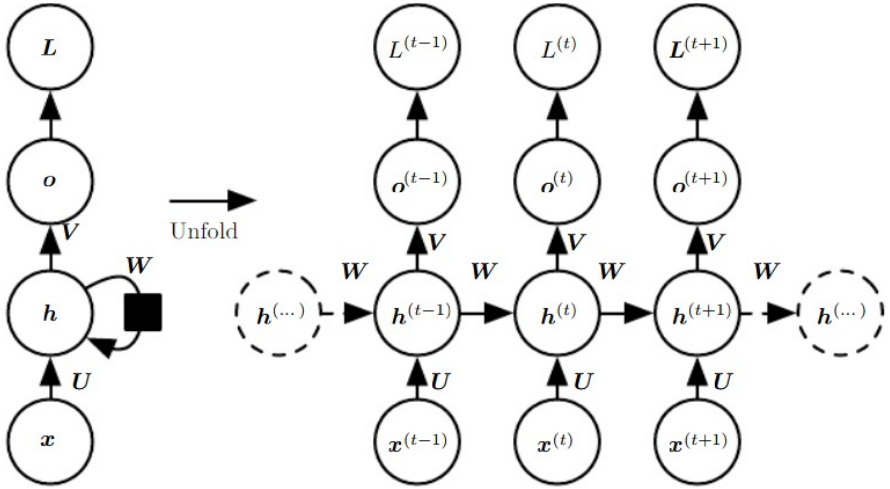


3-layer Neural Network

Backpropagation Trough Time (BPTT)

- Using the generalized backpropagation algorithm, and by unrolling the RNN we can obtain the back propagation trough time algorithm.
- Basically, we can use the chain rule again on the unfolded computational graph.
- We just need to take into consideration that gradients survive trough time.
- We need to calculate the gradient of the Loss function with respect to our parameters: $\frac{\partial \mathcal{L}}{\partial U}, \frac{\partial \mathcal{L}}{\partial V}, \frac{\partial \mathcal{L}}{\partial W}$
- U, V, W are the same for all t
- The gradient flows from multiple paths in contrast with MLPs

Backpropagation Trough Time (BPTT)



$$\boldsymbol{h^{(t)}} = \tanh\left(\boldsymbol{W h^{(t-1)}} + \boldsymbol{U x^{(t)}}\right)$$

$$\boldsymbol{o^{(t)}} = softmax(\boldsymbol{V h^{(t)}})$$

$$\mathcal{L} = \sum_t \mathcal{L}^{(t)} = \sum_t \mathcal{L}^{(t)}\left(o^{(t)}, y^{(t)}\right)$$

$$\frac{\partial \mathcal{L}}{\partial V} = \sum_t \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{(t)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{(t)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial W}$$

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_t \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{(t)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial U}$$

BPTT: Chain Rule for $\partial \mathcal{L}/\partial W$

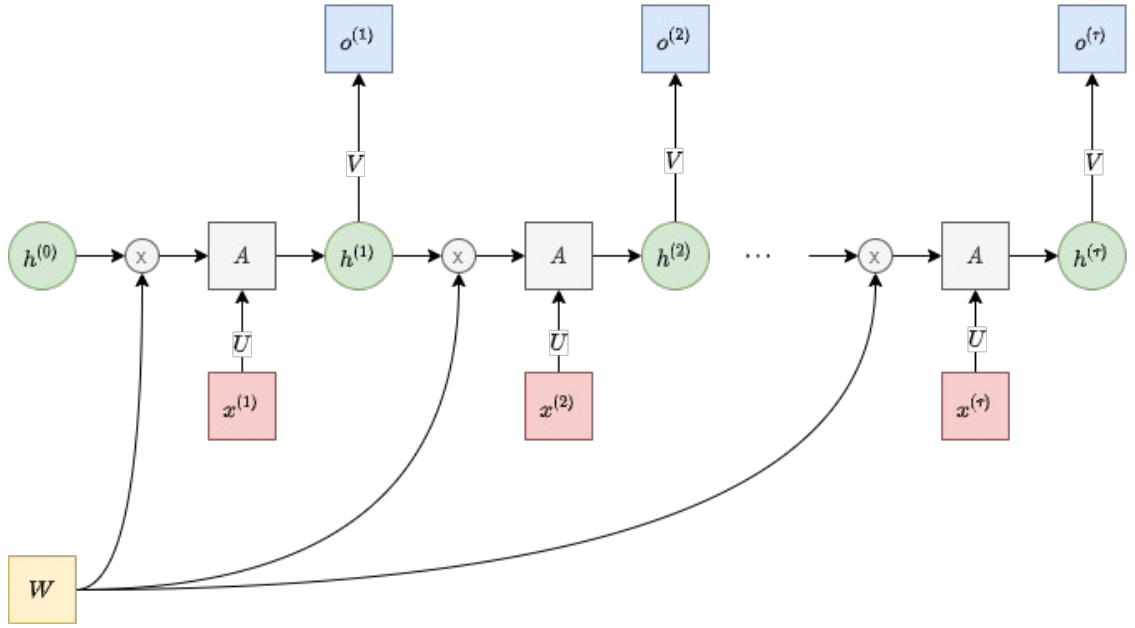
- All gradient path flows from h to W
  - Via $h^{(\tau)}$
  - Via $h^{(\tau-1)}$
  - ...

- Putting everything together:

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial h^{(i)}} \frac{\partial h^{(i)}}{\partial W}$$

- If there are multiple losses over time steps:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t} \frac{\partial \mathcal{L}^{(t)}}{\partial W}$$



min-char-rnn.py

min-char-rnn.py — Raw

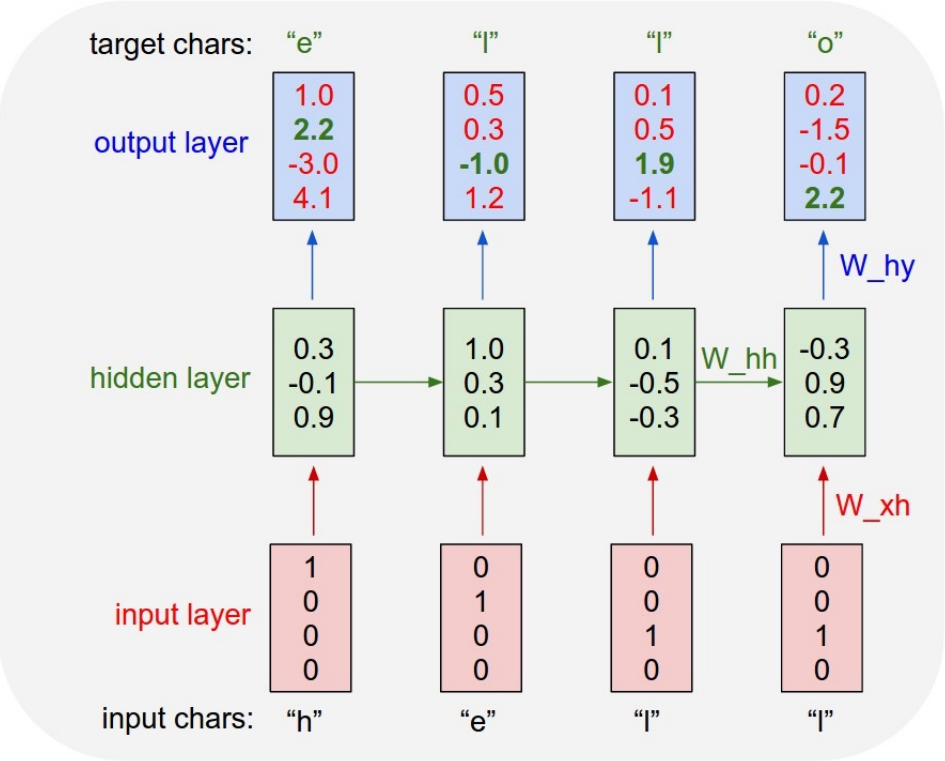
```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-study/#grad if confused here
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

https://gist.github.com/karpathy/d4dee566867f8291f086

Example: Character Level Language Models

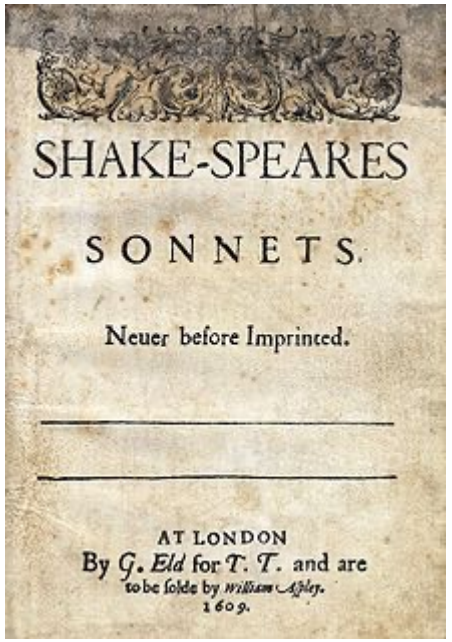
- Example of a simple RNN used in a character level
- Consider a huge text chunk
- If we encode each character to a one-hot vector.
- We can train an RNN to predict the probability distribution of the next character
- As such, we can then generate a text a character at a time.

- At inference time, we sample from the output distribution in order to select the next character



Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015

Shakespeare Database




```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

```
we counter. He stutn co des. His stanted out one ofler that concossions and was
to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn
```

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Andrej Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks”, 2015

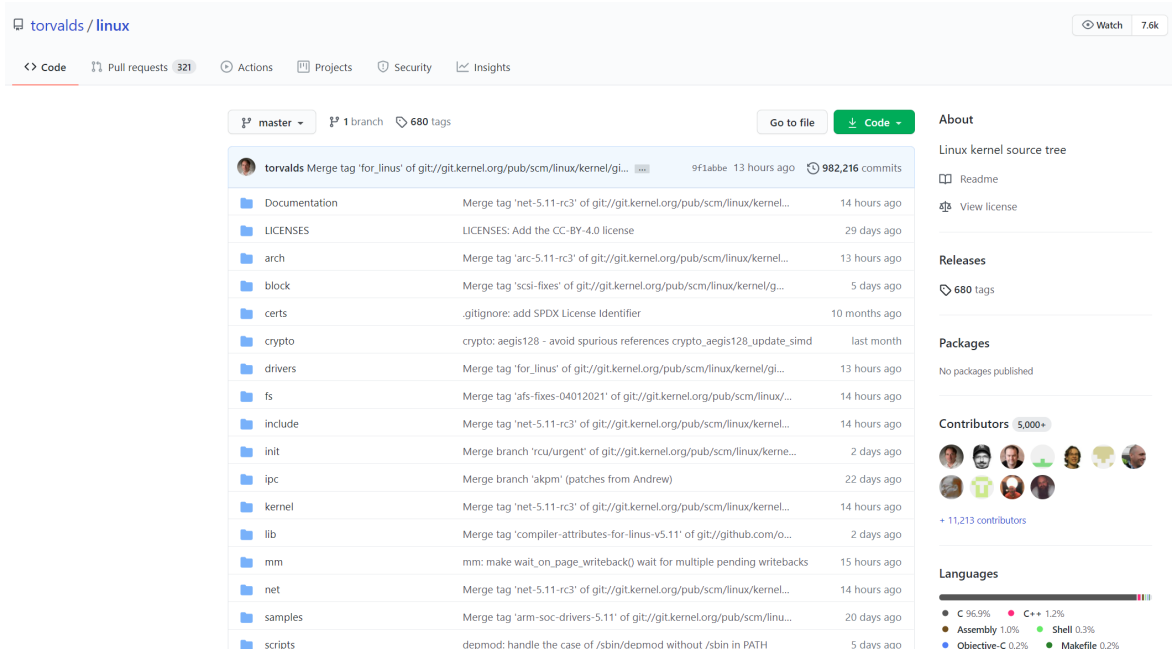
Shakespeare Database

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Andrej Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks”, 2015

Linux Source Code Database




```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

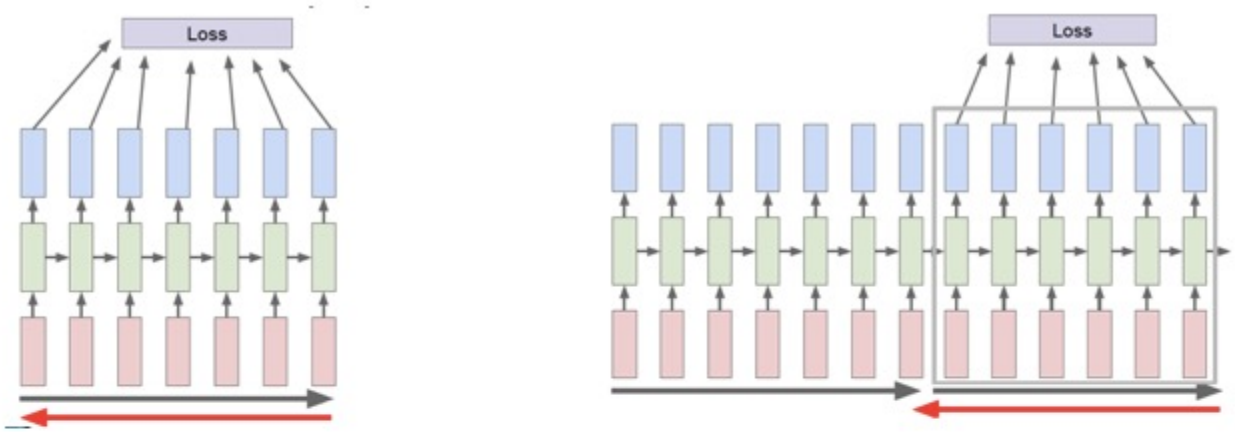
Andrej Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks”, 2015

Standard RNNs are Difficult to Train in Practice

- Unrolling forever is not practical or even feasible
- Gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely)
- Long-term memory difficult to persist.
  - Related with the vanishing gradients problem.

Truncating BPTT

- Truncating to some τ (~100) is a usual strategy.
- Analogous to stochastic gradient descent.
- You can carry the hidden state variables of the previous batch to the current
- More focus on short-term memory
  - Not necessarily undesirable, as long-term dependency might be irrelevant
  - Biases towards simpler models with shorter-term dependencies



(https://srdas.github.io/DLBook/RNNs.html)

Vanishing and Exploding Gradients

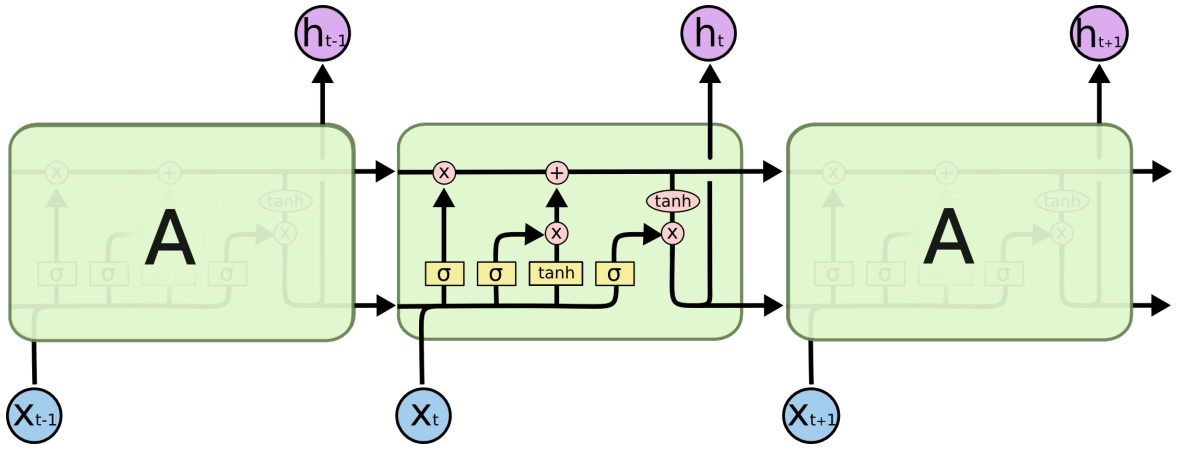
- Gradients vanish or explode even easier because of shared parameters

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial h^{(i)}} \frac{\partial h^{(i)}}{\partial W} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left(\prod_{j=i+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(i)}}{\partial W}$$

- If $\frac{\partial h^{(j)}}{\partial h^{(j-1)}} > 1 \implies \frac{\partial \mathcal{L}^{(t)}}{\partial W} \gg 1 \rightarrow$ Exploding Gradient $\longrightarrow$ Gradient Clipping
- If $\frac{\partial h^{(j)}}{\partial h^{(j-1)}} < 1 \implies \frac{\partial \mathcal{L}^{(t)}}{\partial W} \ll 1 \rightarrow$ Vanishing Gradient $\longrightarrow$ Architecture Change

Long short-term memory (LSTMs)

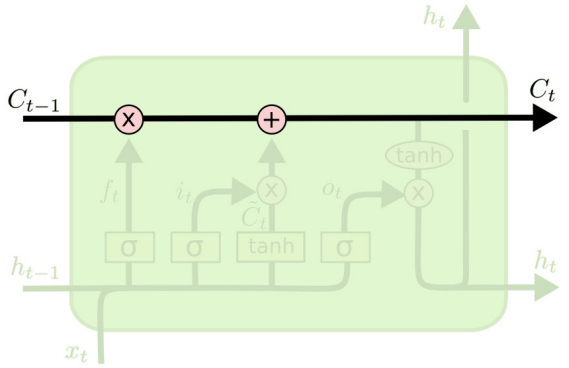
- LSTMs are an RNN architecture.
- They are capable of learning long-term dependencies
- Tackling the vanishing gradient problem
- The LSTM has been found extremely successful in many applications



(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM: Cell State

- The core idea behind LSTMs lies on the cell state (the recurrent part).
- It runs straight down the entire module.
- Minor interactions with the other components.
- It is possible that information flow unchanged.

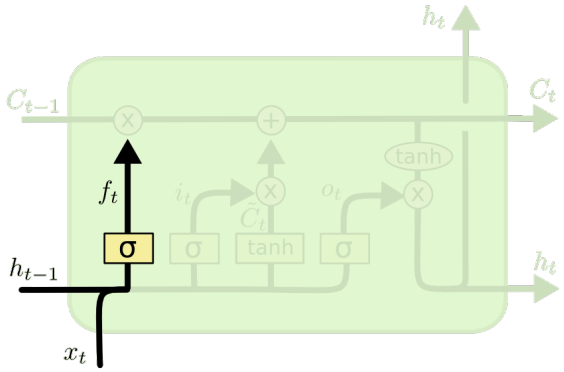


- The LSTM has the ability to add/remove information from this cell state in a regulated manner.

(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM: Forget Gate

- The first step is to decide what information we are going to throw away.
- It looks at the previous output and the current input.
- The sigmoid brings the values to [0,1] ranges
- When multiplied with the cell state the relevant information will be kept and the rest will be discarded



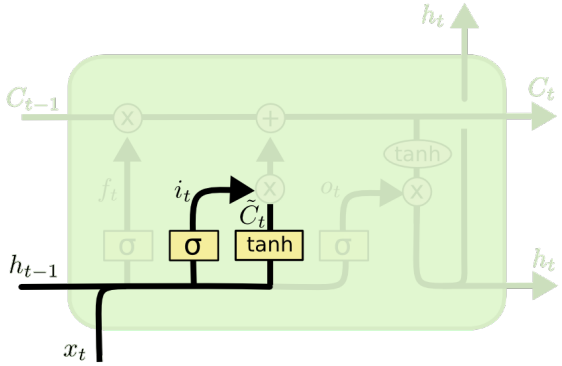
- This is called the forget gate.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM: Input Gate

- Before generating the new cell state:
  - The we identify new candidate values that could be added in the cell state from the new input
  - We scale them with another sigmoid
- We are again looking at the previous output and the current input.



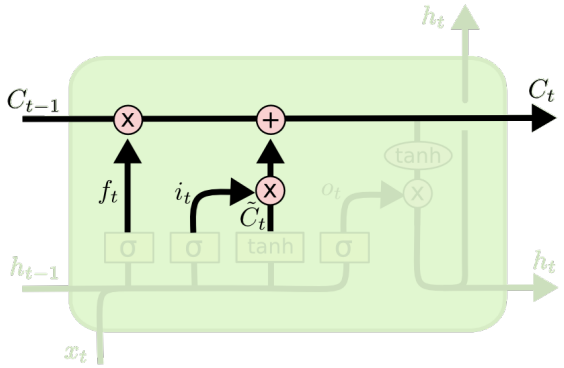
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM: New Cell State

- Now we can apply the forget and write operation in the cell state
- We achieve this with a simple multiplication and addition.
  - Multiplying with the forget signal
  - Adding the new input signal

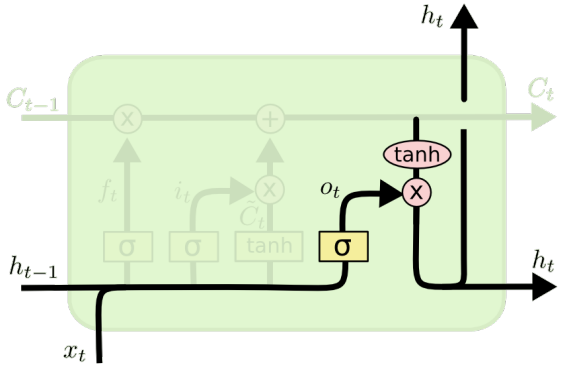


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM: Output

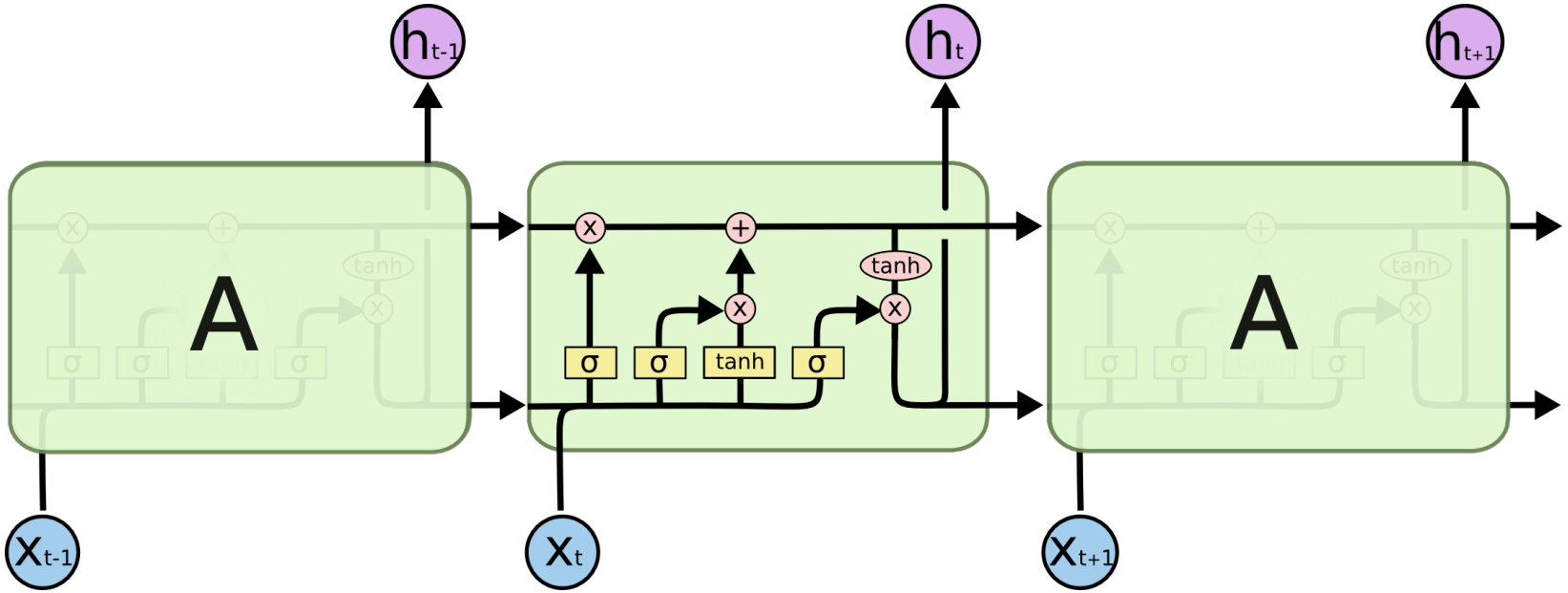
- Lastly, we can use the new cell state to calculate the output
- We can push the values to the range [-1,1] using the tanh
- We select the parts that are relevant for our output using a sigmoid



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

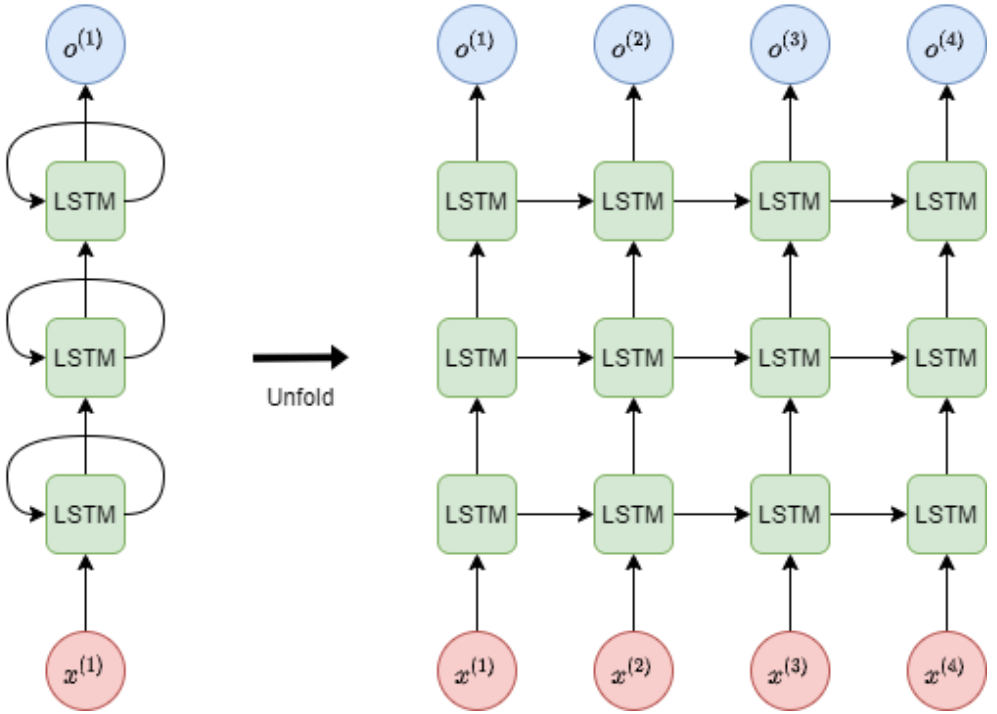
LSTMs



(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

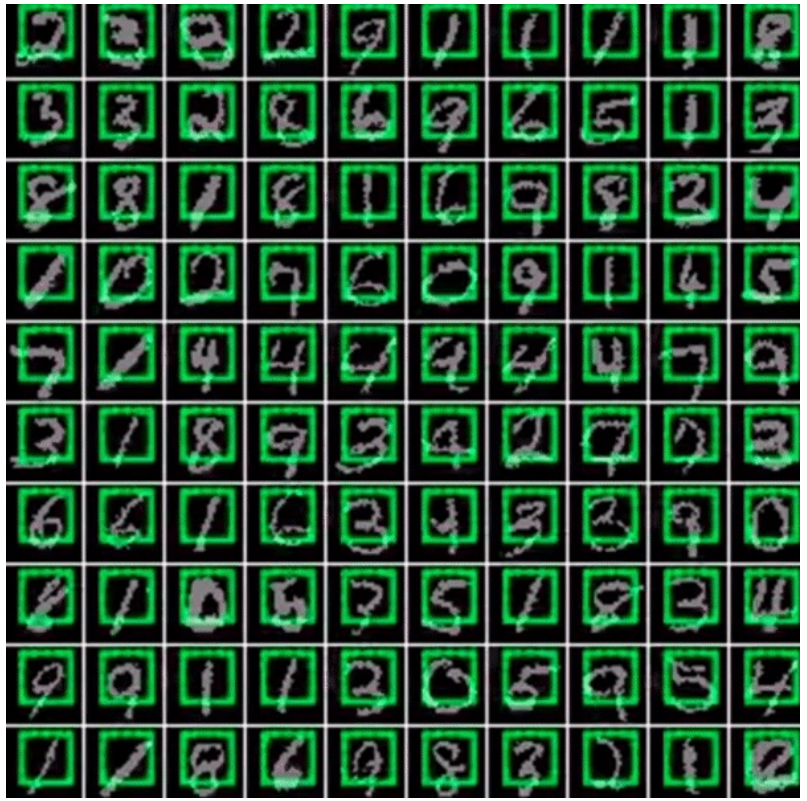
Multi-Layer LSTMs

- RNNs can be stacked in order to for multi-layer architectures.
- In this case we have depth and time.
- BPTT still stands and these can also be optimized
- For many problems you might see 2-4 RNN layers stacked on top of each other



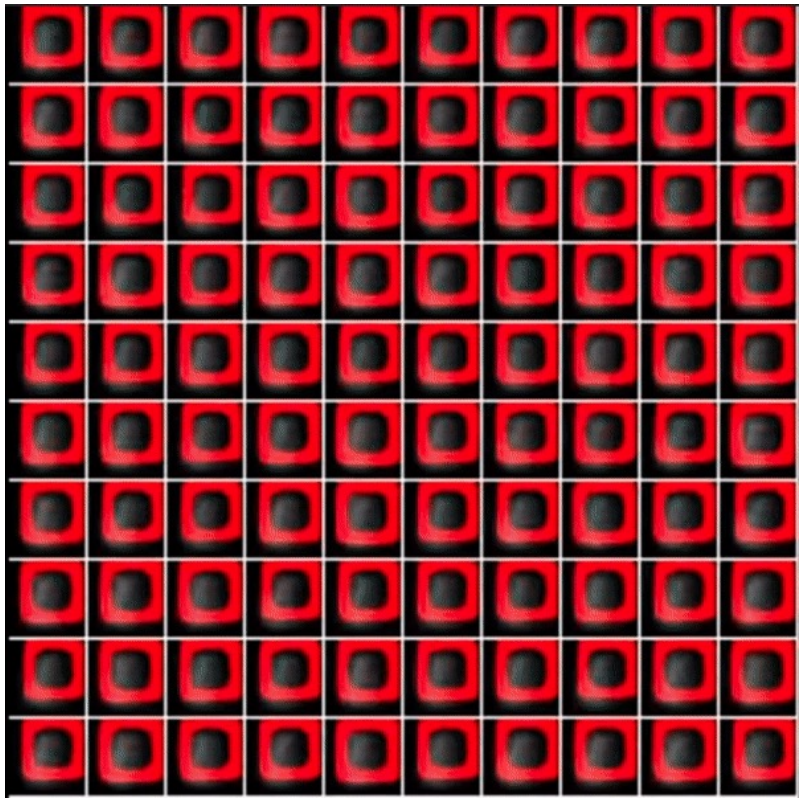
LSTM for Classification

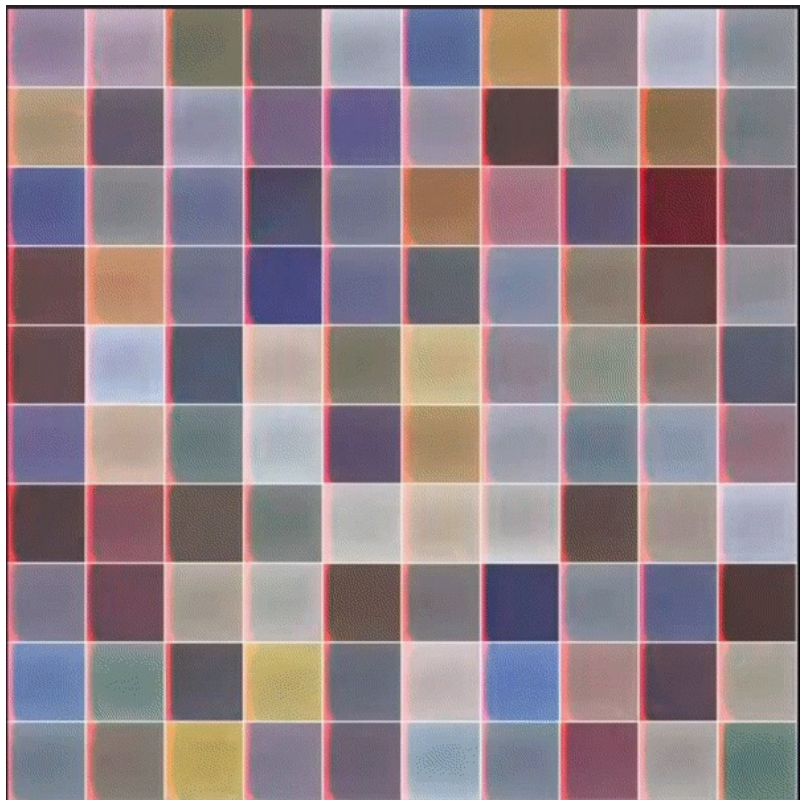
- An LSTM RNN is trained in order to selectively get glimpses of the images
- These glimpses are received in a predefined size (12x12)
- The glimpse sampling is defined in a differential way.
- After a pre-defined number of glimpses the RNN is using the final cell state to perform the classification



Gregor, Karol, et al. "Draw: A recurrent neural network for image generation", ICLR 2015

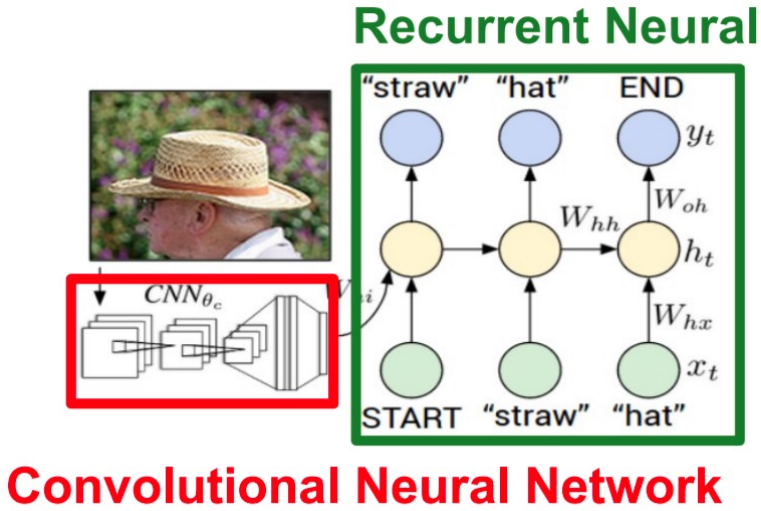
LSTM for Generation





Gregor, Karol, et al. “Draw: A recurrent neural network for image generation”, ICLR 2015

Image Captioning



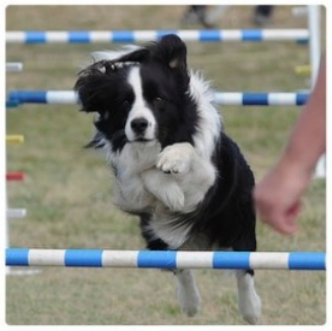

"man in black shirt is playing guitar."


"construction worker in orange safety vest is working on road."


"two young girls are playing with lego toy."


"boy is doing backflip on wakeboard."


"girl in pink dress is jumping in air."

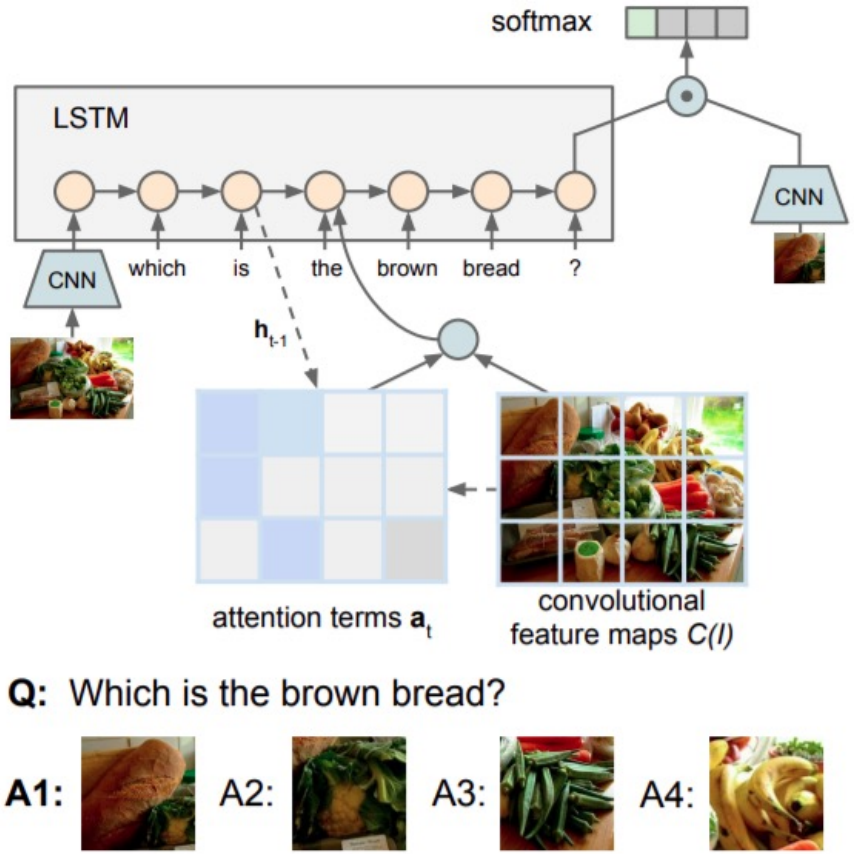

"black and white dog jumps over bar."

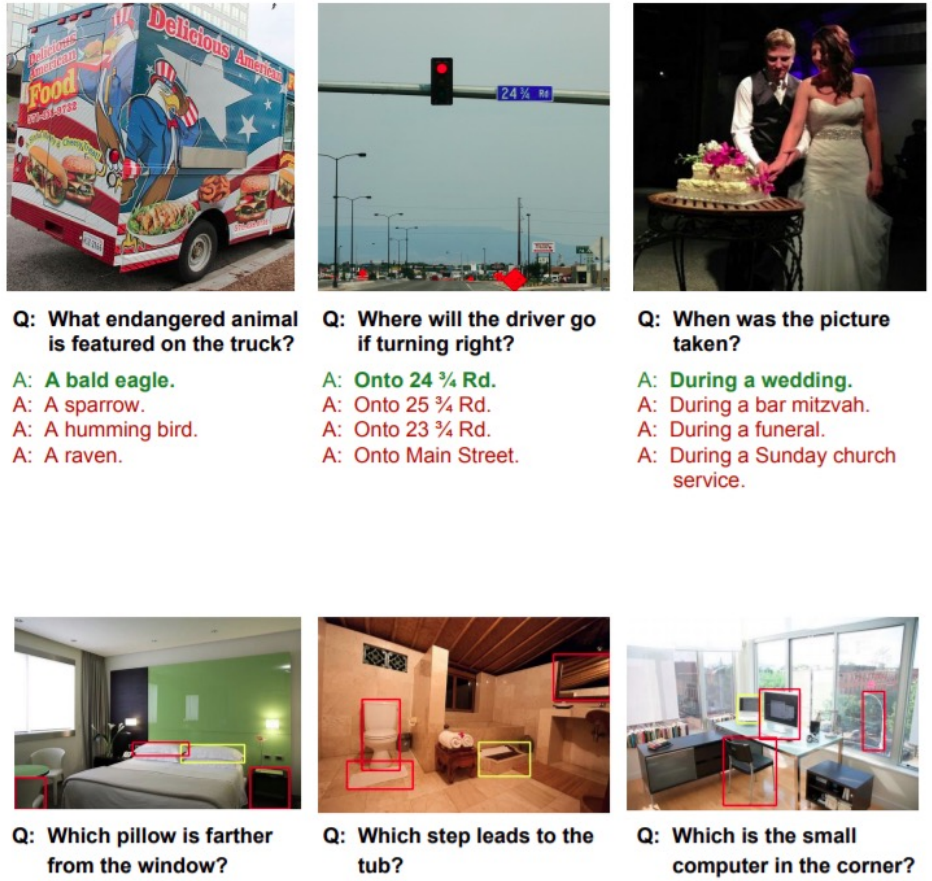

"young girl in pink shirt is swinging on swing."


"man in blue wetsuit is surfing on wave."

Andrej Karpathy, "Automated Image Captioning with ConvNets and Recurrent Nets", 2015

Visual Question Answering





Yuke Zhu, et al., "Visual7W: Grounded Question Answering in Images", 2016

Transformers

- A sequence-to-sequence model with an Encoder-Decoder architecture.
- Even though it is not a reccurent network, it is designed such that it can work with sequences
  - It utilizes positional encoding of the input to capture the relative positions across
  - It utilizes a self-attention mechanism ti decide which other parts of the sequence are important.
- The state-of-the-art for a lot of applications, extended to vision
  - Vision Transformers

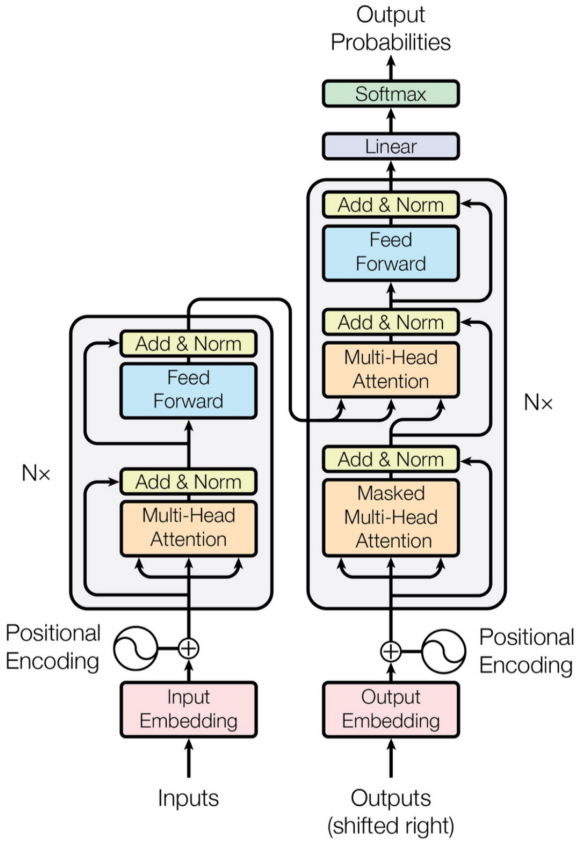


Figure 1: The Transformer - model architecture.

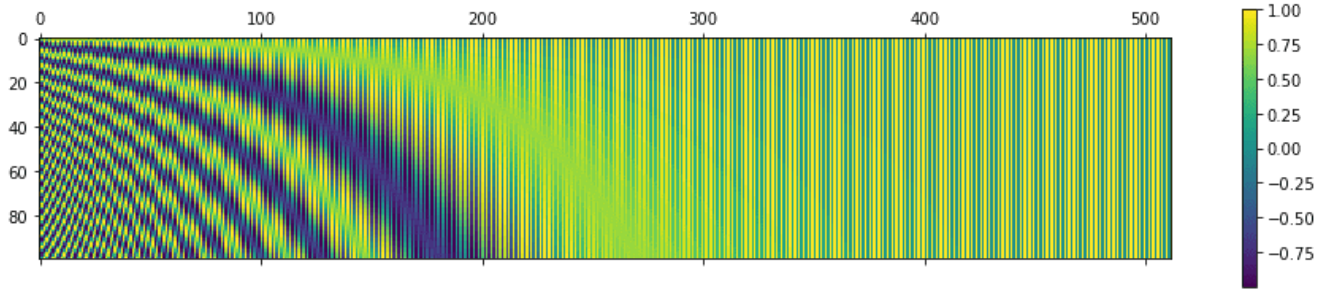
Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Positional Encoding

- Describes the location of each element in a sequence in a unique way
- In transformer architectures, each position is mapped to a vector using sine and cosine functions

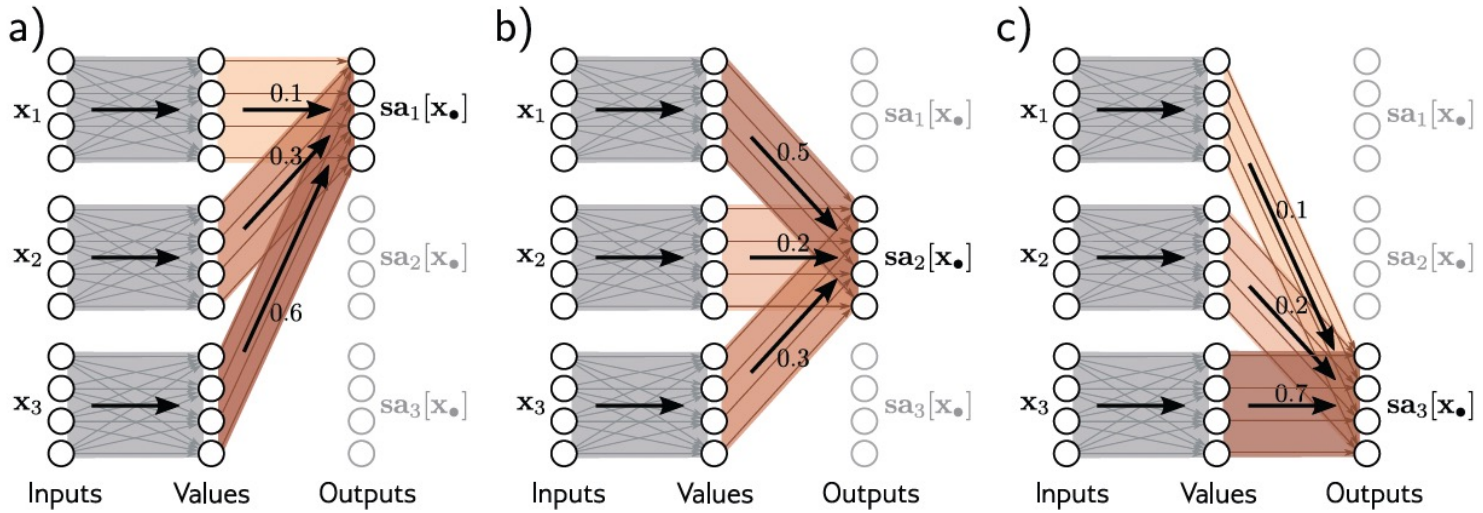
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- With:
  - pos: position of element (x-axis of the figure)
  - i: index of the embeding (y-axis of the figure)
  - d_{model}: length of the embedding



Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Self Attention



- Self-attentions is a mechanism that takes N inputs and processes each separately to compute N value vectors
- These value vectors are then combined using a weighted sum to generate N outputs.
- Sparse matrices with repeated elements can be utilized similarly with Conv layers for apply these efficiently

Understanding Deep Learning by Simon J.D. Prince Published by MIT Press Dec 5th, 2023.

Computing Attention Weights

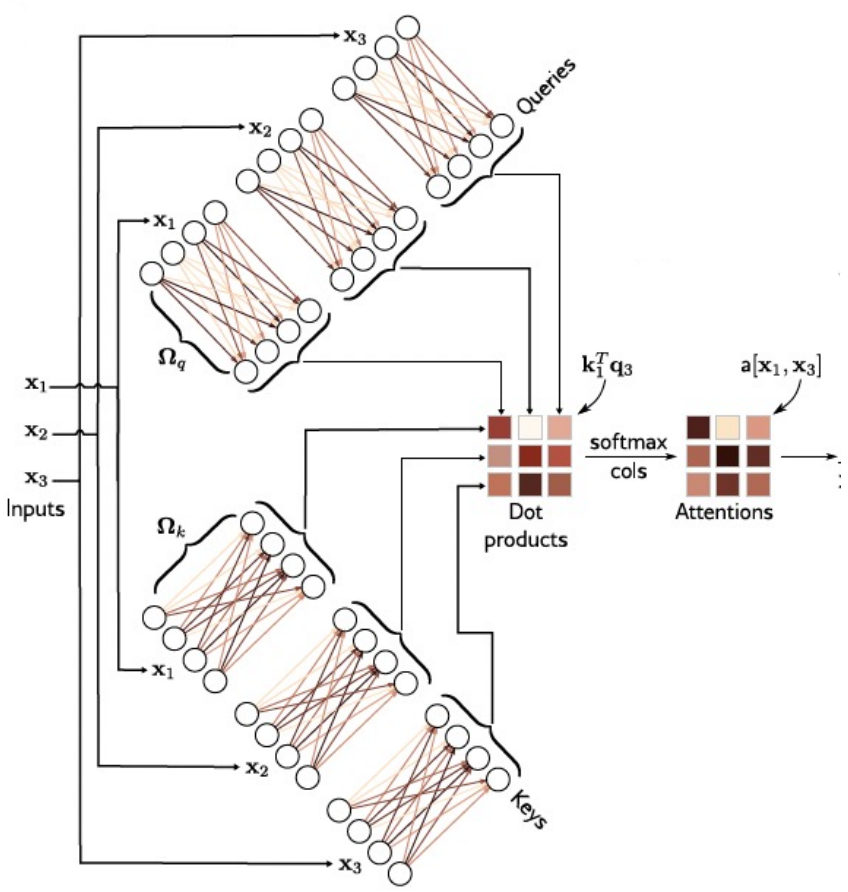
- To calculate the self-attention weights, we are using a nonlinear processing formulation.
- First, we calculate the queries and the keys (terms borrowed from the information retrieval field):

$$q_n = \beta_q + \Omega_q x_n$$
$$k_n = \beta_q + \Omega_q x_n$$

- Then we compute the dot product between the two and we pass the result through a softmax function

$$a[x_m, x_n] = softmax[k^T \cdot q_n]$$

- The dot operations returns a measure of similarity between the two inputs.



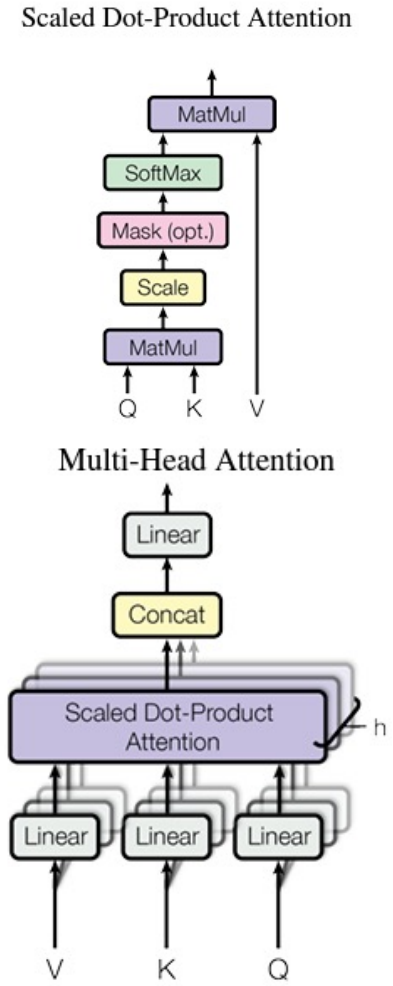
Understanding Deep Learning by Simon J.D. Prince Published by MIT Press Dec 5th, 2023.

Multi-Head Attention

- Eventually, this is extended for multiple heads. The (scaled) attention mechanism is descibed by the following equation:

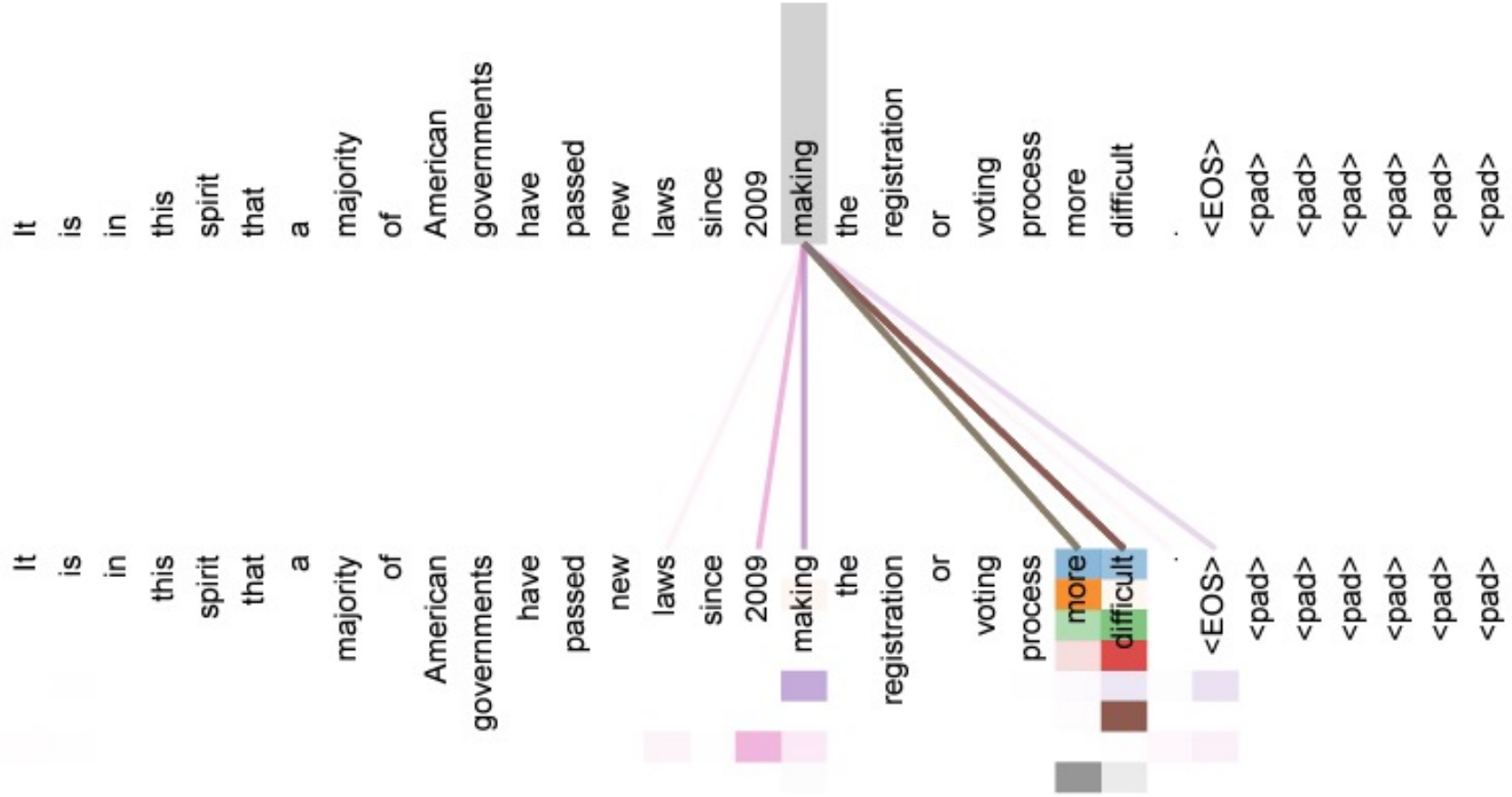
$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Q is a matrix that contains the query, i.e., a vector representation of one word in the sequence
- K are all the keys, i.e., a vector representation of all the words in the sequence
- V are the values, i.e., again a vector representation of all the words in the sequence
- Those matrices (Q,K,V) are different for each position of the attention modules



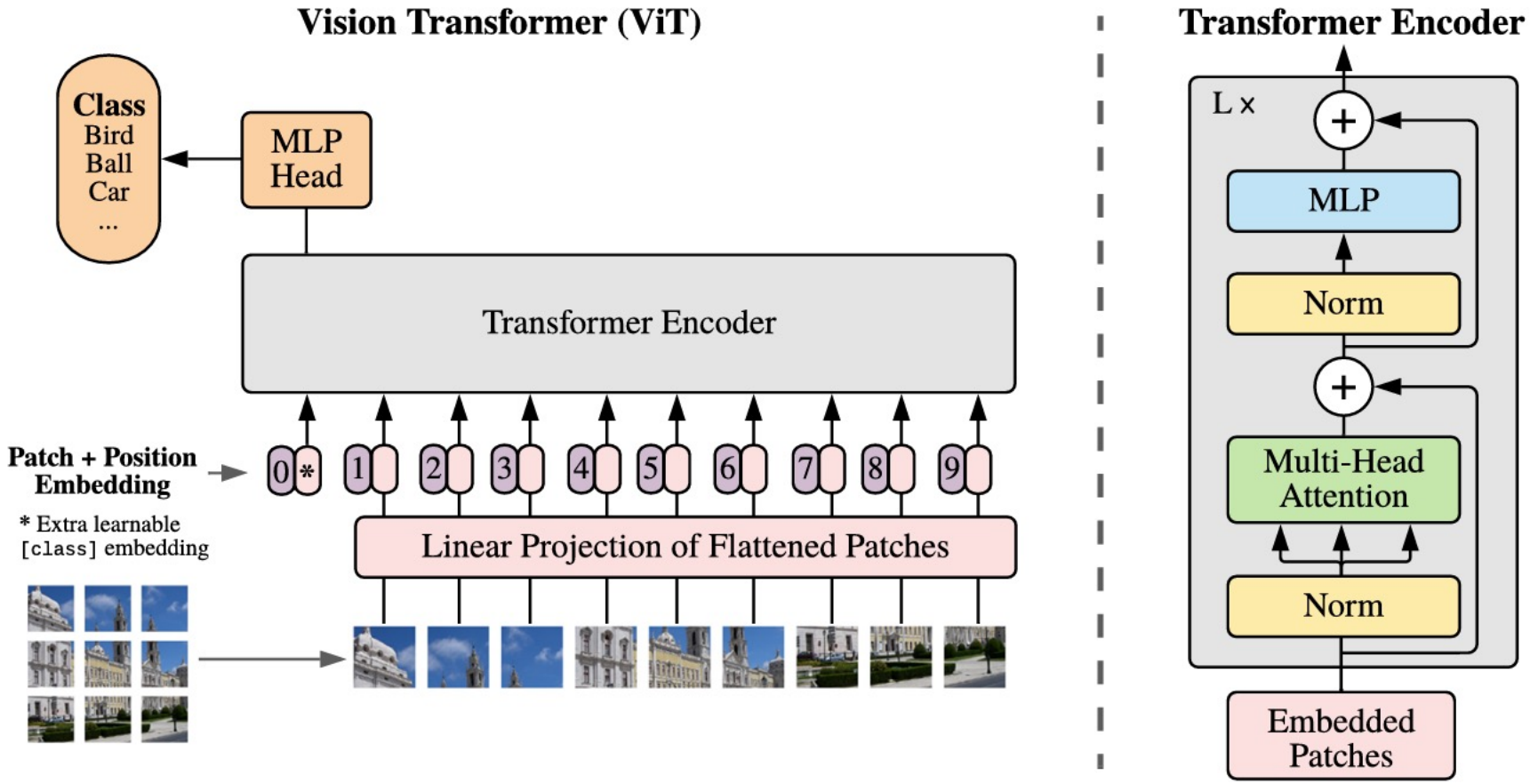
Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Attention Visualization



Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Vision Transformers



Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

Take Home Messages

- Standard RNNs have been proposed in order to handle sequential data.
- The core idea is the use of the recurrent connection to allow memory to persist.
- They can be trained using the backpropagation trough time.
  - Chain rule still holds
- Due to the properties of the data and their structure they are difficult to train.
- LSTMs and variants have been introduced in order to solve some of these difficulties in training.
  - A lot of variants since then have revolutionized the field.
- Transformers are again sequence-to-sequence architectures that introduced self-attention and positional encodings to tackle the issue.
- Further Reading:
  - Deep Learning Book Chapter 10, Ian Goodfellow, Yoshua Bengio, Aaron Courville
  - All the links and references that are provided in these slides