


FACULTY OF MATHEMATICS,
PHYSICS AND INFORMATICS
Comenius University
Bratislava

Neural Networks for Computer Vision

Lecture 9: RNNs and Transformers

Ing. Viktor Kocur, PhD., RNDr. Zuzana Černeková, PhD.

21.11.2024

Contents

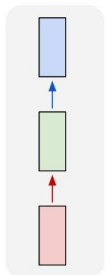
Acknowledgment

Most of the slides are directly adopted from slides for CS231n[1] course at Stanford University!

[1] Fei-Fei Li, Ranjay Krishna, and Danfei Xu. *Stanford CS231n lecture slides*. `http://cs231n.stanford.edu/slides/`

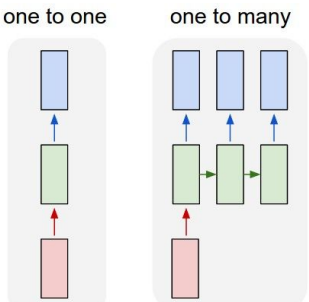
“Vanilla” Neural Network

one to one



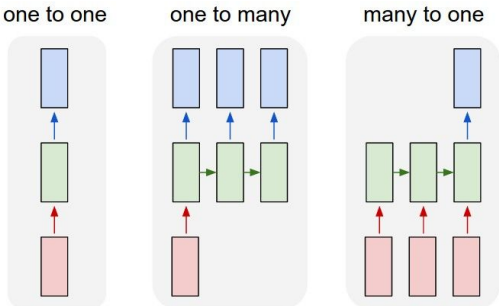
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



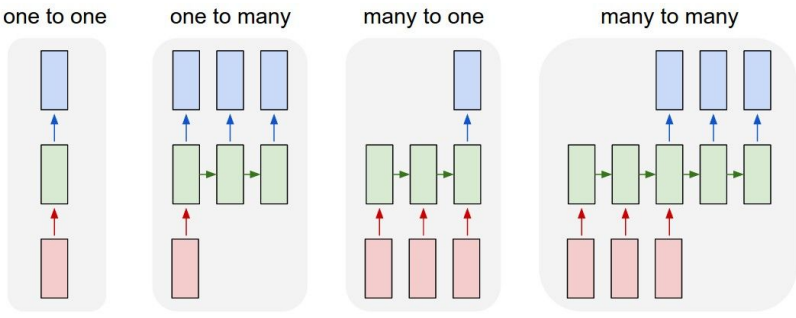
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



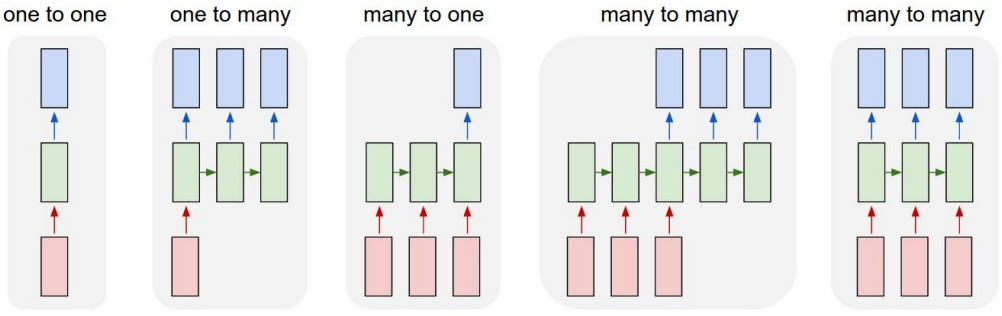
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences

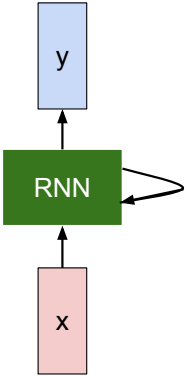


Why do we need RNNs?

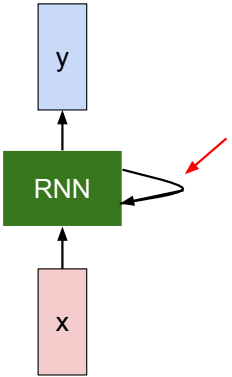
Example task: Video Captioning

- Input length is variable
- Output length is also variable
- Forcing input and output lengths might introduce issues

RNN

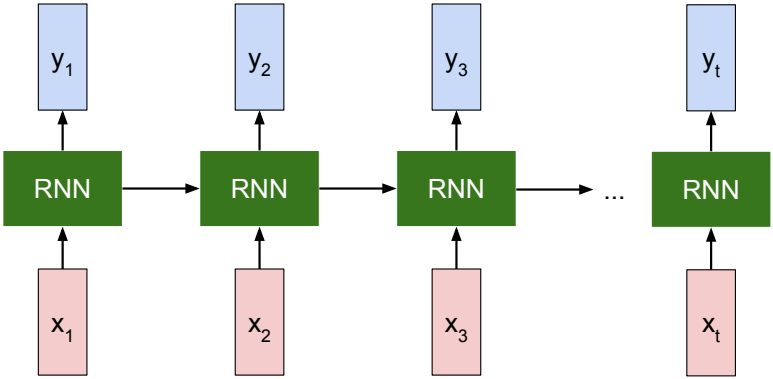


RNN



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

RNN



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

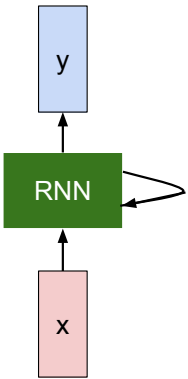
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



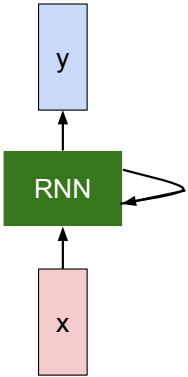
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

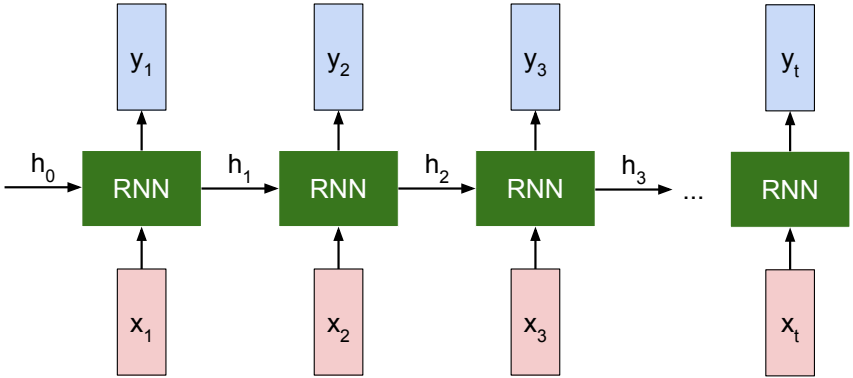
output

new state

another function with parameters W_o

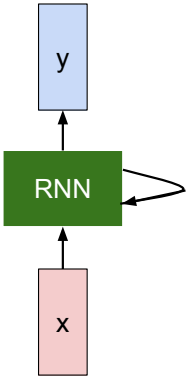


RNN



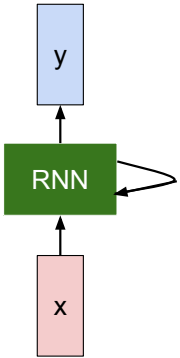
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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



Notice: the same function and the same set of parameters are used at every time step.

RNN



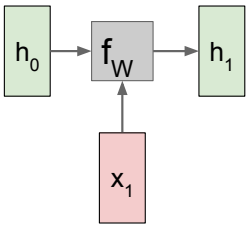
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

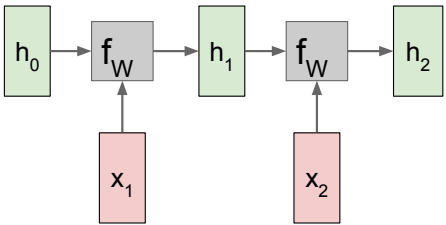
$$y_t = W_{hy}h_t$$

Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman

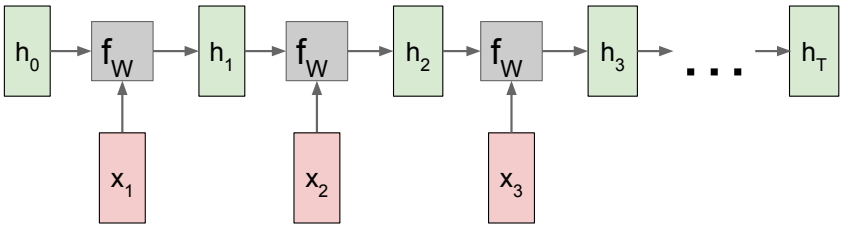
RNN: Computational Graph



RNN: Computational Graph

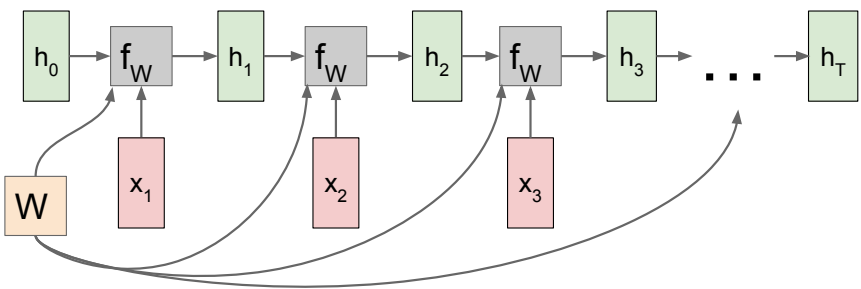


RNN: Computational Graph

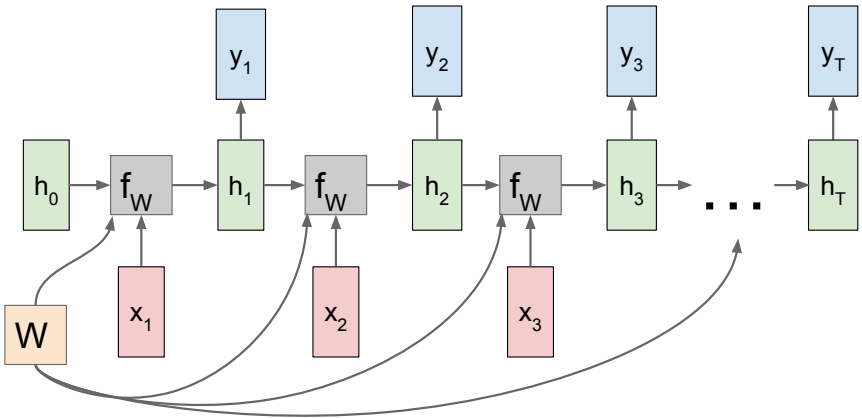


RNN: Computational Graph

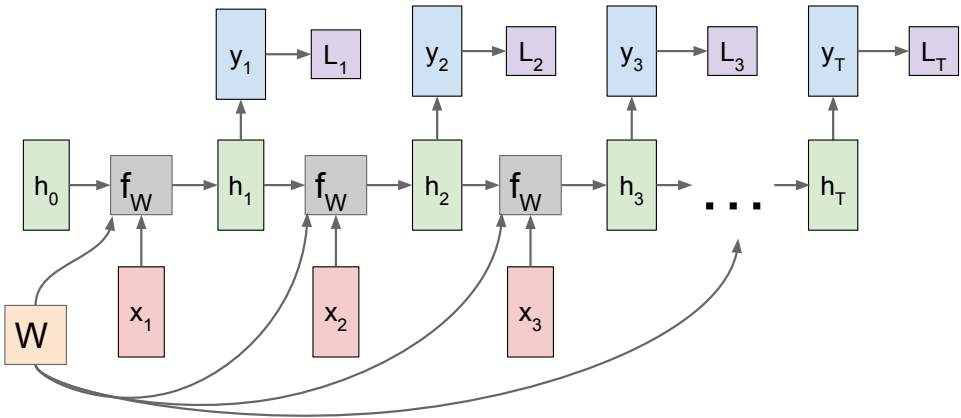
Re-use the same weight matrix at every time-step



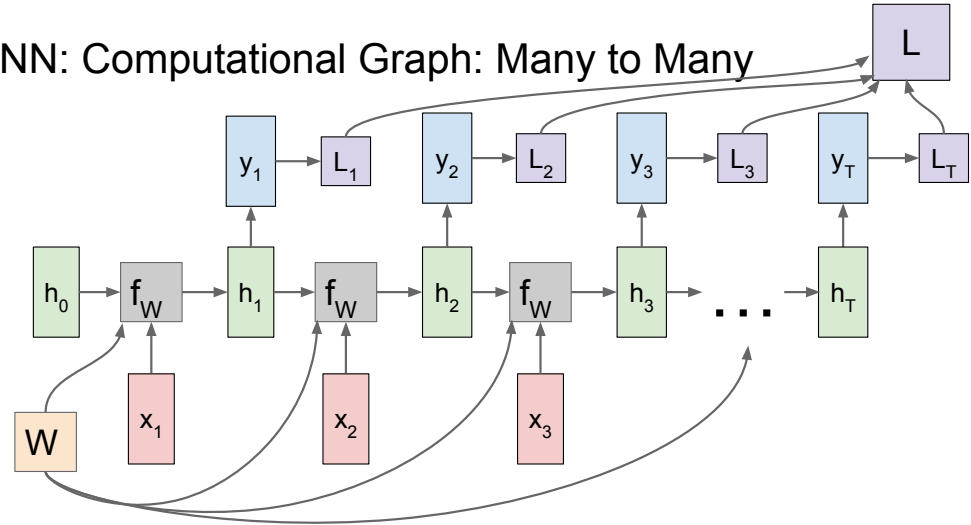
RNN: Computational Graph: Many to Many



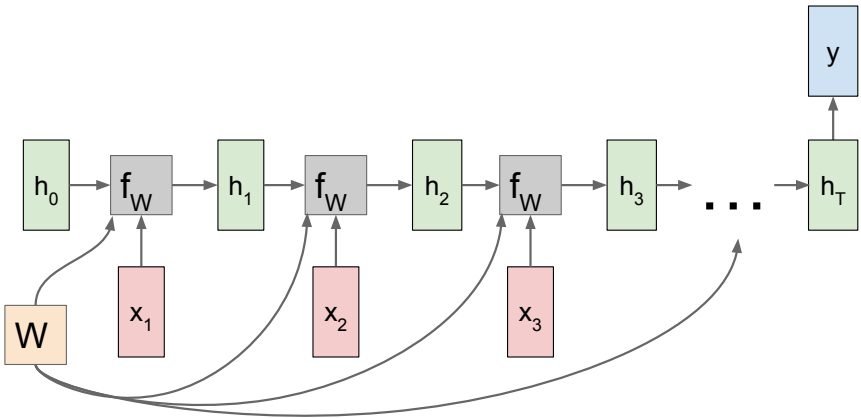
RNN: Computational Graph: Many to Many



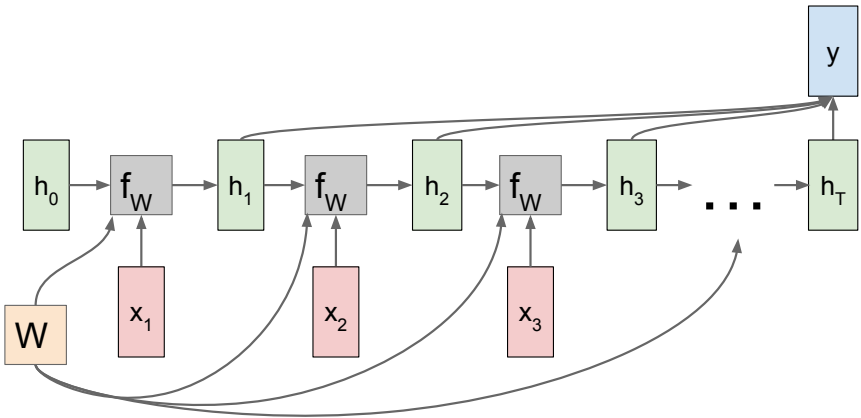
RNN: Computational Graph: Many to Many



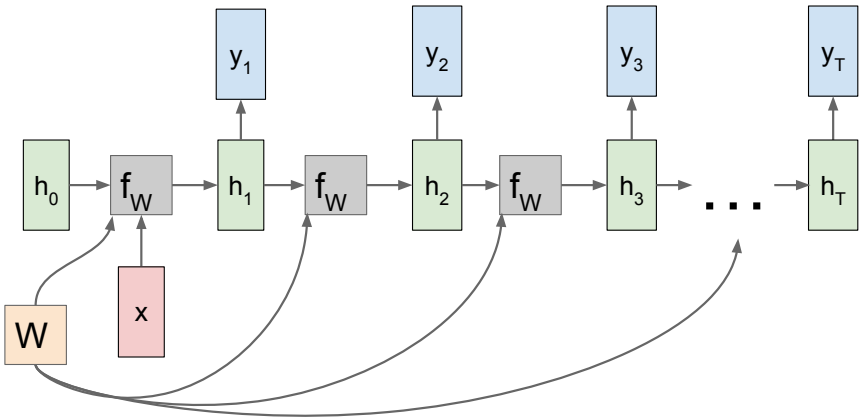
RNN: Computational Graph: Many to One



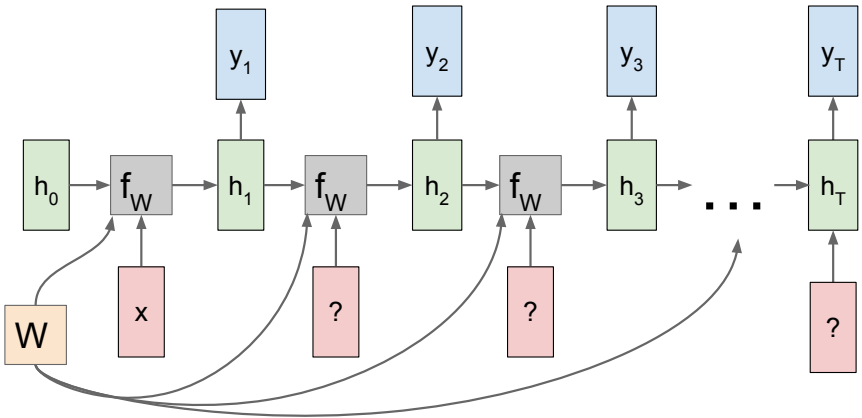
RNN: Computational Graph: Many to One



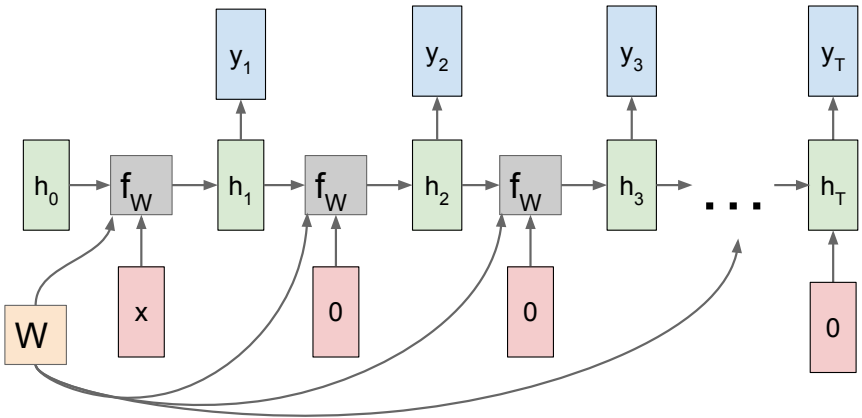
RNN: Computational Graph: One to Many



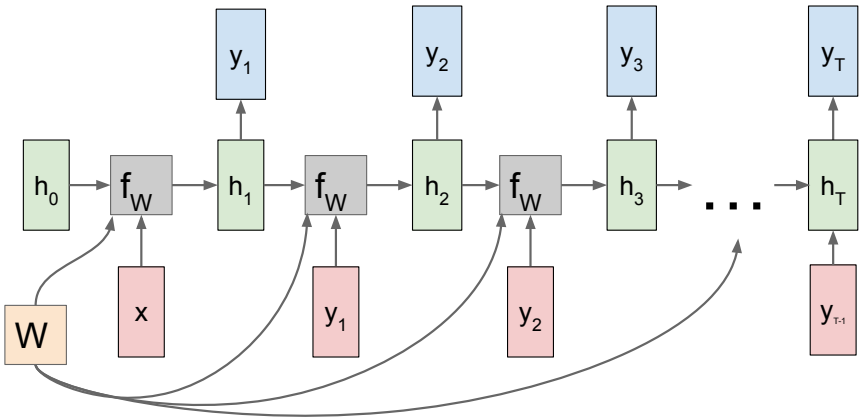
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

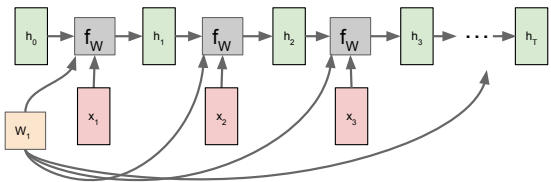


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

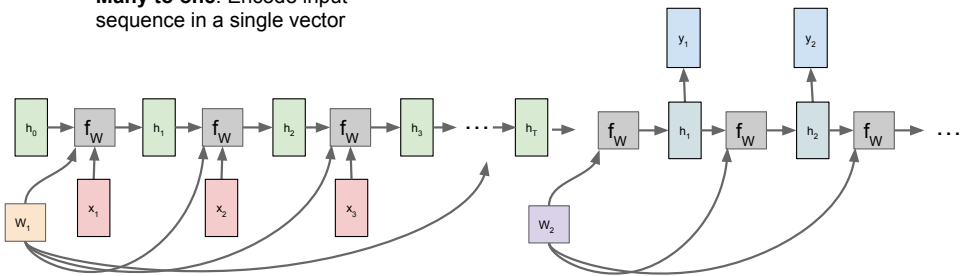


Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



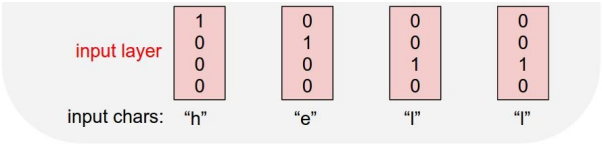
Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Simple Example

Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

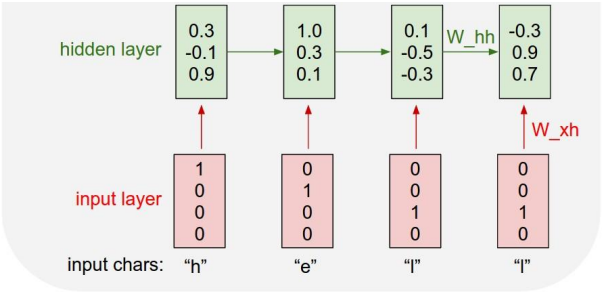


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

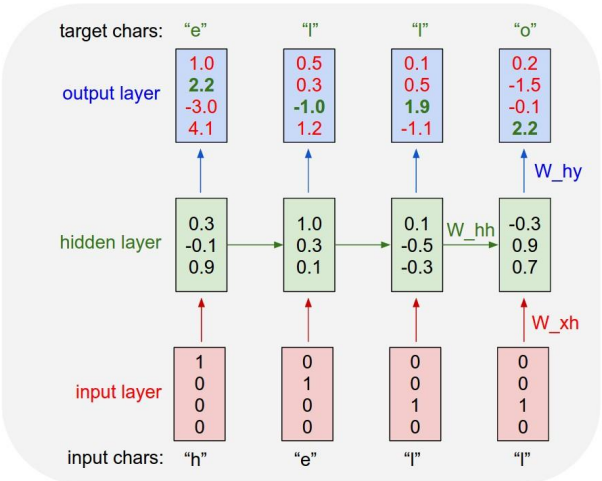


Simple Example

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

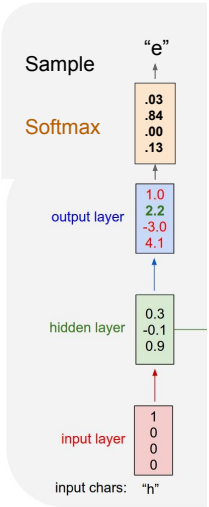


Simple Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

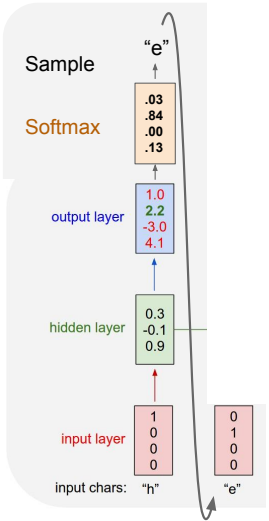


Simple Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

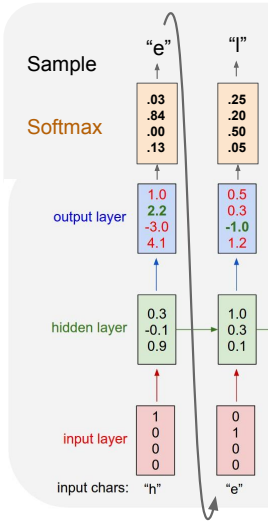


Simple Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

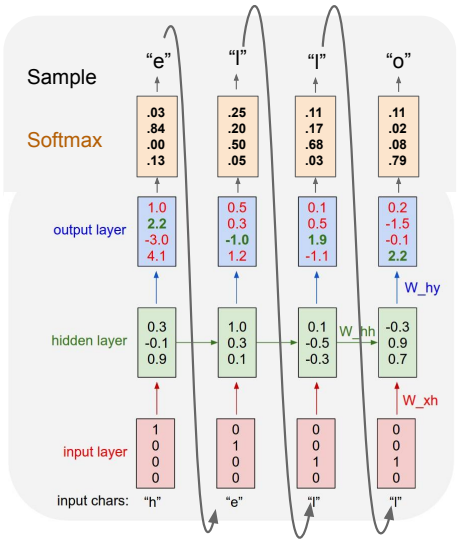


Simple Example

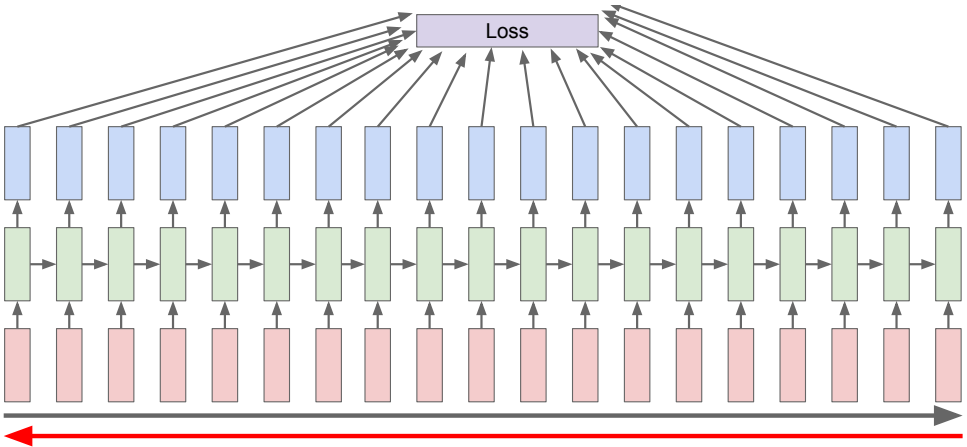
Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

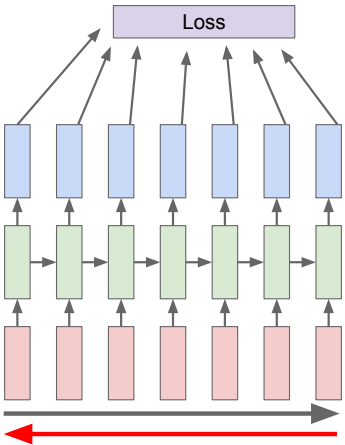
At test-time sample characters one at a time, feed back to model



How to backpropagate

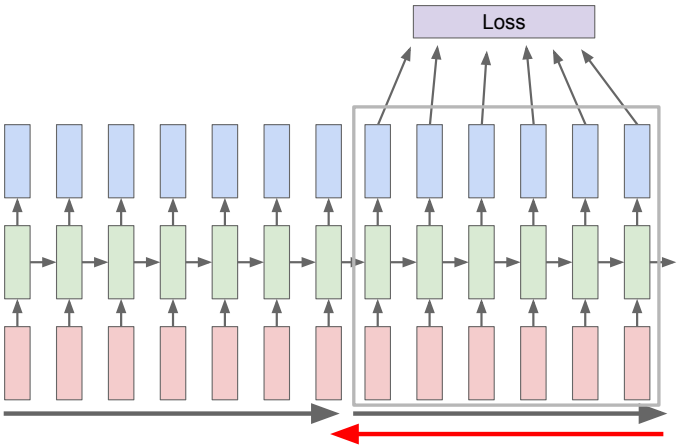


How to backpropagate



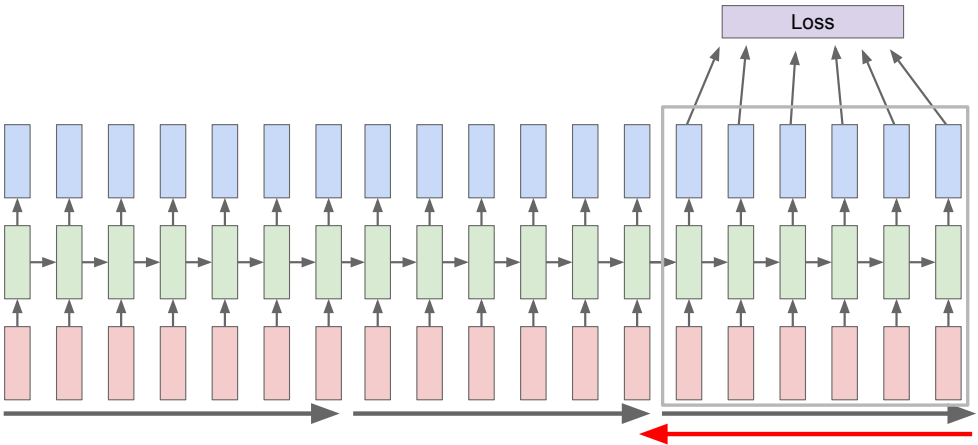
Run forward and backward through chunks of the sequence instead of whole sequence

How to backpropagate



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

How to backpropagate



RNNs pros and cons

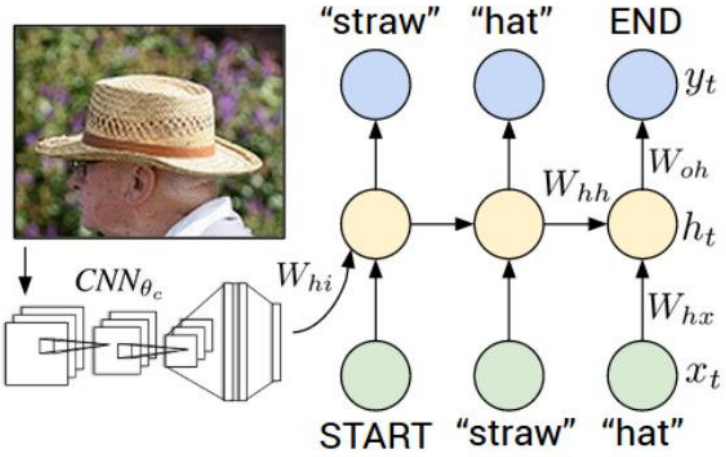
Pros:

- Variable input/output
- Shared weights for steps
- Theoretically can access information in any previous step
- Constant model size for various inputs

Cons:

- Backprop may require too much memory
- Potentially unstable training
- Information from previous steps might be difficult to access
- Slow computation

Image Captioning



Andrej Karpathy and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

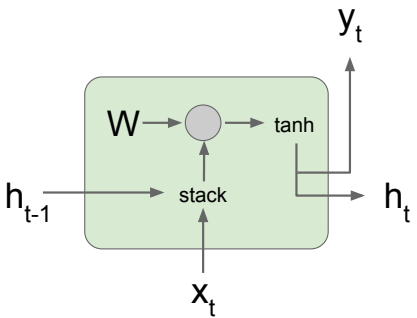
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

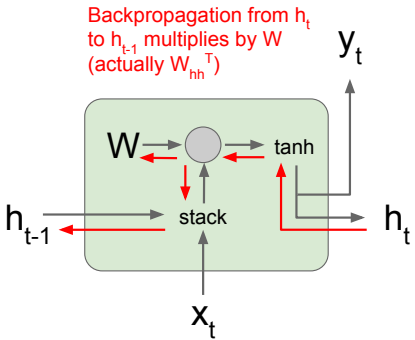
Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In: *Neural computation* 9.8 (1997), pp. 1735–1780

RNN Backprop



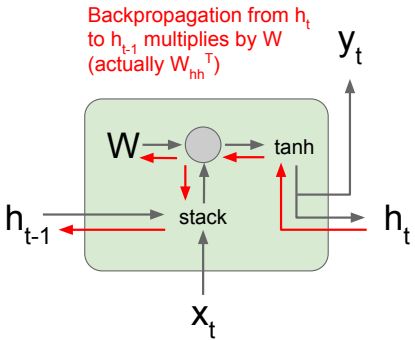
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

RNN Backprop



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

RNN Backprop

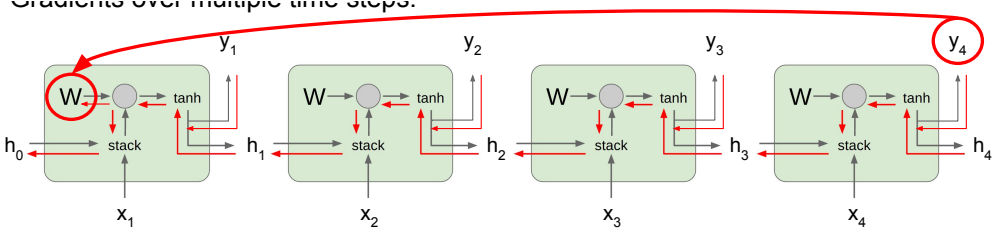


$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

$$
\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}
$$

LSTM

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

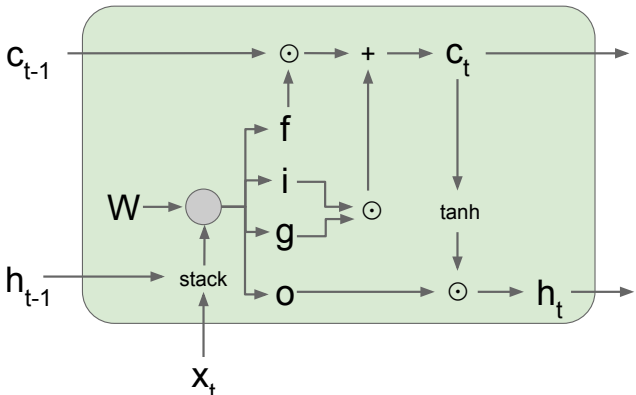
Almost always < 1
Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} tanh'\left(W_{hh} h_{t-1} + W_{xh} x_t\right)\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In: *Neural computation* 9.8 (1997), pp. 1735–1780

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



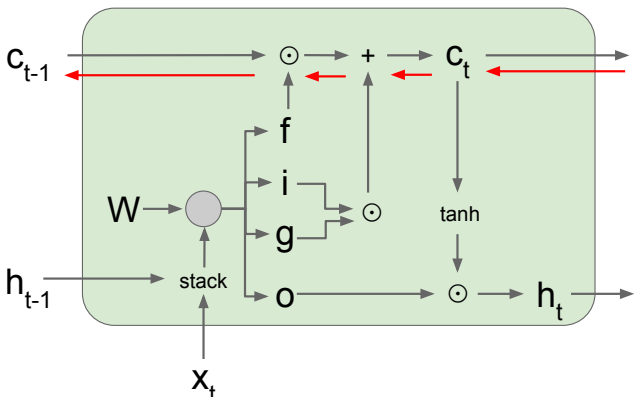
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

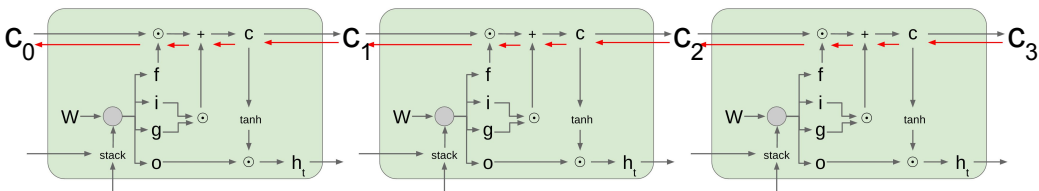
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Notice that the gradient contains the **f** gate's vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the **f, i, g,** and **o** gates

- better balancing of gradient values

Do LSTMs solve the vanishing gradient problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the f = 1 and the i = 0**, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state

LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

GRU

Gated Recurrent Unit[4]:

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

GRU is like LSTM with a forget gate, but no output gate. It has fewer parameters and is usually easier to train!

[4] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *arXiv preprint arXiv:1406.1078* (2014)

Using RNNs

It is probably a better idea to use Transformers nowadays, but if you choose RNNs there are some tips based on my own limited experience:

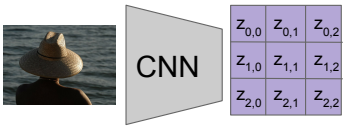
- GRU is easier to train than LSTM.
- Selecting the proper training regime is mostly based on the amount of memory available for training.
- Keeping hidden states during training might introduce problems.
- ELU or PReLU may be better activations than ReLU.
- Nowadays you should probably use a Transformer instead.

Transformers

- 2014-2016 - RNNs were dominant technique to use on sequential data in both NLP and CV.
- 2015-2016 - ResNets and Attention show that there are better ways to solve some issues than LSTM
- 2017 - Transformers dominate in NLP
- 2018-2020 - Transformers get applied to CV with varying levels of success
- 2021 - Transformers start beating CNNs in visual tasks

Attention in CV

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$



Extract spatial features from a pretrained CNN

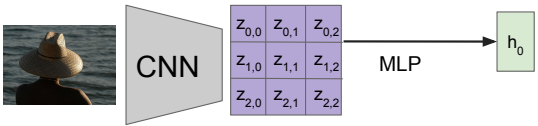
Features: H x W x D

Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Attention in CV

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_W(\mathbf{z})$
where **z** is spatial CNN features
$f_W(.)$ is an MLP



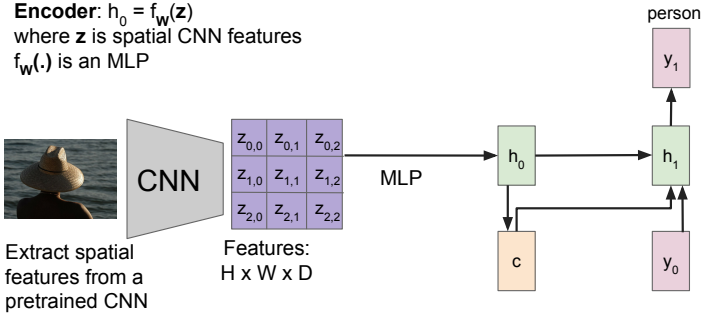
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP



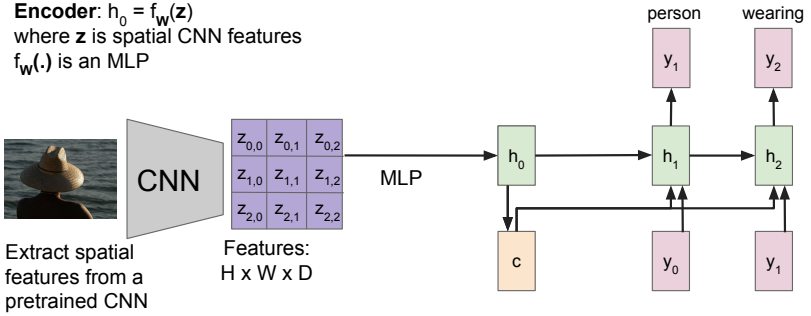
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP



Extract spatial features from a pretrained CNN

Features: H x W x D

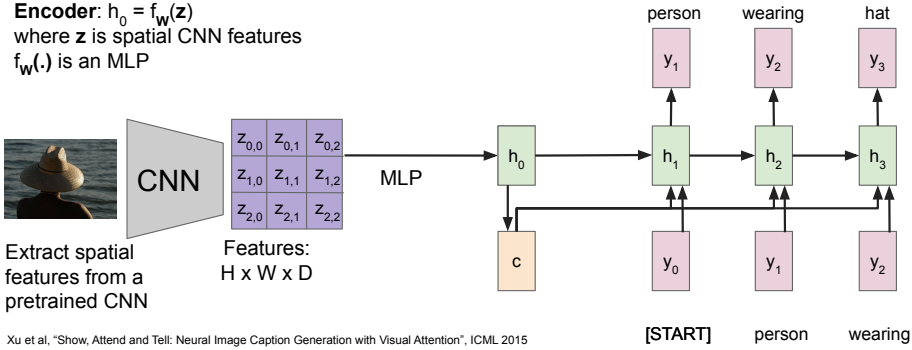
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



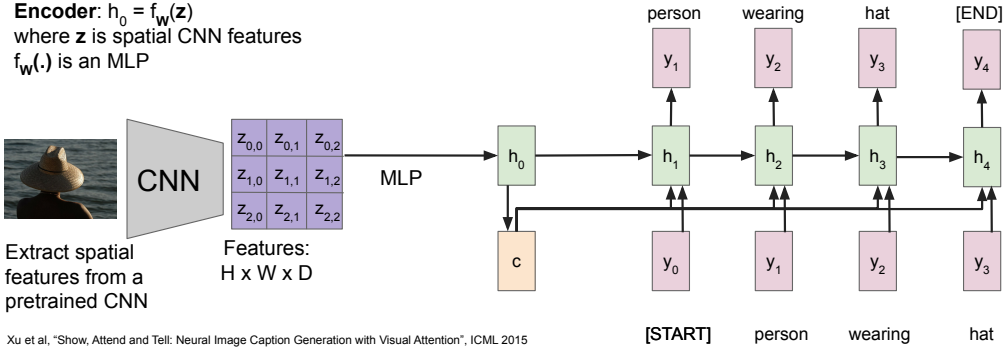
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



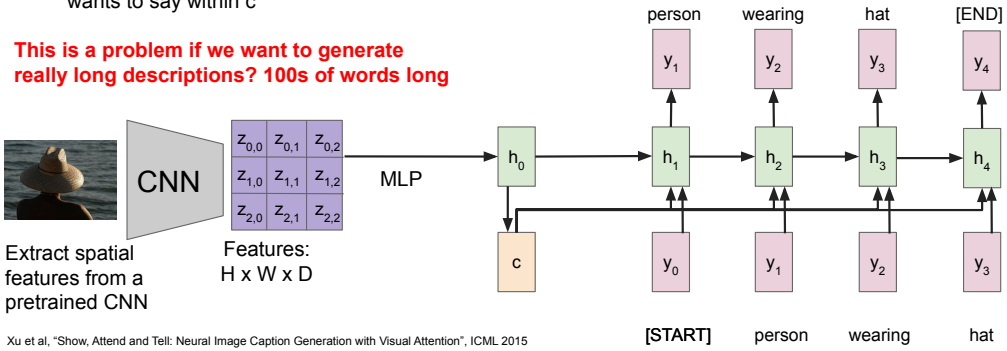
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Problem: Input is "bottlenecked" through c

- Model needs to encode everything it wants to say within c

This is a problem if we want to generate really long descriptions? 100s of words long

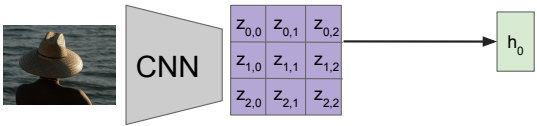


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

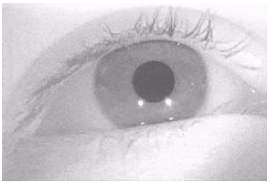
Attention idea: New context vector at every time step.

Each context vector will attend to different image regions



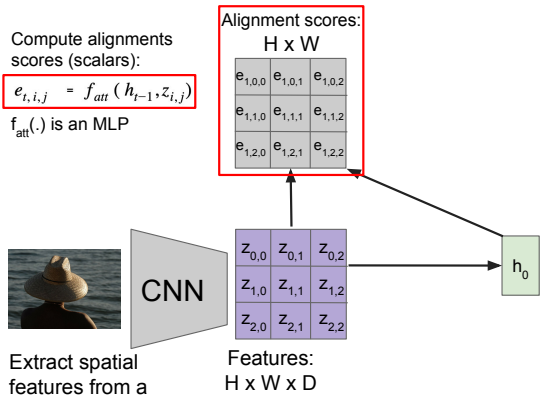
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

gif source



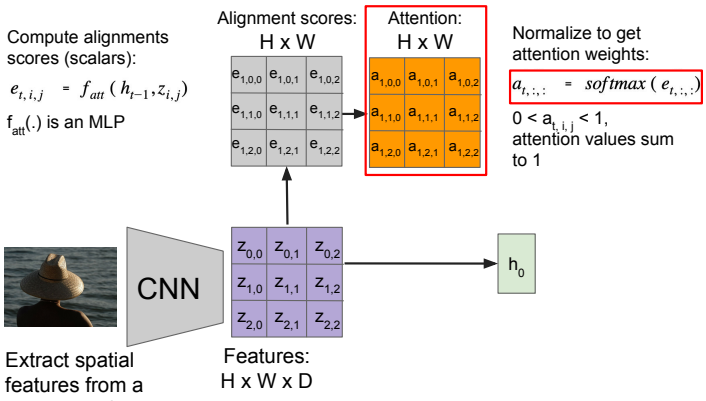
Attention Saccades in humans

Attention in CV



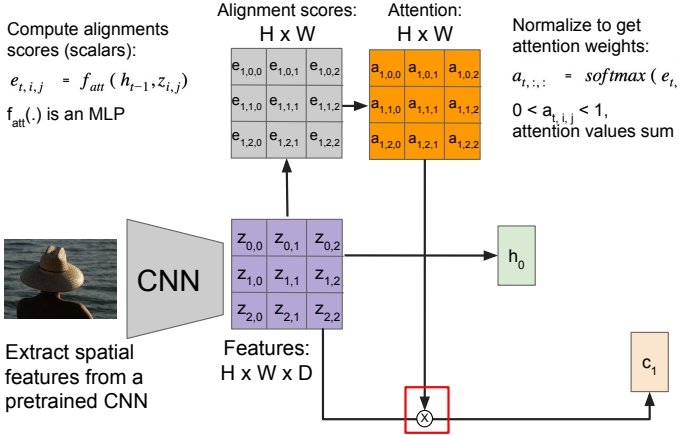
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Attention in CV



Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Attention in CV



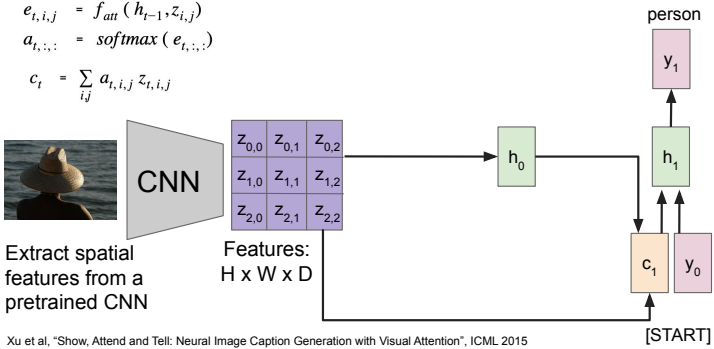
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Attention in CV

Each timestep of decoder uses a different context vector that looks at different parts of the input image

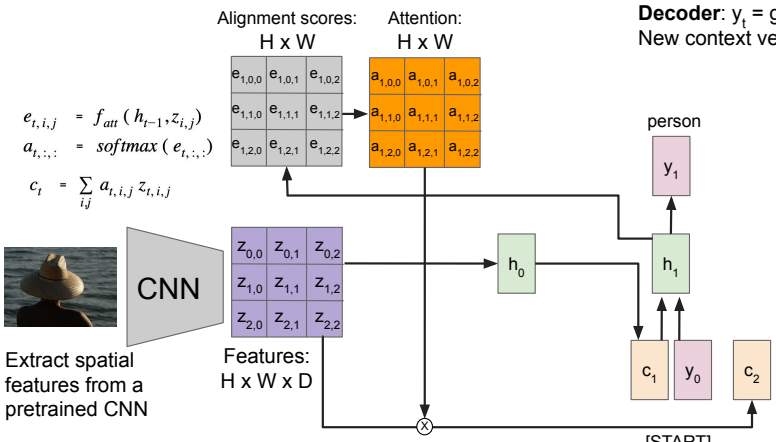
Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

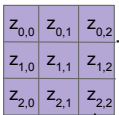
Each timestep of decoder uses a different context vector that looks at different parts of the input image

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

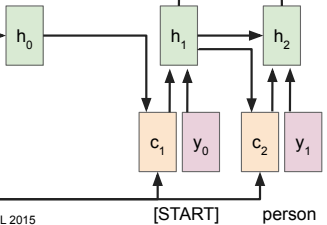
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

CNN

Extract spatial features from a pretrained CNN



Features: H x W x D



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Each timestep of decoder uses a different context vector that looks at different parts of the input image

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

Extract spatial features from a pretrained CNN

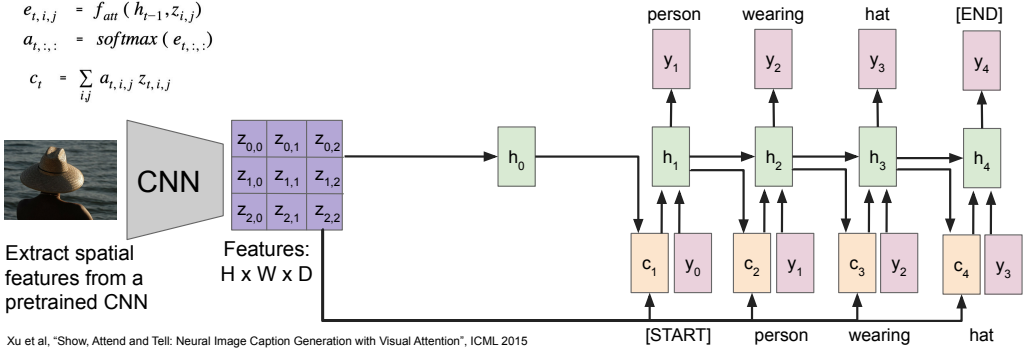
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

Each timestep of decoder uses a different context vector that looks at different parts of the input image

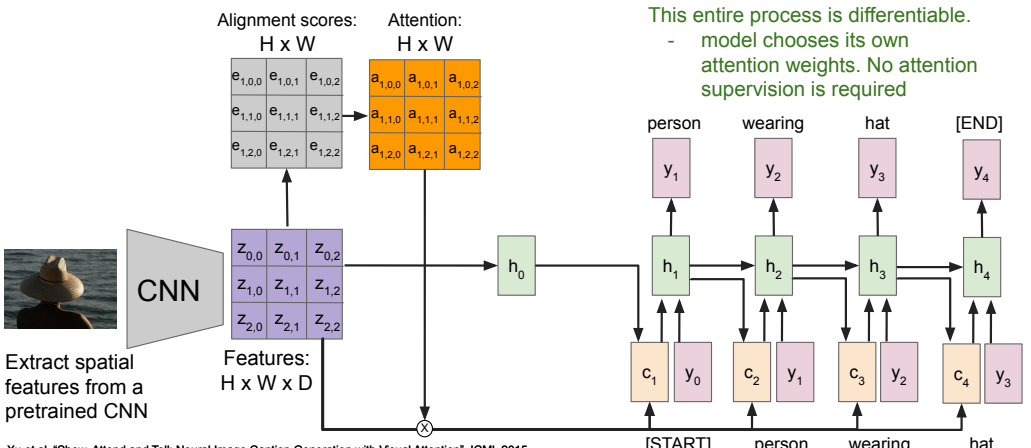
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Transformers

Transformers are based on this attention mechanism. To see how attention works we will check out the amazing illustrations by Jay Alammar at `https://jalammar.github.io/illustrated-transformer/`

Transformers in NLP

Transformers have dominated NLP in recent years:

- Original Transformer Paper - Attention is all you need[5]
- Generative Pre-trained Transformer (GPT)[6]
- BERT[7]

[5] Ashish Vaswani et al. “Attention is all you need.” In: *Advances in neural information processing systems*. 2017, pp. 5998–6008

[6] Alec Radford et al. “Improving language understanding by generative pre-training.” In: (2018)

[7] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding.” In: *arXiv preprint arXiv:1810.04805* (2018)

Transformers in CV

There have been multiple attempts in recent years to bring transformers to CV. We will discuss some of the most influential papers.

- DETR - object detection with transformers[8]
- Visual Transformer (ViT)[9] and DEiT[10]
- Pyramid Vision Transformer[11]
- Swin Transformer[12]

[8] Nicolas Carion et al. "End-to-end object detection with transformers." In: *European Conference on Computer Vision*. Springer. 2020, pp. 213–229

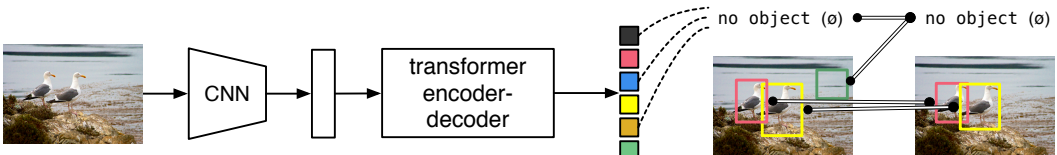
[9] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020)

[10] Hugo Touvron et al. "Training data-efficient image transformers & distillation through attention." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10347–10357

[11] Wenhai Wang et al. "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions." In: *arXiv preprint arXiv:2102.12122* (2021)

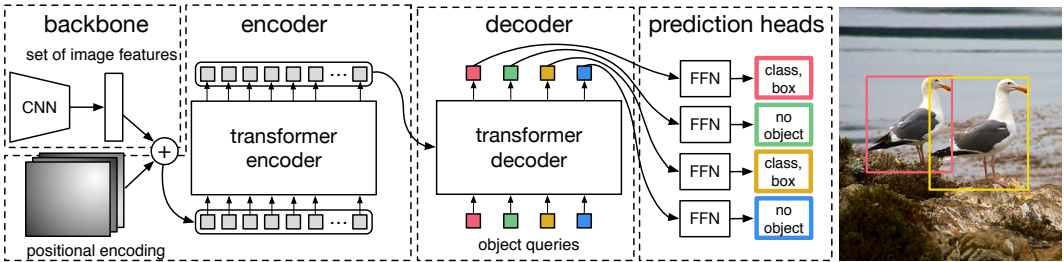
[12] Ze Liu et al. "Swin transformer: Hierarchical vision transformer using shifted windows." In: *arXiv preprint arXiv:2103.14030* (2021)

DETR



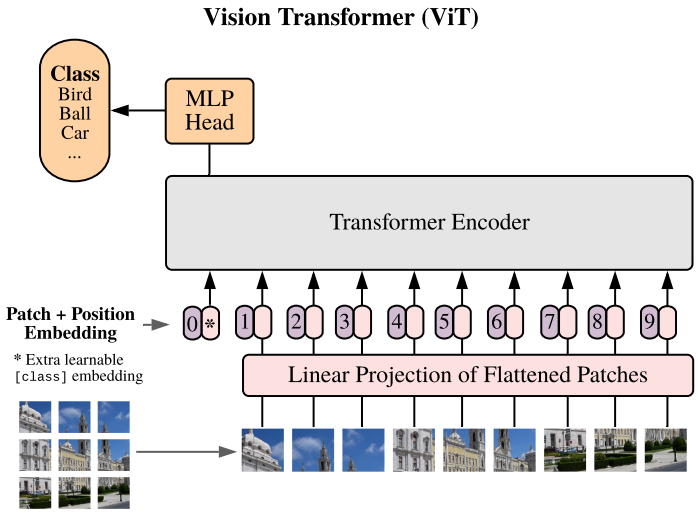
DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a special no-object class prediction.

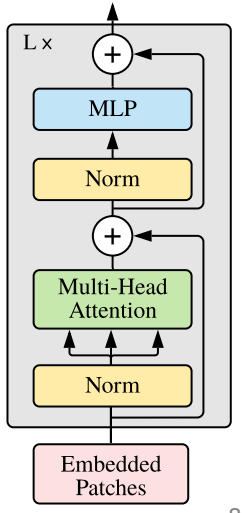
DETR - Architecture



This approach led to competitive results, but authors point out that the training process is tricky. This is an interesting application of transformers, but it still relies on a CNN to extract features.

ViT Model


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches


Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

ViT Model

$$\boldsymbol{z}_0 = [\boldsymbol{x}_{\text{class}};\, \boldsymbol{x}_p^1\boldsymbol{E};\, \boldsymbol{x}_p^2\boldsymbol{E};\, \cdots;\, \boldsymbol{x}_p^N\boldsymbol{E}] + \boldsymbol{E}_{pos}, \tag{1}$$

$$\boldsymbol{z}'_\ell = \text{MSA}(\text{LN}(\boldsymbol{z}_{\ell-1})) + \boldsymbol{z}_{\ell-1}, \qquad \ell = 1 \ldots L \tag{2}$$

$$\boldsymbol{z}_\ell = \text{MLP}(\text{LN}(\boldsymbol{z}'_\ell)) + \boldsymbol{z}'_\ell, \qquad \ell = 1 \ldots L \tag{3}$$

$$\boldsymbol{y} = \text{LN}(\boldsymbol{z}_L^0) \tag{4}$$

$$\boldsymbol{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \boldsymbol{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

ViT - Limitations

There were several issues of ViT which made them very difficult to use:

- Pre-training on huge datasets: ImageNet-21k, JFT-300 (internal Google dataset)
- Training requires a lot of resources
- Results are only reproducible by huge corporations/labs

These are not such issues if a model with pre-trained weights works for your use case.

DEiT

DEiT paper introduced several tricks to train ViT on a reasonable scale. However the results were not that much better than CNNs.

- Self-distillation using a CNN teacher
- CNN features also input into transformer part
- Heavy augmentation + regularization

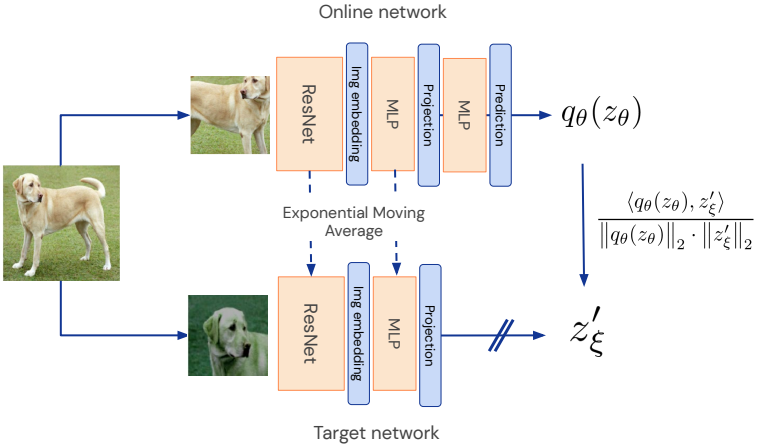
This still requires a pre-trained CNN!

Self-supervised Learning

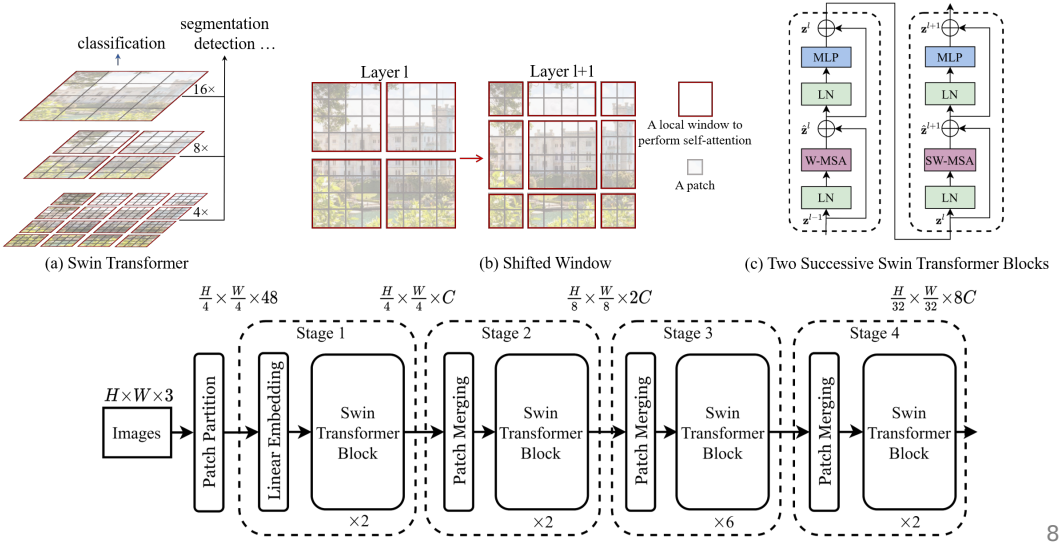
As we saw, visual transformers require large amounts of data. One strategy is to use data without annotations! This is called self-supervised learning.

Several approaches have been quite sucessful in obtaining good features even without labels such as BYOL or DINO.

BYOL - Bootstrap your own latent

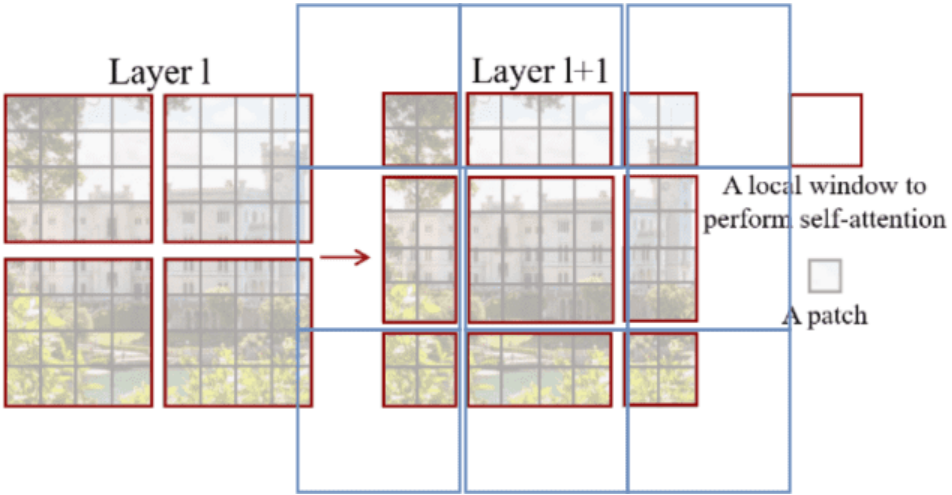


Swin Transformers

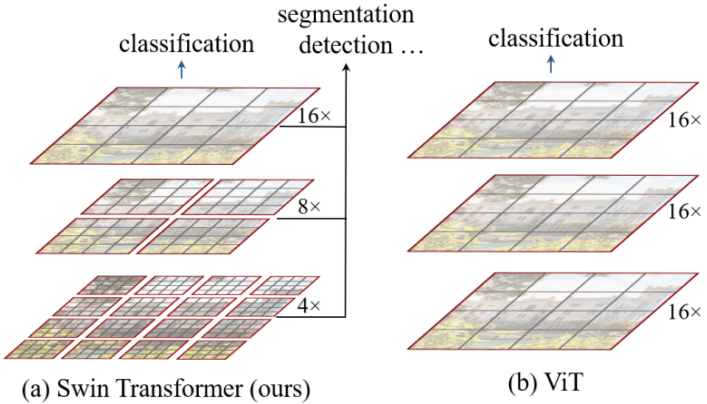


(a) Swin Transformer (b) Shifted Window (c) Two Successive Swin Transformer Blocks

Shifted Window Based Self-Attention



Swin vs ViT Hierarchy



Swin introduces hierarchy to computation. The authors also propose to mirror the spatial resolution hierarchies of popular CNNs (e.g. ResNets) so the swin transformers can easily replace other meta-architectures with CNN backbones.

Multimodal Learning

So far we have shown that it is possible to combine natural language with visual input during training of a network. How far can we go with such approach?

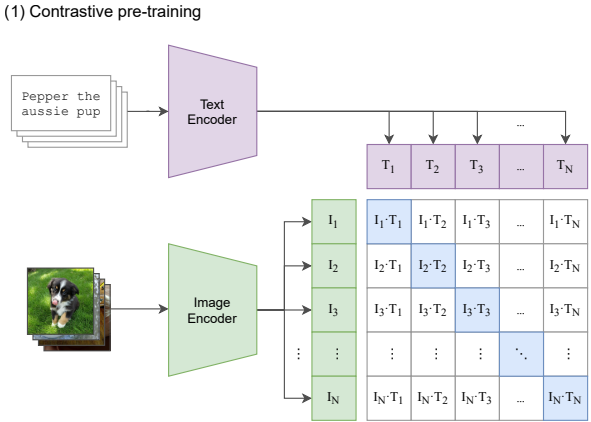
Multimodal Learning

So far we have shown that it is possible to combine natural language with visual input during training of a network. How far can we go with such approach?
A 2021 paper CLIP (Contrastive Language-Image Pre-training) shows how to combine natural language descriptions with images for training. We have two separate networks:

- Image encoder - inputs are images
- Text encoder - inputs is natural language text

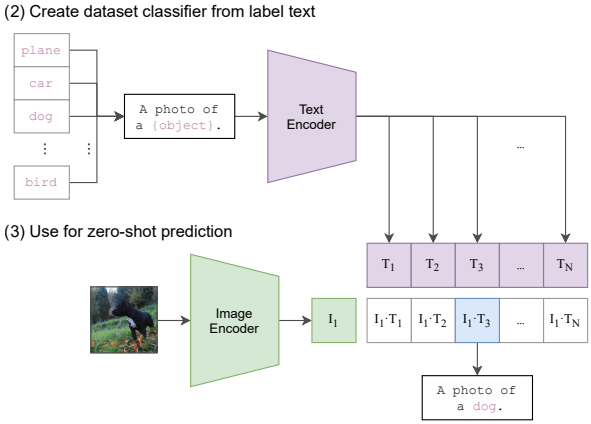
Both of these networks output a vector embedding in a shared multi-modal space. The vectors should be close whenever the text matches the image and far when they do not match.

CLIP Training



Training target is to maximize cosine similarity of N actual pairs while minimizing the similarity $N^2 - N$ incorrect pairings.

CLIP Training



We can use the learned embeddings for zero-shot classification and other tasks!