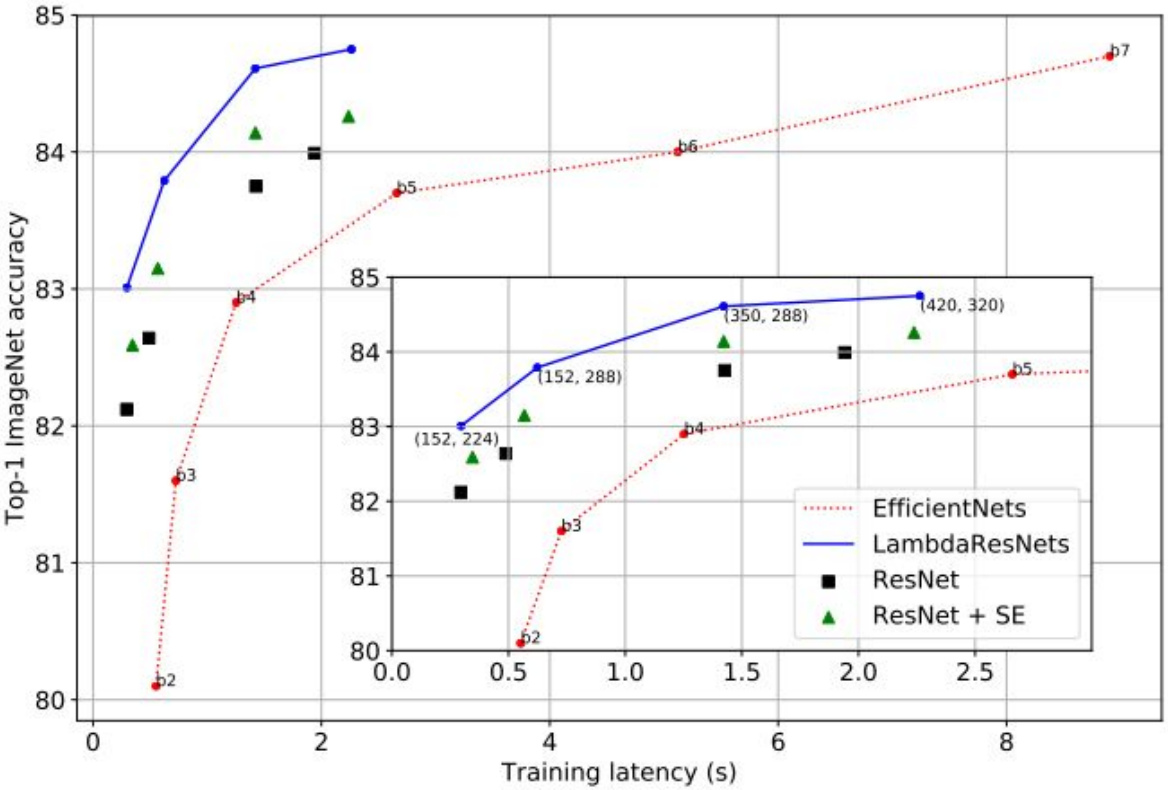
Lambda Networks: Modeling long-range Interactions without Attention

Main Result:

1. Outperforms ResNets and EfficientNets on Top 1 accuracy on ImageNet.
2. 4.5x faster to train than EfficientNets.
3. Low memory requirements compared to Transformers.



Notes:

- Paper is hard to understand.
- Based on Transformer Networks
- Other references:
  - https://www.youtube.com/watch?v=3qxJ2WD8p4w
- Code:
  - https://github.com/lucidrains/lambda-networks (pytorch)
  - https://github.com/leaderj1001/LambdaNetworks (pytorch)
  - https://paperswithcode.com/paper/lambdanetworks-modeling-long-range
-

BackGround - Regular Attention

- When processing large data, some parts of the input are more important than others. (This is attention)
- Attention is basically a weighted average.
- Given two sequences, X (input) & C (what to attend to) what should be the next layer output at each input.

BackGround - Regular Attention

Regular Attention

① Get Queries, Keys and Values

$$\underset{\text{Input}}{X^{n\times d}} \xrightarrow{W_Q^{d\times k}} \underset{\text{queries}}{Q^{n\times k}}$$

$$\underset{\text{Context (what to attend)}}{C^{m\times d}} \xrightarrow{W_K^{d\times k}} \underset{\text{Keys}}{K^{m\times k}}$$

$$C^{m\times d} \xrightarrow{W_V^{d\times v}} \underset{\text{Values}}{V^{m\times v}}$$

n = input length

For image $200 \times 200 = 40{,}000$

d = # of input channels.

k = internal dimensionality of attention layer (dimension of keys)

m = context length. (similar to input)

v = # of output channels

» context can be anything → if same as input = self attention

» Global Attention $n = m$

» Local Attention $m \ll n$.

② Get Attention maps

Ⓐ multiply queries by the keys.

$$A = Q \cdot K^T = A$$

$[n \times k][k \times m] \quad [n \times m]$

Ⓑ row-wise softmax

$$\tilde{A} = \sigma_m(A) = \tilde{A}$$

$[n \times m]$

- $\tilde{A}$ = n attention maps of size m

$$\begin{bmatrix} [0.1 & 0.7 & 0.1 & \cdots \\ [0.3 & 0.4 & 0.2 & \cdots \\ [0.1 & 0.1 & 0.7 & \cdots \\ \cdots & & & \\ \cdots & & & \end{bmatrix}_{n \times m}$$

- How much pixel n should pay attention to ~~for~~ each of its m neighbours.

③ Get the Output

③ Get the Output

A. Multiply Attention Maps with Values.

$$\underset{[n \times m]}{A} \cdot \underset{[m \times v]}{V} = \underset{[n \times v]}{Y}$$

Advantages compared with RNN Approach:

1. Not Sequential on input, can be parallelized for much faster processing on GPUs.
2. Can directly refer to previous inputs using attention, not only the compact representation of all previous inputs in the hidden state. - much better at picking up long range dependencies.
3. More details: Ali Ghodsi lection on self attention: https://www.youtube.com/watch?v=WFcH7kRNEBc

Problems with Attention in Visual Tasks

- **Global Attention (n=m):**
  - For Image size = 200x200,
  - n = 40,000
  - Attention map [nxm] = Huge memory
  - (aka quadratic memory footprint of attention)
- **Local Attention (m << n):**
  - Much smaller attention maps
  - Context changes for each pixel.
  - Huge computational cost.
  - No Efficient way of doing this currently. (Many proposals active)

Position Embeddings

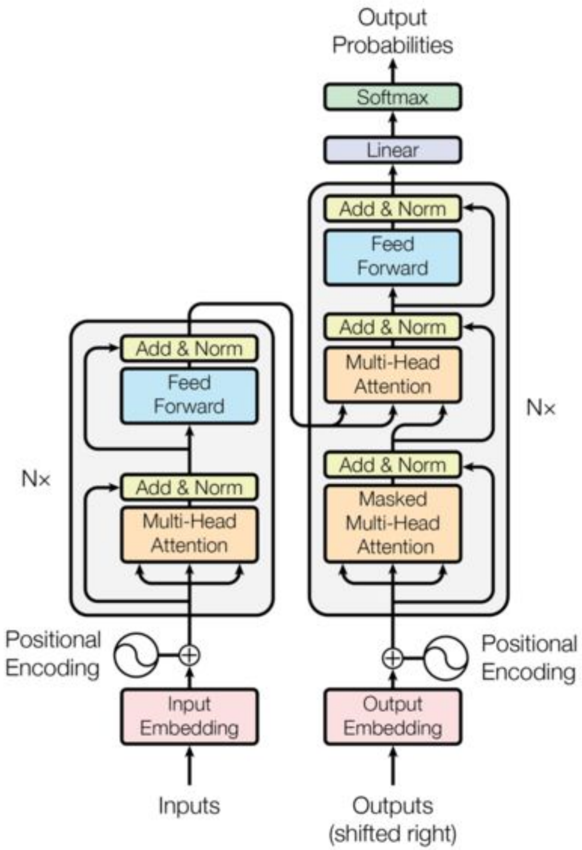
- Transformers do not have inherent awareness of position/order.
- Order of words is important.
- Transformers Networks (main user of attention) add a positional embedding to each input word to give the model knowledge of position
- multi-dimensional vector for each word in the sequence. (same dimension as word embeddings)

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right),$$

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

with $d_{model} = 512$ (thus $i \in [0, 255]$) in the original paper.

- **IMP: Positional Embeddings are fixed.**



Good Reference:
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Higher Order Matrix Multiplication (Tensor Contraction)

⇒ **Higher Order Matrix Multiplication** (Tensor Contraction)

2D Case

$$A = Q \cdot K^T$$

$[n \times m]$ $\quad$ $[n \times k][k \times m]$

→ contract out matching dimension, but summing over it.

$$a_{nm} = \sum_k Q_{nk} \cdot K_{km}$$

higher Dimension Case

$$C = A \cdot B = C_{i,j,l,m}$$

$[i,j,k][k,l,m]$ $\quad$ $[i \times j \times l \times m]$ 4D

$$c_{ijlm} = \sum_k a_{i,j,k} \cdot b_{k,l,m}$$

Ref: https://math.stackexchange.com/questions/63074/is-there-a-3-dimensional-matrix-by-matrix-product

Lambda Networks

LAMBDA NETWORKS

① Get Keys, Queries and Values → same as regular attention

$$X^{n\times d} \xrightarrow{W_Q^{d\times k}} Q^{n\times k}$$

$$\begin{tikzcd}
C^{m\times d} \arrow[r, "W_K^{d\times k}"] \arrow[dr, "W_V^{d\times v}"'] & K^{m\times k} \\
 & V^{m\times v}
\end{tikzcd}$$

② Normalize the Keys

$$\tilde{K} = \sigma_m(K) = \tilde{K}$$
$[m \times k]$

➤ Different from regular attention
→ softmax keys on their own
→ $m \times k$

➤ Regular Attention
→ softmax (Keys x Queries)
→ $n \times m$

③ Get Context Interactions λ^c

➤ This is like the attention map

3. Get Context Interactions λ^c

$$\lambda^c = \underset{[k \times m]}{\tilde{K}^T} \cdot \underset{[m \times v]}{V} = \underset{[k \times v]}{\lambda^c}$$

- This is like the attention map
- Context is a summary of the actual context.

4. Get the Output

$$Y^c = \underset{[n \times k]}{Q} \cdot \underset{[k \times v]}{\lambda^c} = \underset{[n \times v]}{Y^c}$$

Advantages

- much smaller memory map $\lambda^c = [k \times v]$ ~~[illegible]~~
- Regular Attention $A = [n \times m]$
- Especially Important when batch dimension ~~[illegible]~~ and multihead dimensions added.

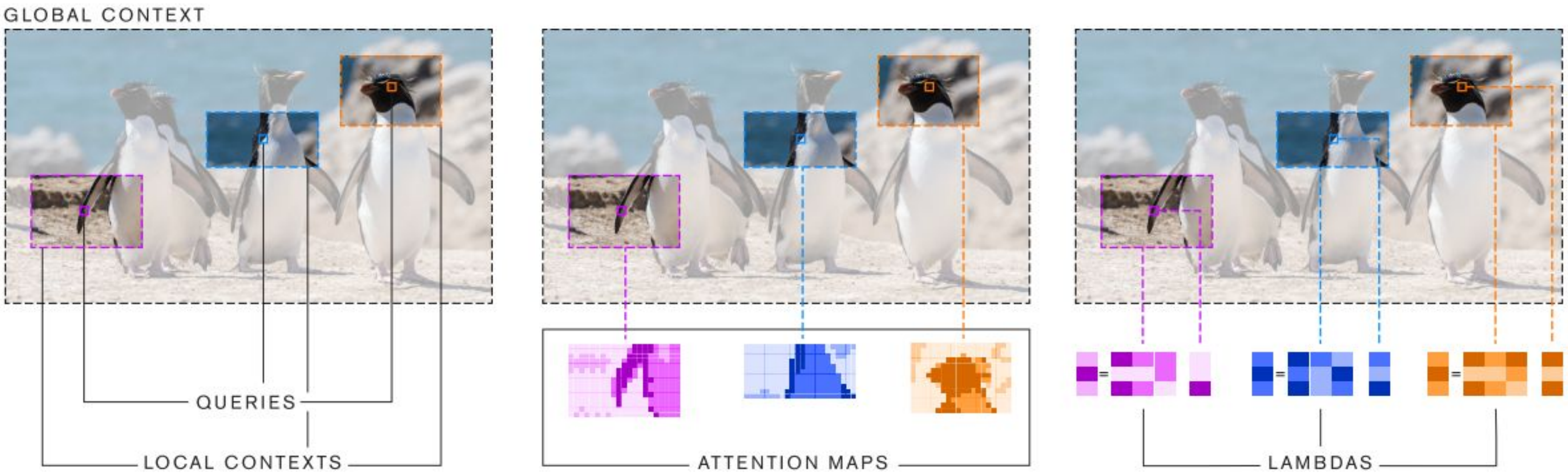
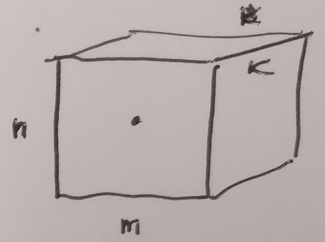


Figure 1: Comparison between attention and lambda layers. (Left) An example of 3 queries and their local contexts within a global context. (Middle) The attention operation associates each query with an attention distribution over its context. (Right) The lambda layer transforms each context into a linear function lambda that is applied to the corresponding query.

Positional Embedding



- Each position n is related to m neighbor positions by a k vector
- $E = [n \times m \times k]$
- BUT, this is the same $\forall$ images (batch size does not affect this)

5 Get the positional Interactions λ^p

contract out the m dimension.

$$\underset{[n \times m \times k]}{\lambda^p = E} \cdot \underset{[m \times v]}{V} = \underset{[n \times k \times v]}{\lambda^p}$$

contract m

6 ~~8~~ Get the Output Y^p

$$Y^p = Q \cdot \lambda^p = Y^p$$

$[n \times k]\ [n \times k \times v] \qquad [n \times v]$

contract over k

Q: What about the second n?

7 ~~8~~ Final Output

$$Y = Y^c + Y^p$$

Advantages

- Not only position embedding, the relationship because of the position (has access to the context)
- Does not grow w batch size

Results - Imagenet Classification Accuracy

Table 3: **Comparison of the lambda layer and attention mechanisms on ImageNet classification with a ResNet50 architecture.** The lambda layer strongly outperforms alternatives at a fraction of the parameter cost. We include the reported improvements compared to the ResNet50 baseline in subscript to account for training setups that are not directly comparable. †: Our implementation.

Layer	Params (M)	top-1
Conv (He et al., 2016)†	25.6	$76.9_{+0.0}$
Conv + channel attention (Hu et al., 2018b)†	28.1	$77.6_{+0.7}$
Conv + double attention (Chen et al., 2018)	33.0	77.0
Conv + efficient attention (Shen et al., 2018)	-	$77.3_{+1.2}$
Conv + relative self-attention (Bello et al., 2019)	25.8	$77.7_{+1.3}$
Local relative self-attention (Ramachandran et al., 2019)	18.0	$77.4_{+0.5}$
Local relative self-attention (Hu et al., 2019)	23.3	$77.3_{+1.0}$
Local relative self-attention (Zhao et al., 2020)	20.5	$78.2_{+1.3}$
Lambda layer	**15.0**	$\mathbf{78.4}_{+1.5}$
Lambda layer ($\lvert u\rvert$=4)	**16.0**	$\mathbf{78.9}_{+2.0}$

Imagenet Classification - Memory and Compute

Table 4: **The lambda layer reaches higher accuracies while being faster and more memory-efficient than self-attention alternatives.** Inference throughput is measured on 8 TPUv3 cores for a ResNet50 architecture with input resolution 224x224.

Layer	Complexity	Memory (GB)	Throughput	top-1
Global self-attention	$\Theta(blhn^2)$	120	OOM	OOM
Axial self-attention	$\Theta(blhn\sqrt{n})$	4.8	960ex/s	77.5
Local self-attention (7x7)	$\Theta(blhnm)$	-	440ex/s	77.4
Lambda layer	$\Theta(lkn^2)$	0.96	1160ex/s	**78.4**
Lambda layer (shared embeddings)	$\Theta(kn^2)$	0.31	1210ex/s	78.0
Lambda layer ($\|k\|$=8)	$\Theta(lkn^2)$	0.48	**1640**ex/s	77.9
Lambda convolution (7x7)	$\Theta(lknm)$	-	1100ex/s	78.1

MS COCO Object Detection

Table 7: **COCO object detection and instance segmentation with Mask-RCNN architecture on 1024x1024 inputs**. Mean Average Precision (AP) is reported at three IoU thresholds and for small, medium, large objects (s/m/l).

Backbone	AP^{bb}_{coco}	$AP^{bb}_{s/m/l}$	AP^{mask}_{coco}	$AP^{mask}_{s/m/l}$
ResNet-101	48.2	29.9 / 50.9 / 64.9	42.6	24.2 / 45.6 / 60.0
ResNet-101 + SE	48.5	29.9 / 51.5 / 65.3	42.8	24.0 / 46.0 / 60.2
LambdaResNet-101	**49.4**	**31.7 / 52.2 / 65.6**	**43.5**	**25.9 / 46.5 / 60.8**
ResNet-152	48.9	29.9 / 51.8 / 66.0	43.2	24.2 / 46.1 / 61.2
ResNet-152 + SE	49.4	30.0 / 52.3 / 66.7	43.5	24.6 / 46.8 / 61.8
LambdaResNet-152	**50.0**	**31.8 / 53.4 / 67.0**	**43.9**	**25.5 / 47.3 / 62.0**

Position vs Context Interactions

Table 8: Contributions of content and positional interactions. As expected, positional interactions are crucial to perform well on the image classification task.

Content	Position	Params (M)	FLOPS (B)	top-1
✓	×	14.9	5.0	68.8
×	✓	14.9	11.9	78.1
✓	✓	14.9	12.0	78.4

Lambda Resnets (Replace Conv blocks with Lambda blocks)

Table 12: Inference throughput and top-1 accuracy as a function of lambda (L) vs convolution (C) layers' placement in a ResNet50 architecture on 224x224 inputs.

Architecture	Params (M)	Throughput	top-1
$\mathbf{C} \rightarrow \mathbf{C} \rightarrow \mathbf{C} \rightarrow \mathbf{C}$	25.6	7240ex/s	76.9
$\mathbf{L} \rightarrow \mathbf{C} \rightarrow \mathbf{C} \rightarrow \mathbf{C}$	25.5	1880ex/s	77.3
$\mathbf{L} \rightarrow \mathbf{L} \rightarrow \mathbf{C} \rightarrow \mathbf{C}$	25.0	1280ex/s	77.2
$\mathbf{L} \rightarrow \mathbf{L} \rightarrow \mathbf{L} \rightarrow \mathbf{C}$	21.7	1160ex/s	77.8
$\mathbf{L} \rightarrow \mathbf{L} \rightarrow \mathbf{L} \rightarrow \mathbf{L}$	15.0	1160ex/s	78.4
$\mathbf{C} \rightarrow \mathbf{L} \rightarrow \mathbf{L} \rightarrow \mathbf{L}$	15.1	2200ex/s	78.3
$\mathbf{C} \rightarrow \mathbf{C} \rightarrow \mathbf{L} \rightarrow \mathbf{L}$	15.4	4980ex/s	78.3
$\mathbf{C} \rightarrow \mathbf{C} \rightarrow \mathbf{C} \rightarrow \mathbf{L}$	18.8	7160ex/s	77.3