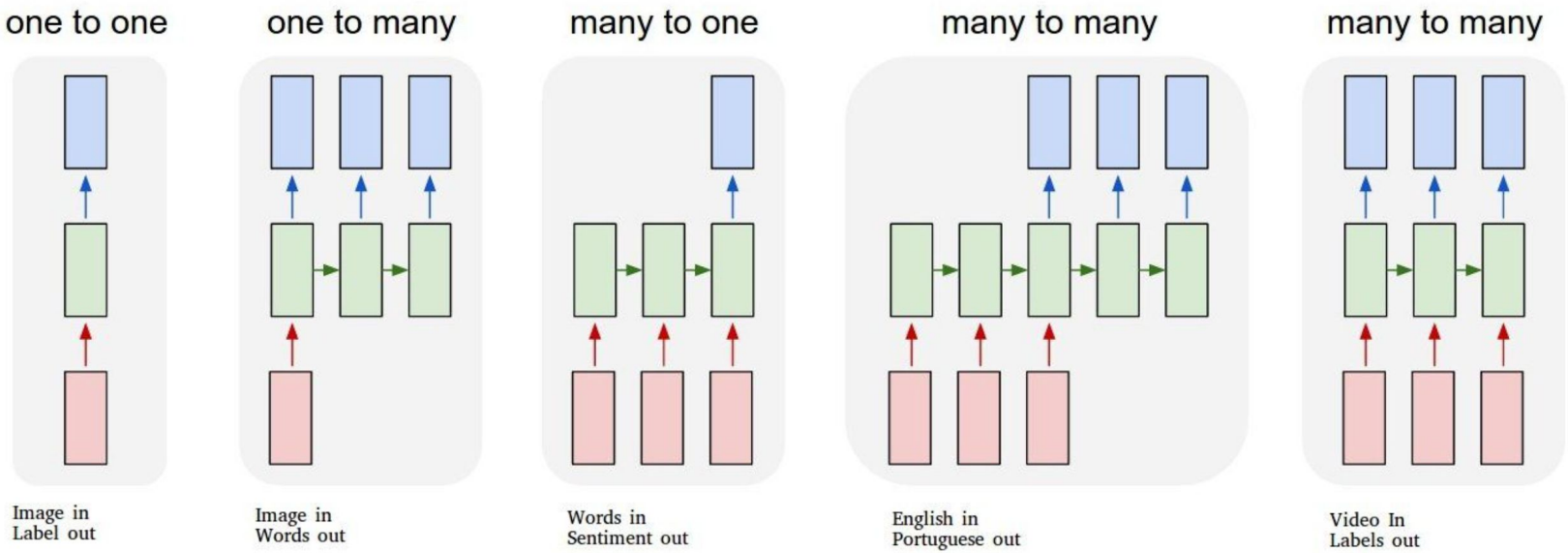
Компьютерное зрение

Лекция 11.
Image captioning. Attention mechanisms.

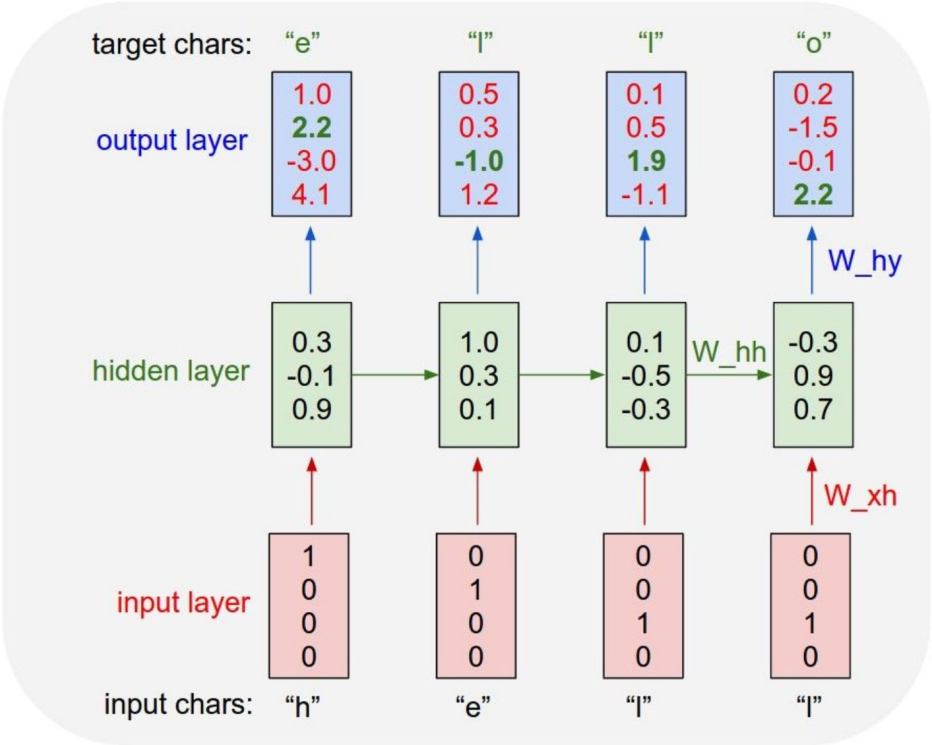
04.06.2020

Руслан Рахимов

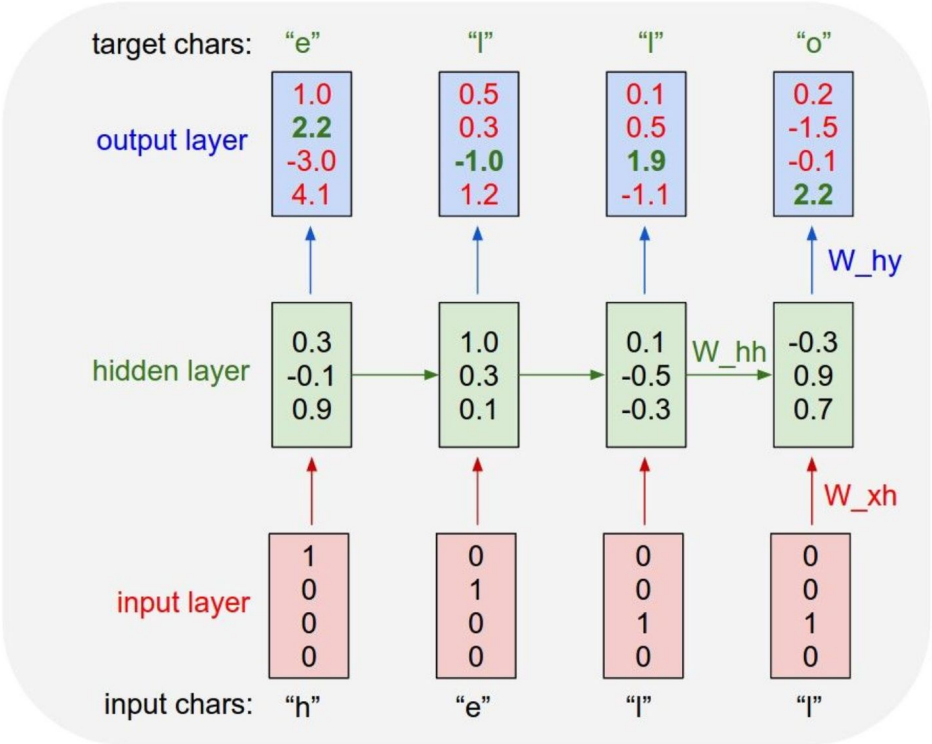
Рекуррентные сети



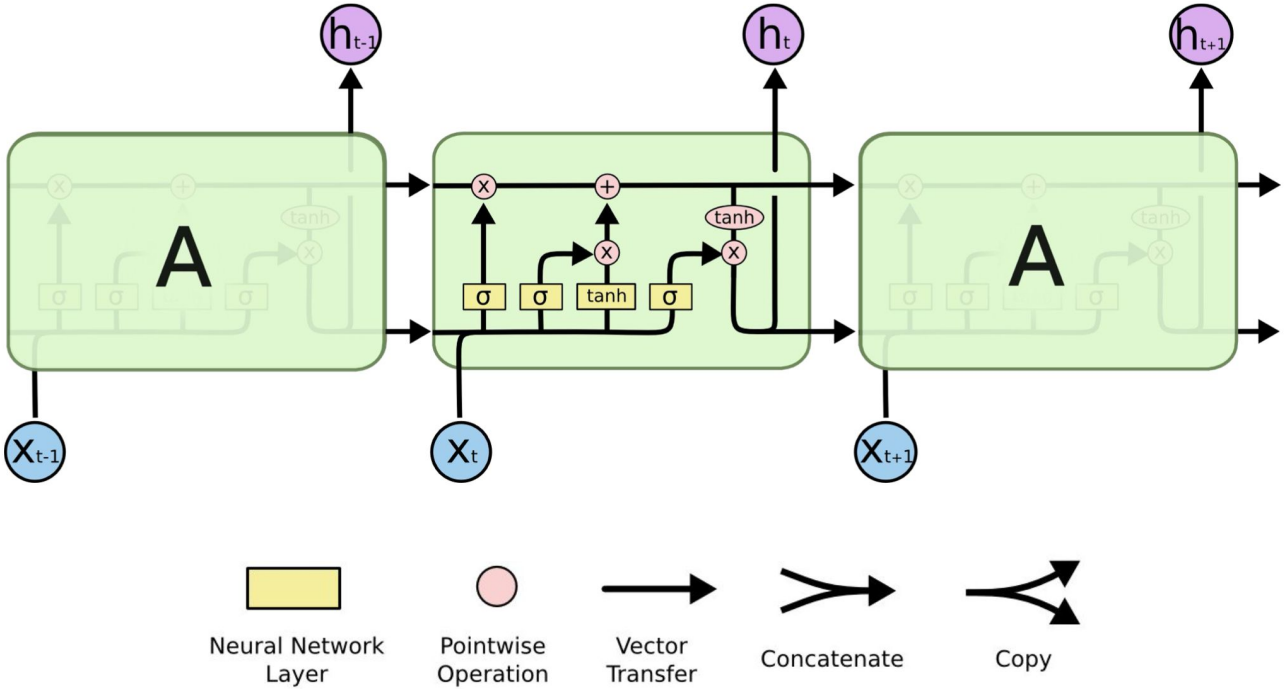
Генерация текста



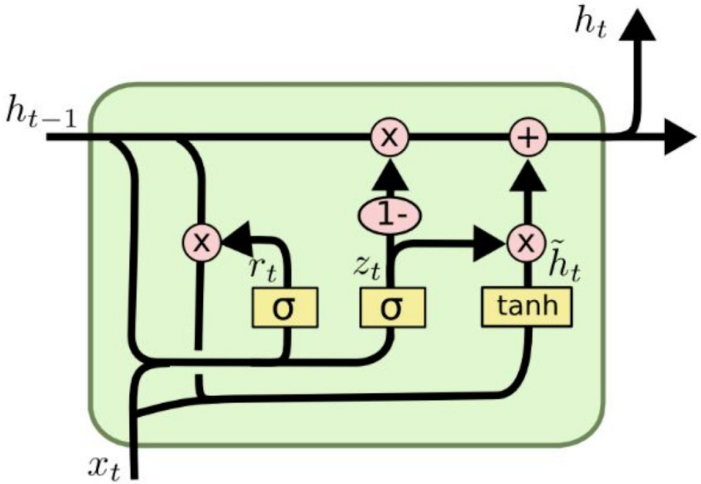
LSTM (Long Short Term Memory)



LSTM (Long Short Term Memory)



GRU (Gated Recurrent Unit)



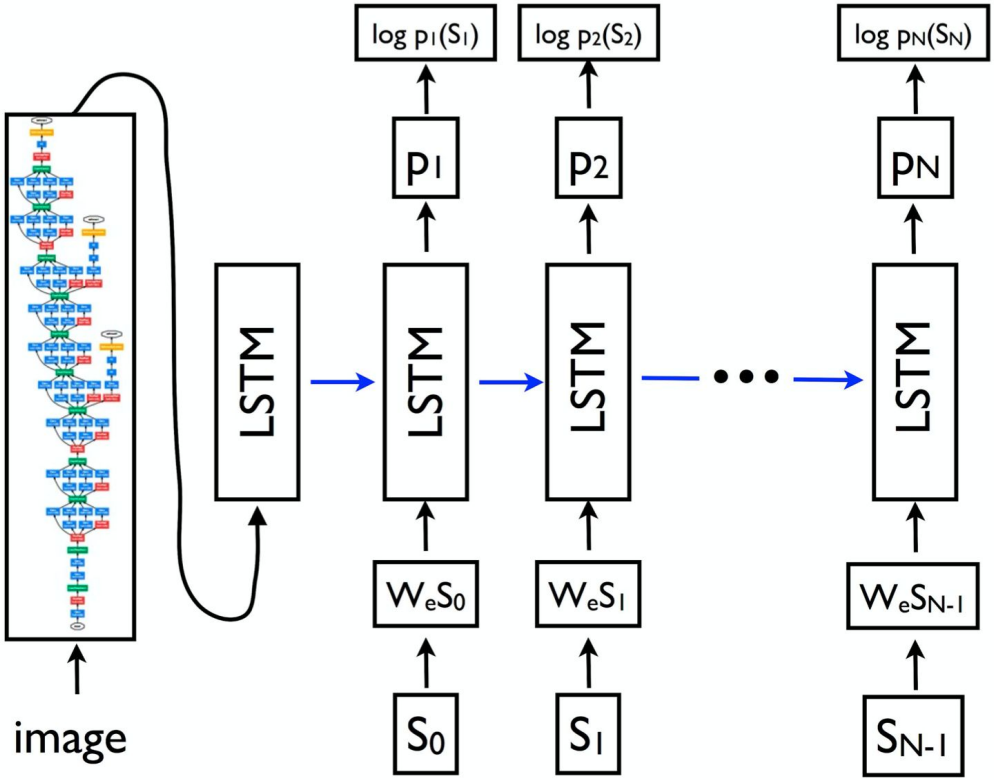
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

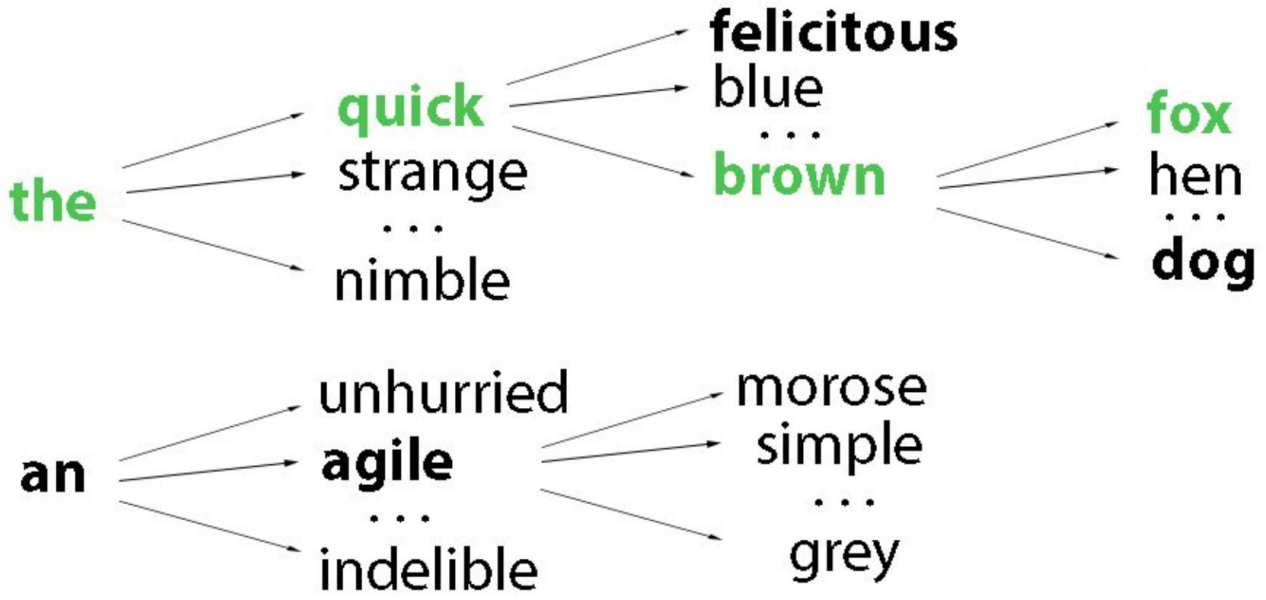
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

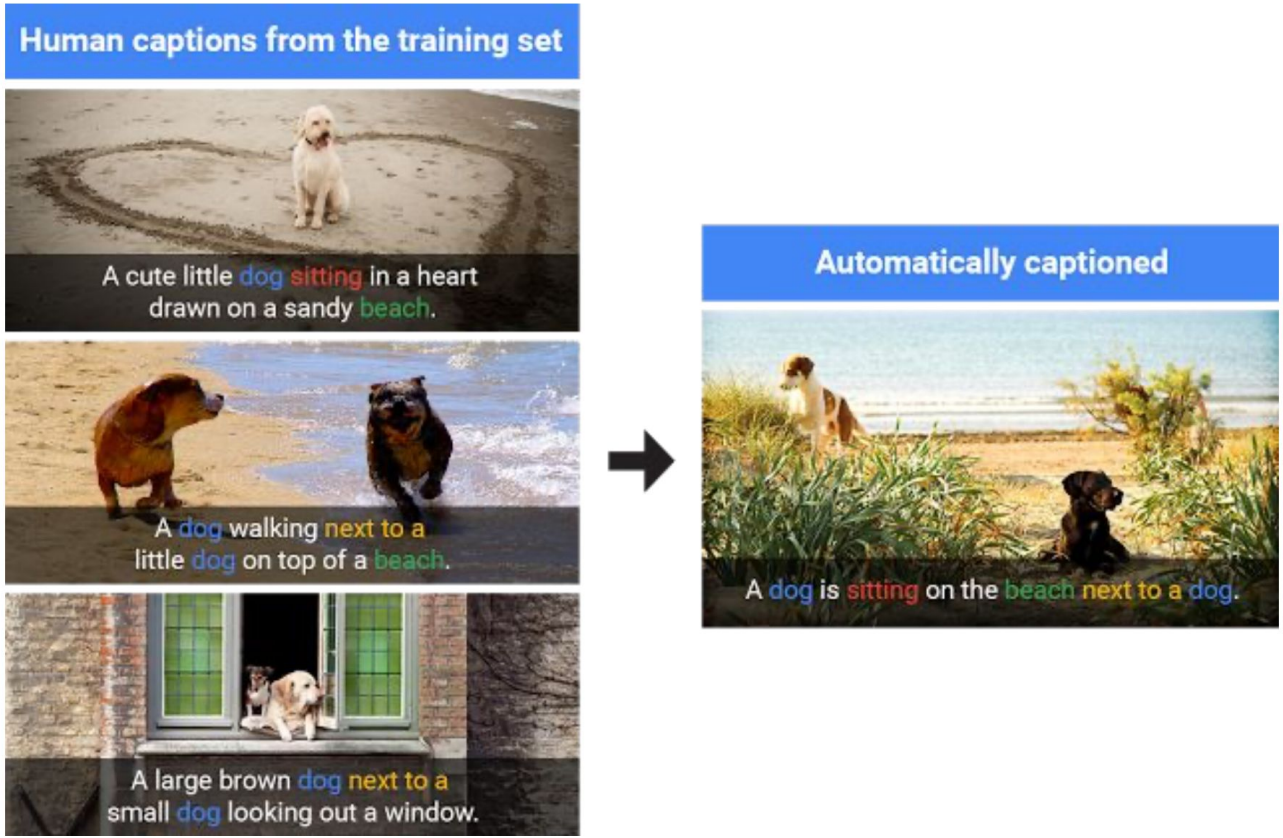
Im2text


log p1(S1)
log p2(S2)
log pN(SN)
p1
p2
pN
LSTM
LSTM
LSTM
LSTM
WeS0
WeS1
WeSN-1
S0
S1
SN-1
image

BeamSearch


the
quick
strange
...
nimble
felicitous
blue
...
brown
fox
hen
...
dog
an
unhurried
agile
...
indelible
morose
simple
...
grey

Im2text


Human captions from the training set
A cute little dog sitting in a heart drawn on a sandy beach.
A dog walking next to a little dog on top of a beach.
A large brown dog next to a small dog looking out a window.
Automatically captioned
A dog is sitting on the beach next to a dog.

В чем проблема LSTM / GRU?

Attention

Implicit Attention in Neural Networks

Deep nets naturally learn a form of **implicit attention** where they respond more strongly to some parts of the data than others

To a first approximation, we can visualise this by looking at the network **Jacobian** — sensitivity of the network outputs with respect to the inputs

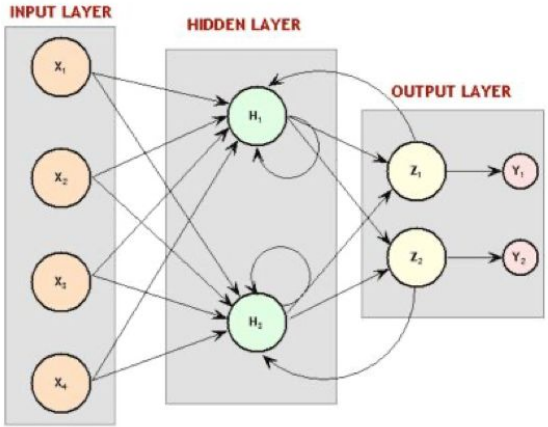
x = size k input vector
y = size m output vector
Jacobian J = m x k matrix

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_k} \end{bmatrix}$$

$$J_{ij} = \frac{\partial y_i}{\partial x_j}$$

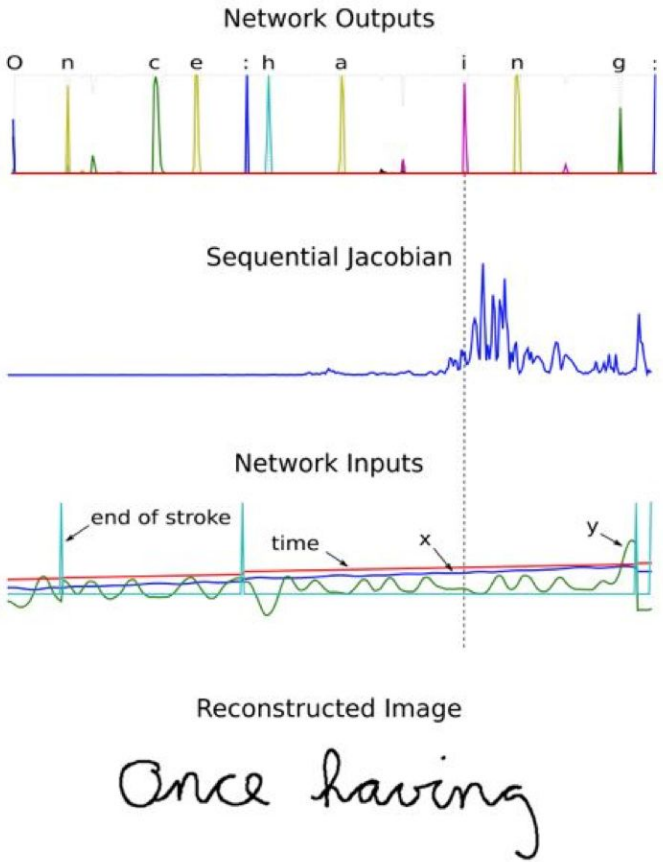
Can compute with ordinary backdrop
(just set output ‘errors’ = output activations)

Attention and memory in Recurrent Networks (RNNs)



"Hello World"

RNNs contain a recursive hidden state and learn functions from sequences of inputs (e.g. a speech signal) to sequences of outputs (e.g. words)

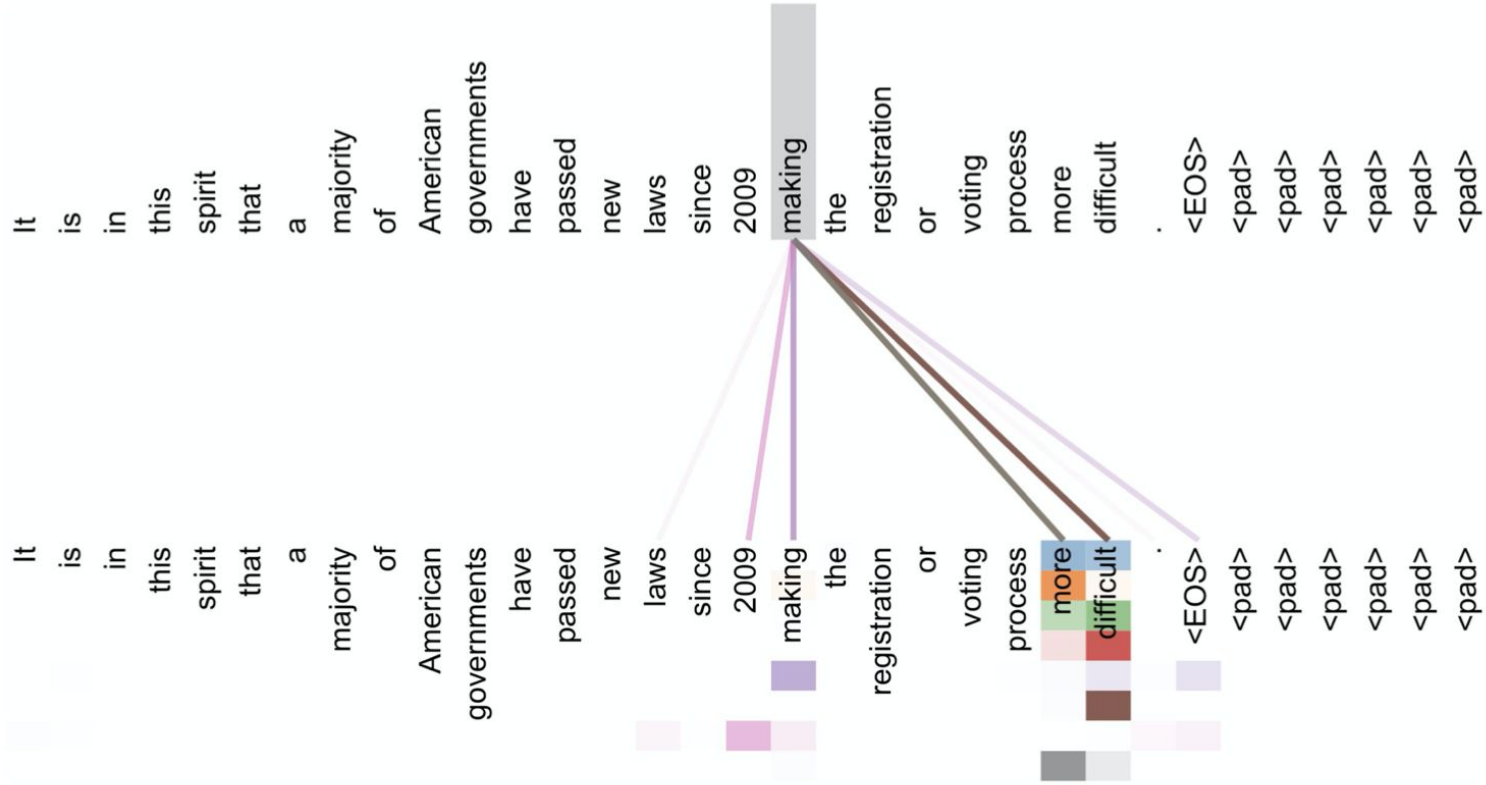


- The Sequential Jacobian is the set of derivatives of one network output with respect to all the inputs

$$J_k^t = \left(\frac{\partial y_k^t}{\partial \mathbf{x}^1}, \frac{\partial y_k^t}{\partial \mathbf{x}^2} \cdots \right)$$

- It shows how the network responds to widely separated, but related, inputs, such as the delayed dot of the 'i' in 'having'

Self-attention



Self-attention

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

Attention is all you need (Vaswani, 2017)

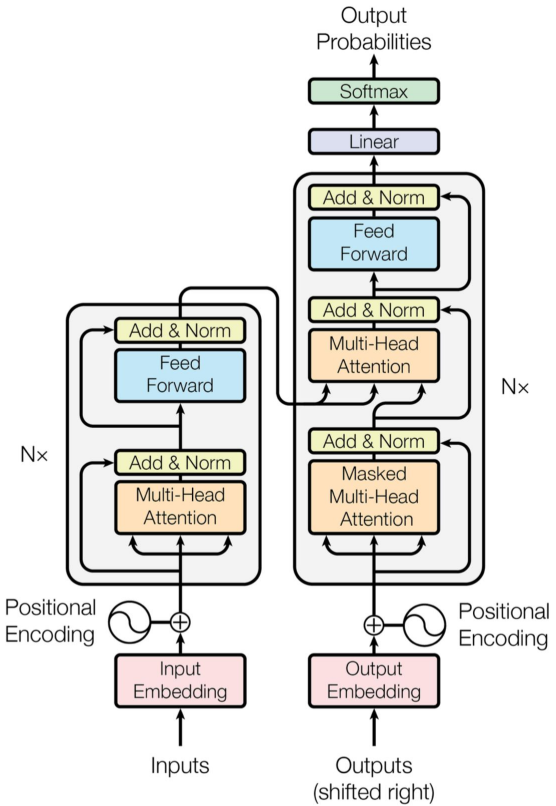


Figure 1: The Transformer - model architecture.

Non-local neural networks (Wang, 2018)

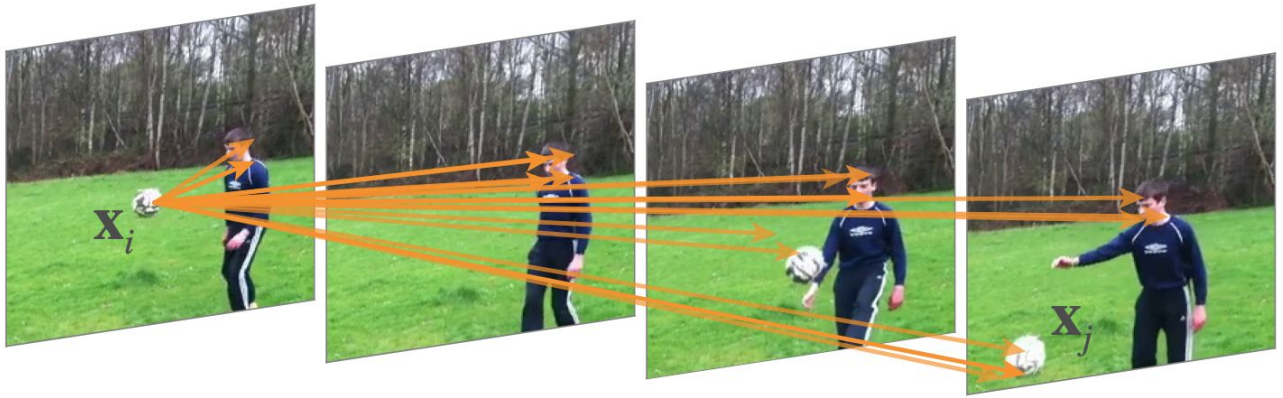


Figure 1. A spacetime ***non-local*** operation in our network trained for video classification in Kinetics. A position $\mathbf{x}_i$'s response is computed by the weighted average of the features of *all* positions $\mathbf{x}_j$ (only the highest weighted ones are shown here). In this example computed by our model, note how it relates the ball in the first frame to the ball in the last two frames. More examples are in Figure 3.

Non-local neural networks (Wang, 2018)

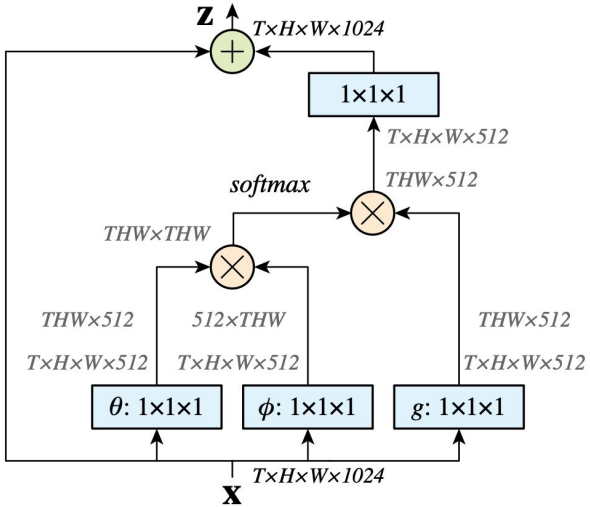


Figure 2. A spacetime **non-local block**. The feature maps are shown as the shape of their tensors, *e.g.*, $T \times H \times W \times 1024$ for 1024 channels (proper reshaping is performed when noted). "$\otimes$" denotes matrix multiplication, and "$\oplus$" denotes element-wise sum. The softmax operation is performed on each row. The blue boxes denote $1 \times 1 \times 1$ convolutions. Here we show the embedded Gaussian version, with a bottleneck of 512 channels. The vanilla Gaussian version can be done by removing θ and ϕ, and the dot-product version can be done by replacing softmax with scaling by $1/N$.

Self-Attention Generative Adversarial Networks (2019)

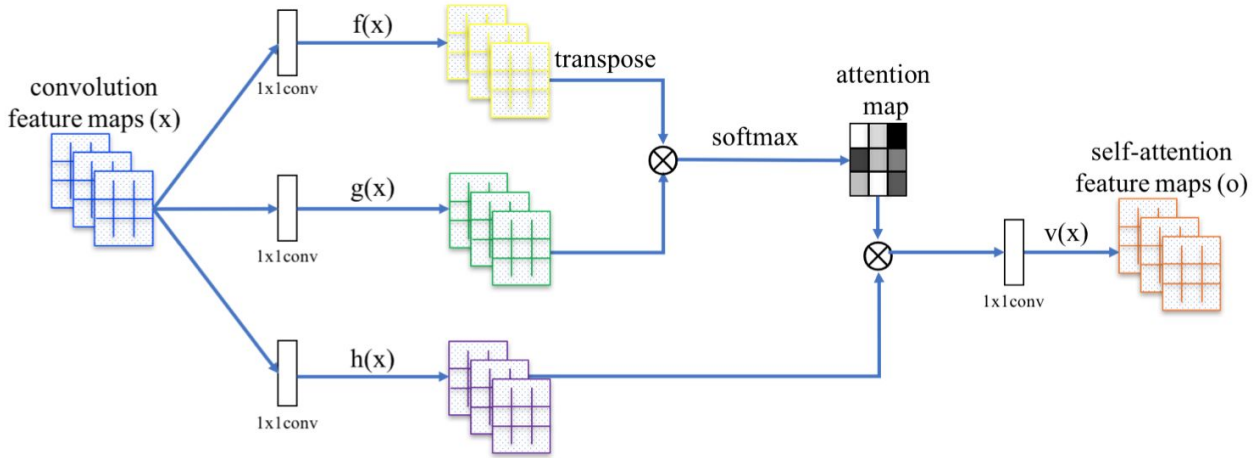


Figure 2. The proposed self-attention module for the SAGAN. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

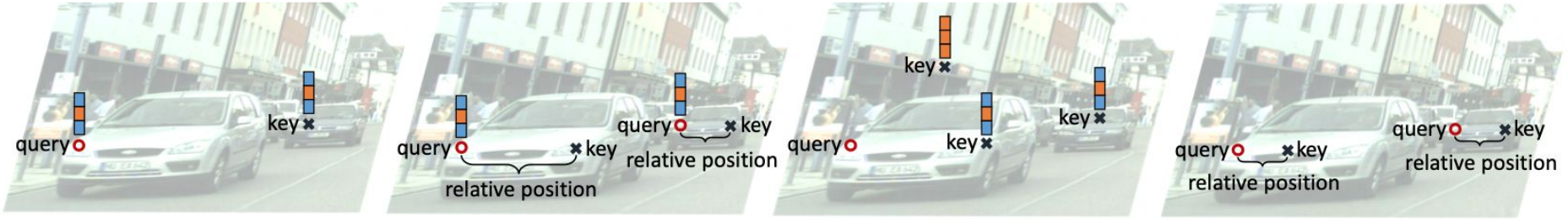
An Empirical Study of Spatial Attention Mechanisms in Deep Networks

Xizhou Zhu[1,3†*] Dazhi Cheng[2,3†*] Zheng Zhang[3*] Stephen Lin[3] Jifeng Dai[3]

[1]University of Science and Technology of China
[2]Beijing Institute of Technology
[3]Microsoft Research Asia

ezra0408@mail.ustc.edu.cn

{v-dachen,zhez,stevelin,jifdai}@microsoft.com



(a) query and key content (b) query content and relative position (c) key content only (d) relative position only

$$y_q = \sum_{m=1}^{M} W_m \Big[\sum_{k \in \Omega_q} A_m(q, k, z_q, x_k) \odot W'_m x_k \Big]$$

Transformer attention

In the most recent instantiation of the Transformer attention module [11], the attention weight of each query-key pair is computed as the sum of four terms $\{\mathcal{E}_j\}_{j=1}^{4}$ that are based on different attention factors, as

$$A_m^{\text{Trans}}(q, k, z_q, x_k) \propto \exp\big(\sum_{j=1}^{4} \mathcal{E}_j\big), \tag{2}$$

normalized by $\sum_{k \in \Omega_q} A_m^{\text{Trans}}(q, k, z_q, x_k) = 1$ where the supporting key region Ω_q spans the key elements (*e.g.*, the

$$\mathcal{E}_1 = z_q^\top U_m^\top \hat{V}_m^{\text{C}} x_k$$

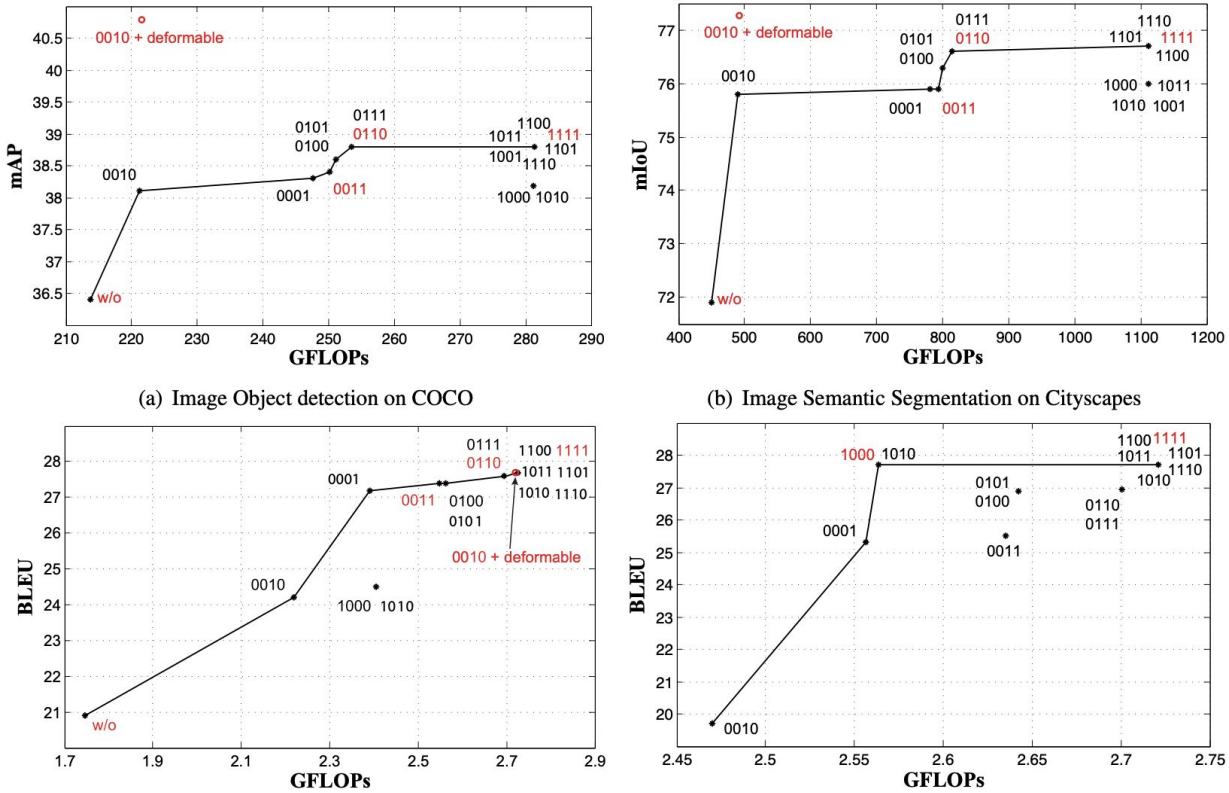
$$\mathcal{E}_2 = z_q^\top U_m^\top V_m^{\text{R}} R_{k-q}$$

$$\mathcal{E}_3 = u_m^\top V_m^{\text{C}} x_k$$

$$\mathcal{E}_4 = v_m^\top V_m^{\text{R}} R_{k-q}$$

attention mechanism		spatial properties	query content	key content	relative position	complexity
Transformer attention	$\mathcal{E}_1$	dense, global	✓	✓		$O(N_s^2C + N_sC^2)$
	$\mathcal{E}_2$	dense, global	✓		✓	$O(N_s^2C + N_sC^2)$
	$\mathcal{E}_3$	dense, global		✓		$O(N_sC^2)$
	$\mathcal{E}_4$	dense, global			✓	$O(N_s^2C + N_sC^2)$
Regular convolution		sparse, local			✓	$O(N_sC^2N_k)$
Deformable convolution		sparse, global	✓		✓	$O(N_sC^2N_k)$
Dynamic convolution		sparse, local	✓		✓	$O(N_sCN_gN_k + N_sC^2)$

Table 1. Comparison of different attention mechanisms. N_s denotes number of spatial elements, i.e. width by height for images, and number of tokens for text; C denotes representation dimension; N_k denotes kernel size of convolution ($N_k = 3 \times 3$ for images and $N_k = 3$ for text, by default); N_g denotes number of feature groups in dynamic convolution.



(a) Image Object detection on COCO

(b) Image Semantic Segmentation on Cityscapes

(c) Translation on newstest2014 (self-attention)

(d) Translation on newstest2014 (encoder-decoder attention)

Figure 3. Accuracy-efficiency tradeoffs of the four terms in Transformer attention ($\mathcal{E}_1$ for query and key content, $\mathcal{E}_2$ for query content and relative position, $\mathcal{E}_3$ for key content only, and $\mathcal{E}_4$ for relative position only). The activation and deactivation of particular terms is set by configuration $\{\beta_j^{\text{Trans}}\}_{j=1}^4$ (*e.g.*, "0011" denotes the activation of $\mathcal{E}_3$ and $\mathcal{E}_4$). Because the encoder-decoder attention mechanism is indispensable for NMT, there is no "w/o" setting in (d). The results of some configurations overlap in the plots because they are of the same accuracy and computational overhead. The key configurations under study are highlighted in red. The recommended configuration of "0010 + deformable" for self-attention in Tab. 2 is also plotted here.

Некоторые из выводов в статье:

1. In self-attention, the query-sensitive terms play a minor role compared to the query-irrelevant terms
2. In encoder-decoder attention, the query and key content term is vital.
3. In self-attention, the attention factors of query content & relative position and the key content only are most important.

On the Relationship between Self-Attention and Convolutional Layers

(Cordonnier, 2020)

Theorem 1. *A multi-head self-attention layer with N_h heads of dimension D_h, output dimension D_{out} and a relative positional encoding of dimension $D_p \geq 3$ can express any convolutional layer of kernel size $\sqrt{N_h} \times \sqrt{N_h}$ and $\min(D_h, D_{out})$ output channels.*

ON THE RELATIONSHIP BETWEEN SELF-ATTENTION AND CONVOLUTIONAL LAYERS

(Cordonnier, 2020)

2.3 POSITIONAL ENCODING FOR IMAGES

There are two types of positional encoding that has been used in transformer-based architectures: the *absolute* and *relative* encoding (see also Table 3 in the Appendix).

With absolute encodings, a (fixed or learned) vector $\mathbf{P}_{\boldsymbol{p},:}$ is assigned to each pixel $\boldsymbol{p}$. The computation of the attention scores we saw in eq. (2) can then be decomposed as follows:

$$\begin{aligned}\mathbf{A}^{\text{abs}}_{\boldsymbol{q},\boldsymbol{k}} &= (\mathbf{X}_{\boldsymbol{q},:} + \mathbf{P}_{\boldsymbol{q},:})\boldsymbol{W}_{qry}\boldsymbol{W}_{key}^{\top}(\mathbf{X}_{\boldsymbol{k},:} + \mathbf{P}_{\boldsymbol{k},:})^{\top} \\ &= \mathbf{X}_{\boldsymbol{q},:}\boldsymbol{W}_{qry}\boldsymbol{W}_{key}^{\top}\mathbf{X}_{\boldsymbol{k},:}^{\top} + \mathbf{X}_{\boldsymbol{q},:}\boldsymbol{W}_{qry}\boldsymbol{W}_{key}^{\top}\mathbf{P}_{\boldsymbol{k},:}^{\top} + \mathbf{P}_{\boldsymbol{q},:}\boldsymbol{W}_{qry}\boldsymbol{W}_{key}^{\top}\mathbf{X}_{\boldsymbol{k},:} + \mathbf{P}_{\boldsymbol{q},:}\boldsymbol{W}_{qry}\boldsymbol{W}_{key}^{\top}\mathbf{P}_{\boldsymbol{k},:} \end{aligned} \tag{7}$$

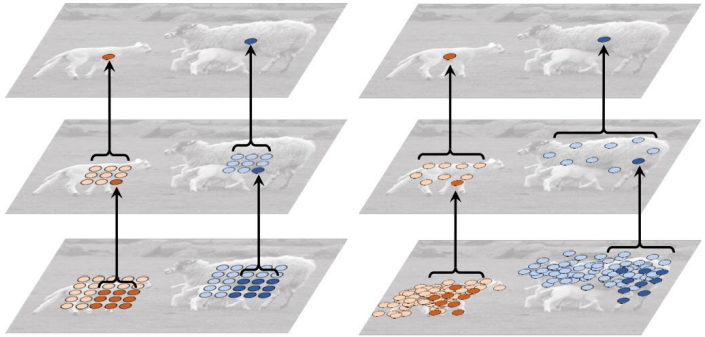
where $\boldsymbol{q}$ and $\boldsymbol{k}$ correspond to the query and key pixels, respectively.

The relative positional encoding was introduced by Dai et al. (2019). The main idea is to only consider the position difference between the query pixel (pixel we compute the representation of) and the key pixel (pixel we attend) instead of the absolute position of the key pixel:

$$\mathbf{A}^{\text{rel}}_{\boldsymbol{q},\boldsymbol{k}} := \mathbf{X}_{\boldsymbol{q},:}^{\top}\boldsymbol{W}_{qry}^{\top}\boldsymbol{W}_{key}\,\mathbf{X}_{\boldsymbol{k},:} + \mathbf{X}_{\boldsymbol{q},:}^{\top}\boldsymbol{W}_{qry}^{\top}\widehat{\boldsymbol{W}}_{key}\,\boldsymbol{r}_{\boldsymbol{\delta}} + \boldsymbol{u}^{\top}\boldsymbol{W}_{key}\,\mathbf{X}_{\boldsymbol{k},:} + \boldsymbol{v}^{\top}\widehat{\boldsymbol{W}}_{key}\,\boldsymbol{r}_{\boldsymbol{\delta}} \tag{8}$$

In this manner, the attention scores only depend on the shift $\boldsymbol{\delta} := \boldsymbol{k} - \boldsymbol{q}$. Above, the learnable vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are unique for each head, whereas for every shift $\boldsymbol{\delta}$ the relative positional encoding $\boldsymbol{r}_{\boldsymbol{\delta}} \in \mathbb{R}^{D_p}$ is shared by all layers and heads. Moreover, now the key weights are split into two types: $\boldsymbol{W}_{key}$ pertain to the input and $\widehat{\boldsymbol{W}}_{key}$ to the relative position of pixels.

Deformable convolutions (2017, v2 - 2018)



(a) standard convolution (b) deformable convolution

Figure 5: Illustration of the fixed receptive field in standard convolution (a) and the adaptive receptive field in deformable convolution (b), using two layers. Top: two activation units on the top feature map, on two objects of different scales and shapes. The activation is from a 3×3 filter. Middle: the sampling locations of the 3×3 filter on the preceding feature map. Another two activation units are highlighted. Bottom: the sampling locations of two levels of 3×3 filters on the preceding feature map. Two sets of locations are highlighted, corresponding to the highlighted units above.

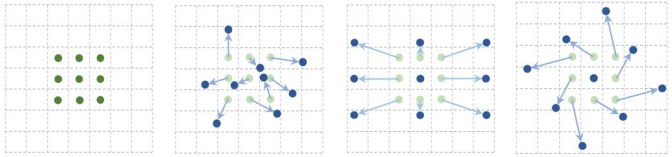


Figure 1: Illustration of the sampling locations in 3×3 standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.

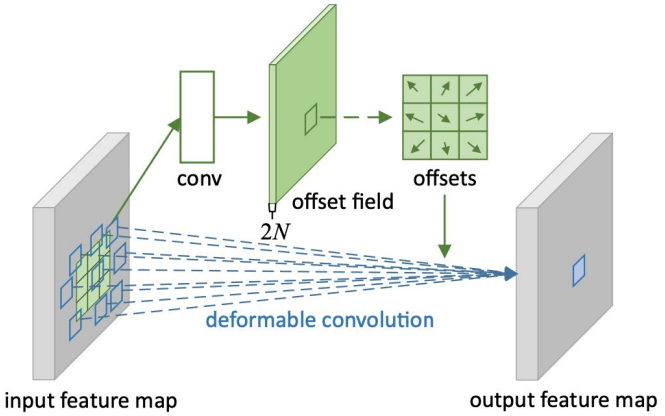


Figure 2: Illustration of 3×3 deformable convolution.

$$\mathcal{R} = \{(-1,-1),(-1,0),\ldots,(0,1),(1,1)\}$$

Usual 3x3 conv

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n)$$

Bilinear interpolation

Deformable 3x3 conv

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$$

Example of channel attention

Squeeze-and-Excitation Networks

Jie Hu[0000−0002−5150−1003] Li Shen[0000−0002−2283−4976] Samuel Albanie[0000−0001−9736−5134]
Gang Sun[0000−0001−6913−6799] Enhua Wu[0000−0002−2174−1428]

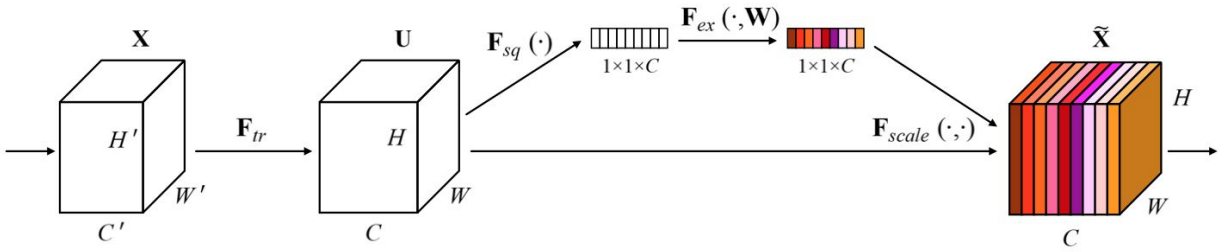


Fig. 1. A Squeeze-and-Excitation block.

Transformers for object detection? Yes!

Francisco Massa
@fvsmassa

I'm super happy to share that our paper End-to-End Object Detection with Transformers (DETR) has been accepted as an oral paper at #ECCV2020 !

> **European Conference on Computer Vision (ECCV)** @eccvconf · 8h
> Congratulations to all who have a paper accepted for #ECCV2020 view the website for more details eccv2020.eu/accepted-paper...

DETR (2020)

```python
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):

    def __init__(self, num_classes, hidden_dim, nheads,
                 num_encoder_layers, num_decoder_layers):
        super().__init__()
        # We take only convolutional layers from ResNet-50 model
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
                                          num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))

    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([
            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
        ], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
                             self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```

Listing 1: DETR PyTorch inference code. For clarity it uses learnt positional encodings in the encoder instead of fixed, and positional encodings are added to the input only instead of at each transformer layer. Making these changes requires going beyond PyTorch implementation of transformers, which hampers readability. The entire code to reproduce the experiments will be made available before the conference.

DETR (2020)

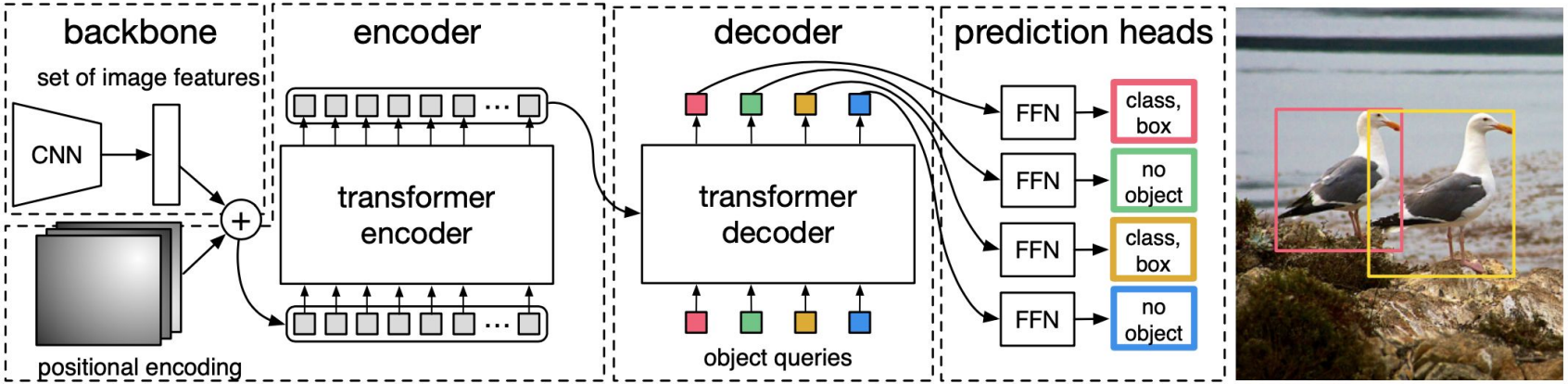


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.