max planck institut
informatik

SIC Saarland Informatics Campus

High Level Computer Vision

Hierarchical Transformer Architectures & Advanced Optimization

@ June 12, 2024

Bernt Schiele

cms.sic.saarland/hlcvss24/

Max Planck Institute for Informatics & Saarland University,
Saarland Informatics Campus Saarbrücken

Overview of Today's Lecture

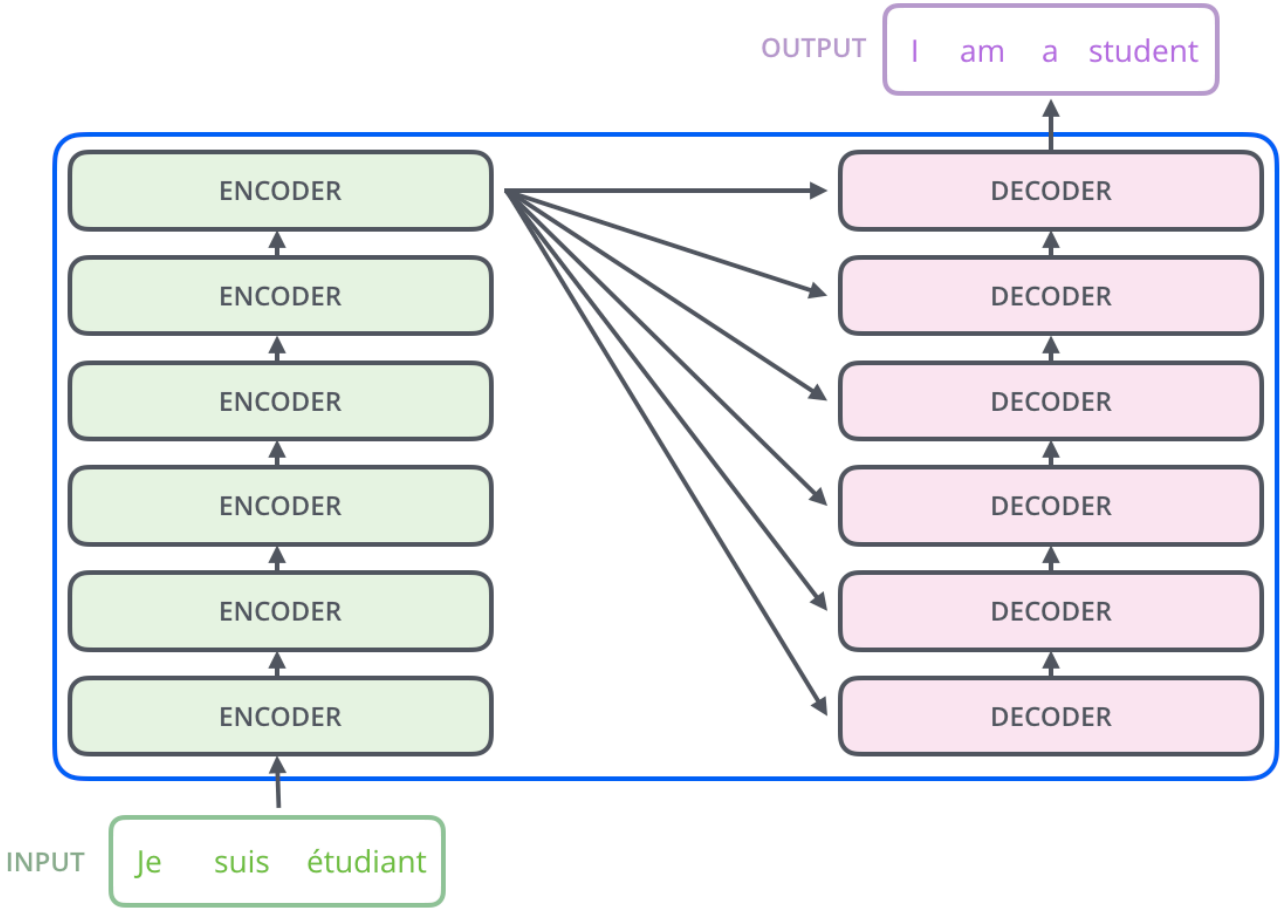
- Recap of Vision Transformer (ViT) for Image Classification

- Hierarchical Transformer Architectures
  - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows @ ICCV 2021 — https://arxiv.org/abs/2103.14030
  - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers @ NeurIPS 2021 — https://arxiv.org/abs/2105.15203

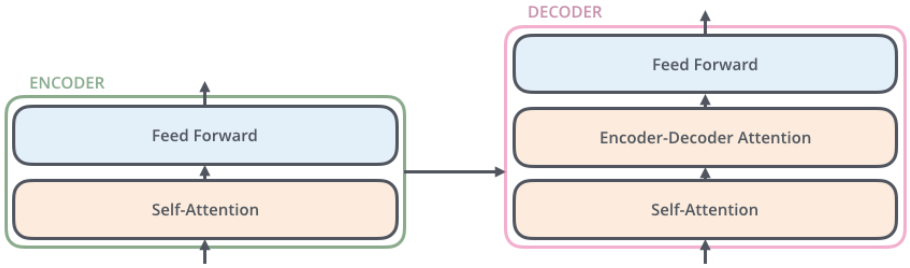
- Beyond Vanilla Gradient Descent
  - Adam Optimizer (and its components)
  - Second Order Optimization

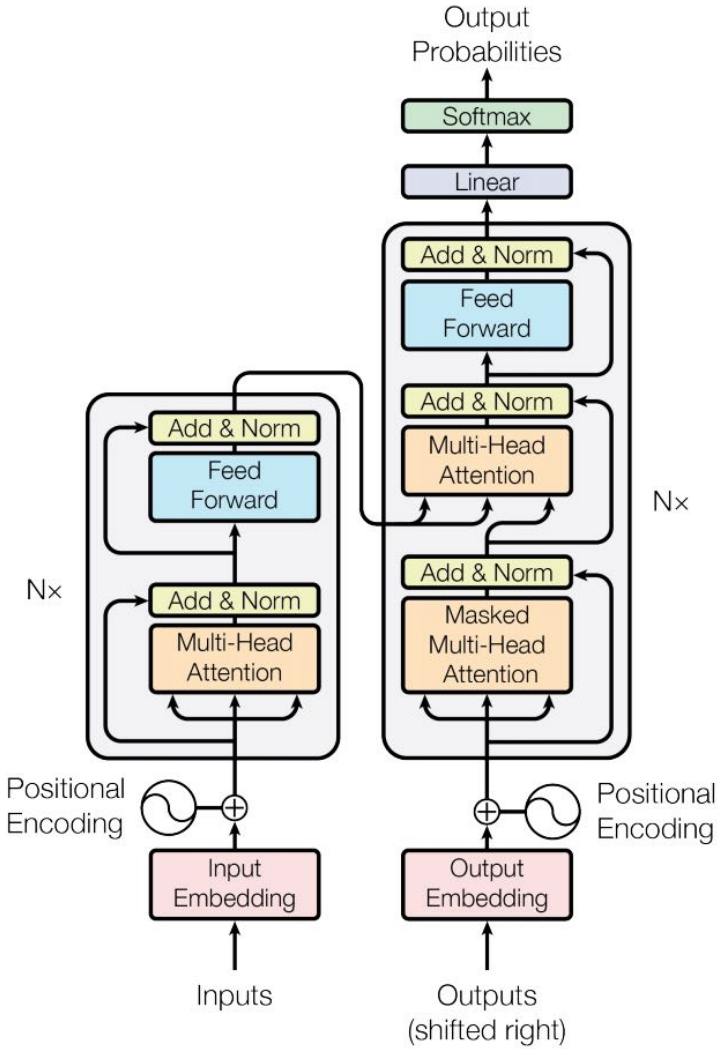
pictures are from http://jalammar.github.io/illustrated-transformer/

Transformer: Architecture Overview



Transformer: Architecture Overview

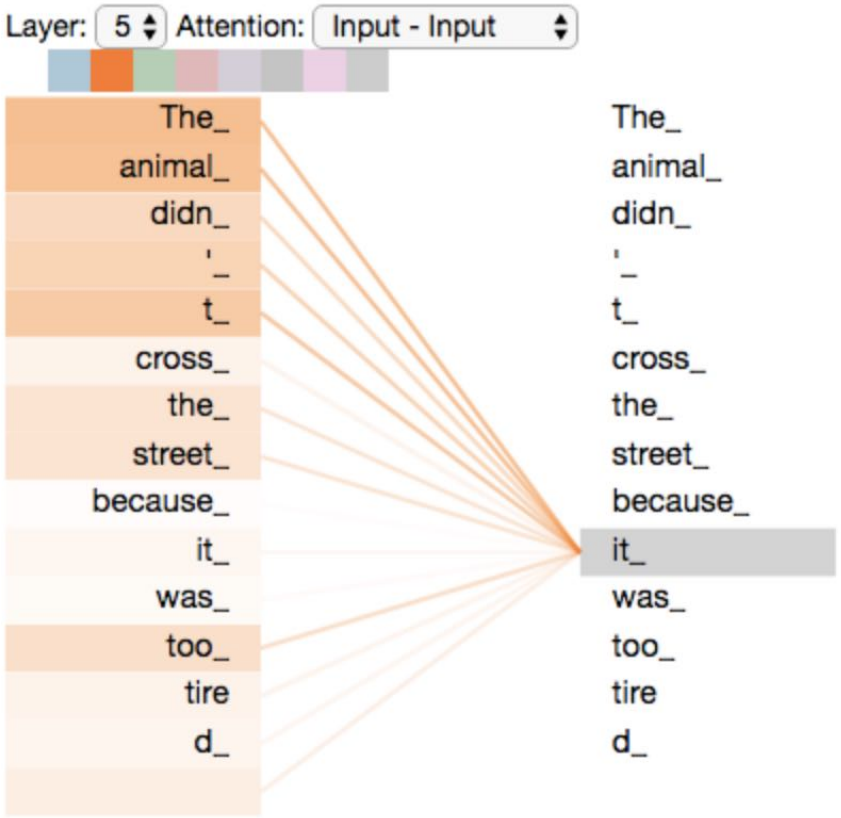




Attention is all you need. Vaswani et al. NeurIPS 2017

pictures are from http://jalammar.github.io/illustrated-transformer/

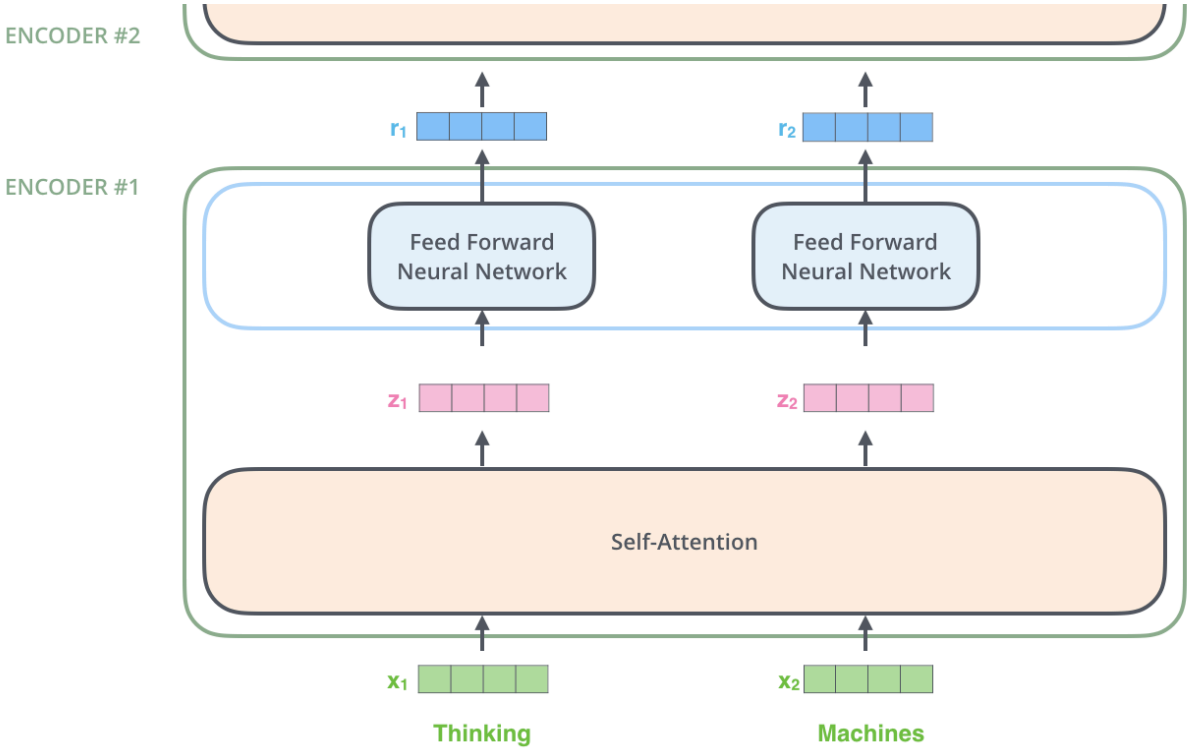
What is Self-Attention?



Attention is all you need. Vaswani et al. NeurIPS 2017

pictures are from http://jalammar.github.io/illustrated-transformer/

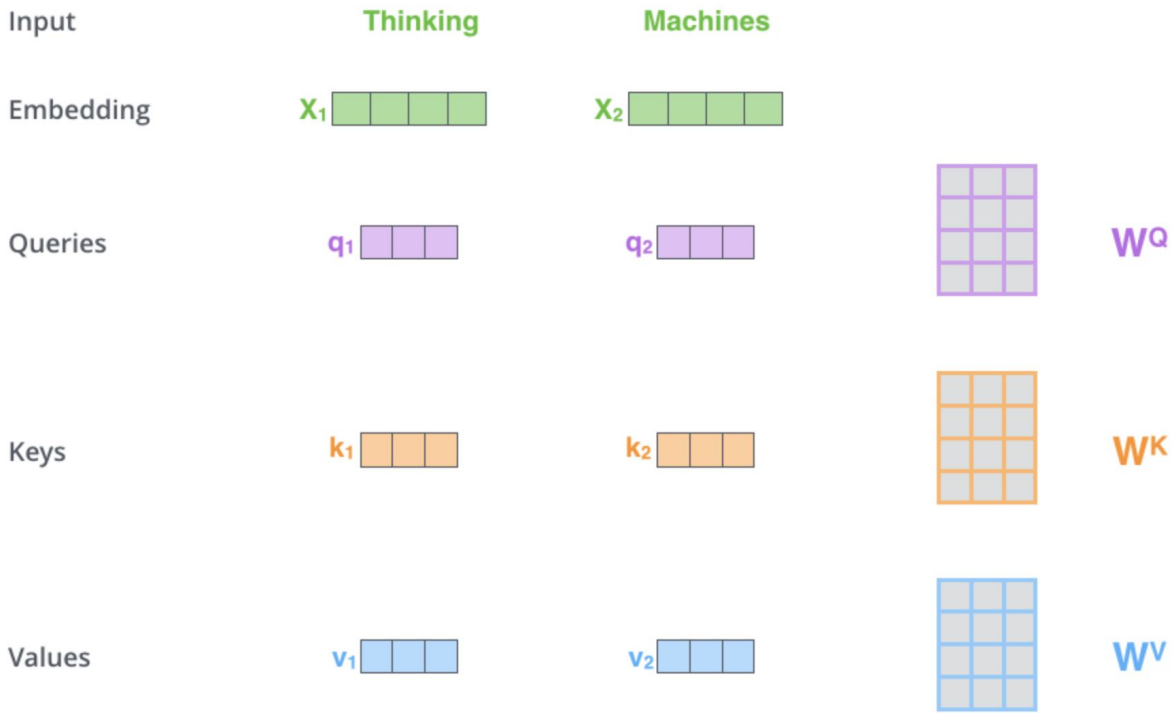
Encoder — Consumes a Set of (Word) Tokens

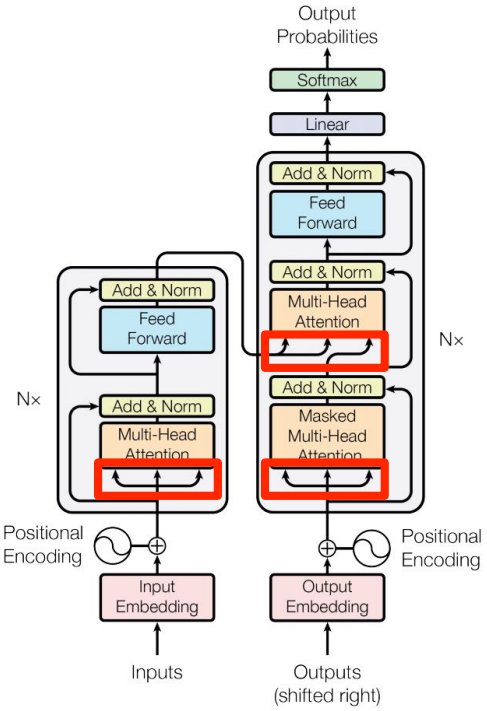


Attention is all you need. Vaswani et al. NeurIPS 2017

pictures are from http://jalammar.github.io/illustrated-transformer/

Self-Attention: Query, Key and Value



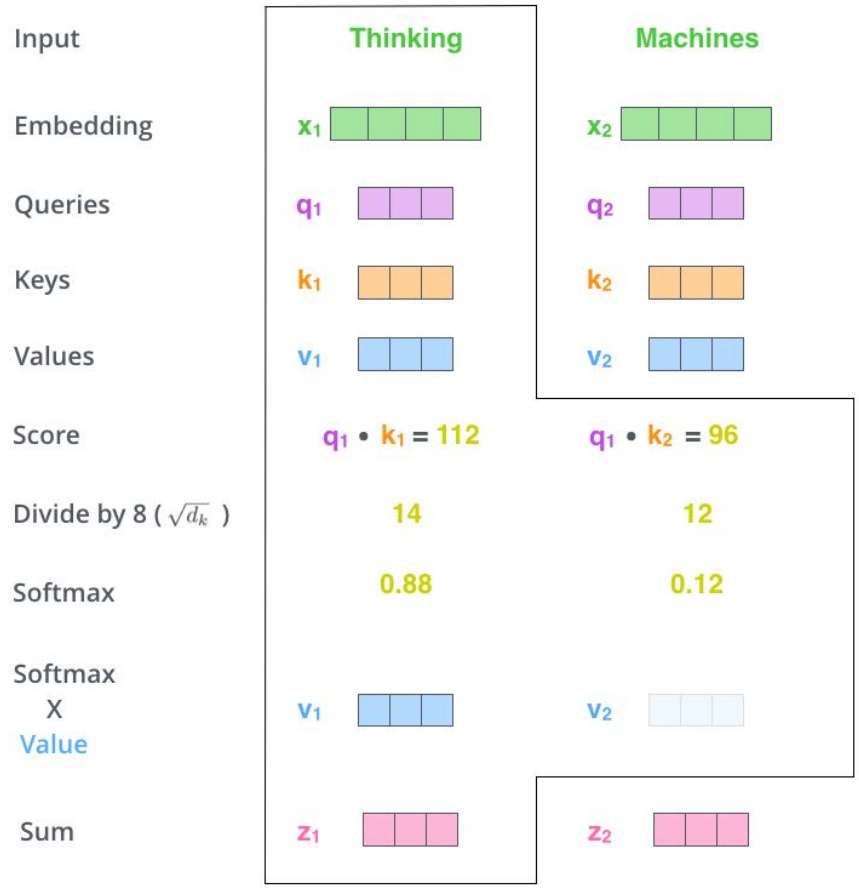


Attention is all you need. Vaswani et al. NeurIPS 2017

slide credit: Yongqin Xian

pictures are from http://jalammar.github.io/illustrated-transformer/

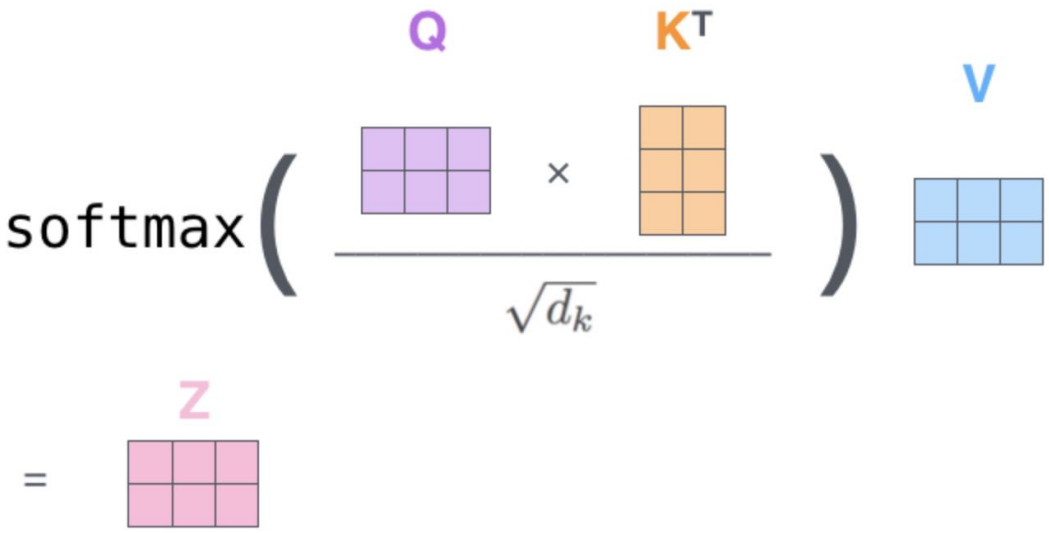
Self-Attention: Query, Key and Value



slide credit: Yongqin Xian

pictures are from http://jalammar.github.io/illustrated-transformer/

Self-Attention: Output

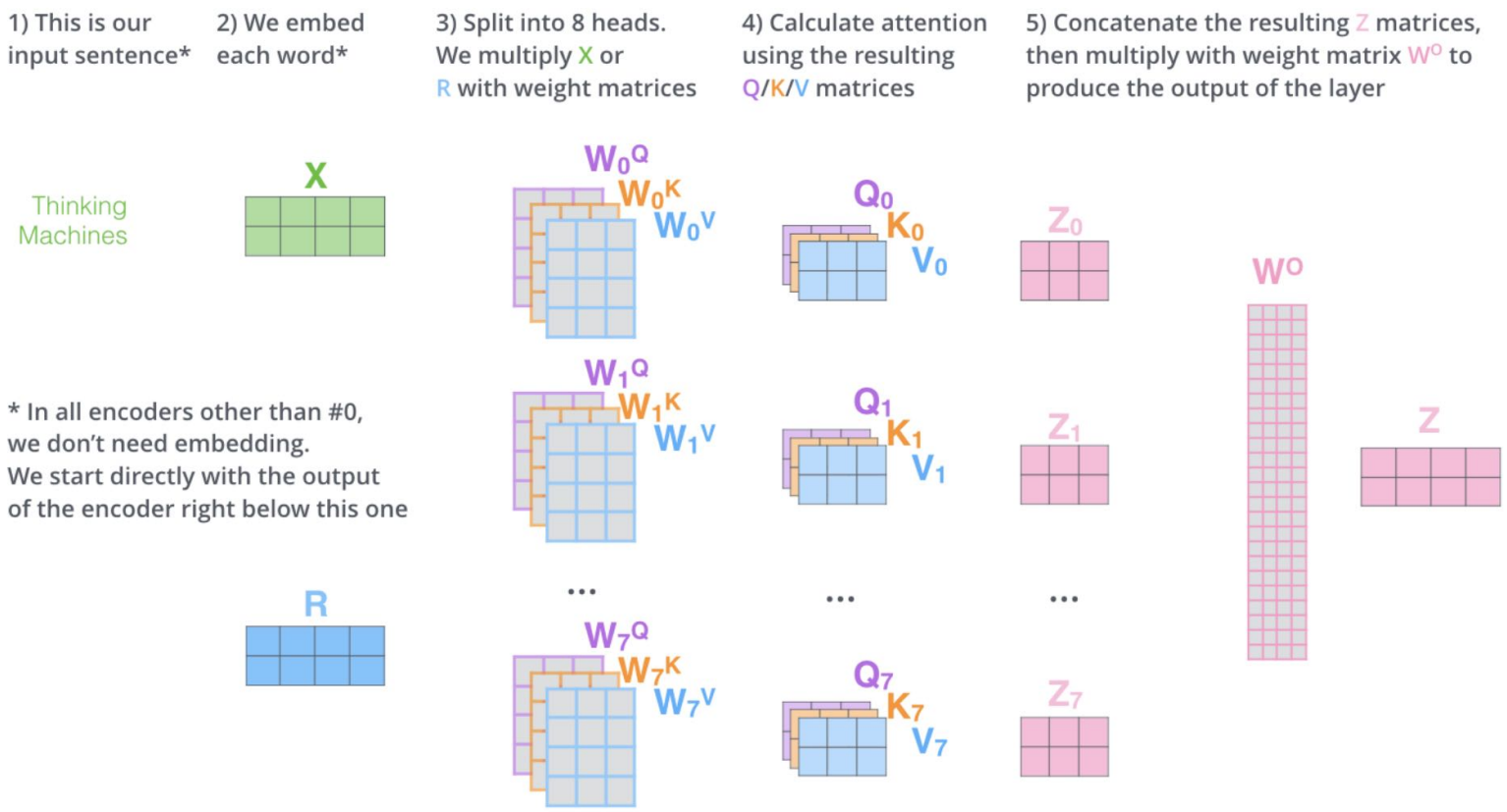


Attention is all you need. Vaswani et al. NeurIPS 2017

slide credit: Yongqin Xian

pictures are from http://jalammar.github.io/illustrated-transformer/

Multi-Head Self-Attention



Attention is all you need. Vaswani et al. NeurIPS 2017

slide credit: Yongqin Xian

Vision Transformer (ViT)

- What are sensible tokens for images?

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021

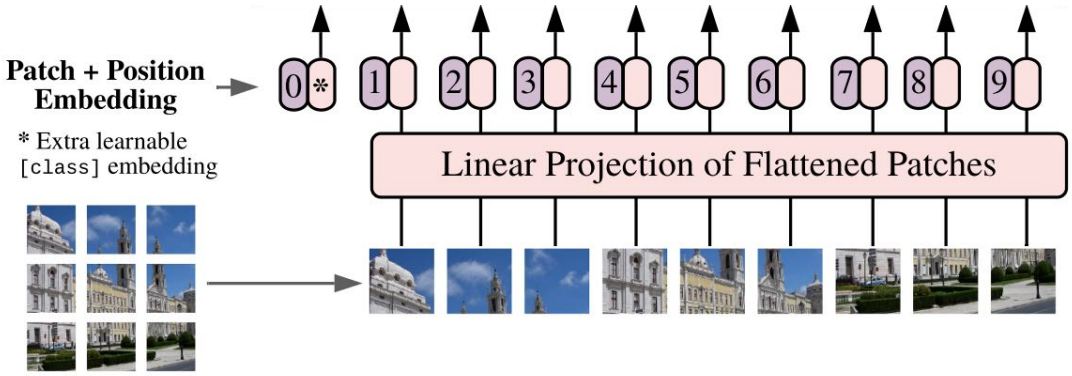
Vision Transformer (ViT)

- What are sensible tokens for images?
  - Image Patches…
  - … here: each image patch of size 16x16

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021

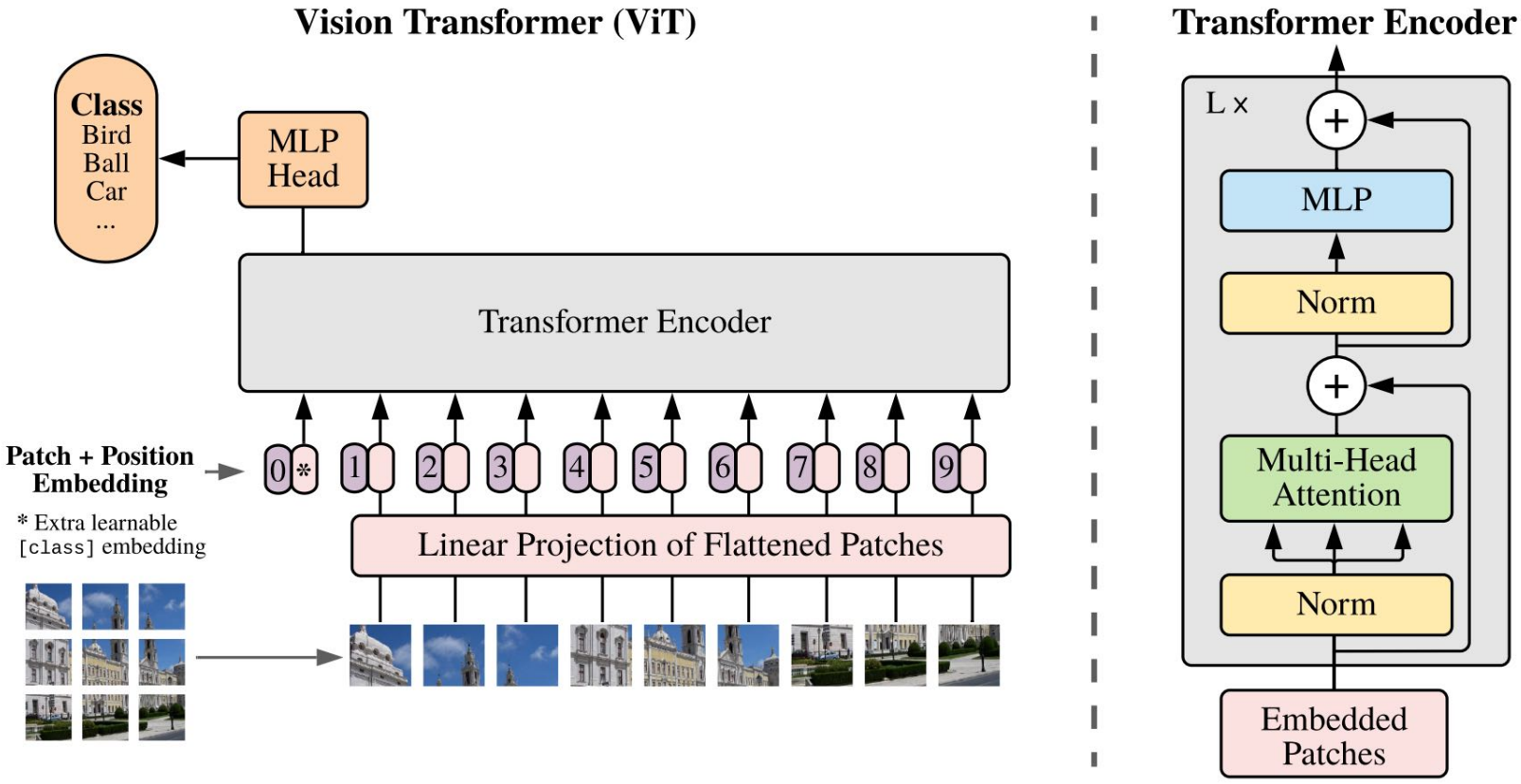
Vision Transformer (ViT)

- Positional encoding
  - learnable embedding for each patch position (sometimes also fixed positional encoding)
- Additional token for “class embedding”
  - learnable, its output embedding used for global image representation (and image classification — see next slide)



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021

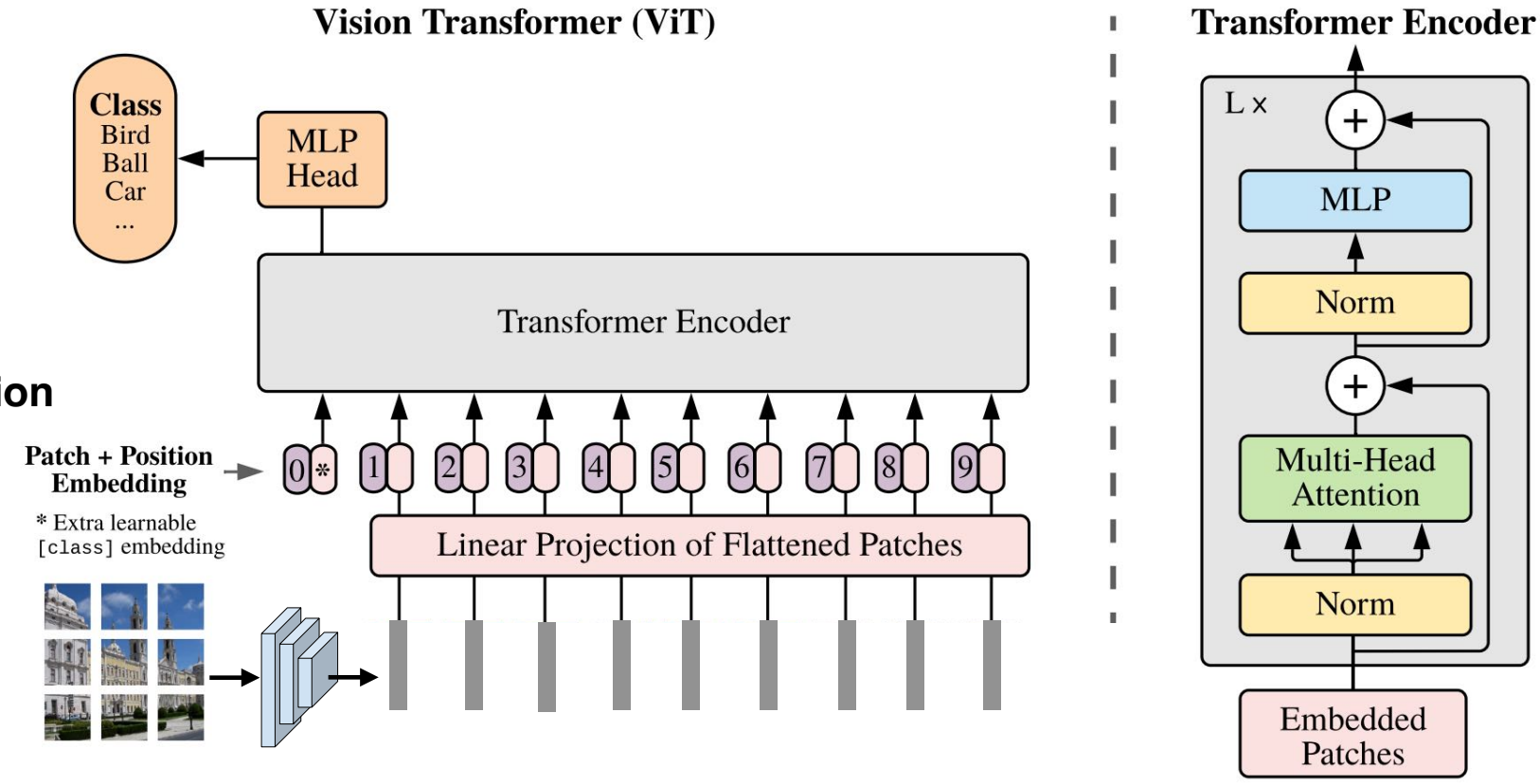
Vision Transformer (ViT)



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021

Hybrid ViT Architecture (with CNN-embedding of image patches)

multimodal:
cross-attention



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021

Overview of Today's Lecture

- Recap of Vision Transformer (ViT) for Image Classification

- Hierarchical Transformer Architectures
  - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows @ ICCV 2021 — https://arxiv.org/abs/2103.14030
  - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers @ NeurIPS 2021 — https://arxiv.org/abs/2105.15203

- Beyond Vanilla Gradient Descent
  - Adam Optimizer (and its components)
  - Second Order Optimization

not popular nowadays but was

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu[1,2†*] Yutong Lin[1,3†*] Yue Cao[1*] Han Hu[1*‡] Yixuan Wei[1,4†]
Zheng Zhang[1] Stephen Lin[1] Baining Guo[1]
[1]Microsoft Research Asia [2]University of Science and Technology of China
[3]Xian Jiaotong University [4]Tsinghua University
{v-zeliu1,v-yutlin,yuecao,hanhu,v-yixwe,zhez,stevelin,bainguo}@microsoft.com

ICCV 2021 — Best Paper Award (Marr Prize)

slide credit: Justin Johnson

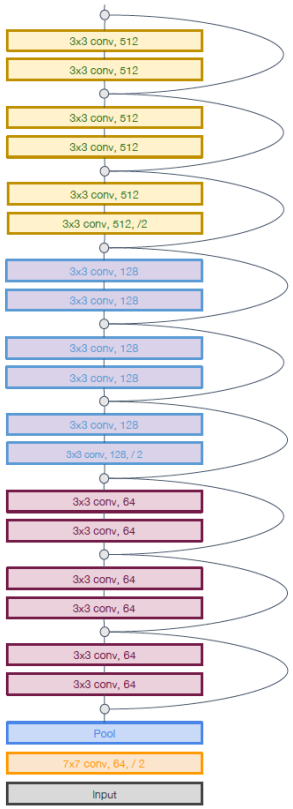
ViT vs CNN

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224

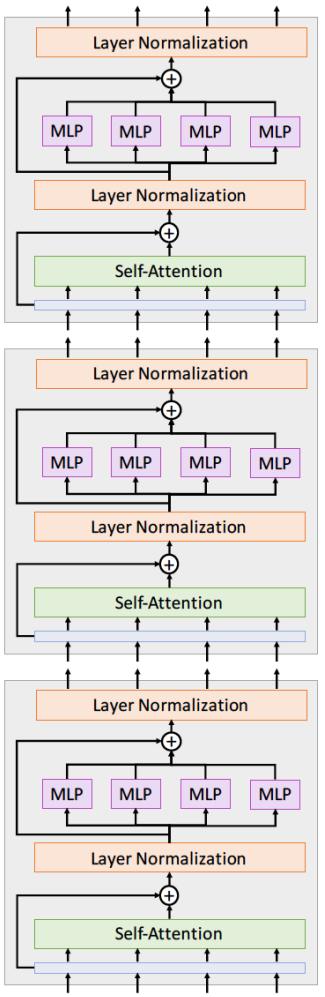


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)

Can we build a **hierarchical** ViT model?



3rd block:
768 x 14 x 14

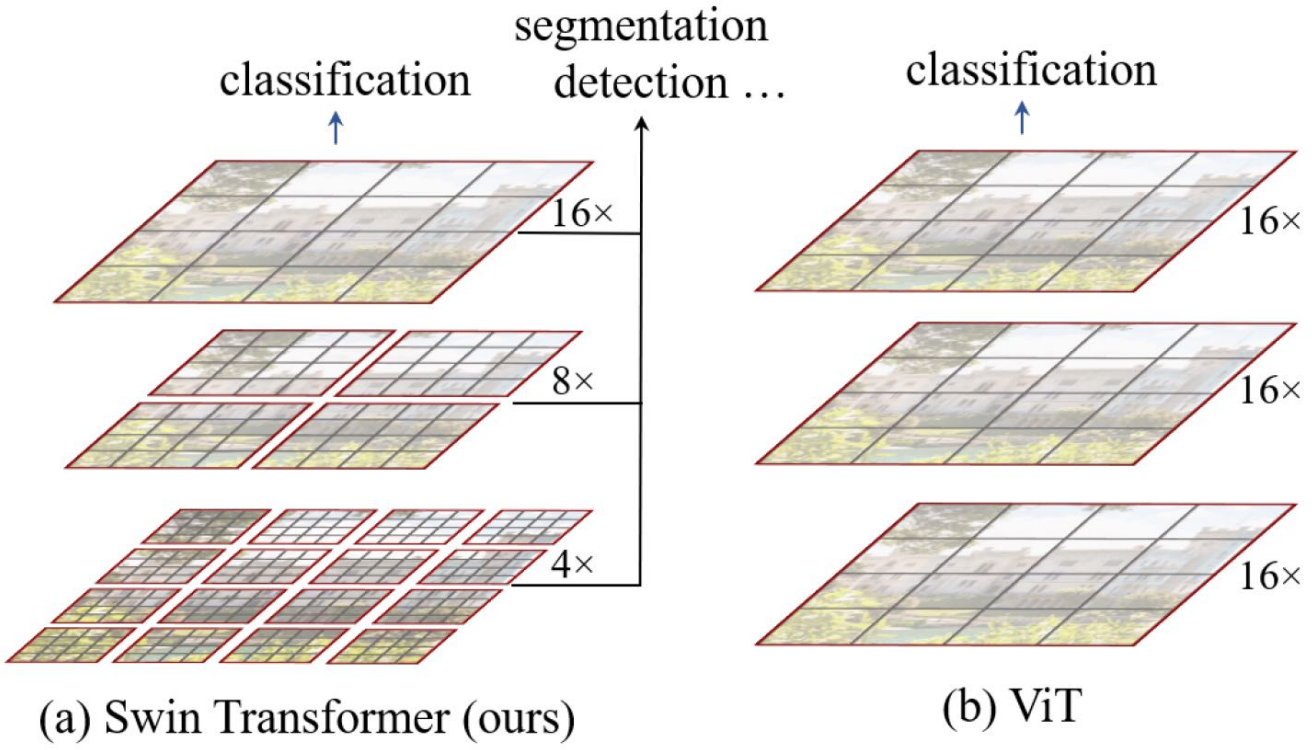
2nd block:
768 x 14 x 14

1st block:
768 x 14 x 14

Input:
3 x 224 x 224

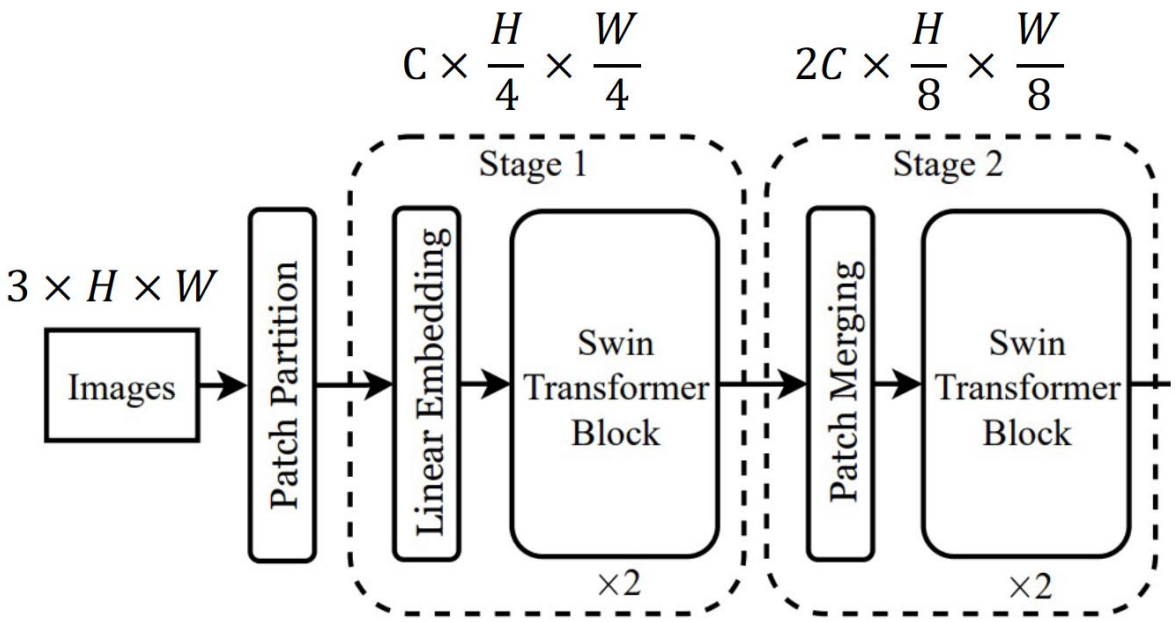
Swin Transformer vs. Standard ViT (Vision Transformer)

- Swin Transformer
  - hierarchical: token size increases from 4x4 to 16x16 pixels
  - architecture can be used for
    - image classification, but also
    - semantic segmentation and
    - object detection
    - instance segmentation
  - computationally more efficient than ViT



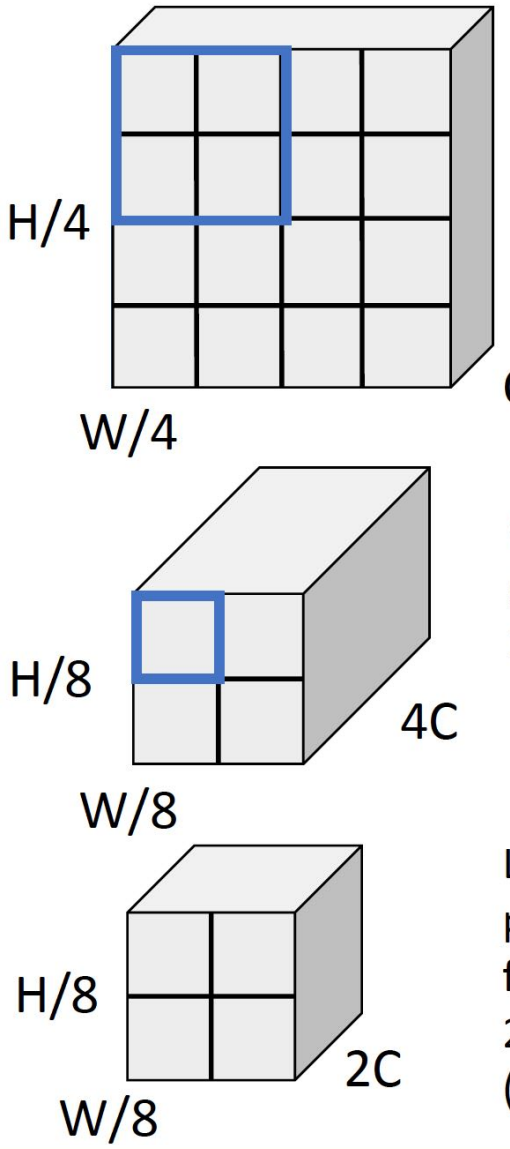
Hierarchical ViT: Swin Transformer

slide credit: Justin Johnson



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8



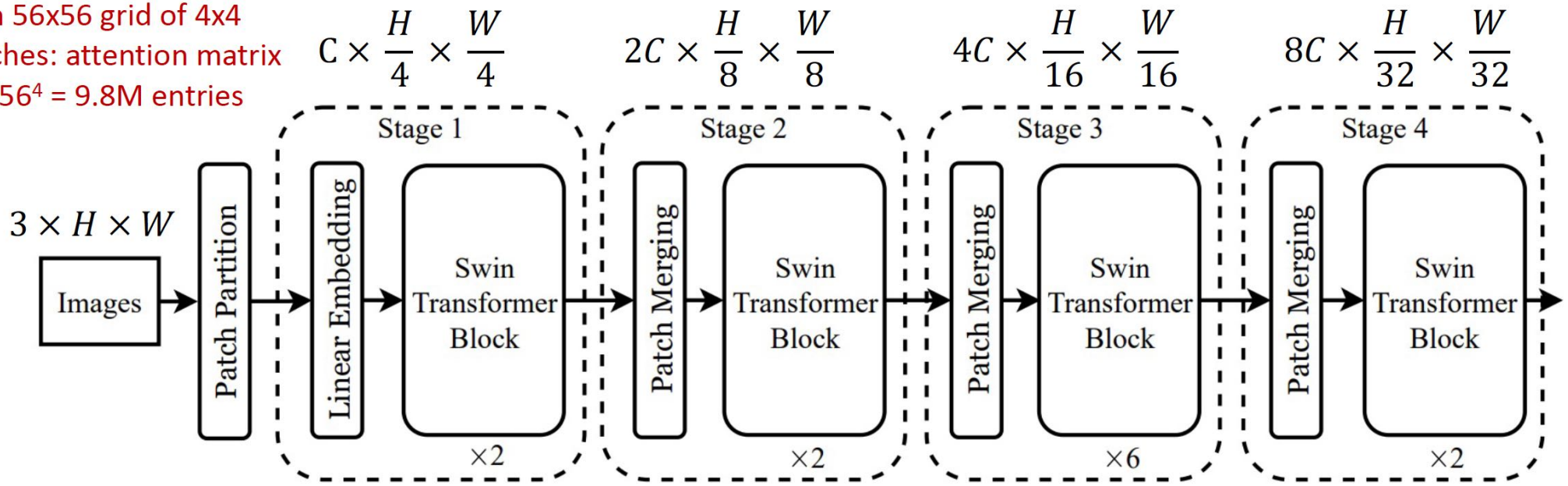
Concatenate groups of 2x2 features

Linear projection from 4C to 2C channels (1x1 conv)

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

Hierarchical ViT: Swin Transformer

Problem: 224x224 image with 56x56 grid of 4x4 patches: attention matrix has 56^4 = 9.8M entries



Solution: don't use full attention, instead use attention over patches

Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson

Swin Transformer: Window Attention

With H x W grid of **tokens**, each attention matrix is H^2W^2 – **quadratic** in image size

Rather than allowing each **token** to attend to all other tokens, instead divide into **windows** of M x M tokens (here M=4); only compute attention within each window

Total size of all attention matrices is now:

$$\underbrace{M^4(H/M)(W/M)}_{\textbf{\# of windows}} = M^2HW$$

Linear in image size for fixed M!

Swin uses M=7 throughout the network

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson

Swin Transformer: Window Attention

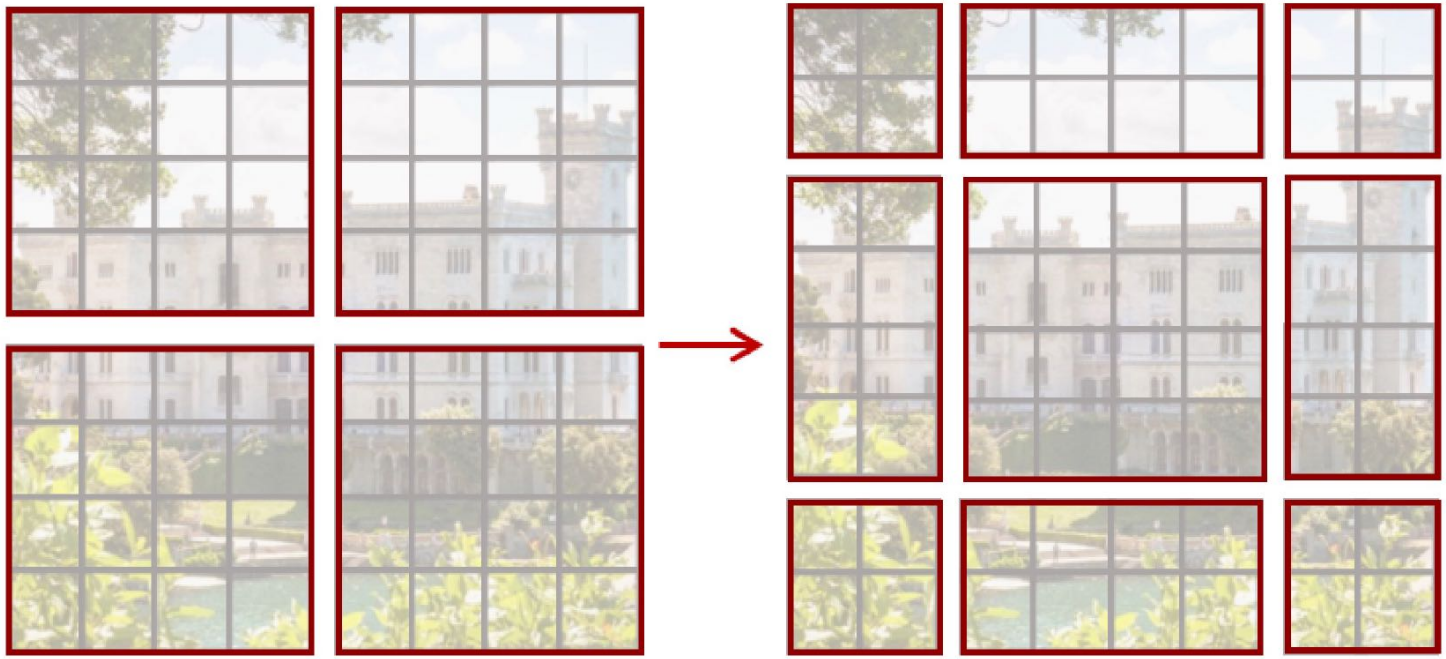
Problem: tokens only interact with other tokens within the same window; no communication across windows

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks



Block L: Normal windows

Block L+1: Shifted Windows

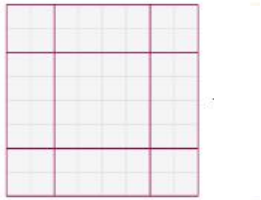
Ugly detail: Non-square windows at edges and corners

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson

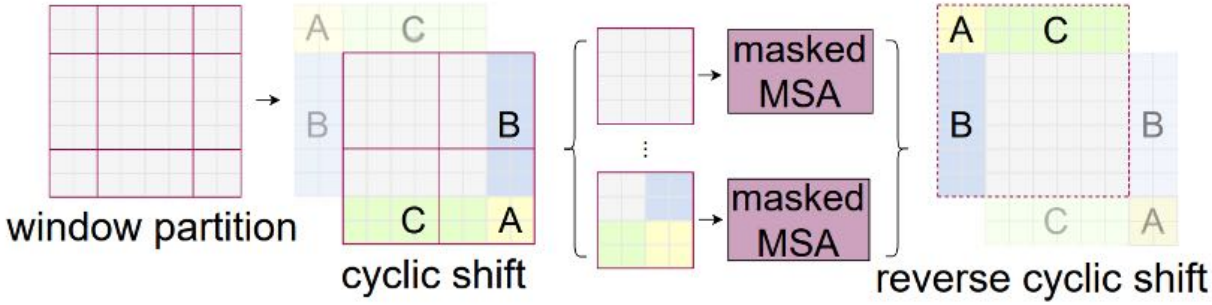
Efficient batch computation for shifted configuration

- First idea
  - add padding… works, but increased computation



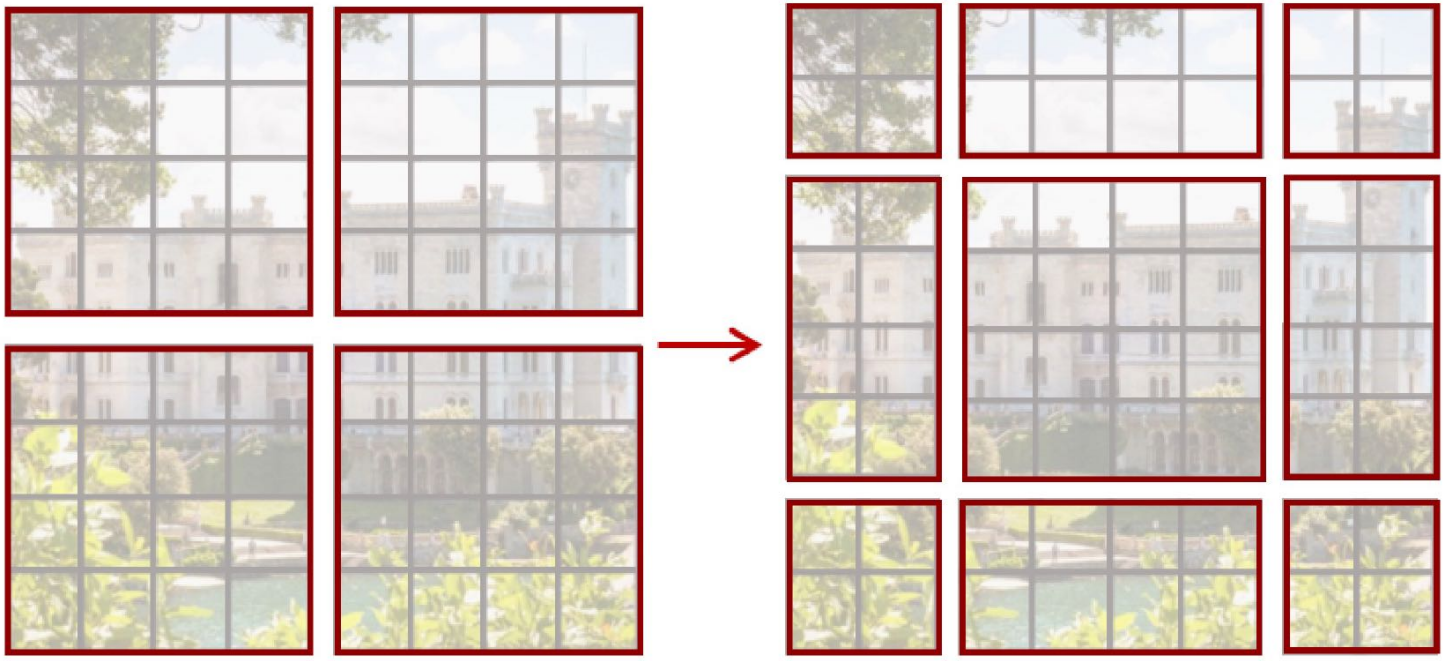
window partition

- Their idea: cyclic-shift:



Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks



Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

Standard Attention:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D}}\right)V$$

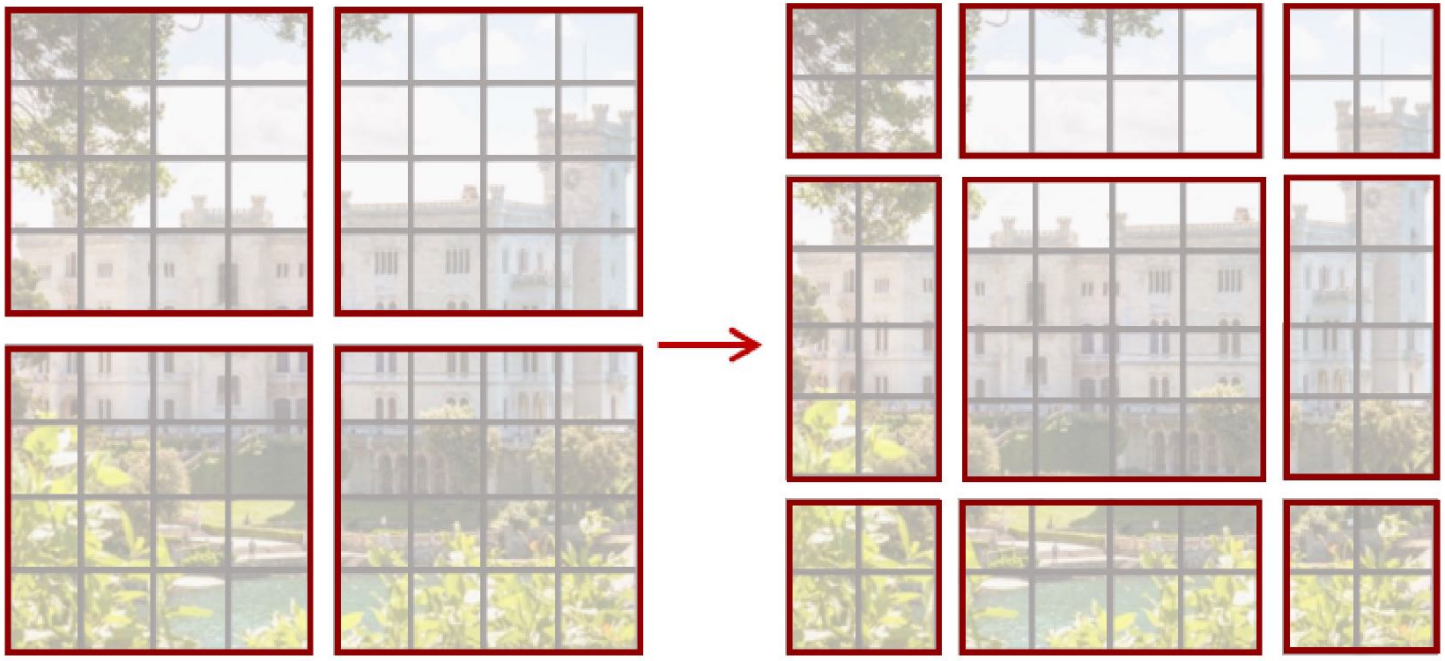
$Q, K, V: M^2 \times D$ (Query, Key, Value)

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks



Block L: Normal windows

Block L+1: Shifted Windows

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

Attention with relative bias:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D}} + B\right)V$$

$Q, K, V: M^2 \times D$ (Query, Key, Value)

$B: M^2 \times M^2$ (learned biases)

relative encoding

slide credit: Justin Johnson

CNN for Semantic Segmentation:

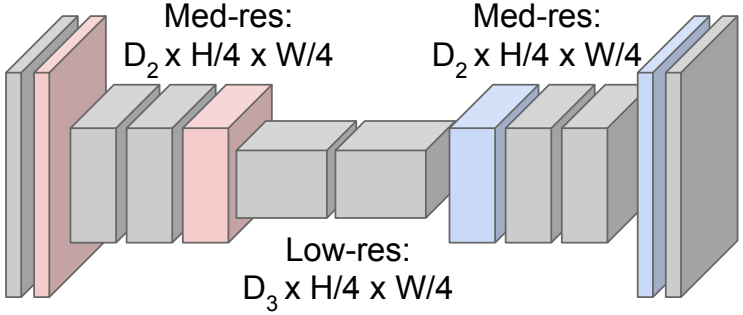
resolution -> lower
channel -> higher

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
Unpooling or strided transpose convolution

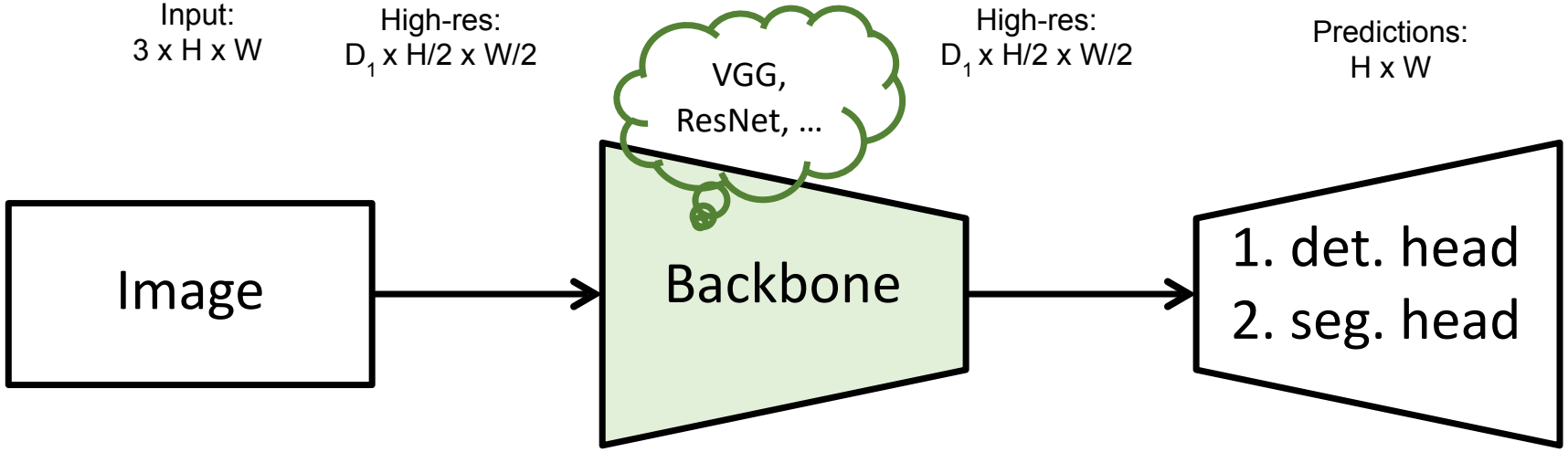
Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

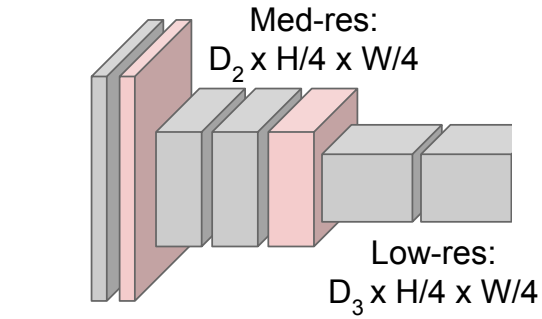
Predictions:
H x W

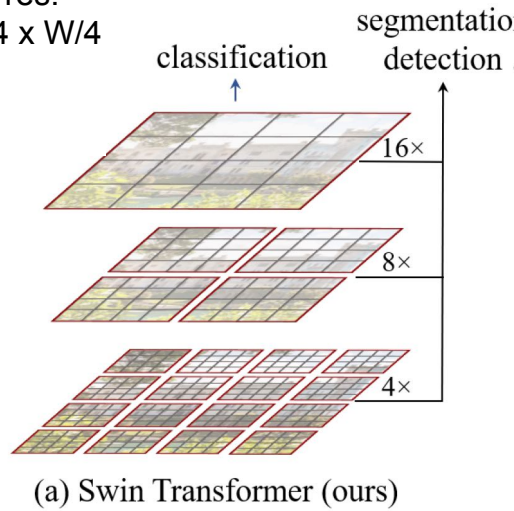


Swin Transformer

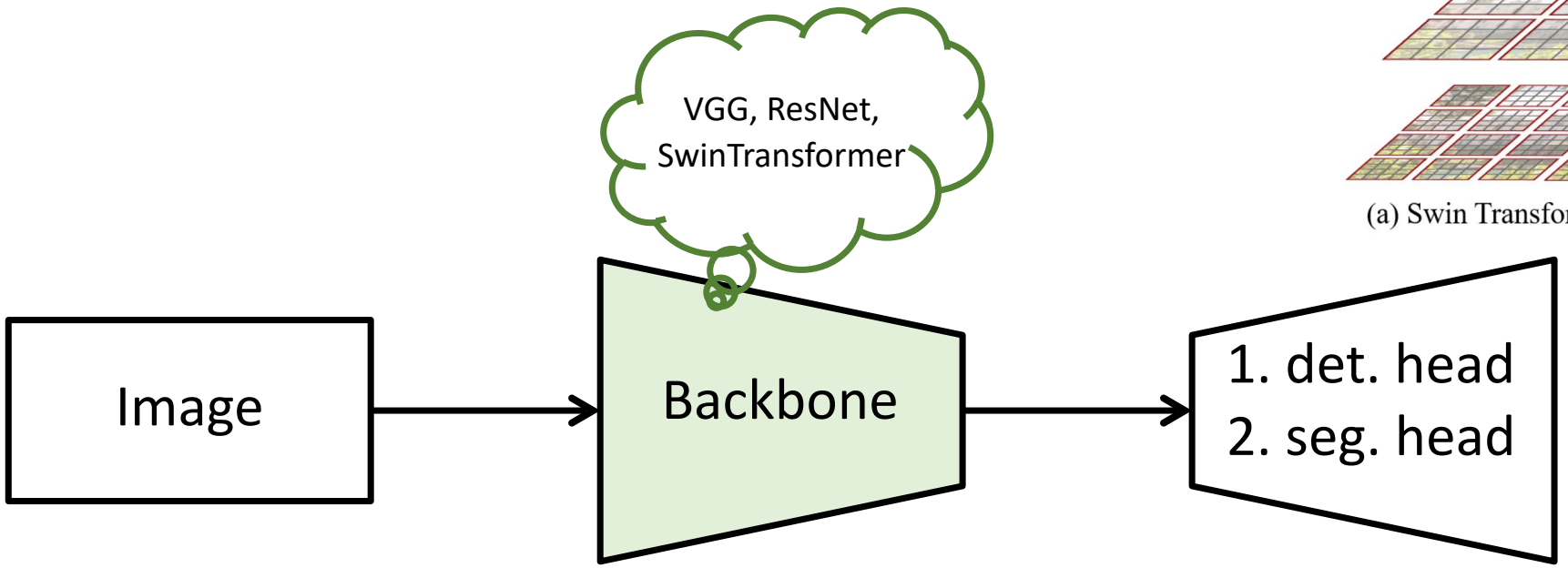
How to apply Swin Transformer to detection and segmentation tasks?

A: Replacing the backbone.





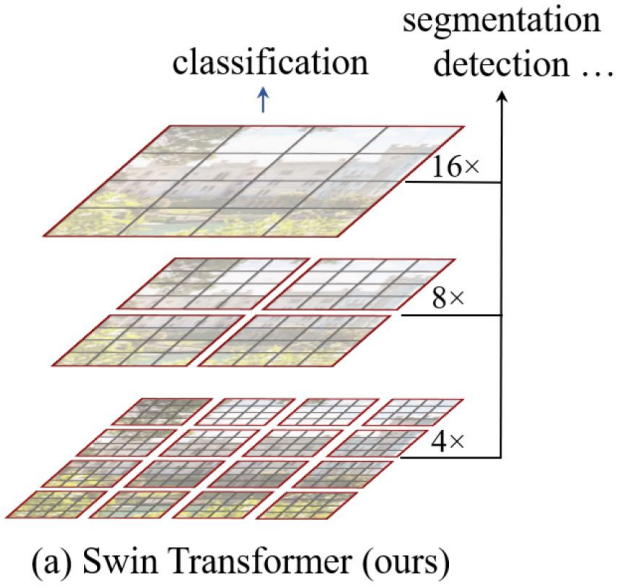
(a) Swin Transformer (ours)



Application to Various Tasks…

- General idea: think of (hierarchical tokens) as feature maps
  - allows to apply standard techniques for
    - detection (e.g. Faster-RNN),
    - instance segmentation (e.g. Mask-RCNN),
    - semantic segmentation (e.g UperNet)

	ImageNet		COCO		ADE20k
	top-1	top-5	AP^{box}	AP^{mask}	mIoU
w/o shifting	80.2	95.1	47.7	41.5	43.3
shifted windows	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**
no pos.	80.1	94.9	49.2	42.6	43.8
abs. pos.	80.5	95.2	49.0	42.4	43.2
abs.+rel. pos.	81.3	95.6	50.2	43.4	44.0
rel. pos. w/o app.	79.3	94.7	48.2	41.9	44.1
rel. pos.	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**



(a) Swin Transformer (ours)

Overview of Today's Lecture

- Recap of Vision Transformer (ViT) for Image Classification

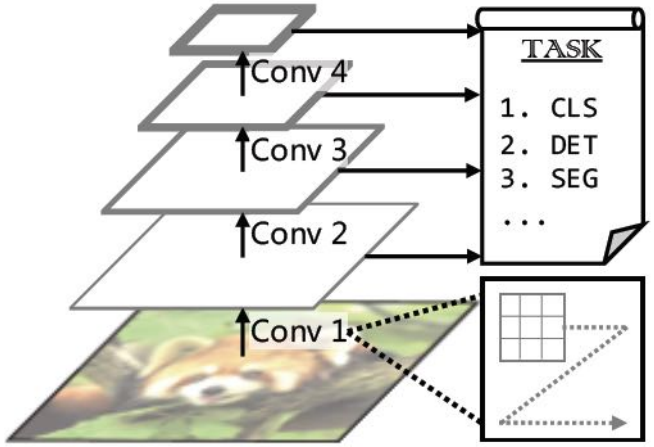
- Hierarchical Transformer Architectures
  - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows @ ICCV 2021 — https://arxiv.org/abs/2103.14030
  - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers @ NeurIPS 2021 — https://arxiv.org/abs/2105.15203

- Beyond Vanilla Gradient Descent
  - Adam Optimizer (and its components)
  - Second Order Optimization

Pyramid Vision Transformer Motivation

[22] He K, et al. "Deep residual learning for image recognition." in CVPR, 2016.
[54] Simonyan K, et al. "Very deep convolutional networks for large-scale image recognition." in ICLR, 2015.
[13] Dosovitskiy A, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." in ICLR 2021.

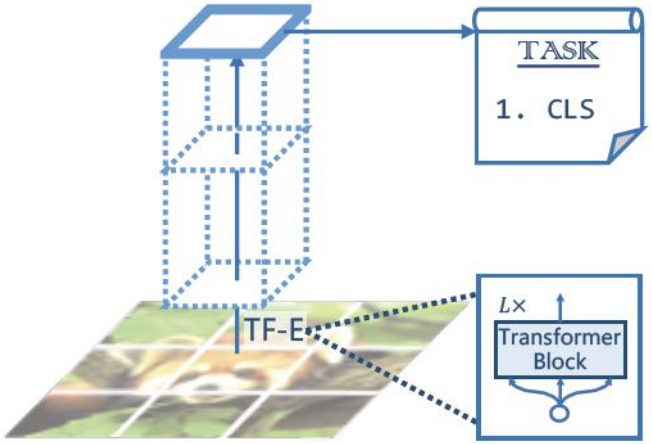
slide credit: Wenhai Wang & Enze Xie



(a) CNNs: VGG [54], ResNet [22], *etc.*

CNN's: Limitations

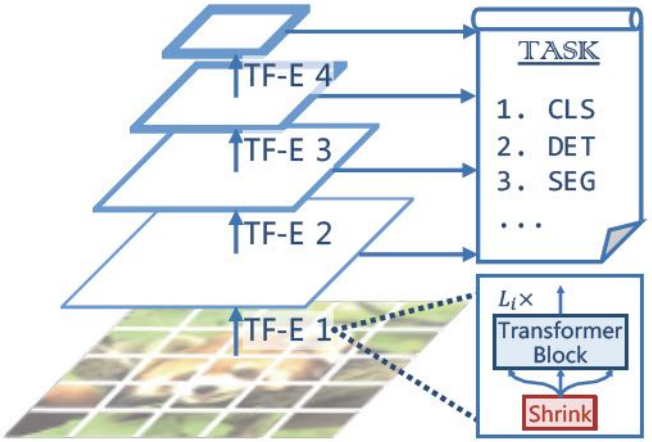
- *Local Receptive Field*
- *Fixed Weights*



(b) Vision Transformer [13]

ViT's Limitations

- *Column Structure*
- *Low-Resolution Output*
- *Unsuitable for DET/SEG*



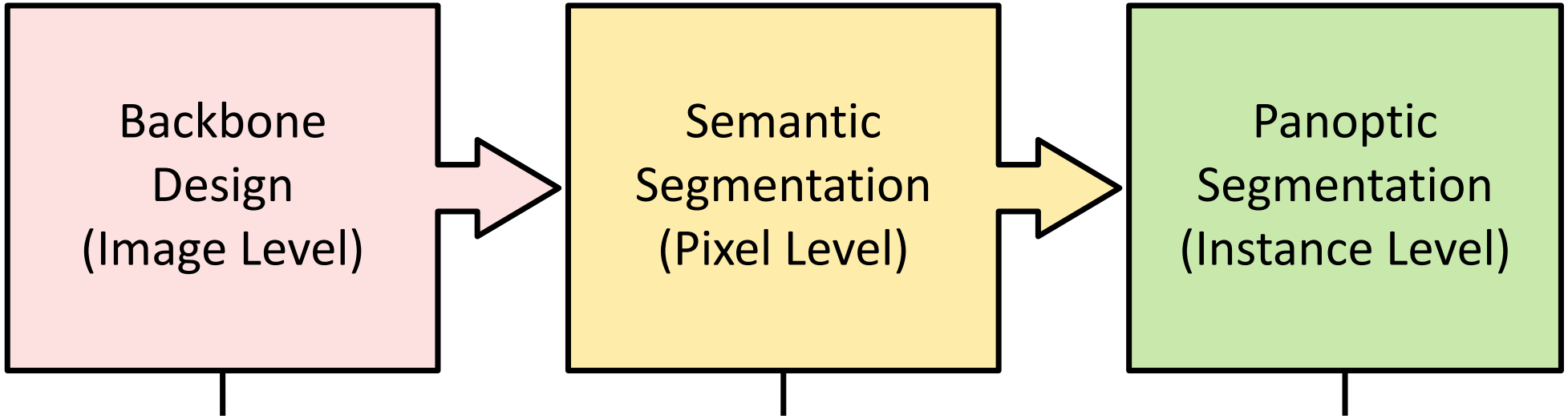
(c) Pyramid Vision Transformer (ours)

PVT (ours)

- *A Transformer backbone as versatile as CNN*

slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer & SegFormer



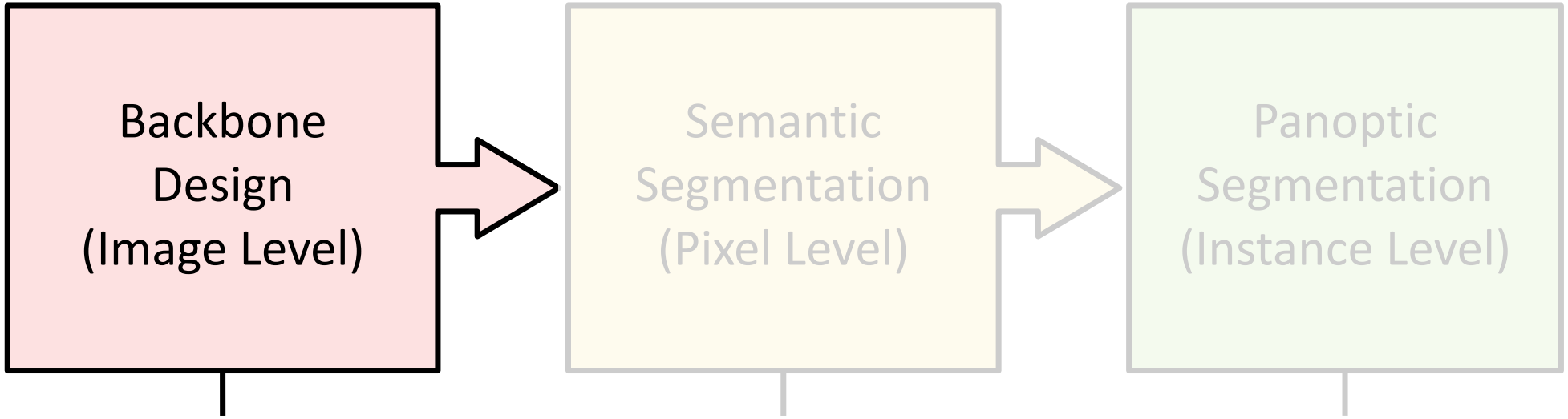
[1] **Pyramid Vision Transformer**: A Versatile Backbone for Dense Prediction without Convolutions. *In ICCV, 2021.**(Oral)***

[2] **SegFormer**: Simple and Efficient Design for Semantic Segmentation with Transformers. *In NeurIPS, 2021.*

[3] Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers. *In CVPR 2022.*

slide credit: Wenhai Wang & Enze Xie

Backbone Design



[1] Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. *In ICCV, 2021.**(Oral)***

[2] SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. *In NeurIPS, 2021.*

[3] Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers. *In CVPR 2022.*

slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer

- **Overall Architecture**

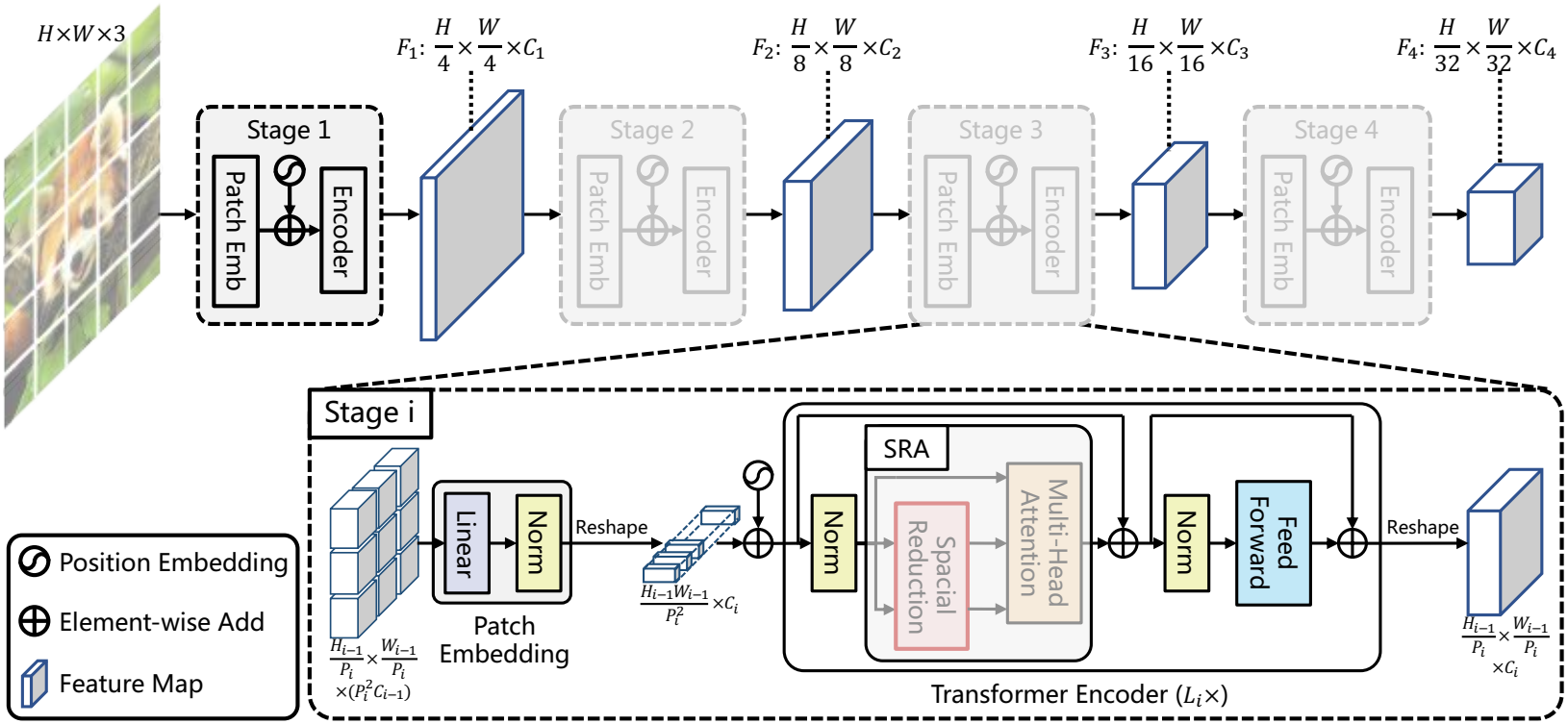
Key Points

- *Pyramid Structure*
- *Four Stages with:*

(1) Patch Emb.

(2) Transformer Enc.

- *Spatial-Reduction Attention (SRA)*

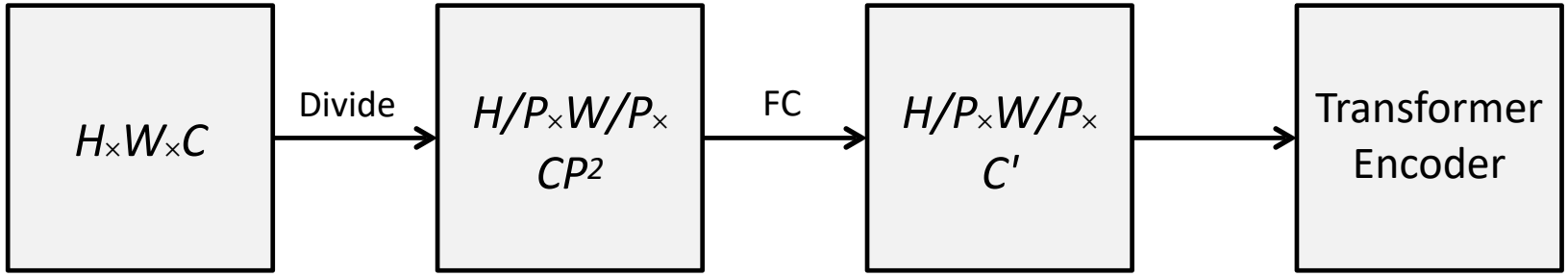


slide credit: Wenhai Wang & Enze Xie

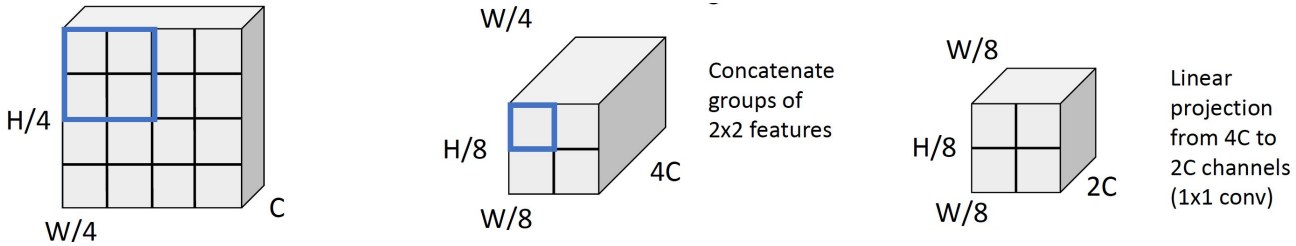
Pyramid Vision Transformer

Q1: How to obtain the **patch encoding**?

A: Patch embeddings with different patch sizes (P)



The process of the patch embedding in Stage *i*.



slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer

Q2: How to **reduce** the **computational cost** of **large number of tokens**?

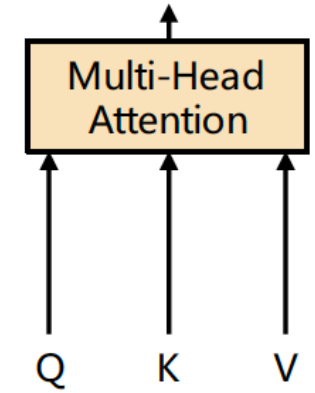
A: **Spatial-Reduction Attention (SRA)**

$$\mathrm{SRA}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_0, ..., \mathrm{head}_{N_i})W^O$$

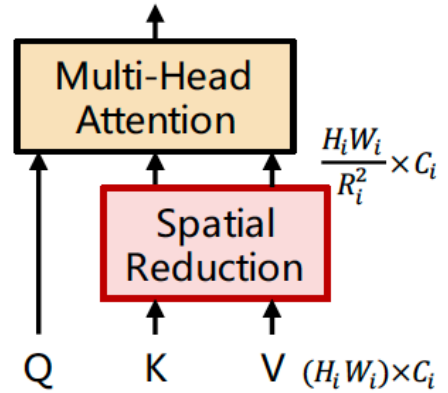
$$\mathrm{head}_j = \mathrm{Attention}(QW_j^Q, \mathrm{SR}(K)W_j^K, \mathrm{SR}(V)W_j^V)$$

$$\mathrm{SR}(\mathbf{x}) = \mathrm{Norm}(\mathrm{Reshape}(\mathbf{x}, R_i)W^S)$$

$$\mathrm{Attention}(\mathbf{q},\mathbf{k},\mathbf{v}) = \mathrm{Softmax}(\frac{\mathbf{q}\mathbf{k}^\mathsf{T}}{\sqrt{d_{\mathbf{head}}}})\mathbf{v}$$



Multi-Head Attention



Spatial-Reduction Attention (ours)

Compared to original multi-head attention, *the complexity of SRA is R_i^2 times lower!*

slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer

Q2: How to **reduce** the **computational cost** of **large number of tokens**?

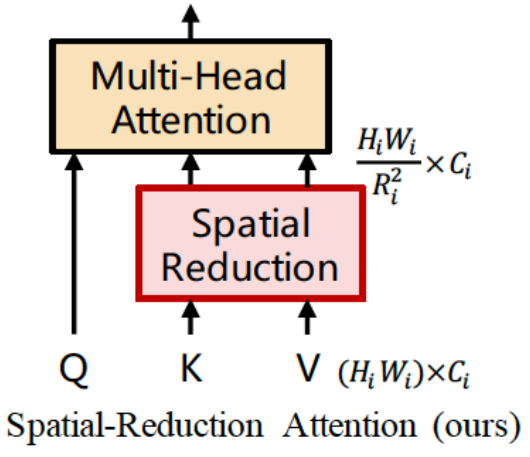
A: **Spatial-Reduction Attention (SRA)**

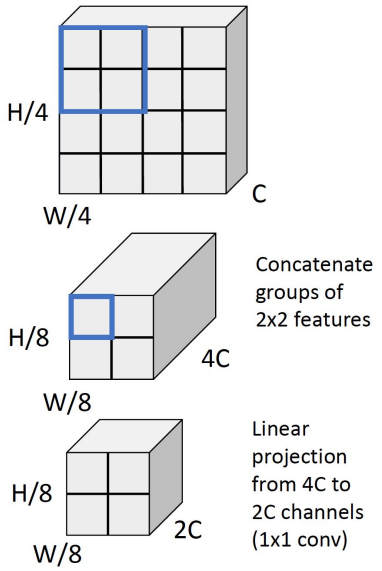
$$\mathrm{SRA}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_0, ..., \mathrm{head}_{N_i})W^O$$

$$\mathrm{head}_j = \mathrm{Attention}(QW_j^Q, \mathrm{SR}(K)W_j^K, \mathrm{SR}(V)W_j^V)$$

$$\mathrm{SR}(\mathbf{x}) = \mathrm{Norm}(\mathrm{Reshape}(\mathbf{x}, R_i)W^S)$$

$$\mathrm{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \mathrm{Softmax}(\frac{\mathbf{q}\mathbf{k}^\mathsf{T}}{\sqrt{d_{\mathbf{head}}}})\mathbf{v}$$





Compared to original multi-head attention, *the complexity of SRA is R_i^2 times lower!*

Pyramid Vision Transformer

- Detailed Settings

- P_i: the patch size of the stage i;
- C_i: the channel number of the output of the stage i;
- L_i: the number of encoder layers in the stage i;
- R_i: the reduction ratio of the SRA in the stage i;
- N_i: the head number of the SRA in the stage i;
- E_i: the expansion ratio of the feed-forward layer [51] in the stage i;

slide credit: Wenhai Wang & Enze Xie

	Output Size	Layer Name	PVT-Tiny	PVT-Small	PVT-Medium	PVT-Large
Stage 1	$\frac{H}{4} \times \frac{W}{4}$	Patch Embedding	$P_1 = 4;\ C_1 = 64$			
		Transformer Encoder	$\begin{bmatrix} R_1 = 8 \\ N_1 = 1 \\ E_1 = 8 \end{bmatrix} \times 2$	$\begin{bmatrix} R_1 = 8 \\ N_1 = 1 \\ E_1 = 8 \end{bmatrix} \times 3$	$\begin{bmatrix} R_1 = 8 \\ N_1 = 1 \\ E_1 = 8 \end{bmatrix} \times 3$	$\begin{bmatrix} R_1 = 8 \\ N_1 = 1 \\ E_1 = 8 \end{bmatrix} \times 3$
Stage 2	$\frac{H}{8} \times \frac{W}{8}$	Patch Embedding	$P_2 = 2;\ C_2 = 128$			
		Transformer Encoder	$\begin{bmatrix} R_2 = 4 \\ N_2 = 2 \\ E_2 = 8 \end{bmatrix} \times 2$	$\begin{bmatrix} R_2 = 4 \\ N_2 = 2 \\ E_2 = 8 \end{bmatrix} \times 3$	$\begin{bmatrix} R_2 = 4 \\ N_2 = 2 \\ E_2 = 8 \end{bmatrix} \times 3$	$\begin{bmatrix} R_2 = 4 \\ N_2 = 2 \\ E_2 = 8 \end{bmatrix} \times 8$
Stage 3	$\frac{H}{16} \times \frac{W}{16}$	Patch Embedding	$P_3 = 2;\ C_3 = 320$			
		Transformer Encoder	$\begin{bmatrix} R_3 = 2 \\ N_3 = 5 \\ E_3 = 4 \end{bmatrix} \times 2$	$\begin{bmatrix} R_3 = 2 \\ N_3 = 5 \\ E_3 = 4 \end{bmatrix} \times 6$	$\begin{bmatrix} R_3 = 2 \\ N_3 = 5 \\ E_3 = 4 \end{bmatrix} \times 18$	$\begin{bmatrix} R_3 = 2 \\ N_3 = 5 \\ E_3 = 4 \end{bmatrix} \times 27$
Stage 4	$\frac{H}{32} \times \frac{W}{32}$	Patch Embedding	$P_4 = 2;\ C_4 = 512$			
		Transformer Encoder	$\begin{bmatrix} R_4 = 1 \\ N_4 = 8 \\ E_4 = 4 \end{bmatrix} \times 2$	$\begin{bmatrix} R_4 = 1 \\ N_4 = 8 \\ E_4 = 4 \end{bmatrix} \times 3$	$\begin{bmatrix} R_4 = 1 \\ N_4 = 8 \\ E_4 = 4 \end{bmatrix} \times 3$	$\begin{bmatrix} R_4 = 1 \\ N_4 = 8 \\ E_4 = 4 \end{bmatrix} \times 3$

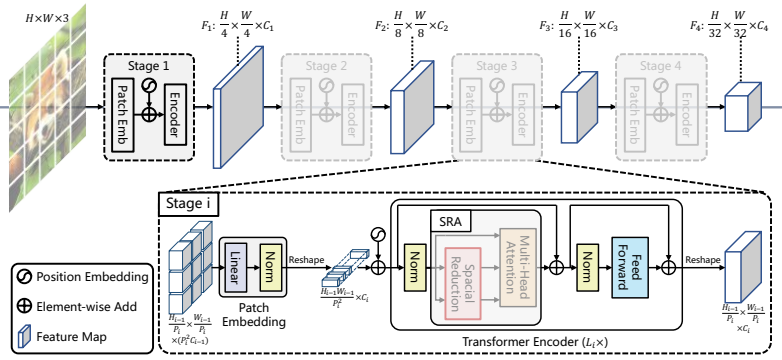
Table A1: **Detailed settings of Pyramid Vision Transformer (PVT) series.** The design follows the two rules of ResNet [5]. (1) With the growth of network depth, the hidden dimension gradually increases, and the output resolution progressively shrinks; (2) The major computation resource is concentrated in Stage 3.

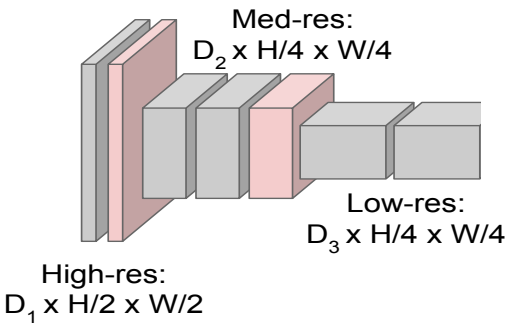
slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer

Q3: How to **apply PVT to detection** and **segmentation** tasks?

A: Replacing the backbone.



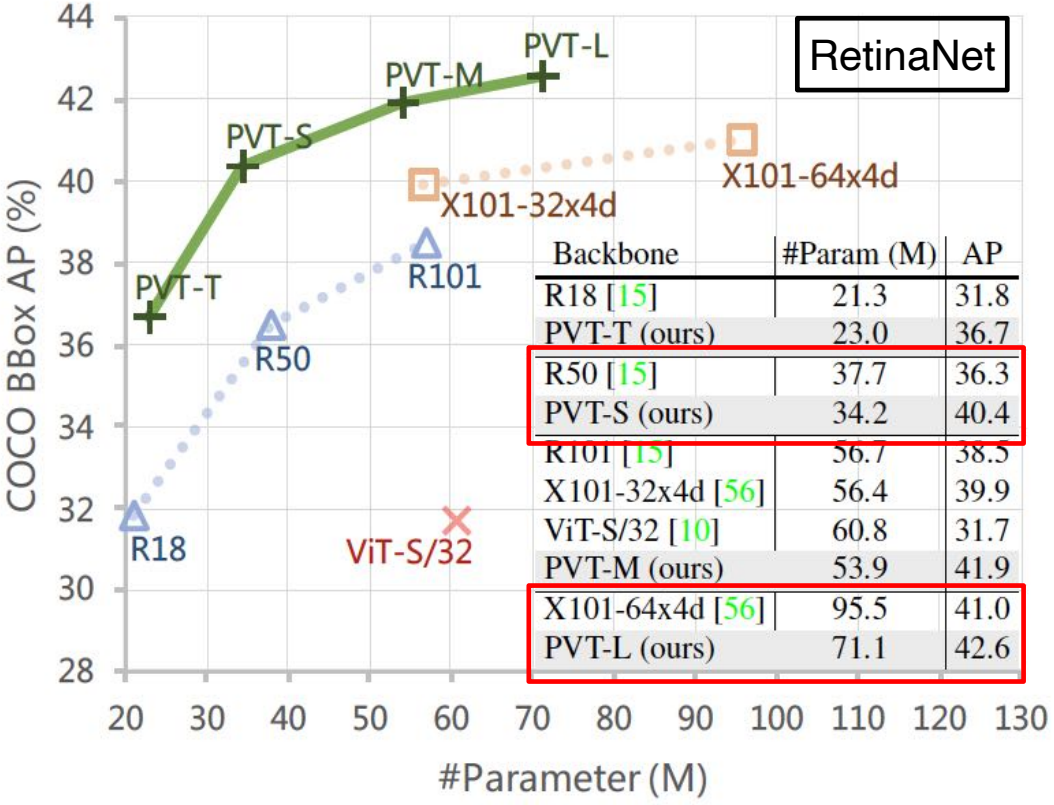


slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer

- Performance
  - Left: Object Detection (RetinaNet) — Right: Semantic Segmentation (Semantic FPN)

Detection on MS COCO



Backbone	#Param (M)	AP
R18 [15]	21.3	31.8
PVT-T (ours)	23.0	36.7
R50 [15]	37.7	36.3
PVT-S (ours)	34.2	40.4
R101 [15]	56.7	38.5
X101-32x4d [56]	56.4	39.9
ViT-S/32 [10]	60.8	31.7
PVT-M (ours)	53.9	41.9
X101-64x4d [56]	95.5	41.0
PVT-L (ours)	71.1	42.6

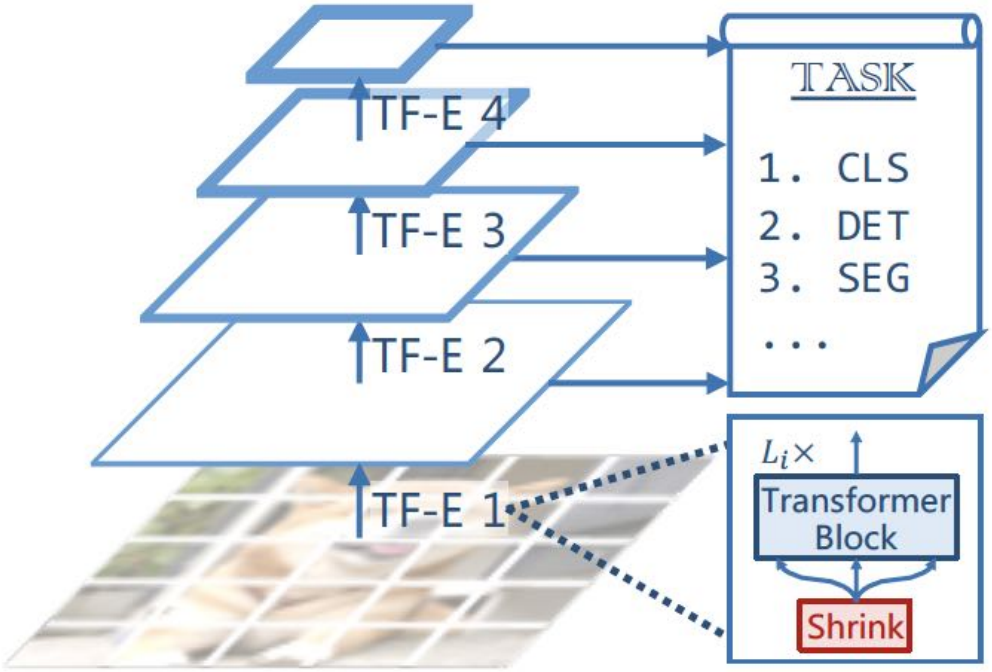
Backbone	Semantic FPN		
	#Param (M)	GFLOPs	mIoU (%)
ResNet18 [22]	15.5	32.2	32.9
PVT-Tiny (ours)	17.0	33.2	35.7(+2.8)
ResNet50 [22]	28.5	45.6	36.7
PVT-Small (ours)	28.2	44.5	39.8(+3.1)
ResNet101 [22]	47.5	65.1	38.8
ResNeXt101-32x4d [73]	47.1	64.7	39.7(+0.9)
PVT-Medium (ours)	48.0	61.0	41.6(+2.8)
ResNeXt101-64x4d [73]	86.4	103.9	40.2
PVT-Large (ours)	65.1	79.6	42.1(+1.9)
PVT-Large* (ours)	65.1	79.6	44.8

Segmentation on ADE20K

slide credit: Wenhai Wang & Enze Xie

Pyramid Vision Transformer

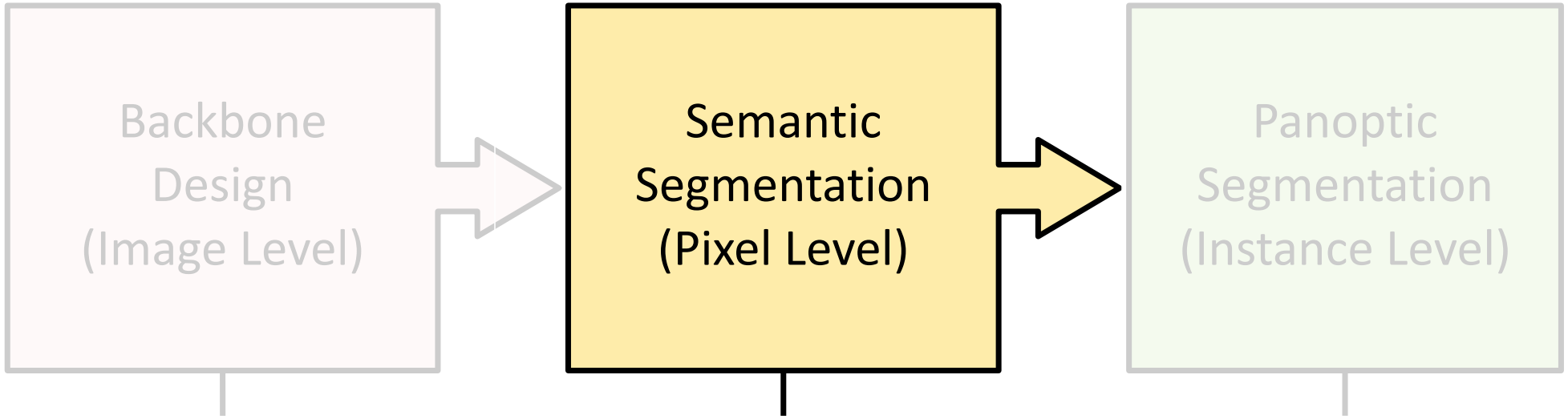
- Advantages
  - *Multi-Scale/High-Resolution Output*
  - *As versatile as CNN, can be applied to detection/segmentation*
  - *Making pure Transformer detection/ segmentation possible, for example*
    - *(1) PVT + DETR [1]*
    - *(2) PVT + Trans2Seg [2]*



[1] Carion N, et al. "End-to-end object detection with transformers." in ECCV 2020.
[2] Xie E, et al. "Segmenting transparent object in the wild with transformer." in IJCAI 2021.

slide credit: Wenhai Wang & Enze Xie

Semantic Segmentation



[1] Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. *In ICCV, 2021.**(Oral)***

[2] SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. *In NeurIPS, 2021.*

[3] Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers. *In CVPR 2022.*

slide credit: Wenhai Wang & Enze Xie

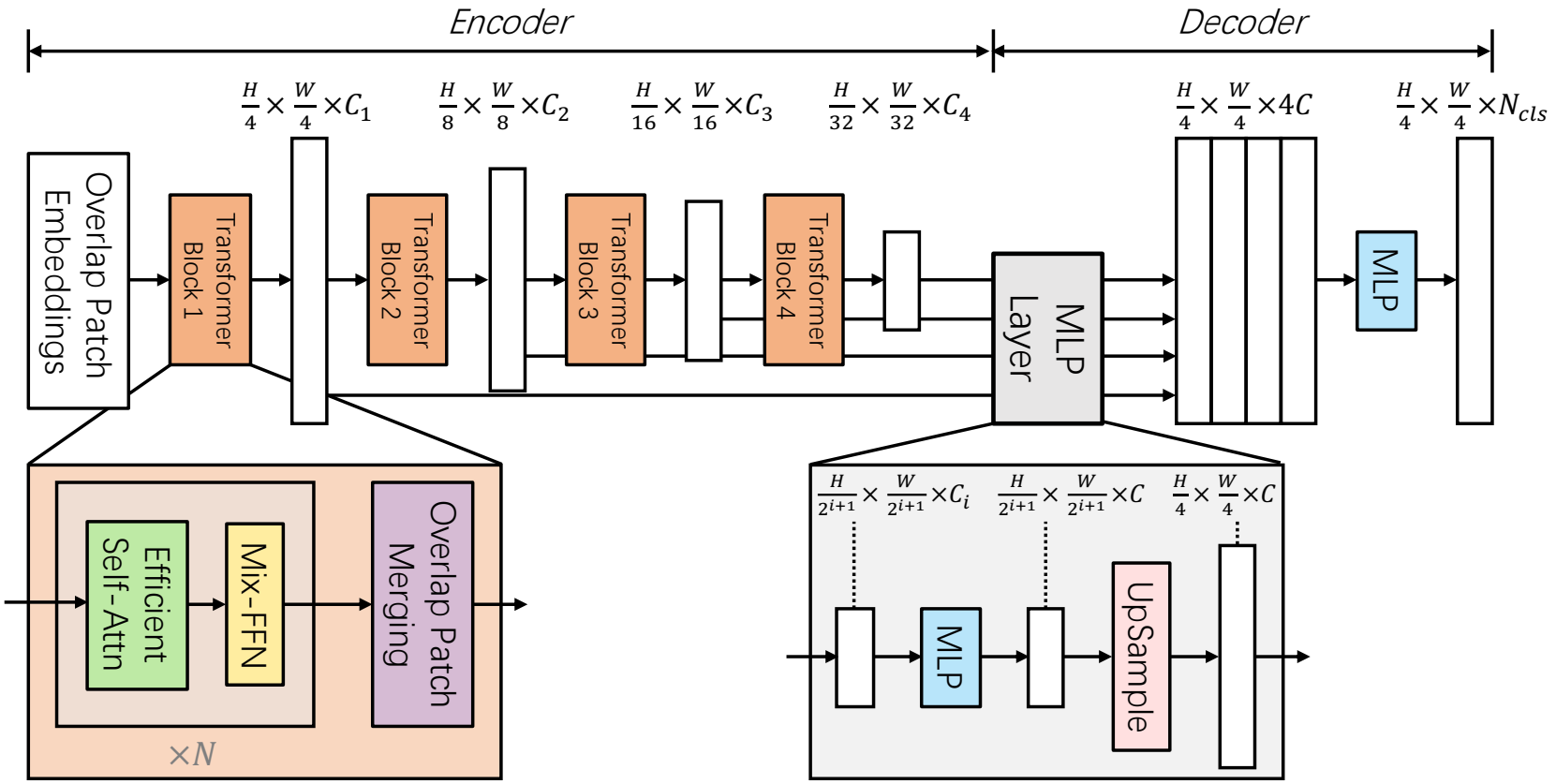
SegFormer — in particular adds MLP-based decoder…

- Overall Architecture

Key Points

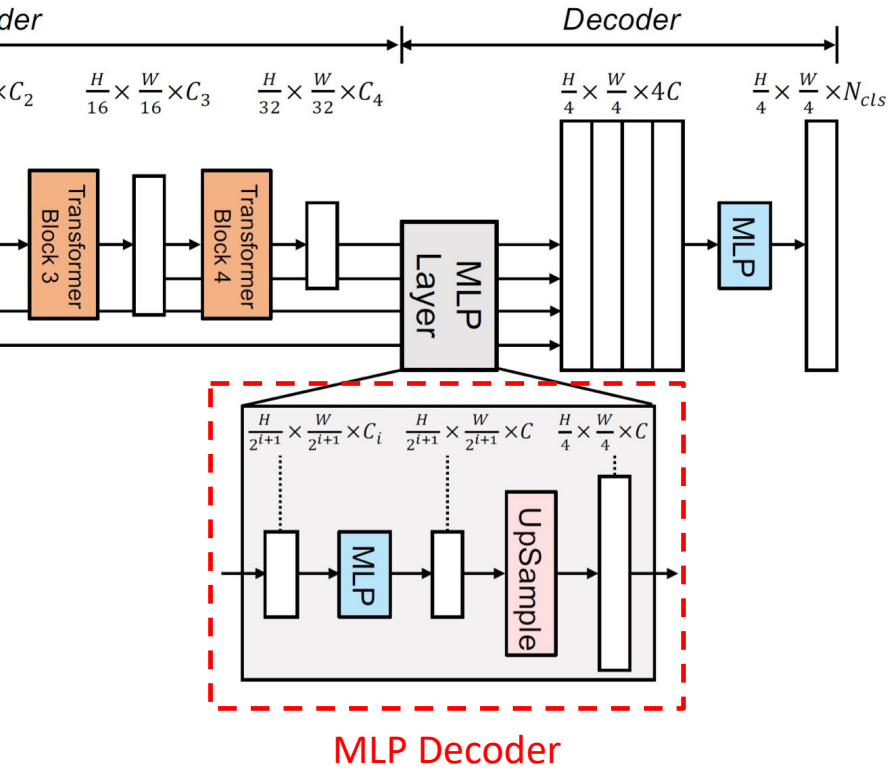
- *Encoder: PVT (No Pos. Encoding, but Mix-FFN (3x3 Conv + FFN))*
- *MLP decoder & MLP head is sufficient for seg.*

Due to the large ERF given by Transformer.



slide credit: Wenhai Wang & Enze Xie

A Closer Look at MLP Decoder



MLP Decoder

$$\hat{F}_i = \text{Linear}(C_i, C)(F_i), \forall i$$

$$\hat{F}_i = \text{Upsample}(\frac{W}{4} \times \frac{W}{4})(\hat{F}_i), \forall i$$

$$F = \text{Linear}(4C, C)(\text{Concat}(\hat{F}_i)), \forall i$$

$$M = \text{Linear}(C, N_{cls})(F),$$

1. unify to C channel dimensions for all transformer blocks

2. upsample

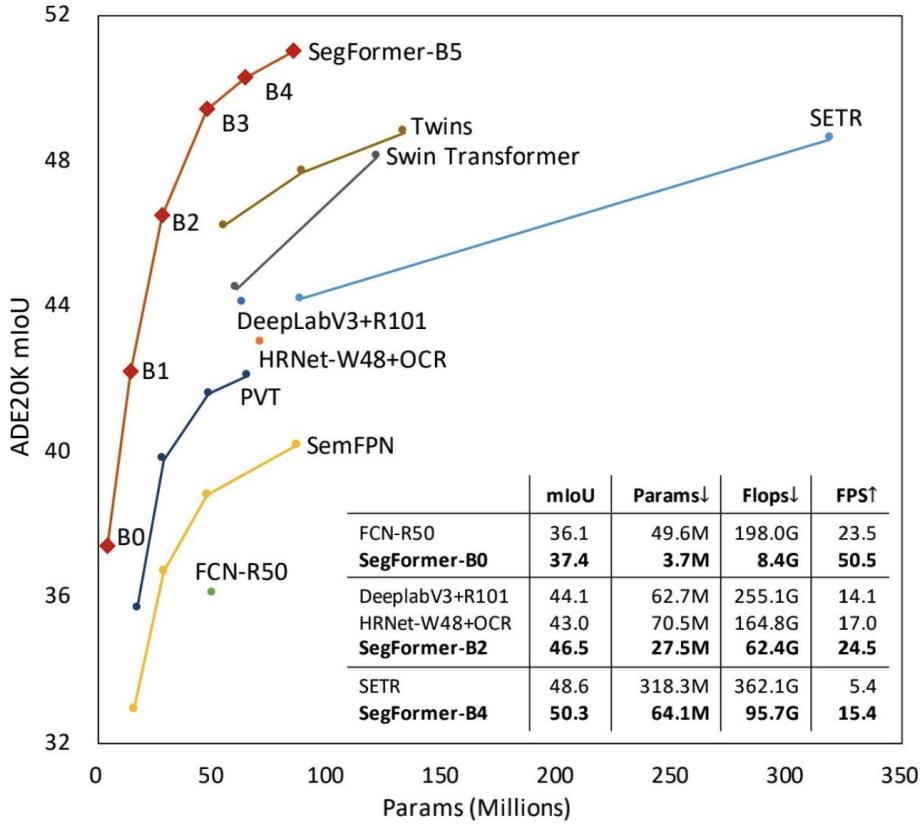
3. concatenate all blocks (4C channels) & project to C channels

4. predict the semantic classes from C channels

slide credit: Wenhai Wang & Enze Xie

SegFormer

- **Performance**



	mIoU	Params↓	Flops↓	FPS↑
FCN-R50	36.1	49.6M	198.0G	23.5
SegFormer-B0	**37.4**	**3.7M**	**8.4G**	**50.5**
DeeplabV3+R101	44.1	62.7M	255.1G	14.1
HRNet-W48+OCR	43.0	70.5M	164.8G	17.0
SegFormer-B2	**46.5**	**27.5M**	**62.4G**	**24.5**
SETR	48.6	318.3M	362.1G	5.4
SegFormer-B4	**50.3**	**64.1M**	**95.7G**	**15.4**

SegFormer

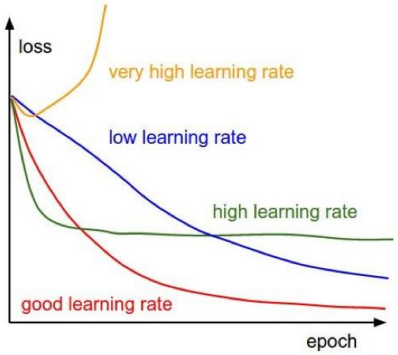
SETR

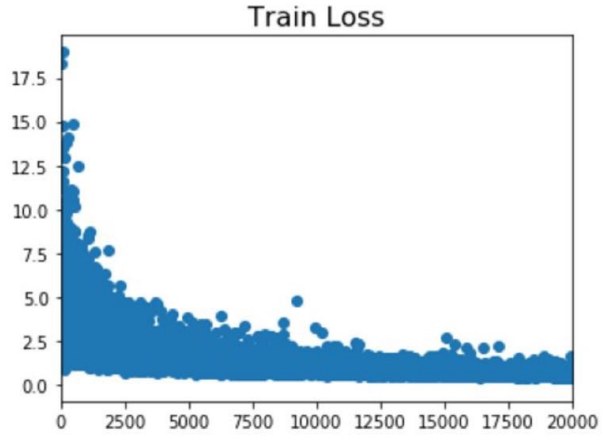
DeepLabV3+

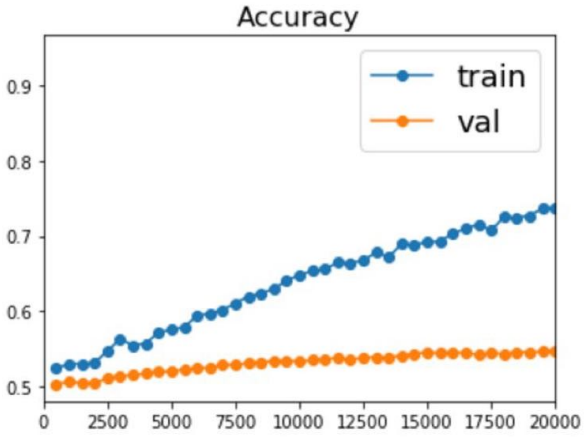
Overview of Today's Lecture

- Recap of Vision Transformer (ViT) for Image Classification
- Hierarchical Transformer Architectures
  - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows @ ICCV 2021 — https://arxiv.org/abs/2103.14030
  - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers @ NeurIPS 2021 — https://arxiv.org/abs/2105.15203
- Beyond Vanilla Gradient Descent
  - Adam Optimizer (and its components)
  - Second Order Optimization

Babysitting Training

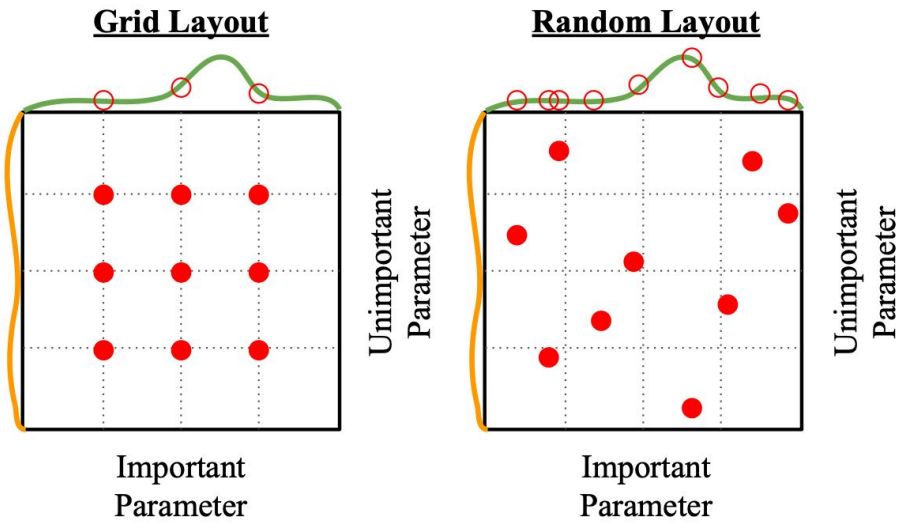






slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Hyperparameter Search



Coarse to fine search

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

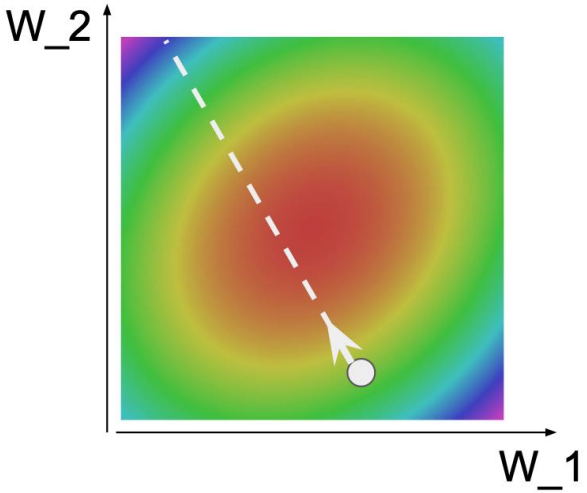
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



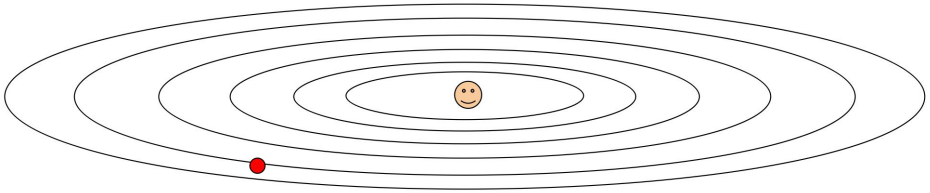
SGD

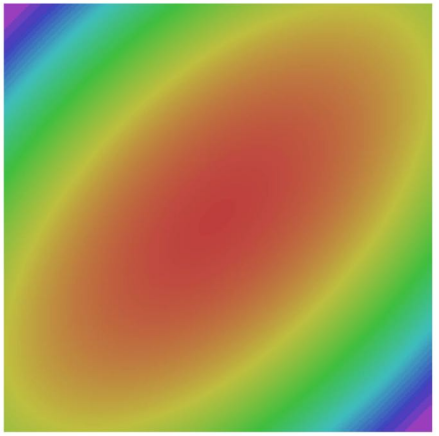
$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?



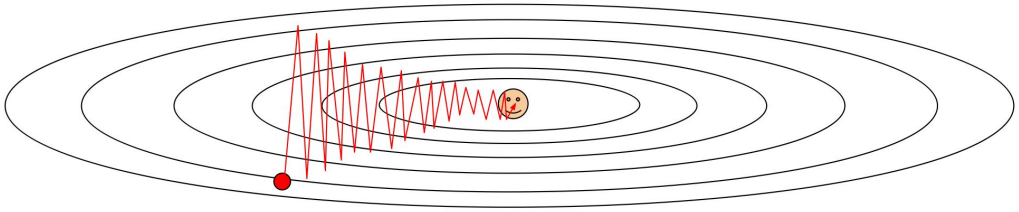


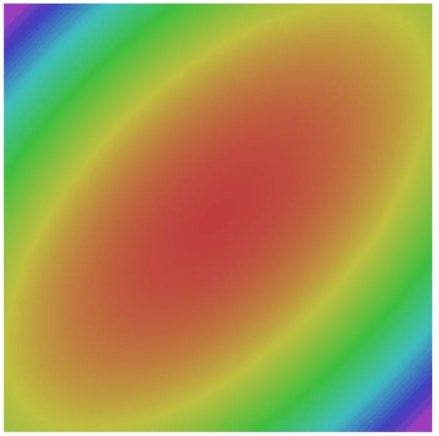
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction





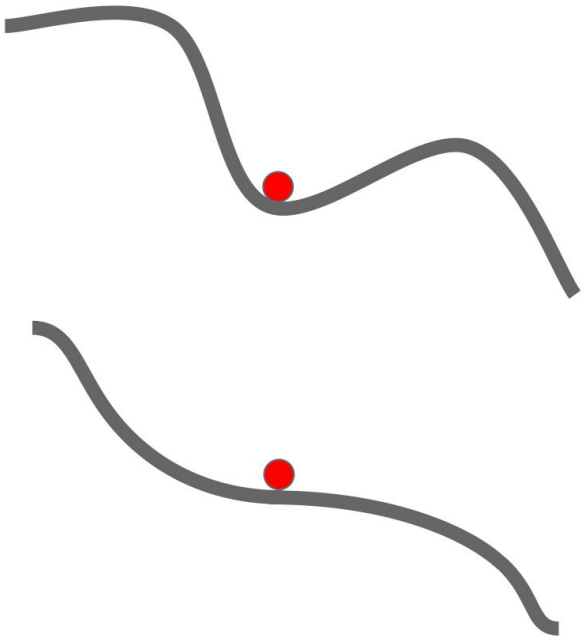
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck

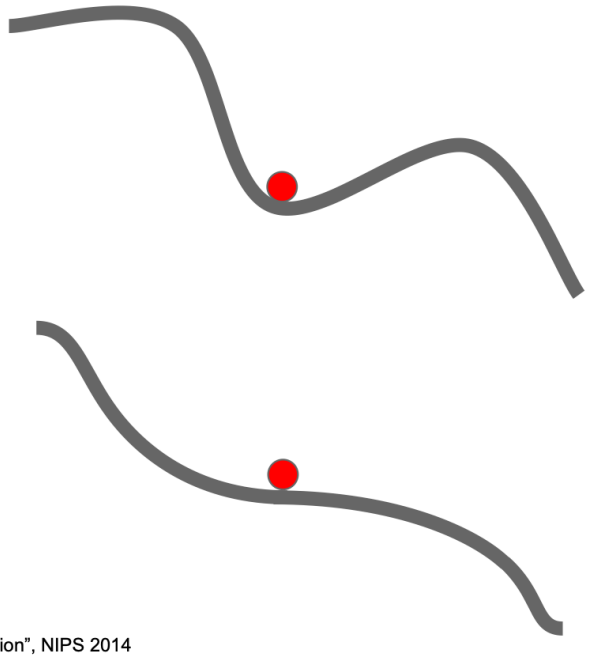


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Saddle points much more common in high dimension



Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

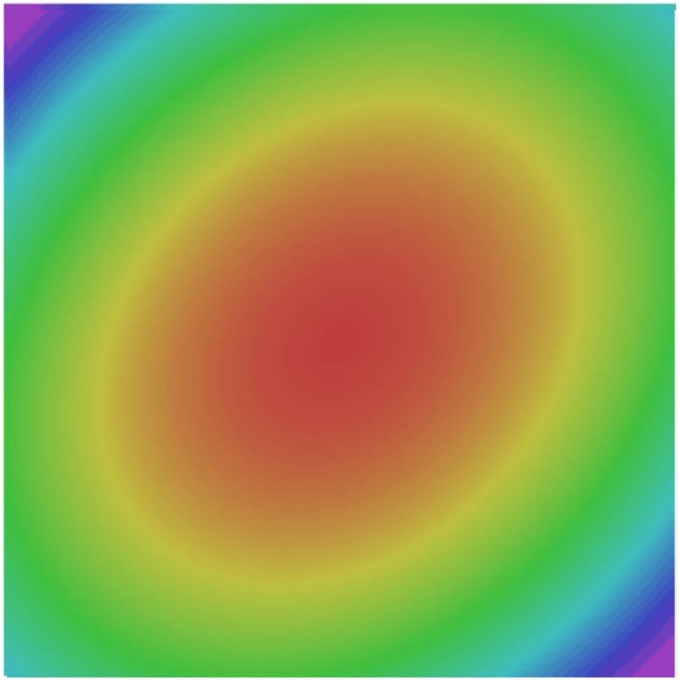
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

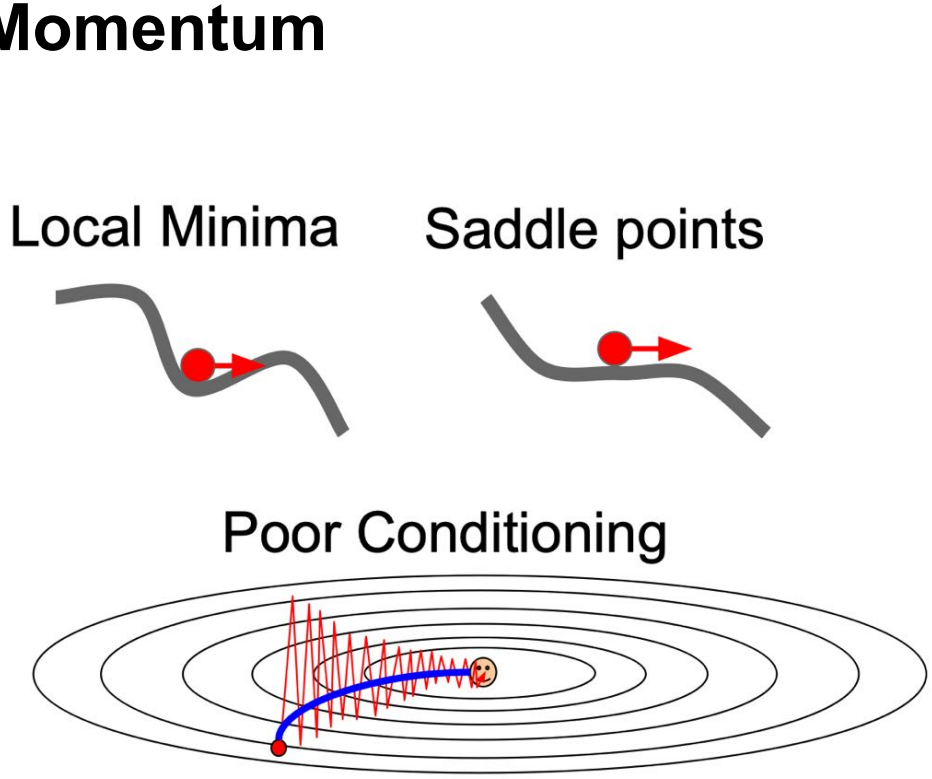
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

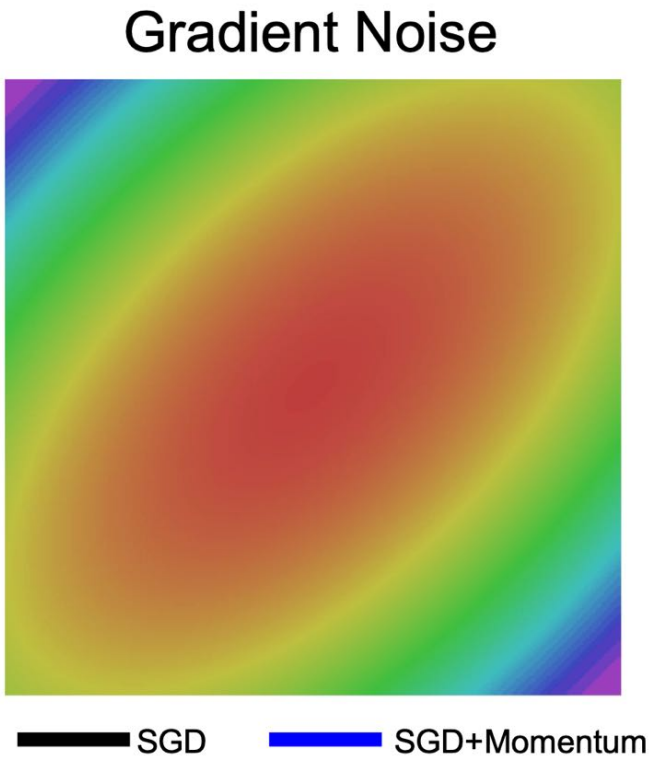
- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD + Momentum

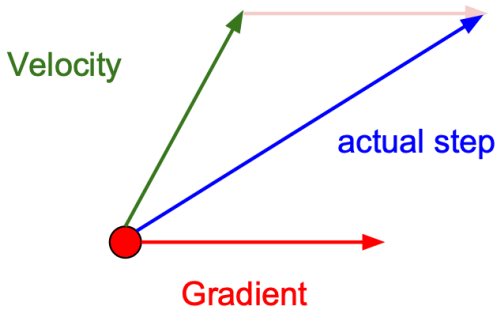




slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD + Momentum

Momentum update:



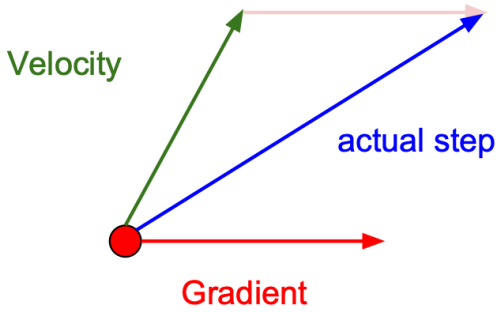
Combine gradient at current point with velocity to get step used to update weights

Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k^2)”, 1983
Nesterov, “Introductory lectures on convex optimization: a basic course”, 2004
Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

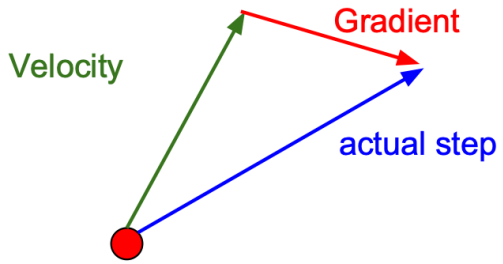
Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate O(1/k^2)", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

Nesterov Momentum



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

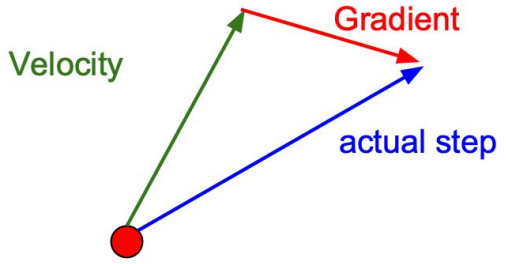
Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$
$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1}$$
$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

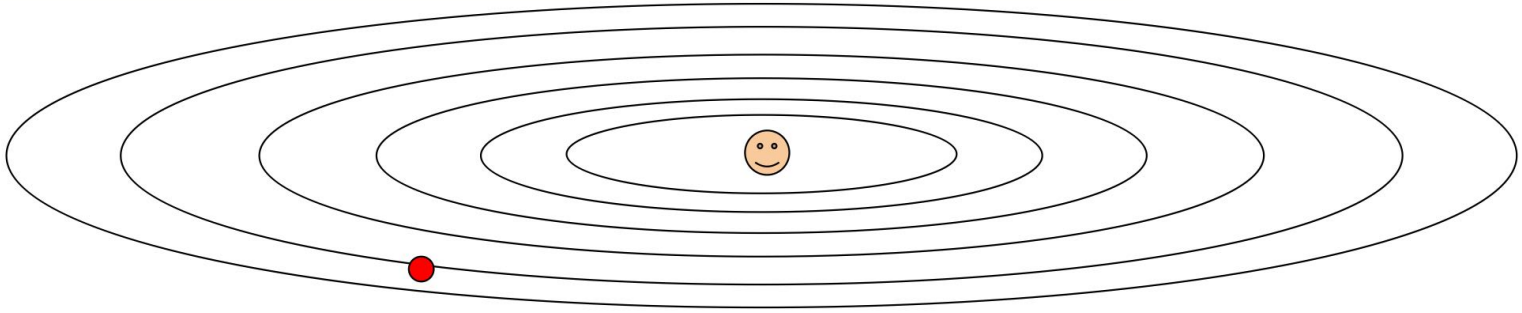
"Per-parameter learning rates"
or "adaptive learning rates"

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q: What happens with AdaGrad? Progress along “steep” directions is damped; progress along “flat” directions is accelerated

Q2: What happens to the step size over long time? Decays to zero

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RMSProp

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Sort of like RMSProp with momentum

Q: What happens at first timestep?

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

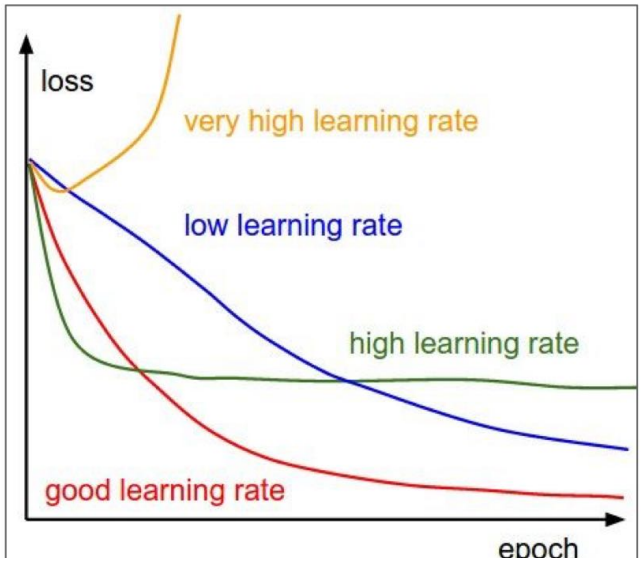
Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Learning Rate Schedule…

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



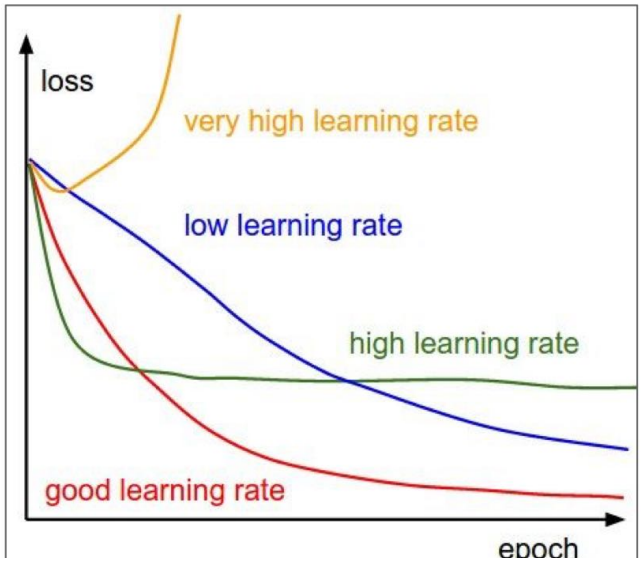
Q: Which one of these learning rates is best to use?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Learning Rate Schedule…

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:
e.g. decay learning rate by half every few epochs.

exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

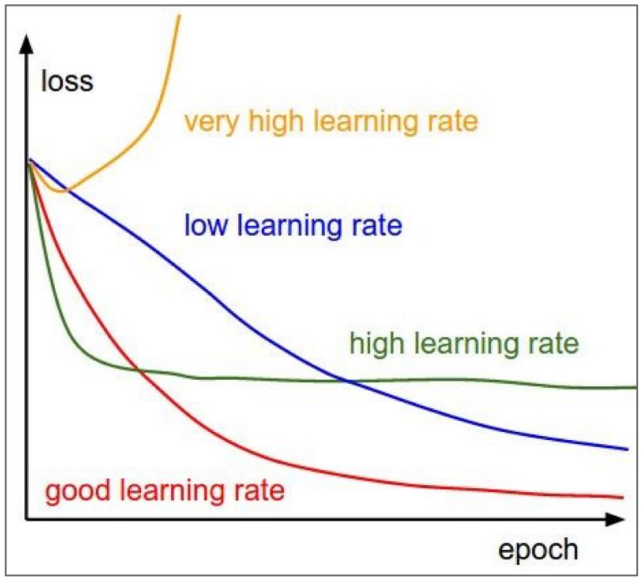
1/t decay:

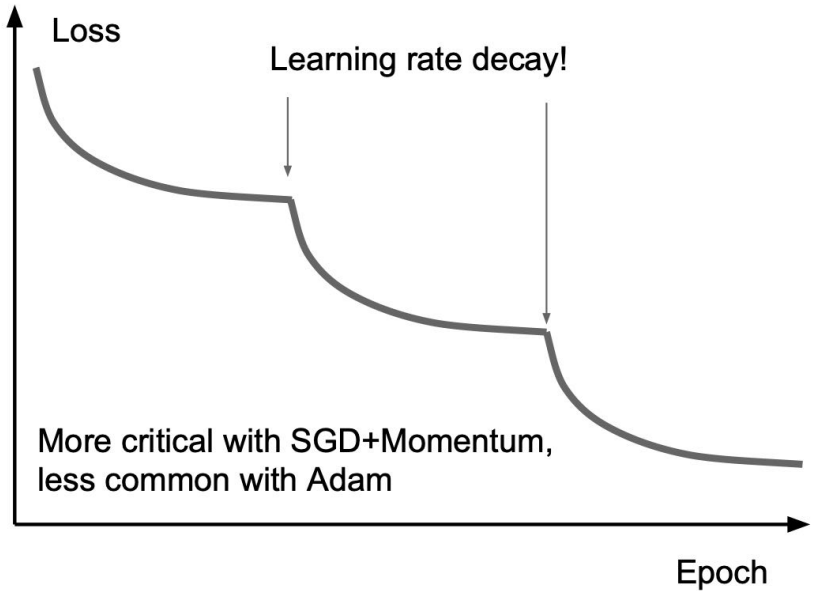
$$\alpha = \alpha_0 / (1 + kt)$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Learning Rate Schedule…

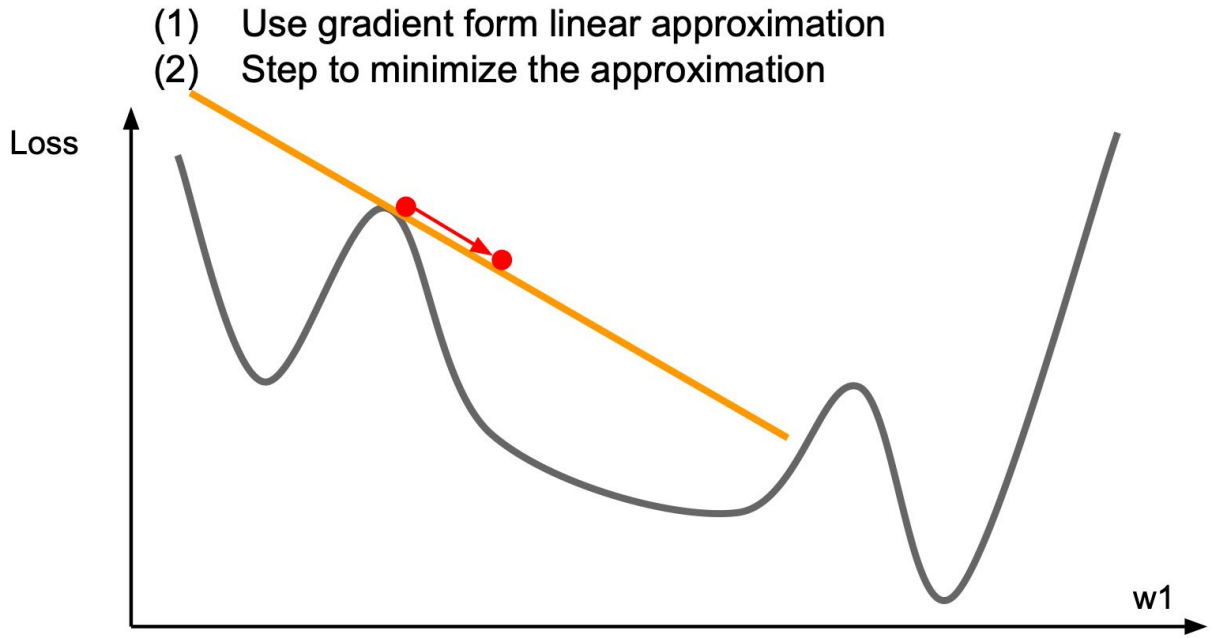
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.





slide credit: Fei-Fei, Justin Johnson, Serena Yeung

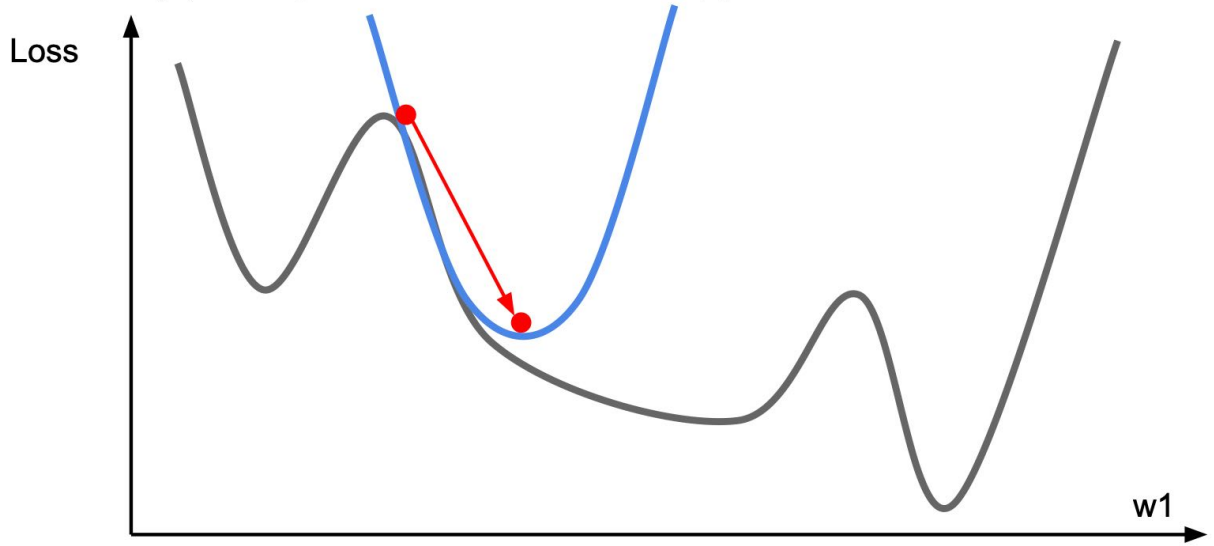
First-Order Optimization



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

(1) Use gradient **and Hessian** to form **quadratic** approximation
(2) Step to the **minima** of the approximation



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

No hyperparameters!
No learning rate!
(Though you might use one in practice)

Q: What is nice about this update?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Hessian has O(N^2) elements
Inverting takes O(N^3)
N = (Tens or Hundreds of) Millions

Q2: Why is this bad for deep learning?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

$$\theta^* = \theta_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- Quasi-Newton methods (**BGFS** most popular):
  instead of inverting the Hessian (O(n^3)), approximate inverse Hessian with rank 1 updates over time (O(n^2) each).

- **L-BFGS** (Limited memory BFGS):
  Does not form/store the full inverse Hessian.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

L-BFGS

- **Usually works very well in full batch, deterministic mode** i.e. if you have a single, deterministic f(x) then L-BFGS will probably work very nicely

- **Does not transfer very well to mini-batch setting**. Gives bad results. Adapting second-order methods to large-scale, stochastic setting is an active area of research.

Le et al, “On optimization methods for deep learning, ICML 2011”
Ba et al, “Distributed second-order optimization using Kronecker-factored approximations”, ICLR 2017

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

In Practice:

- **Adam** is a good default choice in many cases
- **SGD+Momentum** with learning rate decay often outperforms Adam by a bit, but requires more tuning

- If you can afford to do full batch updates then try out **L-BFGS** (and don't forget to disable all sources of noise)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung