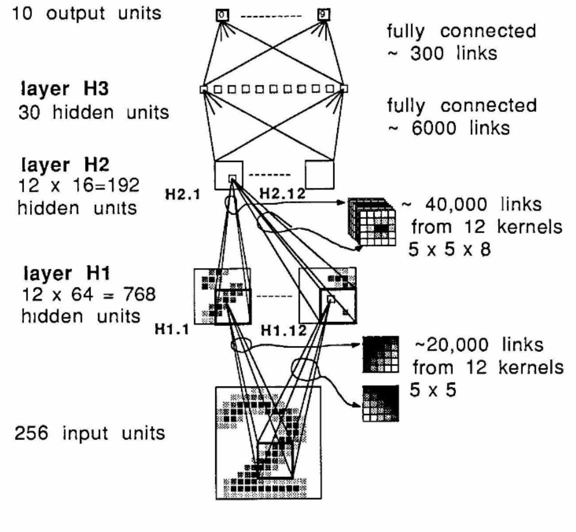
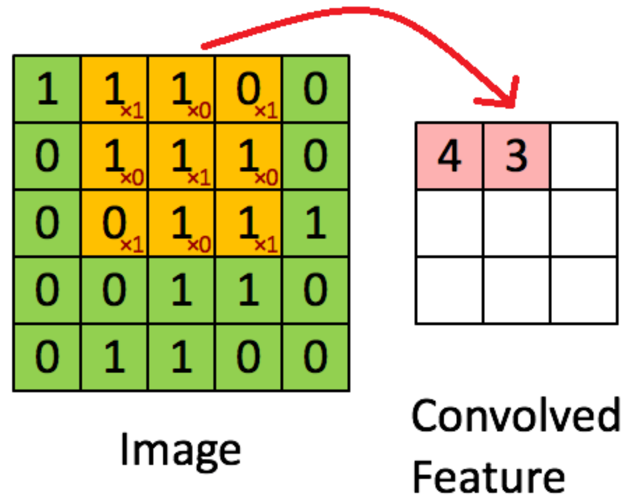
The Birth of Convolutional Neural Nets

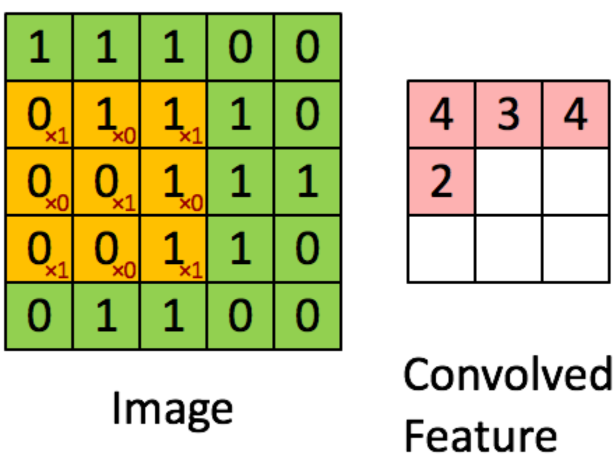


Backpropagation Applied to Handwritten Zip Code Recognition. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. Neural Computation, vol.1(4), 1989.

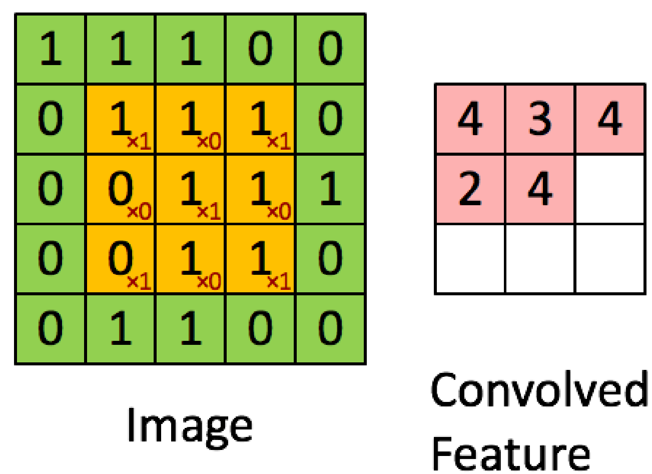
CNN- Convolution



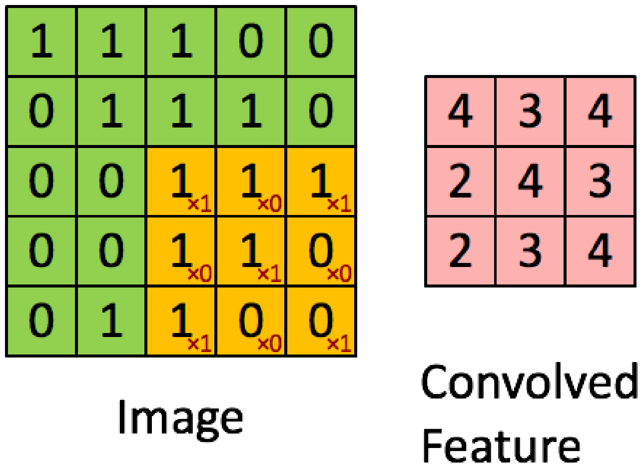
CNN - Convolution



CNN - Convolution



CNN - Convolution



CNN - Convolution with stride=1

$$\begin{array}{|c|c|c|c|c|}\hline p_{11} & p_{12} & p_{13} & p_{14} & p_{15} \\ \hline p_{21} & p_{22} & p_{23} & p_{24} & p_{25} \\ \hline p_{31} & p_{32} & p_{33} & p_{34} & p_{35} \\ \hline p_{41} & p_{42} & p_{43} & p_{44} & p_{45} \\ \hline p_{51} & p_{52} & p_{53} & p_{54} & p_{55} \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline q_{11} & q_{12} & q_{13} \\ \hline q_{21} & q_{22} & q_{23} \\ \hline q_{31} & q_{32} & q_{33} \\ \hline \end{array}$$

Note: $kernelwidth = 3$, $stride = 1$

$$q_{ij} = \sum_{r=i}^{(i+kernelwidth-1)} \sum_{s=j}^{(j+kernelwidth-1)} p_{rs} w_{(r-i+1)(s-j+1)}$$

Exercise: Find the formula for $stride > 1$

CNN - Significance of Convolution

- Convolution operation is robust to ??
- Helps to extract local features which might be useful for the task.
- Significance of striding?

CNN - Significance of Convolution

- Changing the filter size (or kernel width) $\Longrightarrow$??
- Can we use a number of filters? If so, what sizes?
- Recall: In LeCun et al's work, **24** filters were used !
- What features do these convolutions capture?
  Answer:

CNN - Zero Padding

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

- Generally used to take care of the corner and border pixels
- Used to make the convolution operations unbiased towards these pixels

CNN - Convolution on single channel image

$$n_{out} = \left\lfloor \frac{n_{in} + 2 * \text{padsize} - kernelwidth}{stride} \right\rfloor + 1$$

- n_{out} = number of features in output image
- n_{in} = number of features in input image

CNN - RGB Convolution

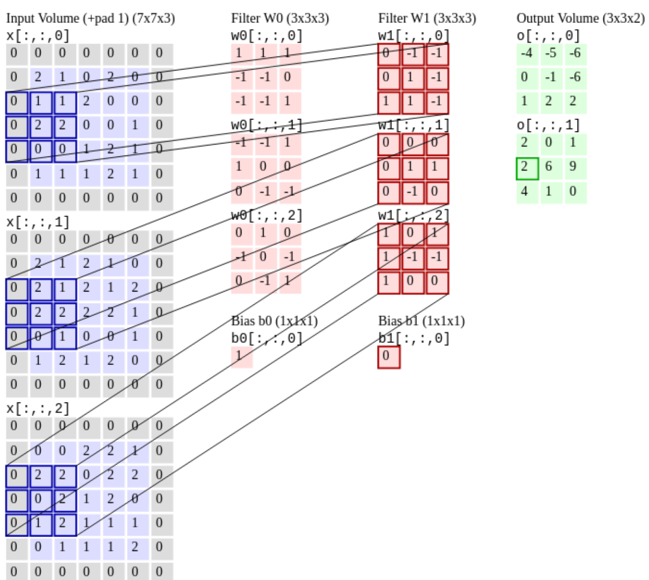


Image Source: http://cs231n.github.io/convolutional-networks/

CNN - Pooling

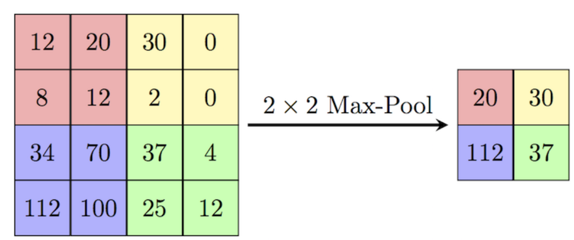
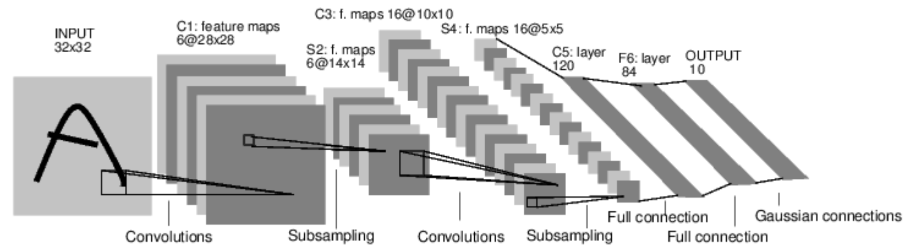


Image Source:
https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

CNN - Pooling

- Max-pooling sometimes called sub-sampling. Why?
- Max-pooling operation preserves what?
- Effect of successive max-pooling operations with zero-padding ?

CNN - LeNet Architecture



Gradient-based learning applied to document recognition.
Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. Proc. of the IEEE, 1998.

- Achieved $\sim$ 92% accuracy on handwritten digit recognition

Image Net Data

- **Detection Task:** 200 classes, $\sim$ 457K images in training set, $\sim$ 60K images in test and validation set.
  - Aim: To predict the boundaries of bounding boxes.
- **Classification and Localization Task:** 1000 categories, validation and test data consist of 150,000 photographs collected from Flickr and other search engines.
  - Aim: To find the categories of objects and their localization (presence in the form of bounding boxes) in the images.

www.kaggle.com/c/imagenet-object-localization-challenge

Pascal VOC Data

- Goal of the challenge: To recognize objects from a number of visual object classes in realistic scenes (i.e. not pre-segmented objects)
- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

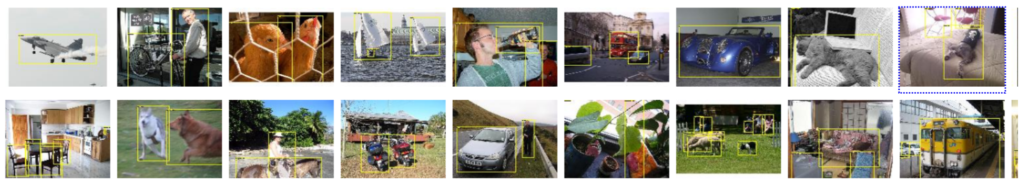
Webpage: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/
http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

Pascal VOC Data

Two main objectives:

- Classification
- Detection

20 classes



CNN - Alex Net Architecture

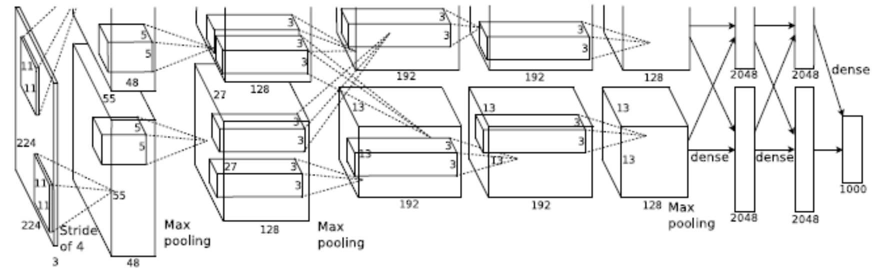


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

CNN - Alex Net

Peculiarities:

- First efficient GPU implementation of 2D convolution.
- ReLU non-linearity used
- Training done on multiple GPUs
- Local response normalization applied after ReLU

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=\max\{0,i-n/2\}}^{\min\{N-1,i+n/2\}} (a^j_{x,y})^2\right)^\beta}$$

- $k = 2$, $\alpha = 10^{-4}$, $n = 5$ and $\beta = 0.75$ chosen using cross-validation
- Local response normalization decreased the error rate
- Overlapping pooling was employed instead of usual max pooling. (3×3 window with stride of length 2)

CNN - Alex Net

Peculiarities during Training:

- Data augmentation using
  - image translations
  - horizontal reflections
- Original image size: 256×256.
  - For larger images in the data set, the shorter side was first rescaled to 256 and then central 256×256 patch was cropped.
- Mean was subtracted from individual pixel values.
- Patches of size 224×224 considered by random patching and their reflections
- Dropout considered
- Training took 5 to 6 days to train on two GTX 580 3GB GPUs.

CNN - Alex Net

Peculiarities during Testing:

- M patches of size 224×224 considered for an input image
- Predictions were averaged over M patches
- All neurons' output was multiplied by 0.5 (to take dropout into account)

CNN - Alex Net Architecture

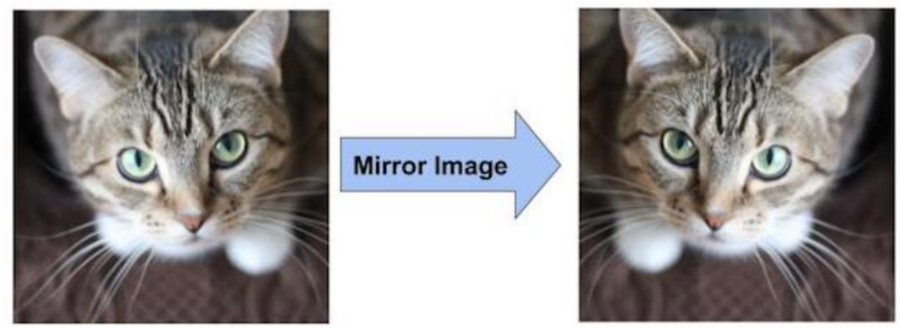


Figure: Data augmentation - mirroring*

* https://learnopencv.com/understanding-alexnet/

CNN - Alex Net Architecture

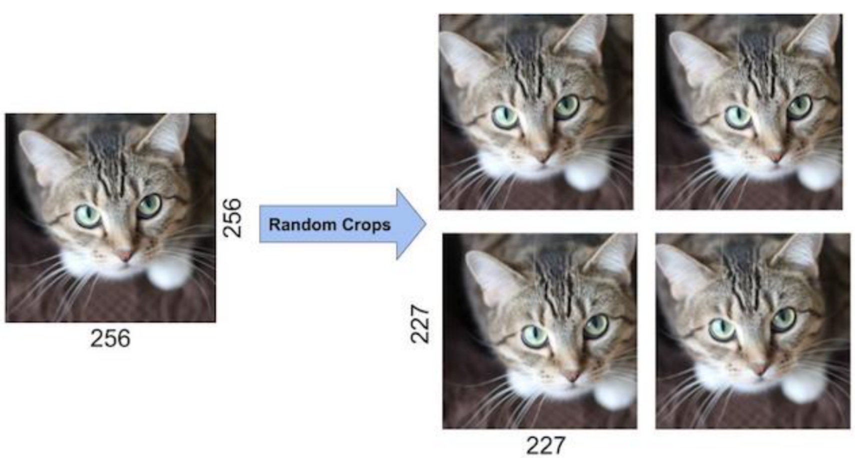


Figure: Data augmentation - random crop*

* https://learnopencv.com/understanding-alexnet/

CNN - Alex Net Architecture

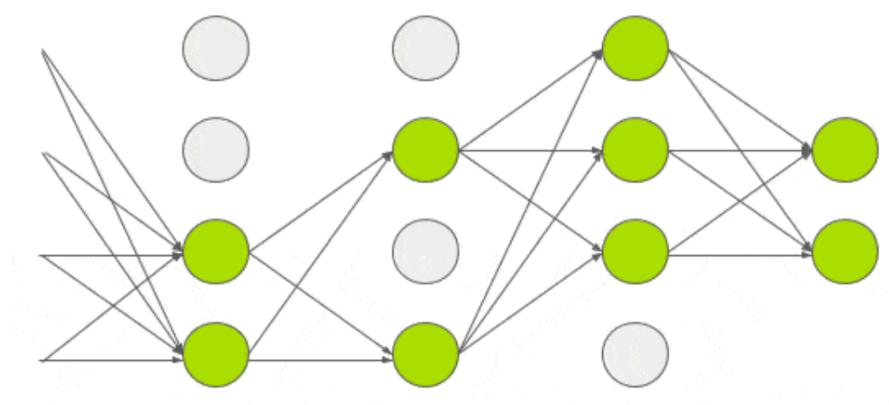


Figure: Dropout mechanism*

* https://learnopencv.com/understanding-alexnet/

CNN - Alex Net Architecture

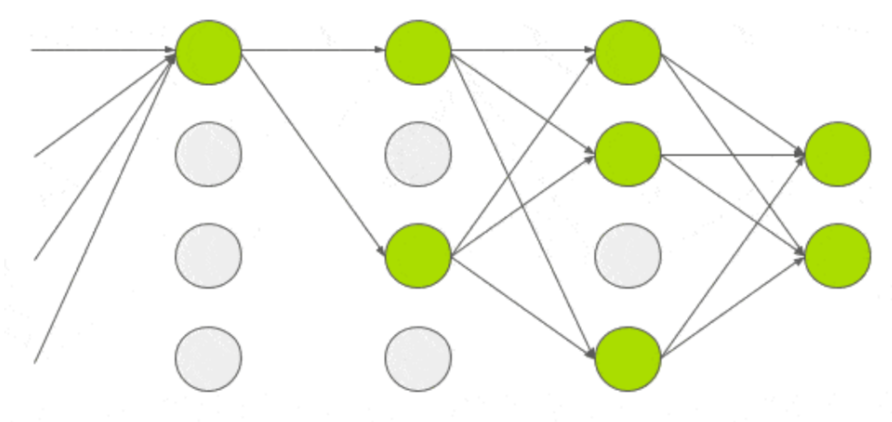


Figure: Dropout mechanism

CNN - Alex Net Architecture

Dropout

- Typically 50% of neurons are dropped in each pass.
- This leads to an increase in the number of training epochs (roughly twice).
- During testing, the whole network is used. Activations are scaled by a factor of 0.5 to account for the dropped neurons during training.
- Since all neurons are not involved in the forward pass, this makes the neurons somewhat robust to variations in the input and leads to less overfitting to training data and better generalization. (empirical observation.)

CNN - Alex Net Architecture

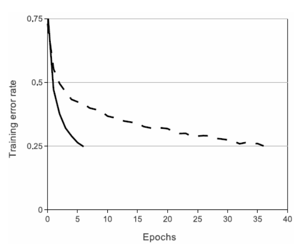


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

Figure: Impact of ReLU activation function

CNN - Alex Net Kernels After Training

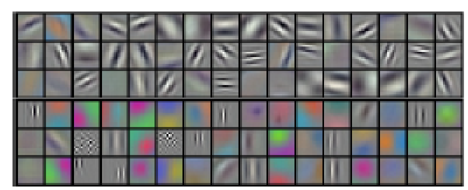


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

CNN - Alex Net Results

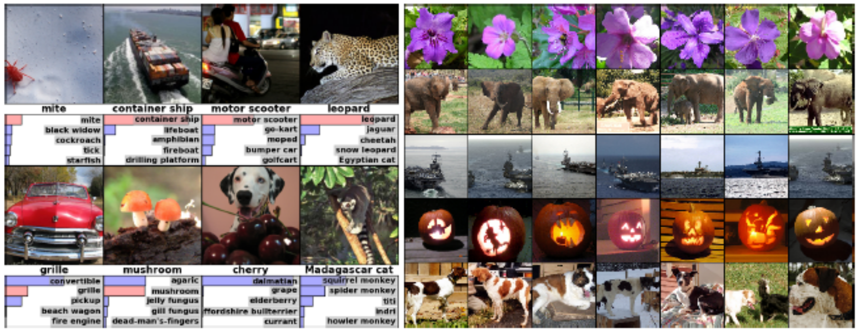
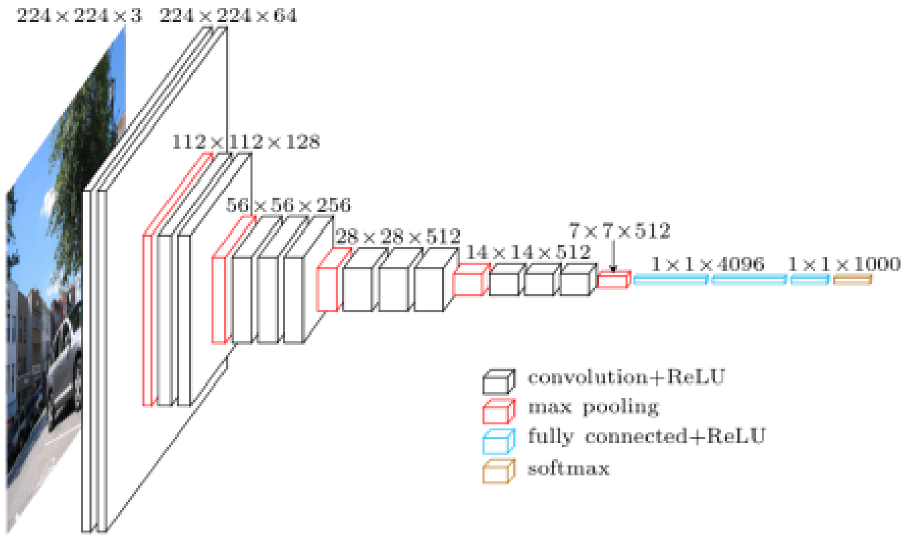


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

CNN - A VGG Net Architecture



CNN - VGG Net

Peculiarities:

- Minimal pre-processing (*e.g.* Subtraction of mean value from RGB pixel values)
- Small receptive field of size 3×3 with stride 1 (as opposed to 7×7 with stride 2 and 11×11 with stride 4 used in other previous approaches)
- Spatial resolution is preserved after convolution with 3×3 filter by using padding of appropriate size. (what padding size is suitable?)
- Max-pooling of size 2×2 with stride 2 was used after some convolution layers.
- Small receptive field leads to reduction in number of weights to be learned (check this claim!)

CNN - VGG Net

Peculiarities:

- Multiple networks with different depths (11 to 19 layers) were experimented with.
  - The depth was increased by increasing the number of convolution layers.
  - The three FC layers were same across the networks.
- The number of convolution filters used in the layers ranged from 64 to 512. This increase was done after each max-pooling operation.
- Using a stack of 2 convolution layers successively each with 3×3 filters effectively achieves a receptive field size of that of using 5×5 filter once. (check this claim!)

CNN - VGG Net

Peculiarities:

- Multinomial logistic regression objective
- Mini-batch SGD with momentum
- Weight decay employed for regularization
- Dropout used for some FC layers.
- Initial learning rate set to 10^{-2} and then decreased when validation error stalls
- Pre-initialization helped

CNN - VGG Net

Peculiarities in Training:

- Multiple scales considered ($S = 256$, $S = 384$, $S \in [256, 512]$)
- Training for larger S value performed by initializing with weights obtained from smaller S values
- Training was done on multiple GPUs where a single batch of images was further divided into GPU batches and passed to the GPUs for parallel processing.
- Titan Black GPUs of NVIDIA were used. Training took around 2 to 3 weeks.

CNN - VGG Net

Peculiarities in Testing:

- Multiple scales considered in testing too
- Scale values not exactly the same as training scales
- FC layers converted to convolution maps
- Dense vs multi-crop evaluation
  - Multi-crop evaluation was similar to that used in AlexNet.
  - In dense evaluation, arbitrary sized inputs were considered without any cropping, and conversion to an appropriate dimension was automatically done by using appropriate convolutions so that the final classification can be obtained on any arbitrary sized input.

CNN - VGG Net

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

CNN - VGG Net

Table 2: **Number of parameters (in millions).**

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

CNN - VGG Net

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

CNN - VGG Net

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

CNN - VGG Net

Table 5: **ConvNet evaluation techniques comparison.** In all experiments the training scale S was sampled from $[256; 512]$, and three test scales Q were considered: $\{256, 384, 512\}$.

ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

CNN - VGG Net

Table 7: **Comparison with the state of the art in ILSVRC classification.** Our method is denoted as "VGG". Only the results obtained without outside training data are reported.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

CNN - VGG Net

Table 10: **Comparison with the state of the art in ILSVRC localisation.** Our method is denoted as "VGG".

Method	top-5 val. error (%)	top-5 test error (%)
VGG	**26.9**	**25.3**
GoogLeNet (Szegedy et al., 2014)	-	26.7
OverFeat (Sermanet et al., 2014)	30.0	29.9
Krizhevsky et al. (Krizhevsky et al., 2012)	-	34.2

CNN - VGG Net

Table 11: **Comparison with the state of the art in image classification on VOC-2007, VOC-2012, Caltech-101, and Caltech-256.** Our models are denoted as "VGG". Results marked with * were achieved using ConvNets pre-trained on the *extended* ILSVRC dataset (2000 classes).

Method	VOC-2007 (mean AP)	VOC-2012 (mean AP)	Caltech-101 (mean class recall)	Caltech-256 (mean class recall)
Zeiler & Fergus (Zeiler & Fergus, 2013)	-	79.0	86.5 ± 0.5	74.2 ± 0.3
Chatfield et al. (Chatfield et al., 2014)	82.4	83.2	88.4 ± 0.6	77.6 ± 0.1
He et al. (He et al., 2014)	82.4	-	$\mathbf{93.4 \pm 0.5}$	-
Wei et al. (Wei et al., 2014)	81.5 (85.2*)	81.7 (**90.3***)	-	-
VGG Net-D (16 layers)	89.3	89.0	91.8 ± 1.0	85.0 ± 0.2
VGG Net-E (19 layers)	89.3	89.0	92.3 ± 0.5	85.1 ± 0.3
VGG Net-D & Net-E	**89.7**	**89.3**	92.7 ± 0.5	$\mathbf{86.2 \pm 0.3}$

CNN - VGG Net

Table 12: **Comparison with the state of the art in single-image action classification on VOC-2012.** Our models are denoted as "VGG". Results marked with * were achieved using ConvNets pre-trained on the *extended* ILSVRC dataset (1512 classes).

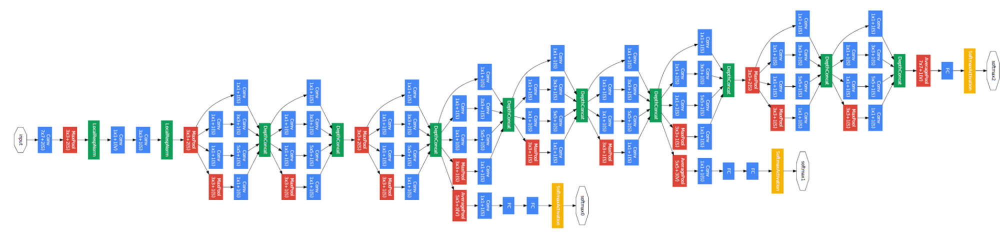
Method	VOC-2012 (mean AP)
(Oquab et al., 2014)	70.2*
(Gkioxari et al., 2014)	73.6
(Hoai, 2014)	76.3
VGG Net-D & Net-E, image-only	**79.2**
VGG Net-D & Net-E, image and bounding box	**84.0**

CNN - GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table : GoogLeNet incarnation of the Inception architecture

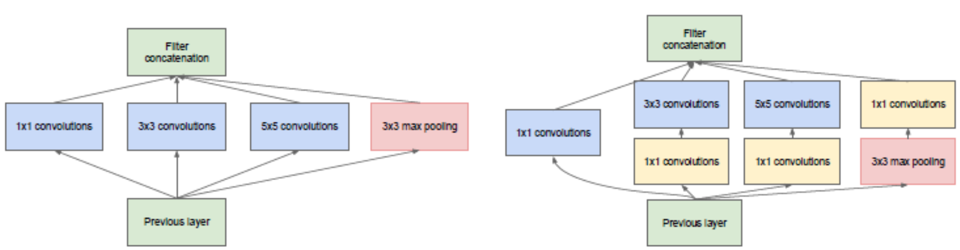
CNN - GoogLeNet



GoogLeNet Network (From Left to Right)

- Testing is a bit complicated: involves 7 models, 144 crops per image, softmax probabilities averaged over all crops
- Achieved ∼ **93.3%** accuracy on ILSVRC 2014 Classification Challenge

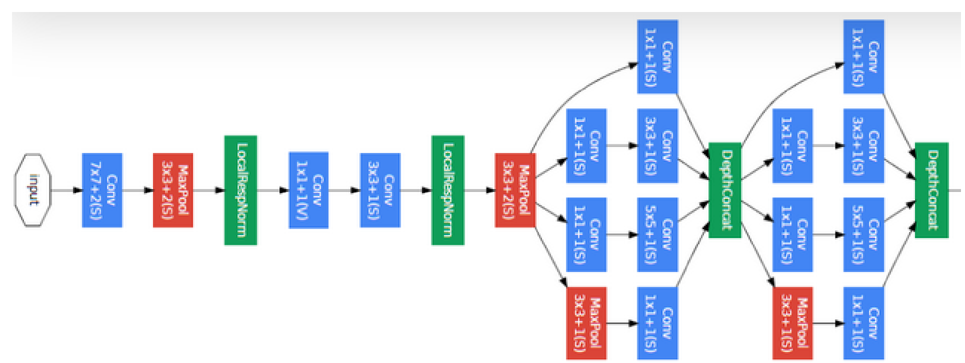
CNN - GoogLeNet



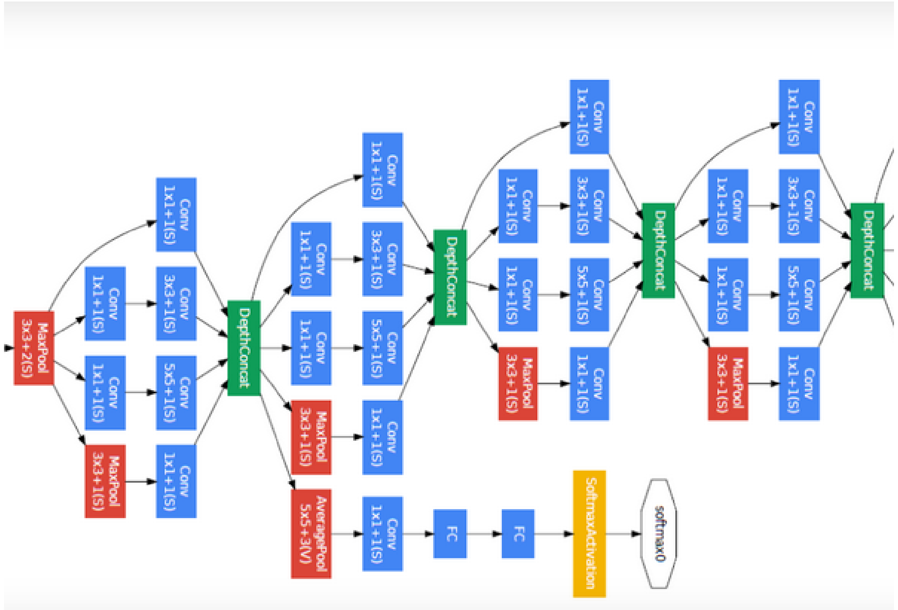
(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

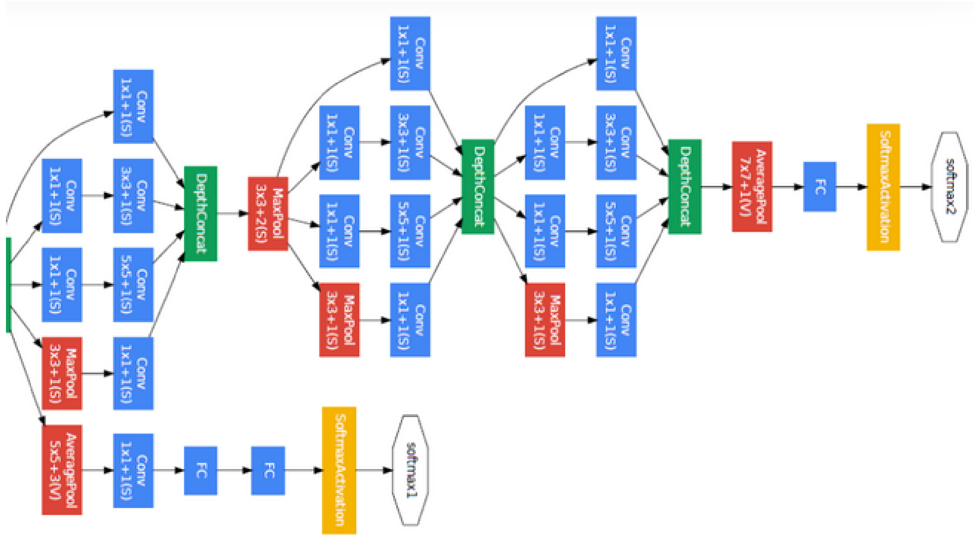
CNN - GoogLeNet



CNN - GoogLeNet



CNN - GoogLeNet



CNN - ResNet

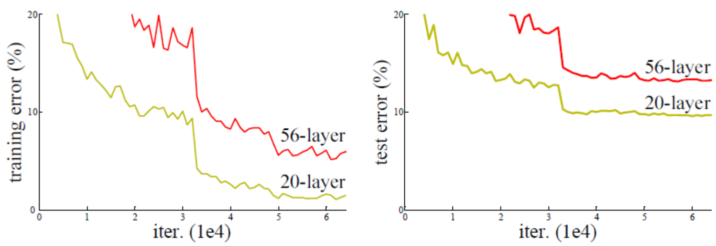
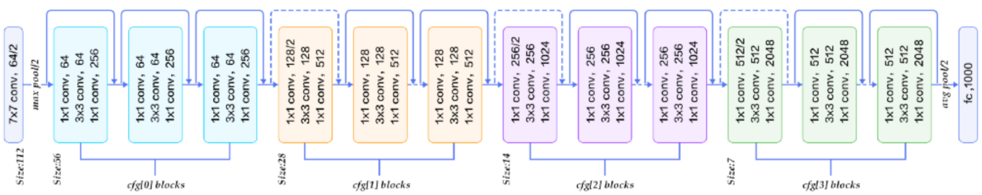


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

CNN - ResNet



- Testing involves 6 models, multiple crops per image, softmax probabilities averaged over all crops
- ResNet-152 Achieved ∼ **97%** accuracy on ILSVRC 2015 Classification Challenge

CNN - ResNet

- **Aim**: To model a map $\mathcal{H} : I \to S$
- I can denote the input space or some suitable intermediate representation space.
- S can denote the output space or some suitable intermediate representation space.

CNN - ResNet

- **Idea**: To model $\mathcal{H}$ using $\mathcal{H}(x) = \mathcal{F}(x) + Wx$ and learn $\mathcal{F}$ and W.
- If I and S are of same dimensions, W can be considered (or hypothesized) to be the identity, hence the aim reduces to learning $\mathcal{F}$.

CNN - ResNet

- **Motivation**: Learning the actual map $\mathcal{H}$ is difficult.
- **Hope**: Can we ease the task by focusing the learning task related to $\mathcal{F}$ and then use the original input to get back $\mathcal{H}$.
- **Caveat**: Learning $\mathcal{F}$ is also not that easy! (Exploit deep nets to learn $\mathcal{F}$)

CNN - ResNet

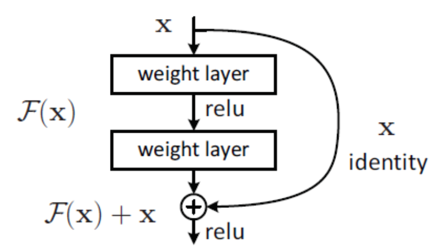


Figure 2. Residual learning: a building block.

CNN - ResNet

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

CNN - ResNet

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	$8.43^{\dagger}$
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

CNN - DenseNet

- **Aim**: To facilitate flow of gradients from last layers to the initial layers **avoiding vanishing gradient** issue.
- A densenet block is designed which can facilitate the flow of gradients.

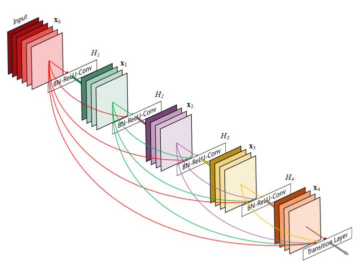


Figure : A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- In each densenet block of L layers, the ℓ-th layer $(0 \leq \ell \leq L-1)$ receives feature maps from **all the previous** $\ell - 1$ **layers**.
- Hence there would be $\frac{L(L+1)}{2}$ feature maps in the densenet block.

CNN - DenseNet

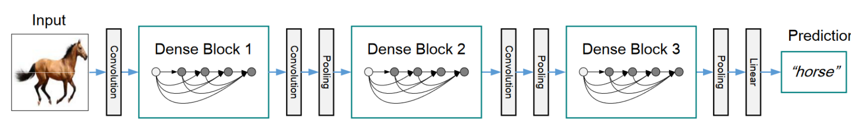


Figure : A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

- At ℓ-th layer the operation would be of the form:

$$M_\ell = H_\ell[M_0; M_1; \ldots; M_{\ell-1}]$$

  where $[M_0; M_1; \ldots; M_{\ell-1}]$ denotes concatenation of feature maps $M_0, M_1, \ldots, M_{\ell-1}$ at corresponding layers.
- H_ℓ is composition of Batch normalization, ReLU and 3×3 convolution.

$$H_\ell(M) = \text{Conv}_{3\times3}(ReLU(BN(M))), \ell \in \{0, 1, \ldots, L-1\}.$$

CNN - DenseNet

Batch Norm Operation

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ref: S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. http://arxiv.org/abs/1502.03167

CNN - DenseNet

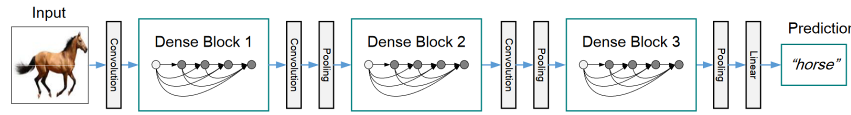


Figure : A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

- The densely connected layers are grouped to form **dense blocks**.
- The convolution and pooling layers between the dense blocks are called **transition layers**.
- At transition layers, the following operations are done:

$$H_{\text{trans}}(M) = \text{AvgPool}_{2\times 2}(\text{Conv}_{1\times 1}(BN(M)))$$

CNN - DenseNet

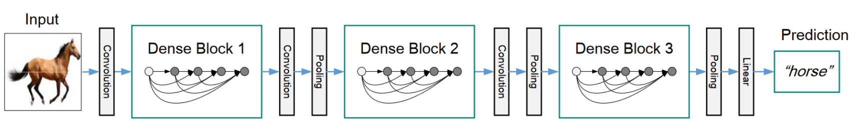


Figure : A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

- If each layer in dense block outputs k feature maps, then the input to the ℓ-th layer in the dense block is $k_0 + k * (\ell - 1)$.
- k_0 is the number of channels at the input layer.
- Thus the number of feature maps at each layer in a dense block is controlled by a factor k called the growth rate.
- k can typically be a small number. (e.g. $k = 12$)

CNN - DenseNet

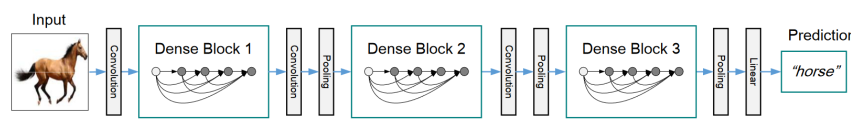


Figure : A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Other heuristics:

- In DenseNet-B, H_ℓ a bottleneck layer of 1×1 convolution precedes the 3×3 convolution:

$$H_\ell(M) = \text{Conv}_{3\times 3}(\text{Conv}_{1\times 1}(ReLU(BN(M))))$$

CNN - DenseNet

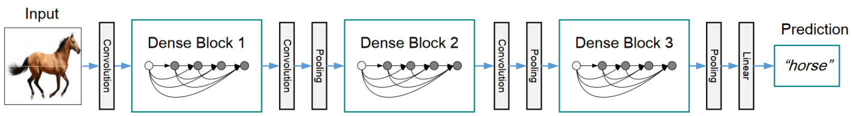


Figure : A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Other heuristics:

- Compression introduced at transition layer where only $\lfloor \theta m \rfloor, \theta \in (0, 1]$ to pass on only a fraction of total m feature maps produced at the dense block.

CNN - DenseNet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table : DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

CNN - DenseNet

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table : Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

CNN - EfficientNet

- A layer i in a CNN can be written as:

$$Y_i = F_i(X_{i_{\langle H_i, W_i, C_i \rangle}})$$

  where F_i is the operation on an input feature map X_i of dimensions $H_i \times W_i \times C_i$ to get the output feature map Y_i.
- **Note:** X_i and Y_i are tensors.
- Hence a CNN $\mathcal{N}$ can be written as:

$$\mathcal{N} = F_k(F_{k-1}(\ldots F_2(F_1(X_1))\ldots)).$$

CNN - EfficientNet

- A layer i in a CNN can be written as:

$$Y_i = F_i(X_{i_{\langle H_i, W_i, C_i \rangle}})$$

  where F_i is the operation on an input feature map X_i of dimensions $H_i \times W_i \times C_i$ to get the output feature map Y_i.
- (H_i, W_i) is called **resolution**.
- C_i (number of channels) is called **width**.
- If an operation F_i is possibly repeated L_i times, L_i is called **network length** or **depth**.

CNN - EfficientNet

We will use an equivalent notation to write the CNN $\mathcal{N}$:

$$\begin{aligned}\mathcal{N} &= F_{k_{\langle H_k, W_k, C_k \rangle}}^{L_k} \odot F_{k-1_{\langle H_{k-1}, W_{k-1}, C_{k-1} \rangle}}^{L_{k-1}} \odot \dots \odot F_{1_{\langle H_1, W_1, C_1 \rangle}}^{L_1} \\ &= \bigodot_{j=1,2,\dots,k} F_{j_{\langle H_j, W_j, C_j \rangle}}^{L_j}.\end{aligned}$$

CNN - EfficientNet

Optimization problem:

$$\begin{aligned}\max_{d,w,r} \quad & Accuracy(\mathcal{N}(d,w,r)) \\ s.t. \quad & \mathcal{N}(d,w,r) = \bigodot_{i=1\dots s} \hat{\mathcal{F}}_i^{d \cdot \hat{L}_i}\langle r \cdot \hat{H}_i, r \cdot \hat{W}_i, w \cdot \hat{C}_i \rangle \\ & \text{Memory}(\mathcal{N}) \leq \text{target\_memory} \\ & \text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}\end{aligned}$$

CNN - EfficientNet

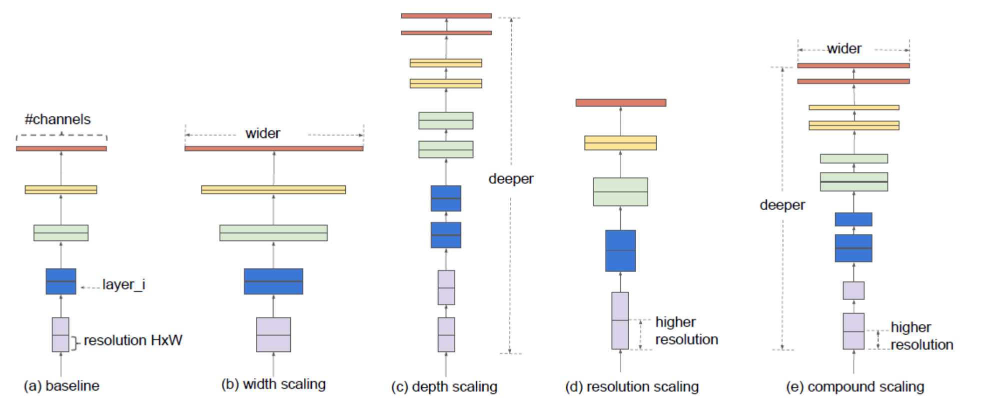


Figure. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

CNN - EfficientNet

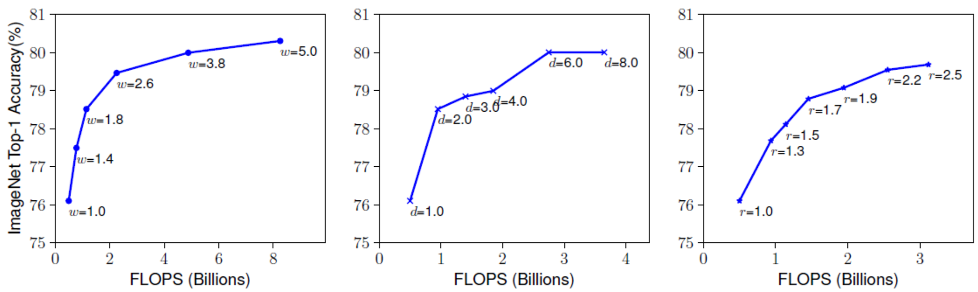


Figure. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling.

CNN - EfficientNet

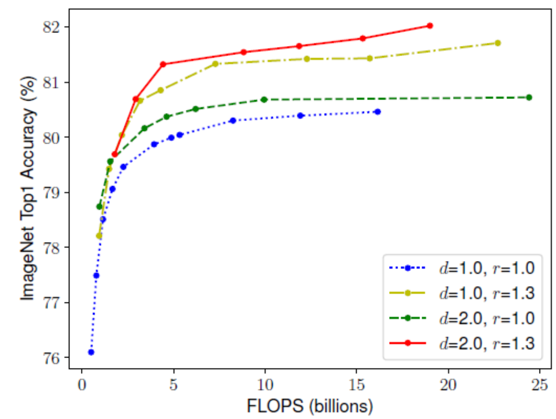


Figure. **Scaling Network Width for Different Baseline Networks.** Each dot in a line denotes a model with different width coefficient (w). The first baseline network (d=1.0, r=1.0) has 18 convolutional layers with resolution 224x224, while the last baseline (d=2.0, r=1.3) has 36 layers with resolution 299x299.

CNN - EfficientNet

- **Optimization problem:**

$$\max_{d,w,r} \quad Accuracy(\mathcal{N}(d,w,r))$$
$$s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right)$$
$$\text{Memory}(\mathcal{N}) \leq \text{target\_memory}$$
$$\text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}$$

- **Parameter selection:**

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

CNN - EfficientNet

Parameter selection procedure:

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ to obtain EfficientNet-B1 to B7

CNN - EfficientNet

Table . **EfficientNet Performance Results on ImageNet** (Russakovsky et al., 2015). All EfficientNet models are scaled from our baseline EfficientNet-B0 using different compound coefficient ϕ . ConvNets with similar top-1/top-5 accuracy are grouped together for efficiency comparison. Our scaled EfficientNet models consistently reduce parameters and FLOPS by an order of magnitude (up to 8.4x parameter reduction and up to 16x FLOPS reduction) than existing ConvNets.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

R-CNN (Fast version)

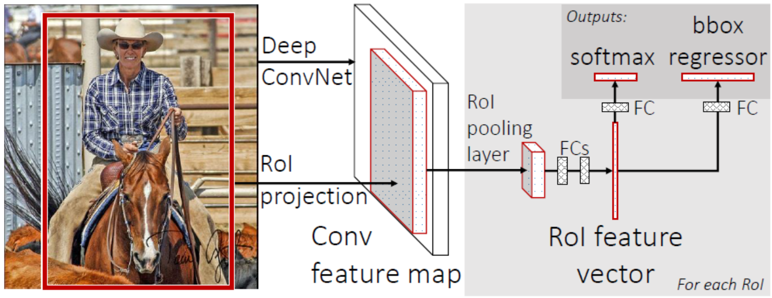


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

R-CNN (Fast version)

- The whole image is first processed by deep convolution networks to produce a conv feature map.
- For each Region of Interest (RoI) proposal, a RoI pooling layer extracts a fixed length feature vector from the feature map.
- Each such feature vector is then fed into sequence of fully connected layers with two output layers.
- The two output layers correspond to object detection and region of interest prediction

R-CNN (Fast version)

- RoI is quantified using (r, c, h, w) where (r, c) represent the top left corner and h, w represent the height and width of regions
- Conv feature map can be constructed from any pre-trained network like Alex Net, VGG Net, etc.

R-CNN (Fast version)

- Two Output Layers:
  - Predict the object class
  - Predict the RoI box
- Loss functions?

R-CNN (Fast version)

- Loss functions
  - $L_{objclass} = L(O_p, O_t)$
  - O_p, O_t represent the predicted class and true class
  - Cross-entropy used for $L_{objclass}$
  - $L_{RoIBox} = L((r_p, c_p, h_p, w_p), (r_t, c_t, h_t, w_t))$
  - (r_p, c_p, h_p, w_p), (r_t, c_t, h_t, w_t) represent predicted and true RoI parameters.
  - Smooth L1 loss used for L_{RoIBox}

$$L_{RoIBox} = L1_{sm}(r_p - r_t) + L1_{sm}(c_p - c_t) + L1_{sm}(h_p - h_t) + L1_{sm}(w_p - w_t)$$

$$L1_{sm}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{else.} \end{cases}$$

- Total Loss: $L = L_{objclass} + \lambda L_{RoIBox}$
- Guess on λ value used?

R-CNN (Fast version)

- Mini-batch SGD used for training
- Testing: Multiple RoI should be sampled and the network is used for predicting the confidence scores and RoI predictions relative to each sampled RoI.

R-CNN (Fast version)

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Table 2. **VOC 2010 test** detection average precision (%). BabyLearning uses a network based on [17]. All other methods use VGG16. Training set key: **12**: VOC12 trainval, **Prop.**: proprietary dataset, **12+seg**: **12** with segmentation annotations, **07++12**: union of VOC07 trainval, VOC07 test, and VOC12 trainval.

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Table 3. **VOC 2012 test** detection average precision (%). BabyLearning and NUS_NIN_c2000 use networks based on [17]. All other methods use VGG16. Training set key: see Table 2, **Unk.**: unknown.

R-CNN (Faster version)

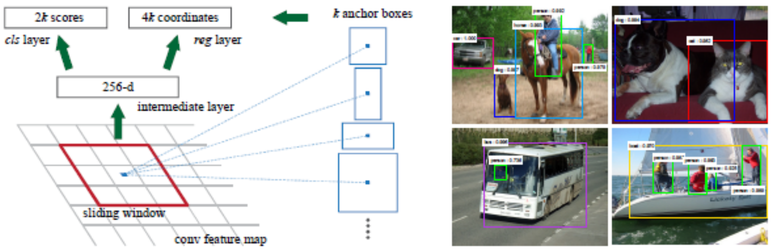


Figure 1: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

R-CNN (Faster version)

- Proposed Region Proposal Networks (RPN)
- Essentially similar to fast R-CNN
- Anchors used in training which overlap with the true RoI boxes
- This leads to some form of robustness in prediction
- Weight sharing between RPNs and Fast R-CNNs employed

R-CNN (Faster version)

Table 1: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

train-time region proposals		**test-time region proposals**		
method	# boxes	method	# proposals	mAP (%)
SS	2k	SS	2k	58.7
EB	2k	EB	2k	58.6
RPN+ZF, shared	2k	RPN+ZF, shared	300	**59.9**

R-CNN (Faster version)

Table 2: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2k. †: this was reported in [5]; using the repository provided by this paper, this number is higher (68.0±0.3 in six runs).

method	# proposals	data	mAP (%)	time (ms)
SS	2k	07	66.9†	1830
SS	2k	07+12	70.0	1830
RPN+VGG, unshared	300	07	68.5	342
RPN+VGG, shared	300	07	69.9	**198**
RPN+VGG, shared	300	07+12	**73.2**	**198**

Table 3: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07++12": union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2k. †: `http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html`. ‡: `http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html`

method	# proposals	data	mAP (%)
SS	2k	12	65.7
SS	2k	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**

3D CNN

Figure: Video as sequence of image frames†

† https://www.cctvsg.net/frame-rate/

3D CNN

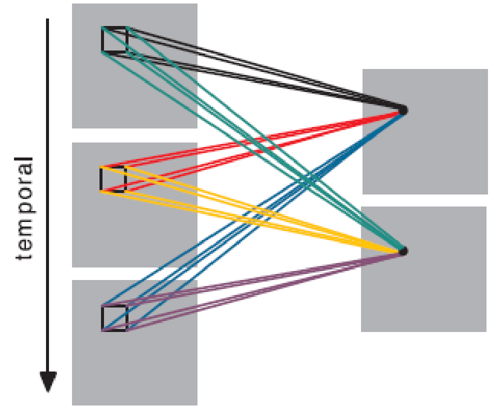


Fig. 2. Extraction of multiple features from contiguous frames. Multiple 3D convolutions can be applied to contiguous frames to extract multiple features. As in Fig. 1, the sets of connections are color-coded so that the shared weights are in the same color. Note that all six sets of connections do not share weights, resulting in two different feature maps on the right.

3D CNN

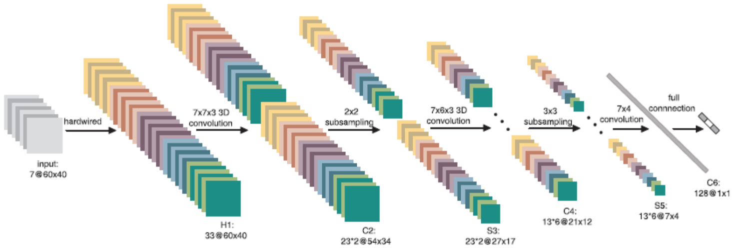


Fig. 3. A 3D CNN architecture for human action recognition. This architecture consists of one hardwired layer, three convolution layers, two subsampling layers, and one full connection layer. Detailed descriptions are given in the text.

3D CNN

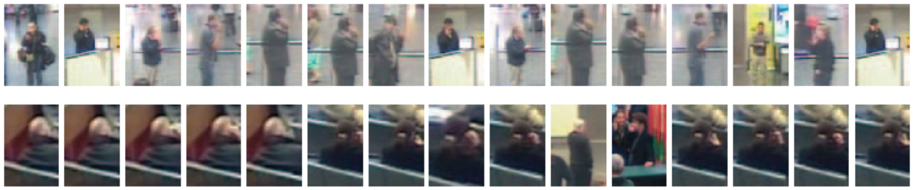


Fig. 10. Sample actions in the CellToEar class. The top row shows actions that are correctly recognized by the combined 3D CNN model, while the bottom row shows those that are misclassified by the model.

CNN for 3D Modeling

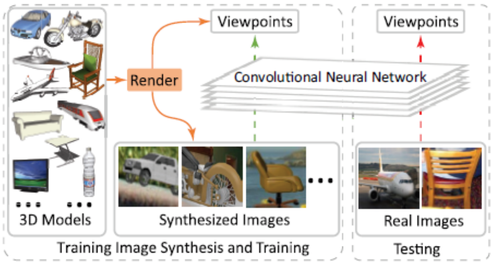


Figure 1. **System overview.** We synthesize training images by overlaying images rendered from large 3D model collections on top of real images. CNN is trained to map images to the ground truth object viewpoints. The training data is a combination of real images and synthesized images. The learned CNN is applied to estimate the viewpoints of objects in real images. Project homepage is at https://shapenet.cs.stanford.edu/projects/RenderForCNN

CNN for 3D Modeling

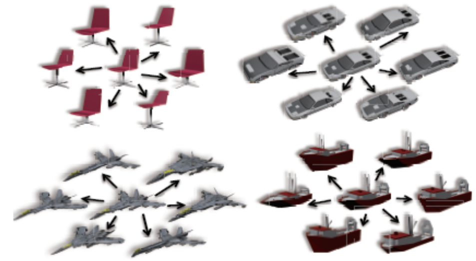


Figure 2. **3D model set augmentation by symmetry-preserving deformation.**

CNN for 3D Modeling

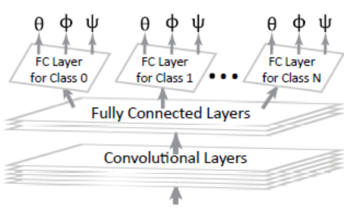


Figure 4. **Network architecture**[2]. Our network includes shared feature layers and class-dependent viewpoint estimation layers.

CNN for 3D Modeling

- Goal: To predict the viewpoint of an object in an image
- Viewpoint is with respect to the camera
- Viewpoint characterized by three parameters (θ, ϕ, ψ) denoting the azimuth, elevation and in-plane rotation angles.
- Opensource ShapeNet† used

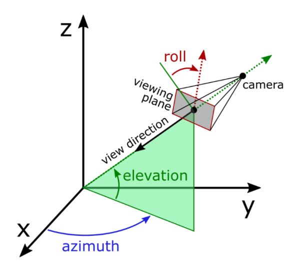


Figure: Azimuth, elevation and roll of camera viewpoint.‡

† https://www.shapenet.org/
‡ https://matplotlib.org/stable/api/toolkits/mplot3d/view_angles.html

CNN for 3D Modeling

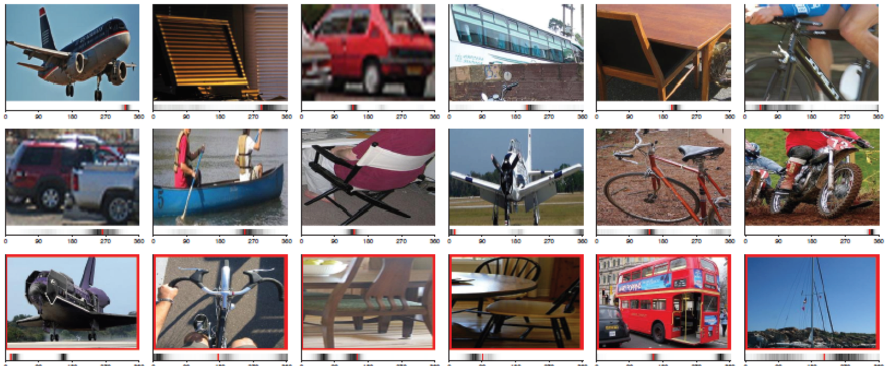
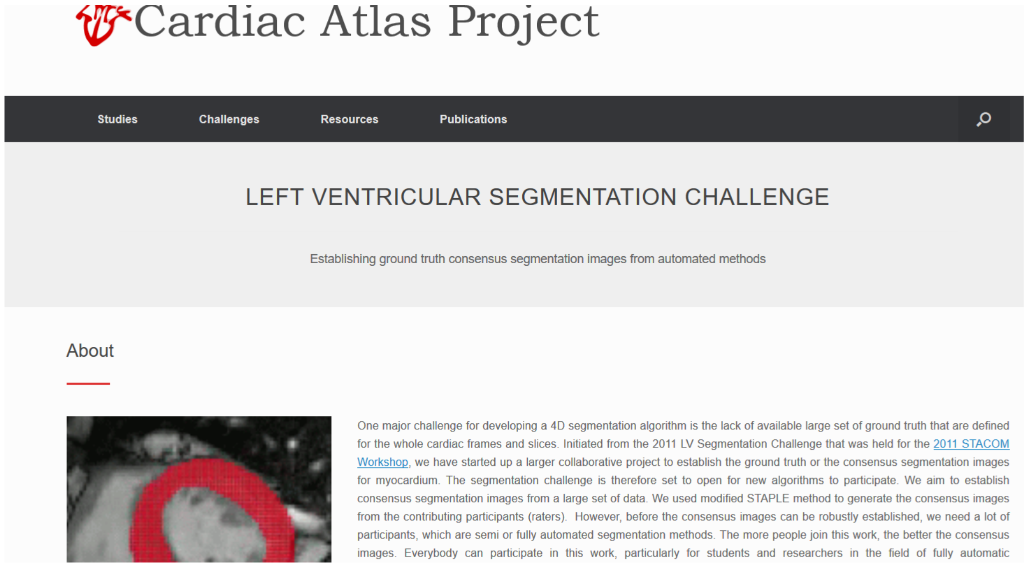


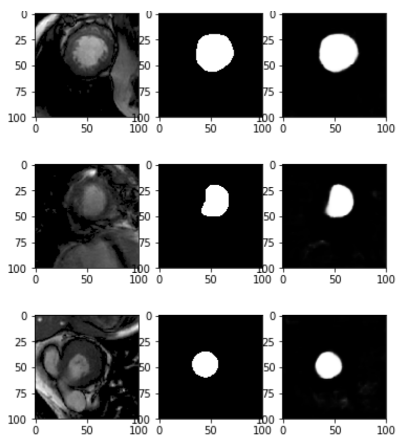
Figure 10. **Viewpoint estimation example results.** The bar under each image indicates the 360-class confidences (black means high confidence) corresponding to $0^\circ \sim 360^\circ$ (with object facing towards us as 0° and rotating clockwise). The red vertical bar indicates the ground truth. The first two rows are positive cases, the third row is negative case (with red box surrounding the image).

Left Ventricle Segmentation



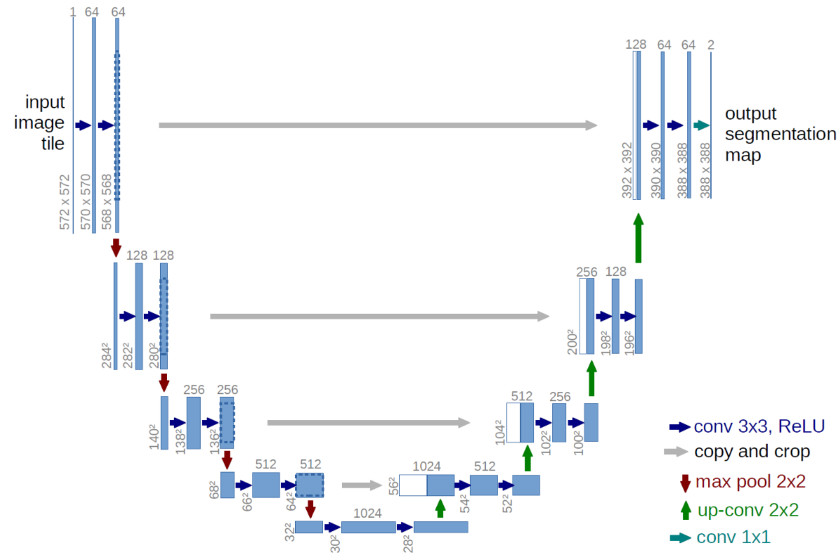
http://www.cardiacatlas.org/challenges/lv-segmentation-challenge/

Left Ventricle Segmentation



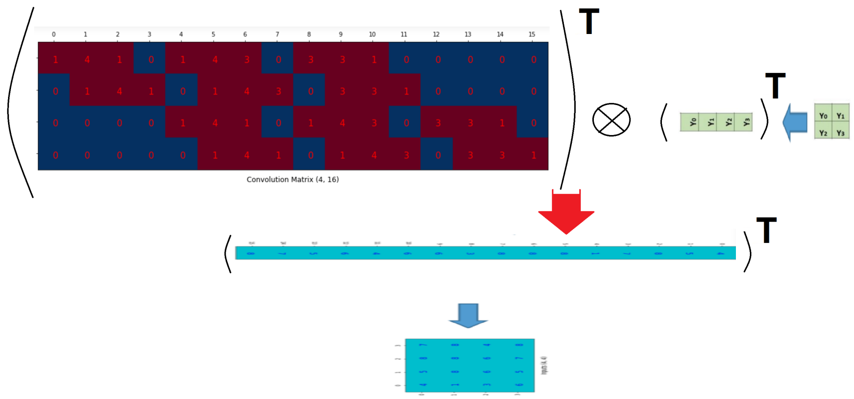
Ventricle of Human Heart | **Mask indicating ventricle location** | **Predicted Mask**

U-Net†



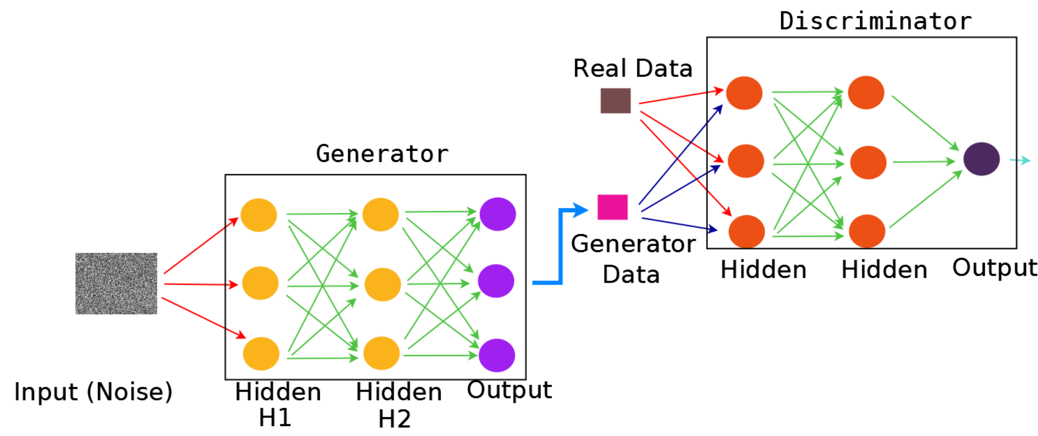
† U-Net: Convolutional Networks for Biomedical Image Segmentation. O. Ronneberger, P. Fischer, T. Brox. MICCAI, 2015.

U-Net: Upsampling



Upsampling operation (or Transpose Convolution)
Source: medium.com. towardsdatascience.com

GAN



GAN

- Aim of Discriminator component D:
  - Maximize likelihood of real data from distribution p_{data}
  - Minimize likelihood of fake data $G(z)$ obtained through generator G, where z comes from noise distribution p_{noise}.
- This is equivalent to
  - Maximize **log** likelihood of real data from distribution p_{data}
  - Minimize **log** likelihood of fake data $G(z)$ obtained through generator G, where z comes from noise distribution p_{noise}.

GAN

- Aim of Discriminator component D:
  - Maximize log likelihood of real data from distribution p_{data}:
    - ★ This is represented as

$$\max \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x).$$

    - ★ Here x denotes real data samples from distribution p_{data}.
    - ★ θ_d represents the parameters of the discriminator D.
    - ★ $D_{\theta_d}(x)$ denotes the output from discriminator parametrized by θ_d, when input to discriminator is x.
    - ★ **Assumption:** $D_{\theta_d}(x)$ denotes a probability or likelihood.

GAN

- Aim of Discriminator component D:
  - Maximize log likelihood of real data from distribution p_{data}:
    - This is represented as
$$\max \mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x).$$
  - Minimize likelihood of fake data $G(z)$ obtained through generator G, where z comes from noise distribution p_{noise}.
    - This is represented as:
$$\min \mathbb{E}_{z\sim p_{noise}} \log D_{\theta_d}(G(z)).$$
    - Here z denotes noise from distribution p_{noise}.
    - $G(z)$ denotes the generator's output when the noise z is passed through the generator.
    - $D_{\theta_d}(G(z))$ denotes the discriminator's output when the generator's output $G(z)$ considered as a fake sample is passed through the discriminator.
    - **Recall:** $D_{\theta_d}(\cdot)$ denotes a probability or likelihood.

GAN

- Aim of Discriminator component D:
  - Maximize log likelihood of real data from distribution p_{data}:
    - This is represented as
$$\max \mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x).$$
  - Minimize log likelihood of fake data $G(z)$ obtained through generator G, where z comes from noise distribution p_{noise}.
    - This is represented as:
$$\min \mathbb{E}_{z\sim p_{noise}} \log D_{\theta_d}(G(z)).$$

GAN

- Objective of Discriminator component D: (straightforward)
$$\max_{\theta_d} \mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p_{noise}} \log(1 - D_{\theta_d}(G(z))).$$

Assumption: $D(x)$ is a probability (or likelihood)

GAN

- Aim of Generator component G:
  - Fool the discriminator by providing $G(z)$ as if it comes from p_{data}.
  - How to do that?
- Idea:
$$\min_{\theta_g} \mathbb{E}_{z\sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))).$$

GAN

- Overall Objective of GAN (seems to be):

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \min_{\theta_g} \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right].$$

- Looks like a max-min problem

GAN

- Overall Objective of GAN (used in paper):

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right].$$

- This is a min-max problem

GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

  end for
  - Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN

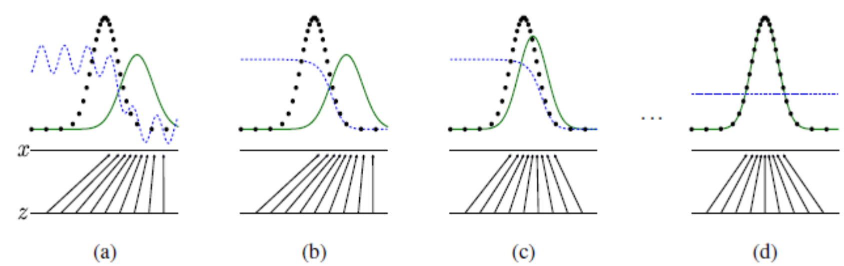


Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)}$. (c) After an update to G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$.

GAN

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

GAN - Mode Collapse Problem

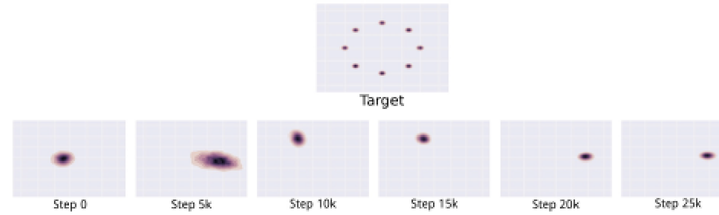


Figure 22: An illustration of the mode collapse problem on a two-dimensional toy dataset. In the top row, we see the target distribution p_{data} that the model should learn. It is a mixture of Gaussians in a two-dimensional space. In the lower row, we see a series of different distributions learned over time as the GAN is trained. Rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. Images from Metz *et al.* (2016).

GAN Training

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
Soumith Chintala

DCGAN

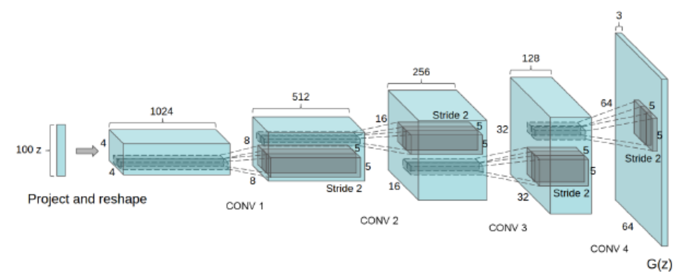


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

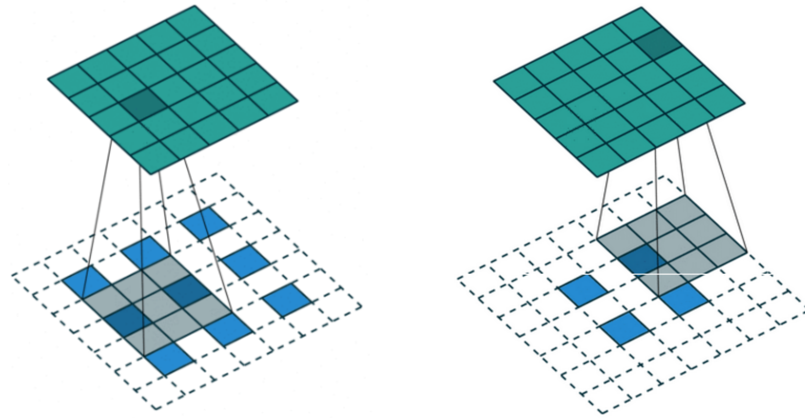
DCGAN Training

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- $\boxed{\text{Use ReLU activation in generator for all layers except for the output, which uses Tanh.}}$
- $\boxed{\text{Use LeakyReLU activation in the discriminator for all layers.}}$

DCGAN Training

Fractional Convolutions:



https://github.com/vdumoulin/conv_arithmetic

DCGAN Results

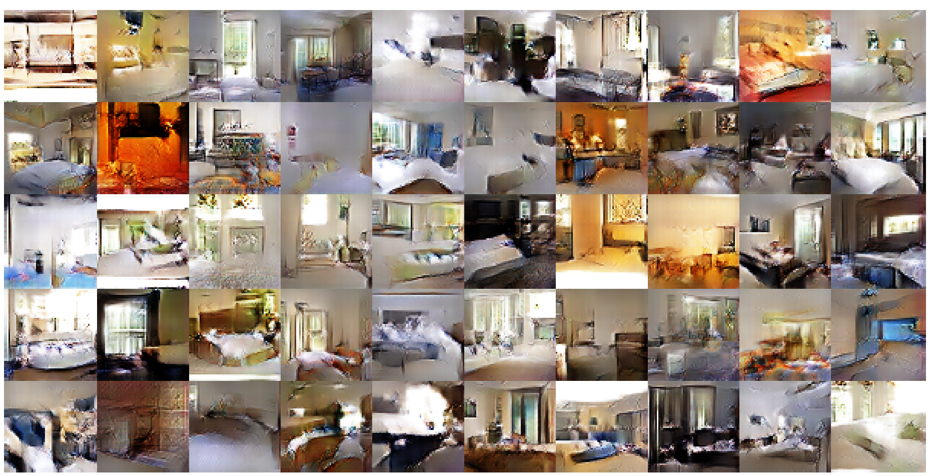


Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

DCGAN Results

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

DCGAN Results

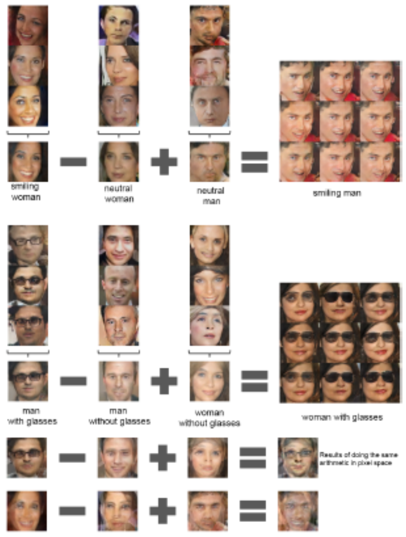


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

DCGAN Results

Figure 8: A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

DCGAN Results

Figure 10: More face generations from our Face DCGAN.

Conditional GAN

Conditional Generative Adversarial Nets

Mehdi Mirza
Simon Osindero

Conditional generative adversarial nets for convolutional face generation

Jon Gauthier

Conditional GAN

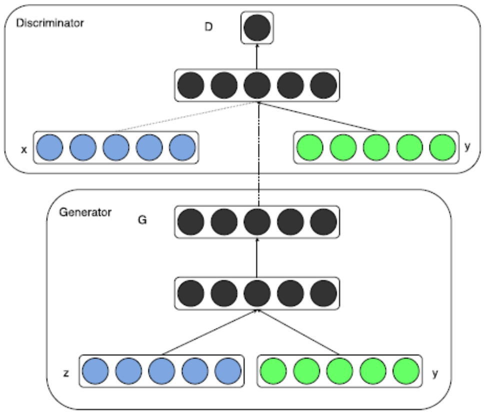
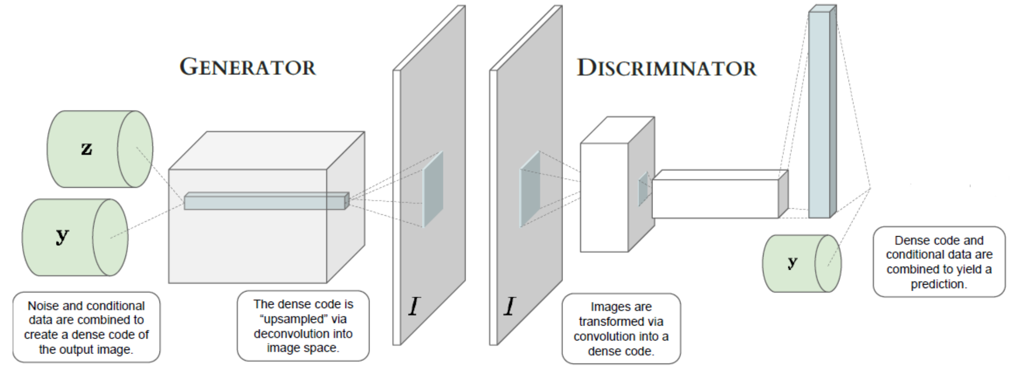


Figure 1: Conditional adversarial net

Conditional GAN



Conditional GAN

- Overall Objective of conditional GAN:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{(x,y)\sim p_{data}} \log D_{\theta_d}(x,y) + \mathbb{E}_{y\sim p_y, z\sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z,y),y)) \right].$$

Conditional GAN

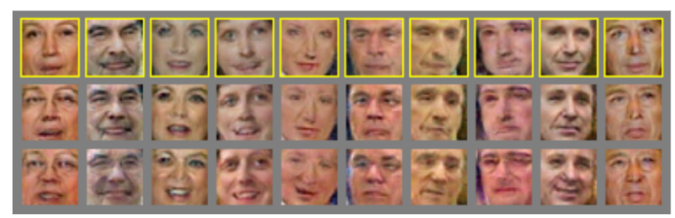


Figure 7: Results of constant additive shifts along particular axes of the conditional data y. (Full details of this shift are given in Section 4.3.1.) Row 1: randomly sampled images from the generator. Row 2: constant additive shift along the SENIOR axis; faces visibly age in the resulting samples. Row 3: constant additive shift along the MOUTHOPEN axis; faces open mouths / smile.

Disco GAN

Learning to Discover Cross-Domain Relations with Generative Adversarial Networks

Taeksoo Kim[1] Moonsu Cha[1] Hyunsoo Kim[1] Jung Kwon Lee[1] Jiwon Kim[1]

Disco GAN

(a) Learning cross-domain relations **without any extra label**

(b) Handbag images (input) & **Generated** shoe images (output)

(c) Shoe images (input) & **Generated** handbag images (output)

Disco GAN

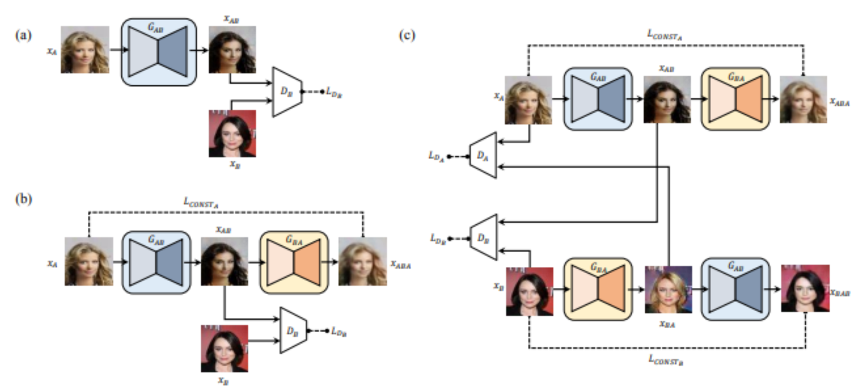
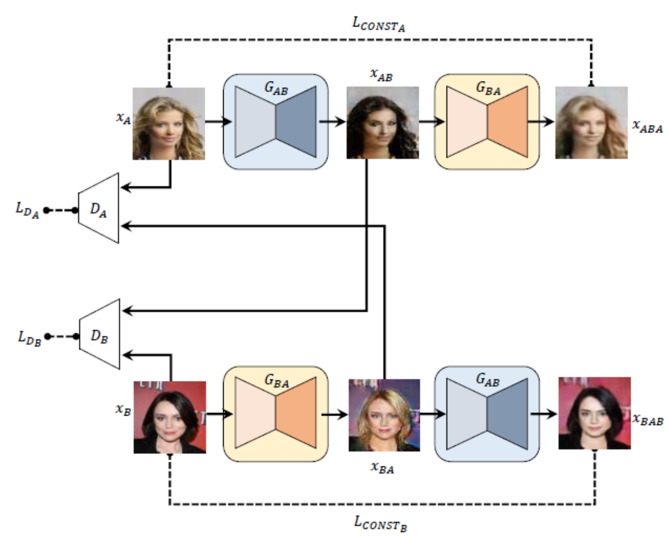


Figure 2. Three investigated models. (a) standard GAN (Goodfellow et al., 2014), (b) GAN with a reconstruction loss, (c) our **proposed model (DiscoGAN)** designed to discover relations between two unpaired, unlabeled datasets. Details are described in Section 3.

Disco GAN



Disco GAN - Components

- Two domains A and B
- Examples:
  - A = Vehicles B = Mesh Diagrams
  - A = Face Emotions B = Music
  - A = Adult animals B = Cubs
  - A = Actors B = Videos
- Two generators G_{AB} and G_{BA}
- Two discriminators D_A and D_B

Disco GAN - Components

- Generator G_{AB}
  - Randomly sample x_A from A
  - Form $x_{AB} = G_{AB}(x_A)$
  - Get back $x_{ABA} = G_{BA}(x_{AB}) = G_{BA}(G_{AB}(x_A))$
  - Reconstruction loss: $R_{G_{AB}} = \mathbb{E}_{x_A \sim A} d(x_A, x_{ABA})$
  - Discriminator loss: $GAN_{G_{AB}} = \mathbb{E}_{x_A \sim A} \log[1 - D_B(G_{AB}(x_A))]$
  - Total loss: $\min L_{G_{AB}} = R_{G_{AB}} + GAN_{G_{AB}}$

Disco GAN - Components

- Generator G_{AB}
  - Randomly sample x_A from A
  - Form $x_{AB} = G_{AB}(x_A)$
  - Get back $x_{ABA} = G_{BA}(x_{AB}) = G_{BA}(G_{AB}(x_A))$
  - Reconstruction loss: $R_{G_{AB}} = \mathbb{E}_{x_A \sim A} d(x_A, x_{ABA})$
  - Discriminator loss: $GAN_{G_{AB}} = \mathbb{E}_{x_A \sim A} \log[1 - D_B(G_{AB}(x_A))]$
  - Total loss: $\min L_{G_{AB}} = R_{G_{AB}} + GAN_{G_{AB}}$
- Generator G_{BA}
  - Randomly sample x_B from B
  - Form $x_{BA} = G_{BA}(x_B)$
  - Get back $x_{BAB} = G_{AB}(x_{BA}) = G_{AB}(G_{BA}(x_B))$
  - Reconstruction loss: $R_{G_{BA}} = \mathbb{E}_{x_B \sim B} d(x_B, x_{BAB})$
  - Discriminator loss: $GAN_{G_{BA}} = \mathbb{E}_{x_B \sim B} \log[1 - D_A(G_{BA}(x_B))]$
  - Total loss: $\min L_{G_{BA}} = R_{G_{BA}} + GAN_{G_{BA}}$

Disco GAN - Components

- Discriminator D_A
  - Randomly sample x_A from A
  - Compute $L_{D_A}^{real} = \mathbb{E}_{x_A \sim A} \log D_A(x_A)$
  - Obtain input from generator G_{BA}
  - Compute $L_{D_A}^{gen} = \mathbb{E}_{x_A \sim A} \log[1 - D_A(G_{BA}(x_B))]$
  - Total loss: $\max L_{D_A} = L_{D_A}^{real} + L_{D_A}^{gen}$.
- Discriminator D_B
  - Randomly sample x_B from B
  - Compute $L_{D_B}^{real} = \mathbb{E}_{x_B \sim B} \log D_B(x_B)$
  - Obtain input from generator G_{AB}
  - Compute $L_{D_B}^{gen} = \mathbb{E}_{x_B \sim B} \log[1 - D_B(G_{AB}(x_A))]$
  - Total loss: $\max L_{D_B} = L_{D_B}^{real} + L_{D_B}^{gen}$.

Disco GAN - Components

- Total Generator loss: $L_{Gen} = \min_{\theta_{gen}} [L_{G_{AB}} + L_{G_{BA}}]$
- Total Discriminator loss: $L_{Disc} = \max_{\theta_{disc}} [L_{D_A} + L_{D_B}]$

Disco GAN

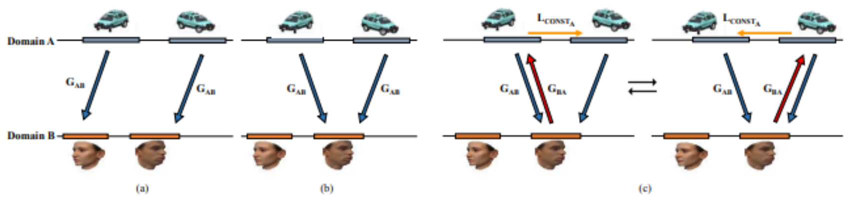


Figure 3. Illustration of our models on simplified one dimensional domains. (a) ideal mapping from domain A to domain B in which the two domain A modes map to two different domain B modes, (b) GAN model failure case, (c) GAN with reconstruction model failure case.

Disco GAN

Figure 6. Face to Face translation experiment. (a) input face images from -90° to +90° (b) results from a standard GAN (c) results from GAN with a reconstruction loss (d) results from our Disco-GAN. Here our model generated images in the opposite range, from +90° to -90°.

Disco GAN

Figure 7. (a,b) Translation of gender in Facescrub dataset and CelebA dataset. (c) Blond to black and black to blond hair color conversion in CelebA dataset. (d) Wearing eyeglasses conversion in CelebA dataset (e) Results of applying a sequence of conversion of gender and hair color (left to right) (f) Results of repeatedly applying the same conversions (upper: hair color, lower: gender)

Disco GAN

(a) Chair to Car (b) Car to Face

Figure 8. Discovering relations of images from visually very different object classes. (a) chair to car translation. DiscoGAN is trained on chair and car images (b) car to face translation. DiscoGAN is trained on car and face images. Our model successfully pairs images with similar orientation.

Disco GAN

Figure 9. Edges to photos experiment. Our model is trained on a set of object sketches and colored images and learns to generate new sketches or photos. (a) colored images of handbags are generated from sketches of handbags, (b) colored images of shoes are generated from sketches of shoes, (c) sketches of handbags are generated from colored images of handbags

Style GAN

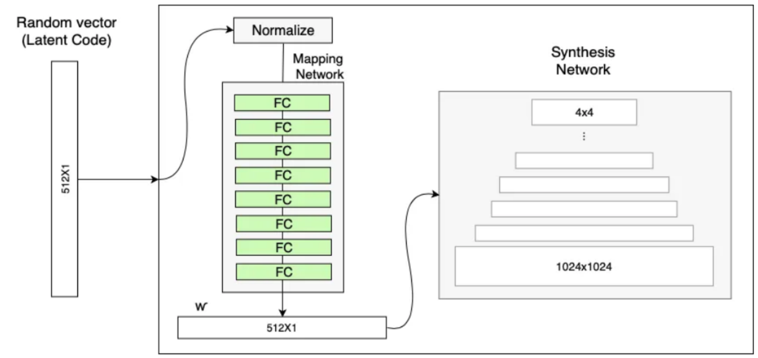
A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras
NVIDIA
tkarras@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

Timo Aila
NVIDIA
taila@nvidia.com

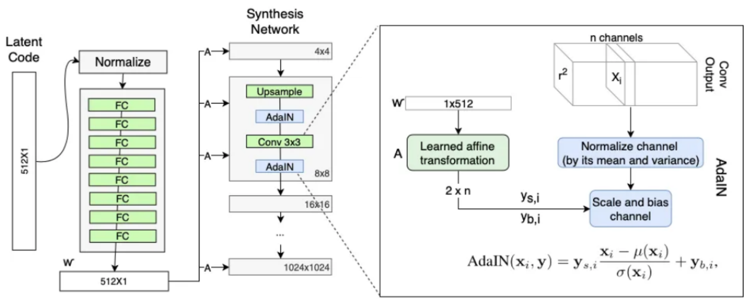
Style GAN



The generator with the Mapping Network (in addition to the ProGAN synthesis network)

Image Source: `https://towardsdatascience.com/`

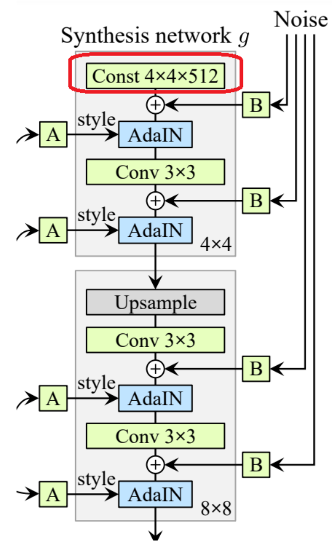
Style GAN



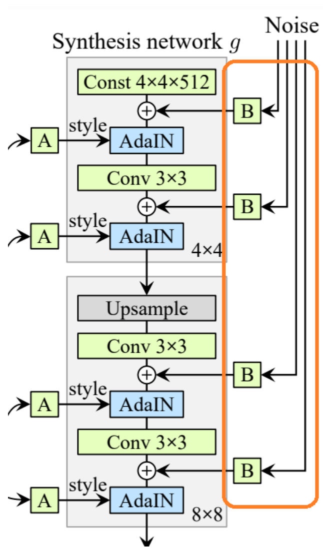
The generator's Adaptive Instance Normalization (AdaIN)

Image Source: `https://towardsdatascience.com/`

Style GAN



Style GAN



Style GAN

(a) Generated image (b) Stochastic variation

Figure 4. Examples of stochastic variation. (a) Two generated images. (b) Zoom-in with different realizations of input noise. While the overall appearance is almost identical, individual hairs are placed very differently.

Style GAN

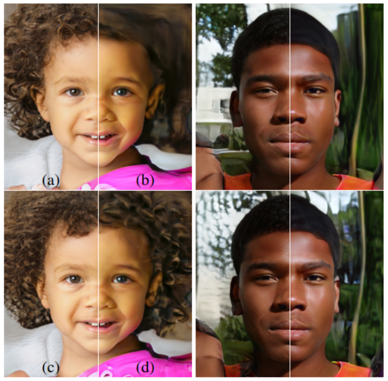
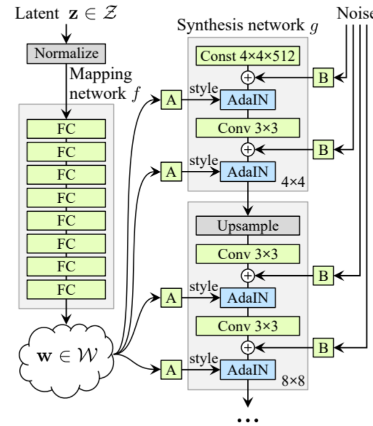


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

Style GAN



Style-based generator

Style GAN

Other techniques tried in the paper

- Mixing multiple latent vectors during training (style mixing)
  - Sample z_1 and z_2 and obtain corresponding w_1 and w_2 from mapping network $\mathcal{W}$.
  - During the adaIn operations, w_1 is considered as inputs for some initial layers and w_2 is considered as inputs for later layers.
  - This has the effect of reducing correlation impacts among different adjacent styles.

Style GAN

Performance metrics proposed in the paper

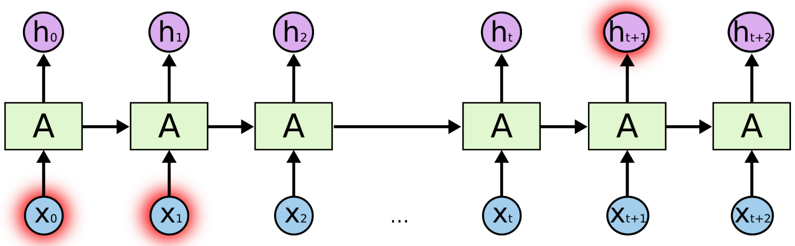
- Frechet Inception Distance (FID)
  - Compares the means and covariances of gaussians fit on the true and generated data. (Check the formula for FID!)
- Perceptual path length
  - $\mathbb{E}[\frac{1}{\epsilon^2} d(G(\text{slerp}(z_1, z_2, t)), G(slerp(z_1, z_2, t+\epsilon)))]$
  - slerp denotes spherical interpolation.
  - G denotes the generator.
  - d denotes the quadratic distance.
  - ϵ denotes a perturbation.
- Linear separability for coarse styles: e.g. gender, pose.

Style GAN

Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions ($4^2 - 8^2$) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions ($16^2 - 32^2$) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles ($64^2 - 1024^2$) from B brings mainly the color scheme and microstructure.

Recurrent Neural Network - Drawbacks

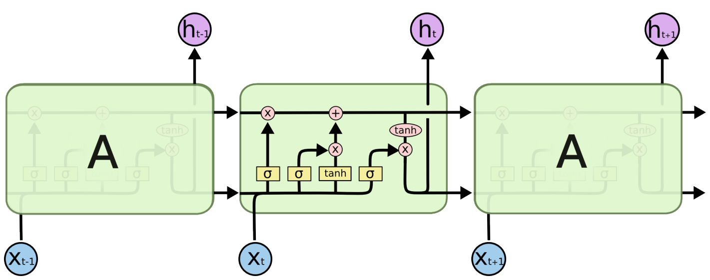
- Long-term dependencies



- Similar to vanishing gradient problem in feed-forward networks

RNNs - Remembering Long Term Dependencies

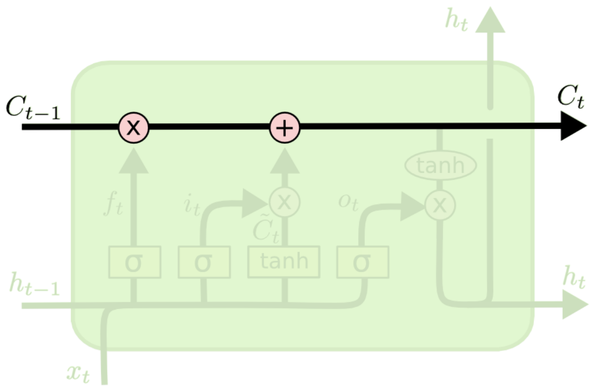
- Long Short Term Memories (LSTM)



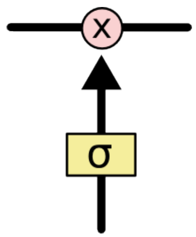
Pictures from:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM - Cell Structure

Cell State Information

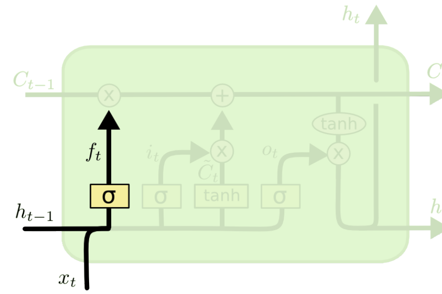


LSTM - Cell Structure



Gate

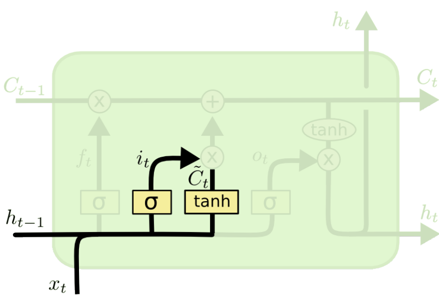
LSTM - Cell Structure



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

Forget Gate

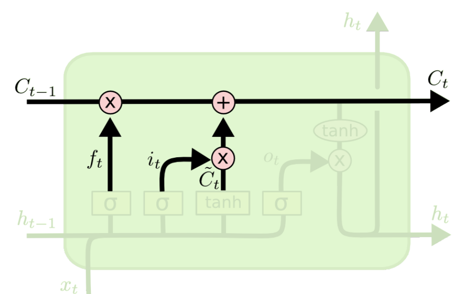
LSTM - Cell Structure



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Input Gate and Update Cell State Information

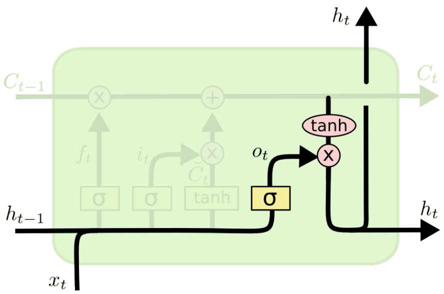
LSTM - Cell Structure



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

Cell State Update

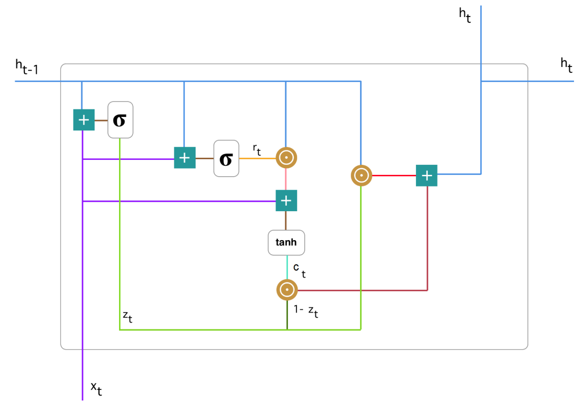
LSTM - Cell Structure



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

Output Gate

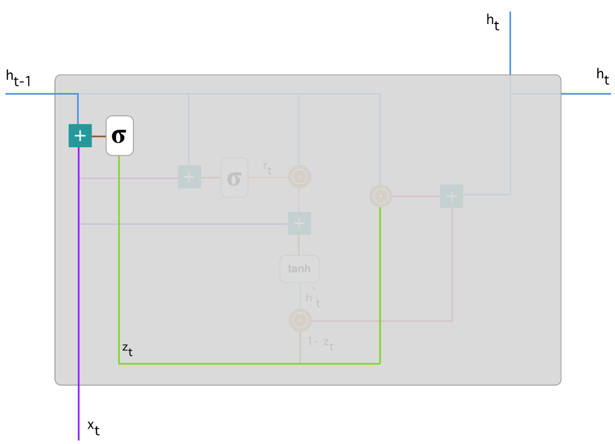
GRU - Cell Structure



Pictures from: `https://towardsdatascience.com/`

GRU - Cell Structure

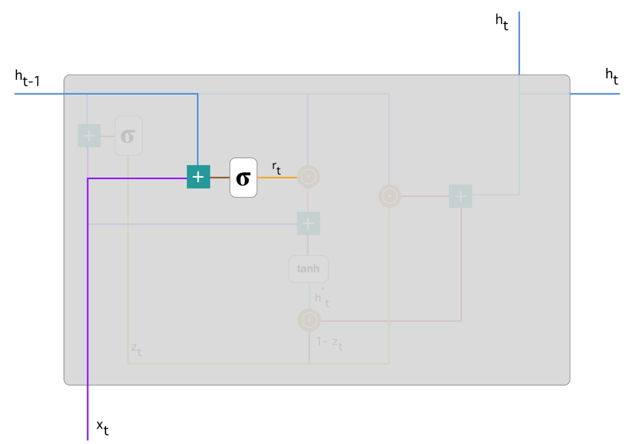
Update Gate



$$z_t = \sigma(W^z x_t + U^z h_{t-1})$$

GRU - Cell Structure

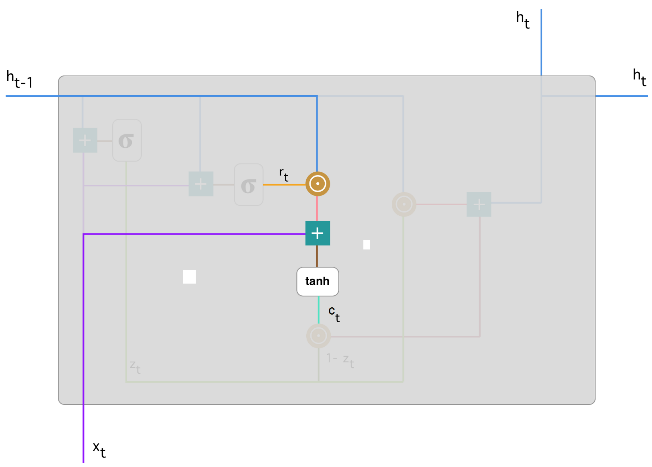
Reset Gate



$$r_t = \sigma(W^r x_t + U^r h_{t-1})$$

GRU - Cell Structure

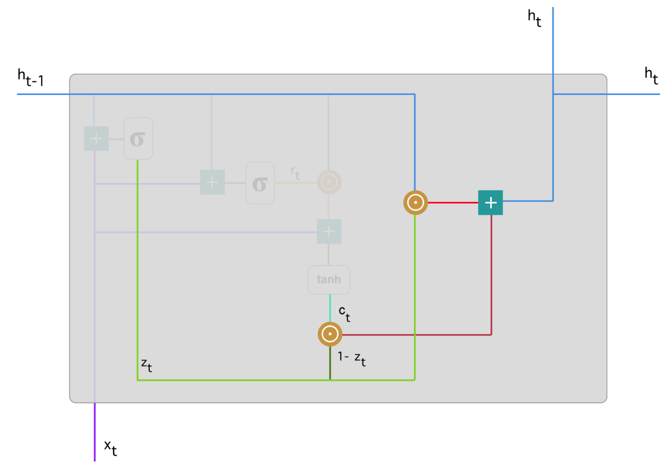
Current Cell State



$$c_t = \tanh(W^c x_t + r_t \odot U^c h_{t-1})$$

GRU - Cell Structure

Final Cell Output



$$h_t = \sigma(z_t \odot c_t + (1 - z_t) \odot h_{t-1})$$

Language Modeling using LSTM

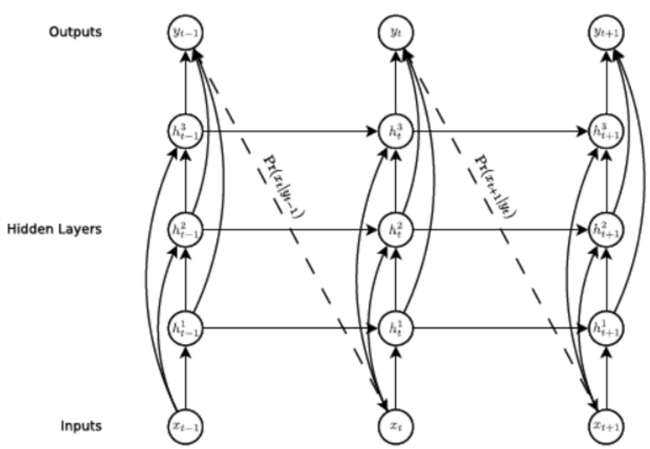


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

Language Modeling using LSTM

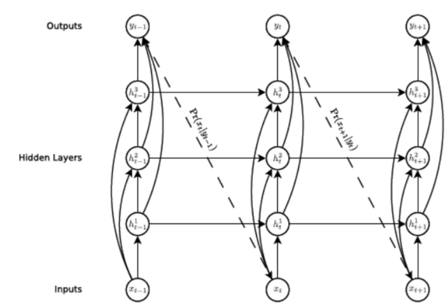


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

- Input sequence: $\mathbf{x} = (x_1, x_2, \ldots, x_T)$.
- Output sequence: $\mathbf{y} = (y_1, x_2, \ldots, y_T)$.

Language Modeling using LSTM

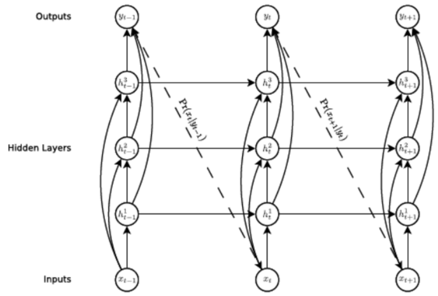


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

- Proposed network consists of a stack of N LSTM cells.
- $h_t^1 = \phi_H(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + b_{h^1})$
- $h_t^n = \phi_H(W_{ih^n}x_t + W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_{h^n}), n = 2, \ldots, N.$
- $\hat{y}_t = \sum_{n=1}^{N} W_{h^ny}h_t^n + b_y.$

Language Modeling using LSTM

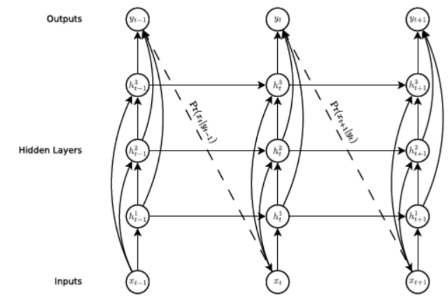


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

- Used to model the probability $P(\mathbf{x}) = \Pi_{t=1}^{T} P(x_{t+1}|y_t)$.
- Loss function: $\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} \log P(x_{t+1}|y_t)$.

Language Modeling using LSTM

Table 1: **Penn Treebank Test Set Results.** 'BPC' is bits-per-character. 'Error' is next-step classification error rate, for either characters or words.

Input	Regularisation	Dynamic	BPC	Perplexity	Error (%)	Epochs
Char	none	no	1.32	167	28.5	9
char	none	yes	1.29	148	28.0	9
char	weight noise	no	1.27	140	27.4	25
char	weight noise	yes	1.24	124	26.9	25
char	adapt. wt. noise	no	1.26	133	27.4	26
char	adapt. wt. noise	yes	1.24	122	26.9	26
word	none	no	1.27	138	77.8	11
word	none	yes	1.25	126	76.9	11
word	weight noise	no	1.25	126	76.9	14
word	weight noise	yes	1.23	117	76.2	14

- BPC: Average of $-\log P(x_{t+1}|y_t)$.
- Perplexity for char prediction: $2^{c*(BPC)}$, where c is the average number of chars per word (or) average length of words.

Language Modeling using LSTM

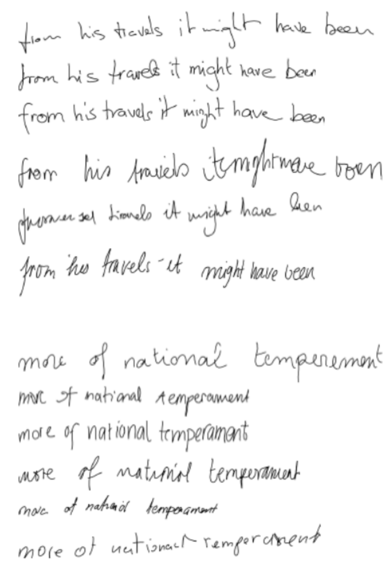


Figure 15: **Real and generated handwriting.** The top line in each block is real, the rest are unbiased samples from the synthesis network. The two texts are from the validation set and were not seen during training.

Language Modeling using LSTM

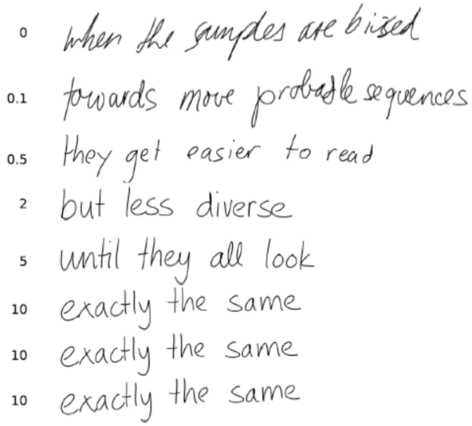


Figure 16: **Samples biased towards higher probability.** The probability biases b are shown at the left. As the bias increases the diversity decreases and the samples tend towards a kind of 'average handwriting' which is extremely regular and easy to read (easier, in fact, than most of the real handwriting in the training set). Note that even when the variance disappears, the same letter is not written the same way at different points in a sequence (for examples the 'e's in "exactly the same", the 'l's in "until they all look"), because the predictions are still influenced by the previous outputs. If you look closely you can see that the last three lines are not quite exactly the same.

Language Modeling using LSTM

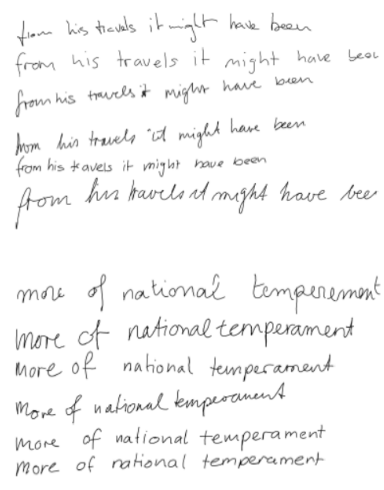


Figure 17: **A slight bias.** The top line in each block is real. The rest are samples from the synthesis network with a probability bias of 0.15, which seems to give a good balance between diversity and legibility.

Cloze Style Comprehension using LSTM

Table 1. An example document excerpt, question and answer.

Document @entity5 (@entity6) an impressive art collection assembled by the late actress and @entity3 icon , @entity4 , has officially been offered for purchase. the collection , which includes works by some of the greatest artists of the 20th century , went under the hammer in @entity13 on march 31 , following a tour of @entity5 , @entity15 , @entity16 and @entity17 . ... the 750 - piece collection , which fetched a total of $ 3.64 million , featured bronze sculptures , jewelry , and a number of decorative arts and paintings , which were sold at @entity50 auction house in @entity13 . ...
Question a collection of 750 items belonging to legendary actress @entity4 has been auctioned off at @entity50 in **@placeholder**
Answer @entity13

Cloze Style Comprehension using LSTM - Multipass Reader

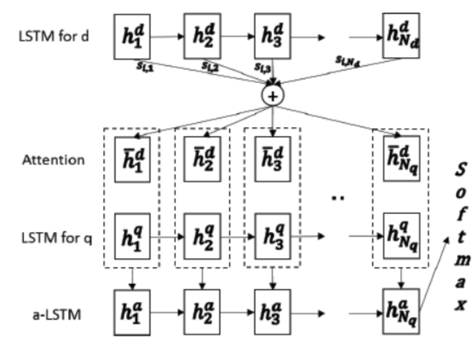


Fig. 1. The Multi-pass Reader model. The dot rectangular is the concatenation of a document attention representation and a question state.

- Input: Document $X^d = (x_1^d, x_2^d, \ldots, x_{N_d}^d)$, Question $X^q = (x_1^q, x_2^q, \ldots, x_{N_q}^q)$
- Expected Output: $a = (x_1^a)$
- Training Data for Multi-pass reader: (X^d, X^q, a)

Cloze Style Comprehension using LSTM - Multipass Reader

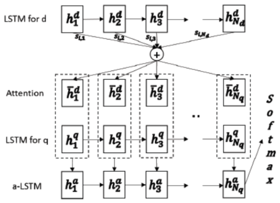


Fig. 1. The Multi-pass Reader model. The dot rectangular is the concatenation of a document attention representation and a question state.

- For i-th word of Question and j-th word in Document, an attention score $s_{i,j}$ is constructed:

$$m_{i,j} = \tanh(W^q h_i^q + W^d h_j^d + W^a h_{i-1}^a)$$
$$s_{i,j} = \exp(w^s m_{i,j})$$

- Then for the i-th word of Question a total score $\bar{h}_i^{q,d} = \sum_j s_{i,j}$ is constructed.
- For the answer LSTM, $\begin{pmatrix} \bar{h}_i^{q,d} & h_i^q \end{pmatrix}$ is used as the inputs.
- Final layer of answer LSTM is expected to give the output.

Cloze Style Comprehension using LSTM - Single Pass Reader

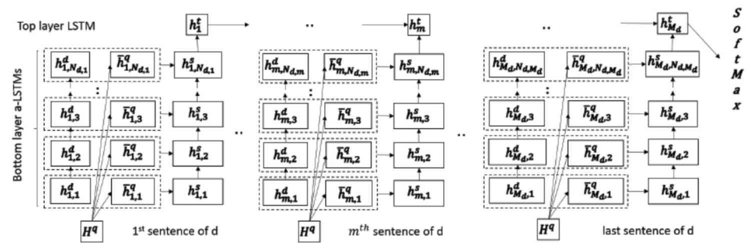


Fig. 2. The Single-pass Reader model. Here $\mathbf{h}_{m,n}^d$ refers to the hidden state produced by some preprocessing LSTM for the n^{th} token in the m^{th} sentence in document d, and $N_{d,m}$ is the number of tokens in the m^{th} sentence in d. $\mathbf{H}^q$ represents the hidden states produced by LSTM for the tokens in question and $\bar{\mathbf{h}}_{m,i}^q$ is the weighted sum of question hidden states $\mathbf{H}^q$.

- Input: Document is split into sentences $X^d = \{X_1^d, X_2^d, \ldots, X_{M_d}^d\}$ where $X_j^d = (x_{1,j}^d, x_{2,j}^d, \ldots, x_{N_d,j}^d)$, Question $X_q = (x_1^q, x_2^q, \ldots, x_{N_q}^q)$.
- Expected Output: $a = (x_1^a)$

Cloze Style Comprehension using LSTM - Single Pass Reader

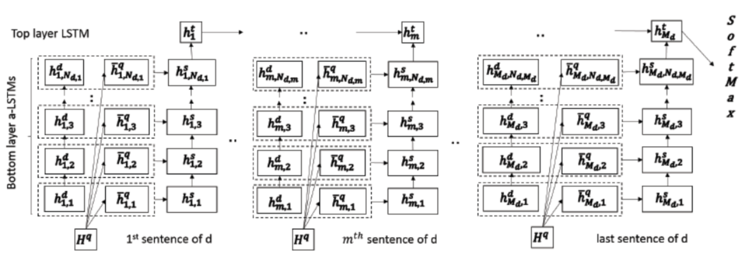


Fig. 2. The Single-pass Reader model. Here $\mathbf{h}_{m,n}^d$ refers to the hidden state produced by some preprocessing LSTM for the n^{th} token in the m^{th} sentence in document d, and $N_{d,m}$ is the number of tokens in the m^{th} sentence in d. $\mathbf{H}^q$ represents the hidden states produced by LSTM for the tokens in question and $\bar{\mathbf{h}}_{m,i}^q$ is the weighted sum of question hidden states $\mathbf{H}^q$.

- Training Data for single-pass reader: $\{(X^q, X_j^d, a)\}_{j=1}^{M_d}$
- For i-th word of Question and j-th word in sentence X_m^d, an attention score $s_{i,j}$ is constructed.

Cloze Style Comprehension using LSTM - Single Pass Reader

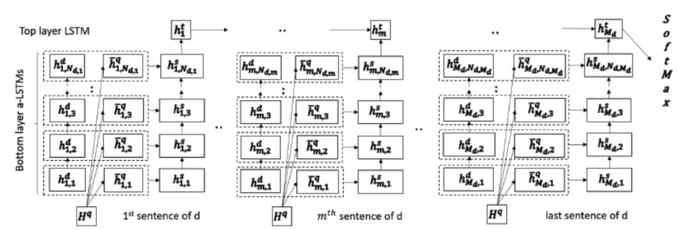


Fig. 2. The Single-pass Reader model. Here $\mathbf{h}_{m,n}^d$ refers to the hidden state produced by some preprocessing LSTM for the n^{th} token in the m^{th} sentence in document d, and $N_{d,m}$ is the number of tokens in the m^{th} sentence in d. $\mathbf{H}^q$ represents the hidden states produced by LSTM for the tokens in question and $\bar{\mathbf{h}}_{m,i}^q$ is the weighted sum of question hidden states $\mathbf{H}^q$.

- Then for the i-th word of Question a total score $\bar{h}_i^{q,d_m} = \sum_j s_{i,j}$ is constructed.
- For each $\{(X^q, X_j^d)\}$ an answer LSTM is constructed.
- For the answer LSTM, $\begin{pmatrix} \bar{h}_i^{q,d_m} & h_i^q \end{pmatrix}$ is used as the inputs.

Cloze Style Comprehension using LSTM - Single Pass Reader

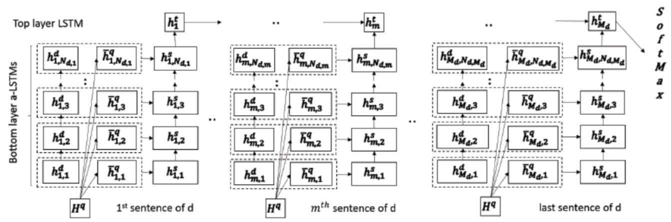


Fig. 2. The Single-pass Reader model. Here $\mathbf{h}^d_{m,n}$ refers to the hidden state produced by some preprocessing LSTM for the n^{th} token in the m^{th} sentence in document d, and $N_{d,m}$ is the number of tokens in the m^{th} sentence in d. $\mathbf{H}^q$ represents the hidden states produced by LSTM for the tokens in question and $\overline{\mathbf{h}}^q_{m,i}$ is the weighted sum of question hidden states $\mathbf{H}^q$.

- Final layer of each answer LSTM is input into a top-layer LSTM which aggregates the information across the document and remembers the answer from sentences which have closest semantics as the question (and therefore might constain the answer). The top-layer LSTM forgets the other irrelevant sentences.

Cloze Style Comprehension using LSTM

Table 2. Some statistics of the CNN/Daily Mail dataset. The last row refers to the relabeled data by [10], where the entities in a document were re-labeled in order, such that the first entity is labeled as @entity1, the second entity as @entity2, etc., rather than being randomly labeled.

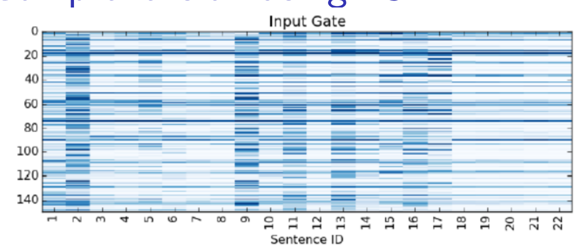
	CNN	Daily Mail
# Train	380,298	879,450
# Dev	3,924	64,835
# Test	3,198	53,182
avg # tokens per doc	762	813
avg # sentences per doc	33.4	32.6
avg # entities per doc	26.2	26.2
vocab size	118,497	208,045
# classes	405	415
# classes (relabeled)	145	156

Cloze Style Comprehension using LSTM

Table 3. Results on the CNN and the Daily Mail data sets with a single model. Note that the bottom section shows the performance of the models trained on the relabeled data sets.

Model	CNN dev	CNN test	Daily Mail dev	Daily Mail test
Attentive Reader [1]	61.6	63.0	70.5	69.0
Impatient Reader [1]	61.8	63.8	69.0	68.0
MemNN [9]	63.4	66.8	N/A	N/A
Entity-Centric Classifier [10]	67.1	67.9	69.1	68.3
Attention Sum Reader [11]	68.6	69.5	75.0	73.9
Stanford Attentive Reader [10]	72.5	72.7	76.9	76.0
DER Network [22]	71.3	72.9	N/A	N/A
EpiReader [7]	73.4	74.0	N/A	N/A
AOA Reader [12]	73.1	74.4	N/A	N/A
ReasoNet [14]	72.9	74.7	77.6	76.6
BiDAF [15]	76.3	76.9	80.3	79.6
GA [13]	77.9	77.9	81.5	80.9
Multi-pass Reader (our model)	67.5	70.0	73.9	73.5
Single-pass Reader (our model)	73.3	74.3	77.7	76.7
Stanford Attentive Reader (on relabeled data) [10]	73.8	73.6	77.6	76.6
Single-pass Reader (on relabeled data) (our model)	75.5	76.6	79.4	78.6

Cloze Style Comprehension using LSTM



Question	a collection of 750 items belonging to legendary actress @entity4 has been auctioned off at @entity50 in @placeholder
ID	**Context sentence**
2	the collection , which includes works by some of the greatest artists of the 20th century , went under the hammer in @entity13 on march 31 , following a tour of @entity5 , @entity15 , @entity16 and @entity17 .
9	the 750 - piece collection , which fetched a total of $ 3.64 million , featured bronze sculptures , jewelry , and a number of decorative arts and paintings , which were sold at @entity50 auction house in @entity13 . "

Fig. 4. The input gates of the top-layer LSTM in the single-pass reader for a document-question pair. Two sentences with high input gate values are also shown together with the question.

Neural Machine Translation using RNN

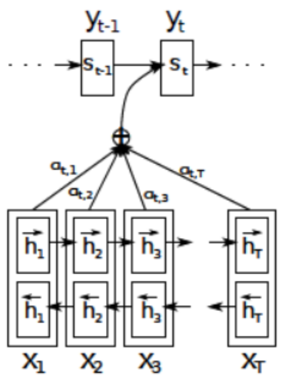


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Neural Machine Translation using RNN

- Input: $\mathbf{x} = (x_1, x_2, \ldots, x_T)$
- Expected Output: $\mathbf{y} = (y_1, y_2, \ldots, y_{T'})$
- Note 1: T and T' need not be same
- Note 2: x_t need not correspond directly to y_t
- Example:
  - $\mathbf{x}$=via digital platforms
  - $\mathbf{y}$=via des plateformes numériques

Neural Machine Translation using RNN

- Idea: To encode the input into a set of context vectors c_i corresponding to an output y_i
- First model annotations $(h_1, h_2, \ldots, h_T)$ for the given $\mathbf{x} = (x_1, x_2, \ldots, x_T)$
- The annotations h_t are modeled using bi-directional RNNs with $h_t = \begin{bmatrix} h_t^{fwd} \\ h_t^{rev} \end{bmatrix}$
- Compute $c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$ where

$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_{k=1}^{T} \exp(a(s_{i-1}, h_k))}$$

- s_i denotes the hidden state and is given by $s_i = f(y_{i-1}, s_{i-1}, c_i)$, where f is some suitable neural network and a is another neural network.

Neural Machine Translation using RNN

- Having constructed c_i corresponding to y_i, the aim is now to predict y_i using

$$p(y_i | y_1, y_2, \ldots, y_{t-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

where g is a neural network designed to maximize the probability of occurrence of y_i.

Neural Machine Translation using RNN

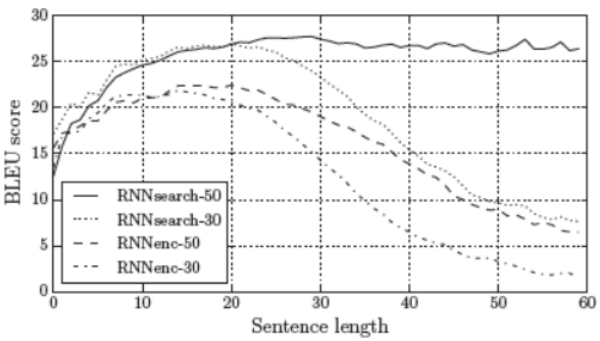


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Neural Machine Translation using RNN

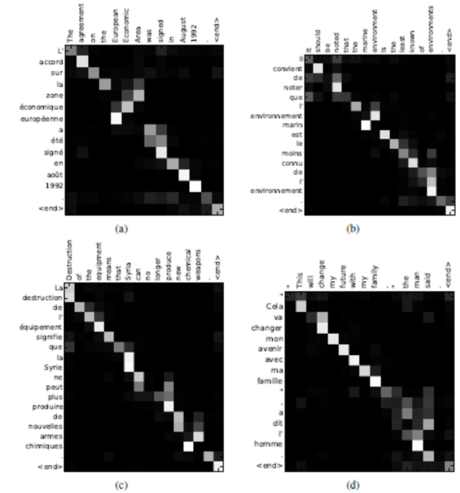


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word (see Eq. (6)), in grayscale (0: black, 1: white). (a) an arbitrary sentence. (b–d) three randomly selected samples among the sentences without any unknown words and of length between 10 and 20 words from the test set.

Neural Machine Translation using RNN

Source	An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.
Reference	Le privilège d'admission est le droit d'un médecin, en vertu de son statut de membre soignant d'un hôpital, d'admettre un patient dans un hôpital ou un centre médical afin d'y délivrer un diagnostic ou un traitement.
RNNenc-50	Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.
RNNsearch-50	Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.
Google Translate	Un privilège admettre est le droit d'un médecin d'admettre un patient dans un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, fondée sur sa situation en tant que travailleur de soins de santé dans un hôpital.
Source	This kind of experience is part of Disney's efforts to "extend the lifetime of its series and build new relationships with audiences via digital platforms that are becoming ever more important," he added.
Reference	Ce type d'expérience entre dans le cadre des efforts de Disney pour "étendre la durée de vie de ses séries et construire de nouvelles relations avec son public grâce à des plateformes numériques qui sont de plus en plus importantes", a-t-il ajouté.
RNNenc-50	Ce type d'expérience fait partie des initiatives du Disney pour "prolonger la durée de vie de ses nouvelles et de développer des liens avec les lecteurs numériques qui deviennent plus complexes.
RNNsearch-50	Ce genre d'expérience fait partie des efforts de Disney pour "prolonger la durée de vie de ses séries et créer de nouvelles relations avec des publics via des plateformes numériques de plus en plus importantes", a-t-il ajouté.
Google Translate	Ce genre d'expérience fait partie des efforts de Disney à "étendre la durée de vie de sa série et construire de nouvelles relations avec le public par le biais des plates-formes numériques qui deviennent de plus en plus important", at-il ajouté.
Source	In a press conference on Thursday, Mr Blair stated that there was nothing in this video that might constitute a "reasonable motive" that could lead to criminal charges being brought against the mayor.
Reference	En conférence de presse, jeudi, M. Blair a affirmé qu'il n'y avait rien dans cette vidéo qui puisse constituer des "motifs raisonnables" pouvant mener au dépôt d'une accusation criminelle contre le maire.
RNNenc-50	Lors de la conférence de presse de jeudi, M. Blair a dit qu'il n'y avait rien dans cette vidéo qui pourrait constituer une "motivation raisonnable" pouvant entraîner des accusations criminelles portées contre le maire.
RNNsearch-50	Lors d'une conférence de presse jeudi, M. Blair a déclaré qu'il n'y avait rien dans cette vidéo qui pourrait constituer un "motif raisonnable" qui pourrait conduire à des accusations criminelles contre le maire.
Google Translate	Lors d'une conférence de presse jeudi, M. Blair a déclaré qu'il n'y avait rien dans cette vido qui pourrait constituer un "motif raisonnable" qui pourrait mener à des accusations criminelles portes contre le maire.

Table 3: The translations generated by RNNenc-50 and RNNsearch-50 from long source sentences (30 words or more) selected from the test set. For each source sentence, we also show the gold-standard translation. The translations by Google Translate were made on 27 August 2014.

Neural Machine Translation using LSTM

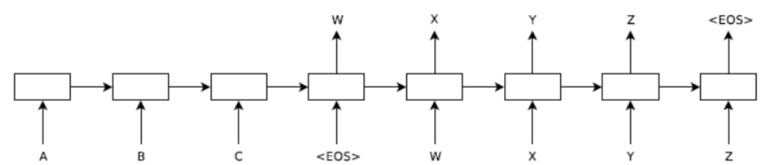


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Neural Machine Translation using RNN

- Input: $\mathbf{x} = (x_1, x_2, \ldots, x_T)$
- Expected Output: $\mathbf{y} = (y_1, y_2, \ldots, y_{T'})$
- Idea: To encode the input $\mathbf{x}$ into a single context vector c, then decode $\mathbf{y}$ from c.
- One LSTM used to construct the encoding from $\mathbf{x}$ to c
- Another LSTM used to construct the decoding of $\mathbf{y}$ from c
- During training, reversed training inputs were used !?

Neural Machine Translation using LSTM

$$p(y_i|y_1, y_2, \ldots, y_{t-1}, \mathbf{x}) = \hat{g}(y_{i-1}, s_i, c)$$

$\hat{g}$ is a suitable neural network and s_i is the hidden state of the RNN.

Neural Machine Translation using LSTM

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Neural Machine Translation using LSTM

Type	Sentence
Our model	Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .
Truth	Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .
Our model	" Les téléphones cellulaires , qui sont vraiment une question , non seulement parce qu' ils pourraient potentiellement causer des interférences avec les appareils de navigation , mais nous savons , selon la FCC , qu' ils pourraient interférer avec les tours de téléphone cellulaire lorsqu' ils sont dans l' air " , dit UNK .
Truth	" Les téléphones portables sont véritablement un problème , non seulement parce qu' ils pourraient éventuellement créer des interférences avec les instruments de navigation , mais parce que nous savons , d' après la FCC , qu' ils pourraient perturber les antennes-relais de téléphonie mobile s' ils sont utilisés à bord " , a déclaré Rosenker .
Our model	Avec la crémation , il y a un " sentiment de violence contre le corps d' un être cher " , qui sera " réduit à une pile de cendres " en très peu de temps au lieu d' un processus de décomposition " qui accompagnera les étapes du deuil " .
Truth	Il y a , avec la crémation , " une violence faite au corps aimé " , qui va être " réduit à un tas de cendres " en très peu de temps , et non après un processus de décomposition , qui " accompagnerait les phases du deuil " .

Table 3: A few examples of long translations produced by the LSTM alongside the ground truth translations. The reader can verify that the translations are sensible using Google translate.

Neural Machine Translation using LSTM

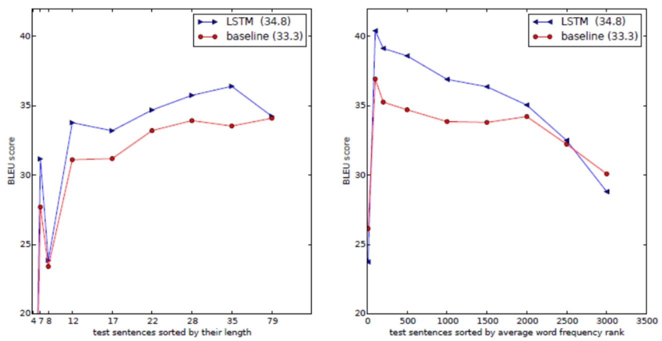


Figure 3: The left plot shows the performance of our system as a function of sentence length, where the x-axis corresponds to the test sentences sorted by their length and is marked by the actual sequence lengths. There is no degradation on sentences with less than 35 words, there is only a minor degradation on the longest sentences. The right plot shows the LSTM's performance on sentences with progressively more rare words, where the x-axis corresponds to the test sentences sorted by their "average word frequency rank".

Sequential outputs - Motivating Applications

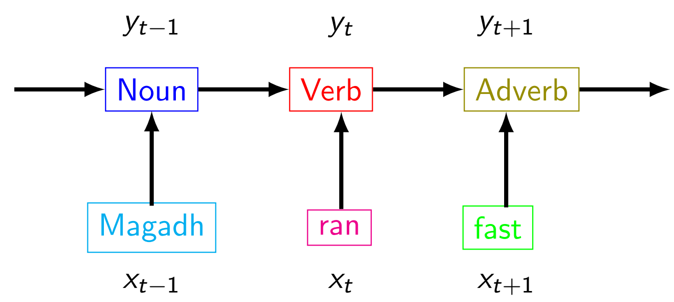


Figure: Part-of-Speech Tagging

Sequential outputs - Motivating Applications

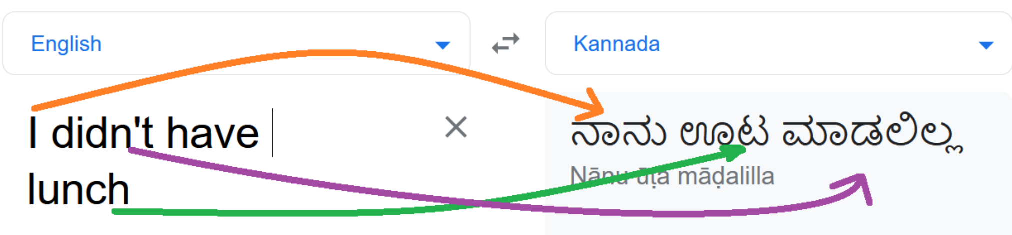


Figure: Language Translation

Sequential outputs - Motivating Applications

Figure: Protein Sequence Alignment

Earliest Recurrent Network Architectures

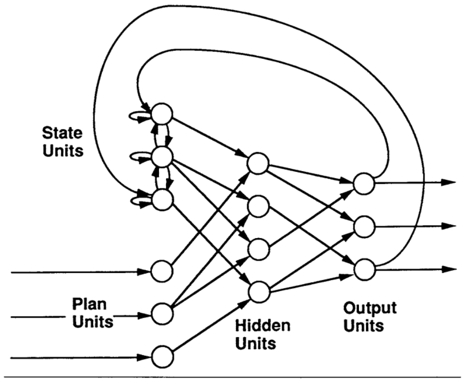


FIGURE 1. The processing units and basic interconnection scheme (not all connections are shown). The plan and state units together constitute the input units for the network.

Figure: Jordan Network†

† Michael Jordan. *Serial order: A parallel distributed processing approach (Tech. Rep. No. 8604).*, Institute for Cognitive Science, UCSD, San Diego, 1986

Earliest Recurrent Network Architectures

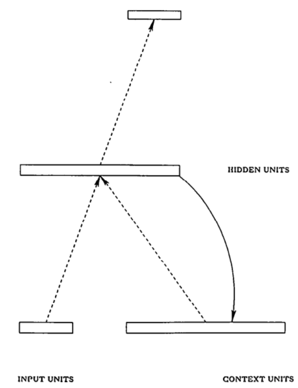


Figure . A simple recurrent network in which activations are copied from hidden layer to context layer on a one-for-one basis, with fixed weight of 1.0. Dotted lines represent trainable connections.

Figure: Elman Network‡

‡ Jeffrey L. Elman, *Finding structure in time*, Cognitive Science, Vol 14(2), pp. 179-211, 1990.

Recurrent Neural Network

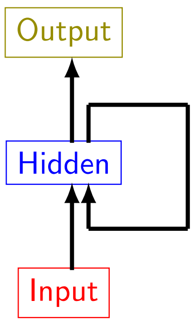


Figure: A simple recurrent network

Recurrent Neural Network

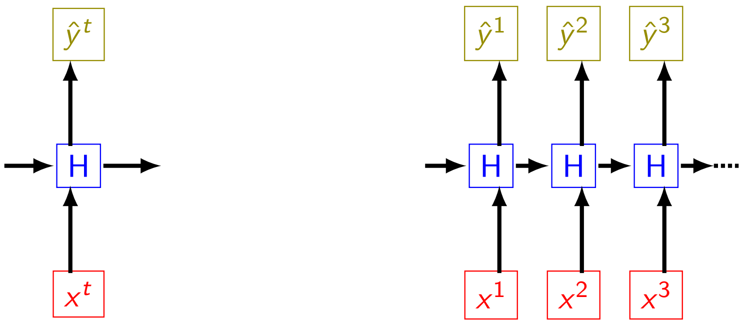


Figure: A simple recurrent network

Recurrent Layers - Formalism

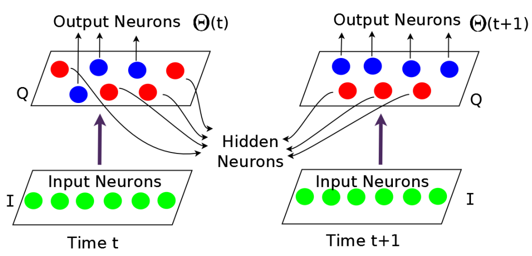


Figure: Recurrent layers

- S be the set of all neurons
- $S = \mathrm{I} \cup Q$
  - I is the set of input neurons
  - Q is the set of hidden neurons
- Some neurons in set Q can be used as output neurons at particular time steps (this set is denoted by $\Theta(t) \subseteq Q$).

Recurrent Layers - Formalism

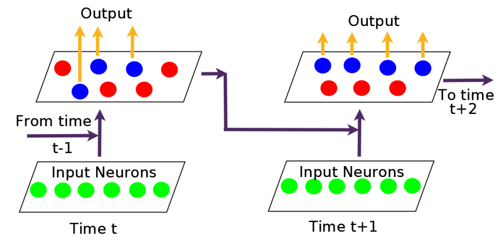


Figure: Recurrent layers

- At the time step t, we define:

$$\xi_k(t) = \begin{cases} a_k(t-1) & \text{if } k \in Q \\ x_k(t) & \text{if } k \in I \end{cases}$$

- $a_k(t-1)$ denotes activation of neuron k at time $t-1$.
- $x_k(t)$ denotes input at neuron k at time t.
- Note: $\hat{y}_k(t-1) = a_k(t-1)$.

Recurrent Layers - Formalism†

- At the time step t, we define:

$$\xi_k(t) = \begin{cases} a_k(t-1) & \text{if } k \in Q \\ x_k(t) & \text{if } k \in I \end{cases}$$

- For all neurons belonging to S we can now define:

$$z_k(t) = \sum_{\ell \in S} w_{k\ell}\xi_\ell(t) \quad \forall k \in S$$

$$a_k(t) = \phi[z_k(t)]$$

- ϕ denotes activation function. (*e.g.* sigmoidal, ReLU)

† R. J. Williams and D. Zipser, *Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity*, Backpropagation, L. Erlbaum Associates Inc. Hillsdale, NJ, USA. 1995.

Recurrent Layers - Formalism

- For time step t, denote the set of output neurons as $\Theta(t) \subseteq Q$.
- Let $y_k(t)$ be the actual output associated with neuron k at time t.
- Error at time step t for a particular neuron k:

$$e_k(t) = \begin{cases} y_k(t) - a_k(t) & \text{when } k \in \Theta(t), \\ 0 & \text{else.} \end{cases}$$

- Error at time step t can be defined as a squared error function:

$$E(t) = \sum_{k \in S} e_k^2(t)$$

- Net error over all time steps (assuming that the network gets initialized at time t_0 and runs during time interval $(t_0, t_f]$) can be defined as:

$$E = \sum_{t=t_0+1}^{t_f} E(t) = \sum_{t=t_0+1}^{t_f} \sum_{k \in S} e_k^2(t)$$

Recurrent Neural Networks - Loss Minimization

Optimization problem

$$\min_{w} E = \sum_{t=t_0+1}^{t_f} E(t) = \sum_{t=t_0+1}^{t_f} \sum_{k \in S} e_k^2(t)$$

Equivalent Optimization problem

$$\min_{w} E = \sum_{t=t_0+1}^{t_f} \sum_{k \in \Theta(t)} (y_k(t) - a_k(t))^2$$

Recall:

- $a_k(t) = \phi[z_k(t)]$
- $z_k(t) = \sum_{\ell \in S} w_{k\ell}\xi_\ell(t) \ \ \forall k \in S$

Recurrent Neural Networks - Parameter Update

Gradient Descent Type Update

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Transformers

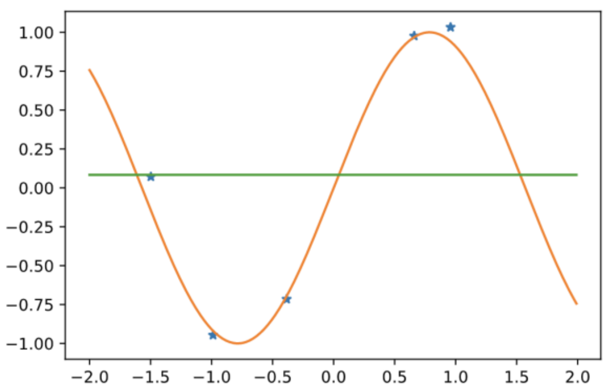
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m}\sum_{i=1}^n y^i$
- Improved estimator: $y = \sum_{i=1}^n \alpha(x^i, x) y^i$

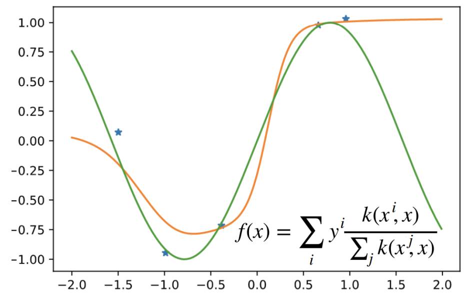


Source: http://alex.smola.org/talks/ICML19-attention.pdf

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m}\sum_{i=1}^n y^i$
- Improved estimator: $y = \sum_{i=1}^n \alpha(x^i, x) y^i$
- Weights $\alpha(x^i, x)$ can be considered to be attention score of i-th sample.

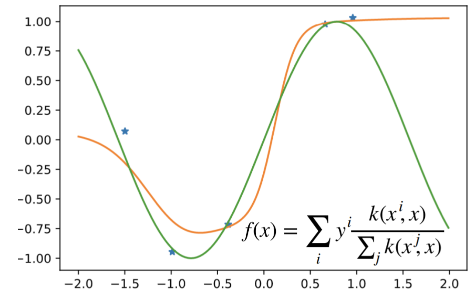


Source: http://alex.smola.org/talks/ICML19-attention.pdf

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m}\sum_{i=1}^n y^i$
- Improved estimator: $y = \sum_{i=1}^n \alpha(x^i, x) y^i$
- Weights $\alpha(x^i, x)$ can be considered to be attention score of i-th sample.
- Popular example: $\alpha(x^i, x) = k(x^i, x)/\sum_{j=1}^n k(x^j, x)$ where $k(x^i, x)$ is some kernel function based score for i-th sample.



Transformers - Idea from history

Watson-Nadaraya Estimator

- In the expression $y = \sum_{i=1}^n \alpha(x^i, x) y^i$,
  - x can be called **query**
  - x^i can be called **key**
  - y^i can be called **value**

Transformers - Architecture

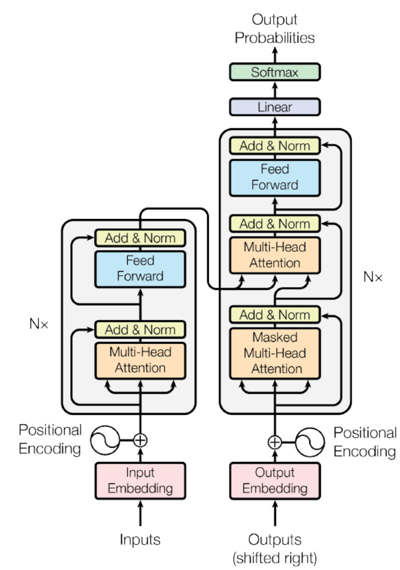
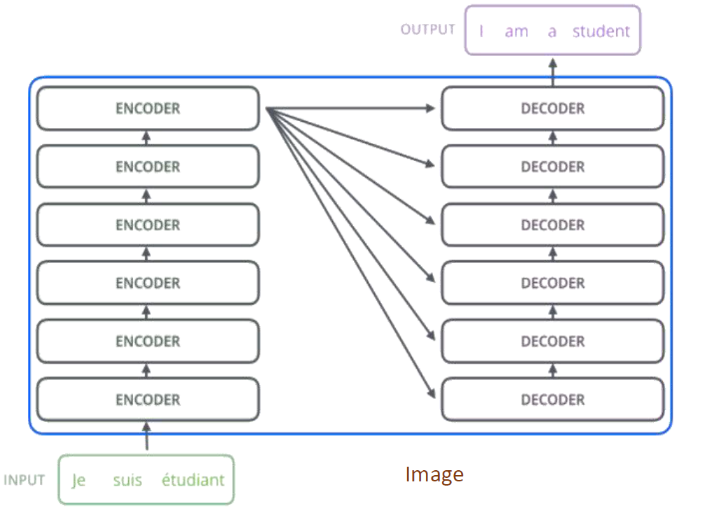


Figure : The Transformer - model architecture.

Transformers - Blocks in Architecture



Image

Image Source: [The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)]

Transformers

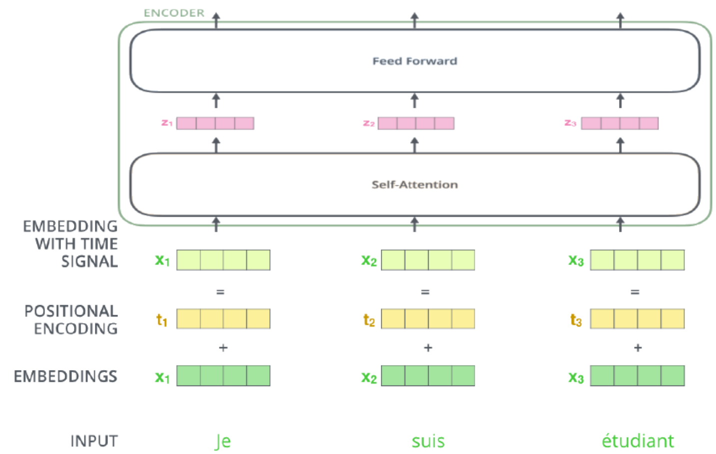
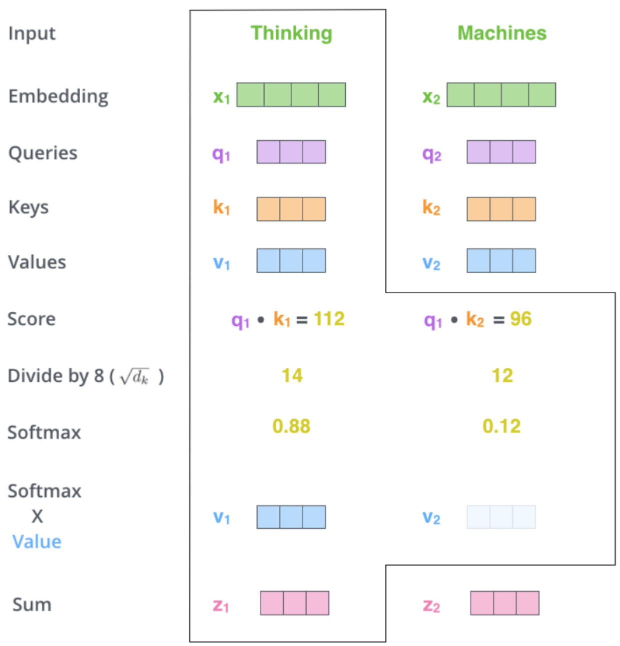


Image Source: [The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)]

Transformers: Self-attention



Transformers: Self-attention

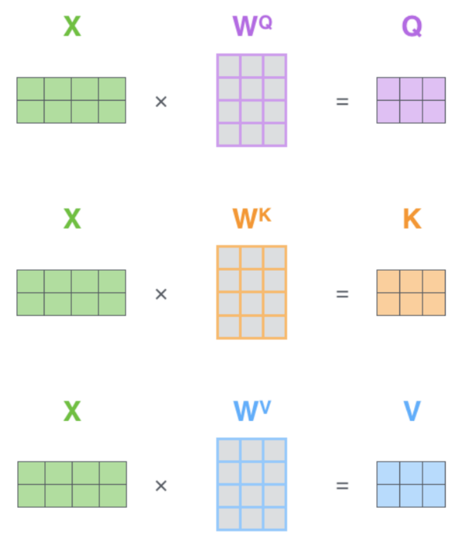


Image Source: [The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)]

Transformers: Self-attention

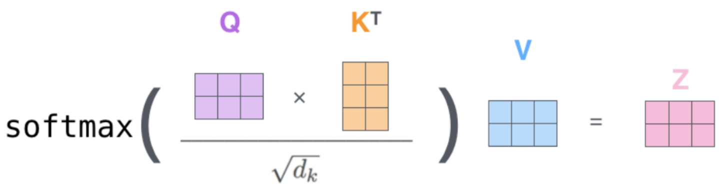
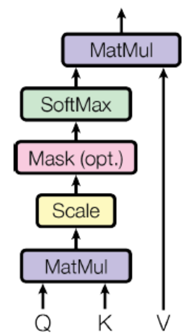


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Self-attention

Scaled Dot-Product Attention



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

Transformers: Multi-head attention

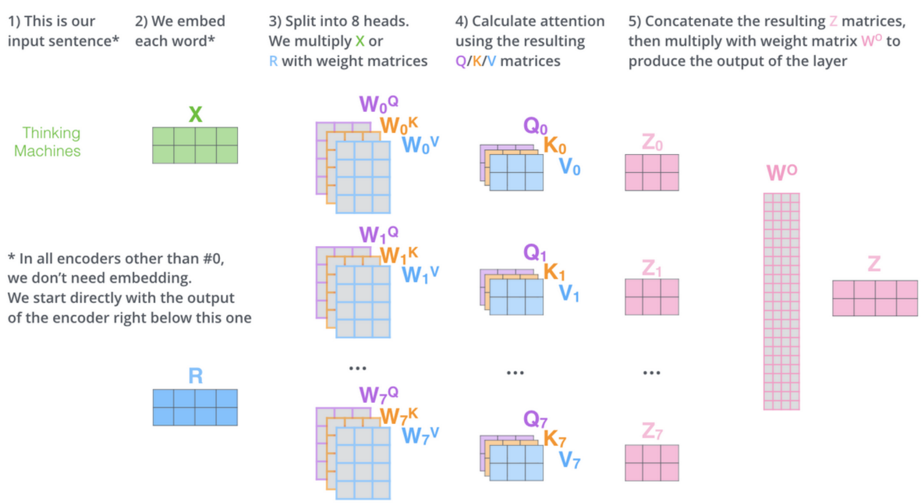
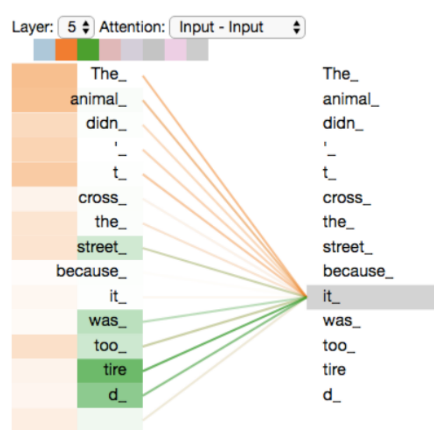
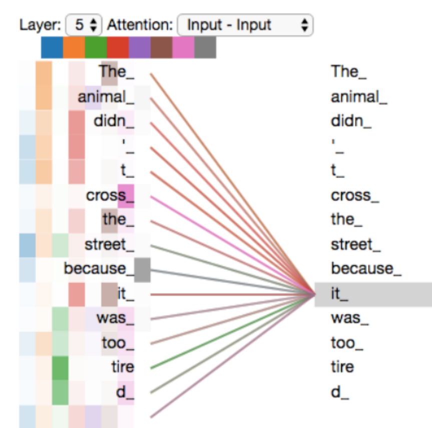


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Multi-head self-attention Example





https://jalammar.github.io/illustrated-transformer/

Transformers: Full Encoder Block

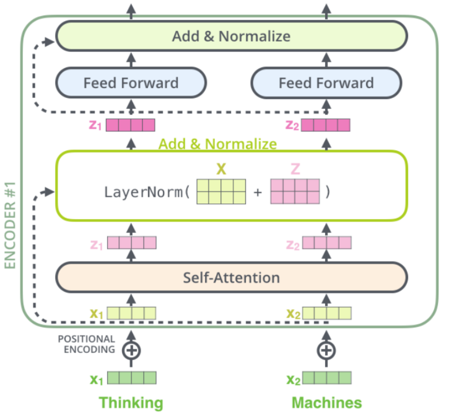


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Layer Normalization:

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

J. L. Ba, J. R. Kiros and G. E. Hinton. Layer normalization.
https://arxiv.org/pdf/1607.06450.pdf

Transformers: Encoder-Decoder Connections

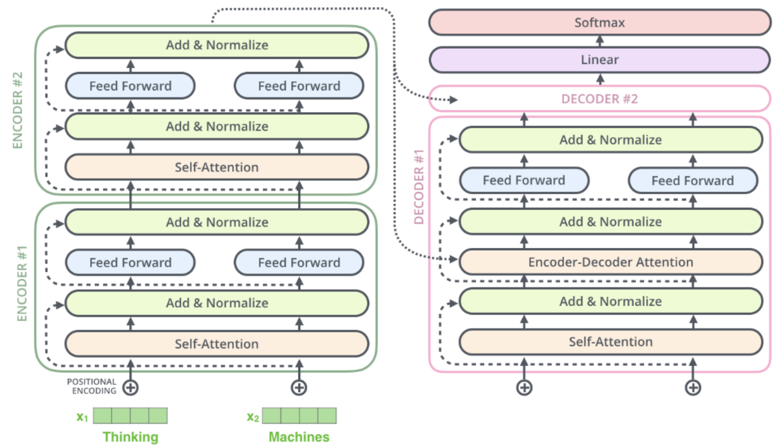


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Decoding

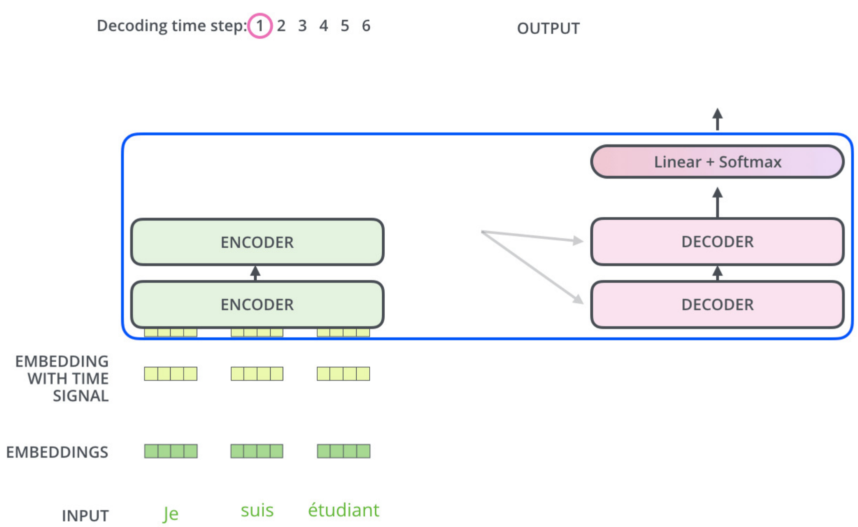


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

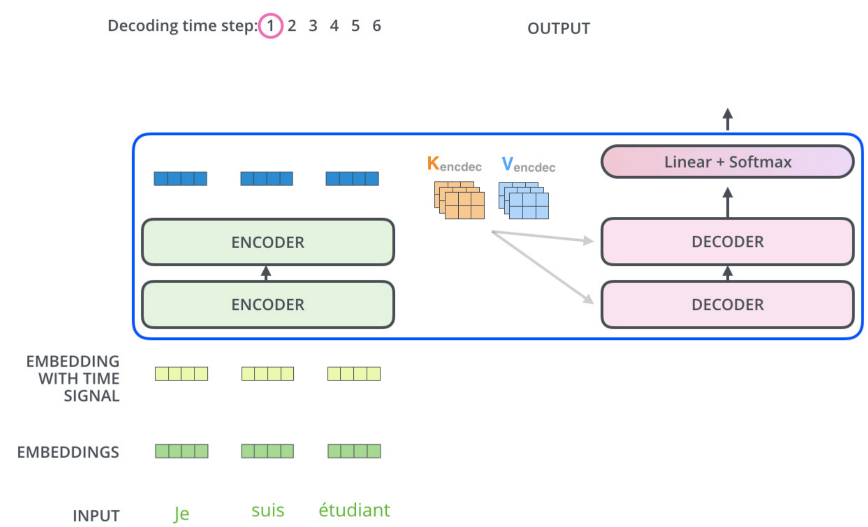


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

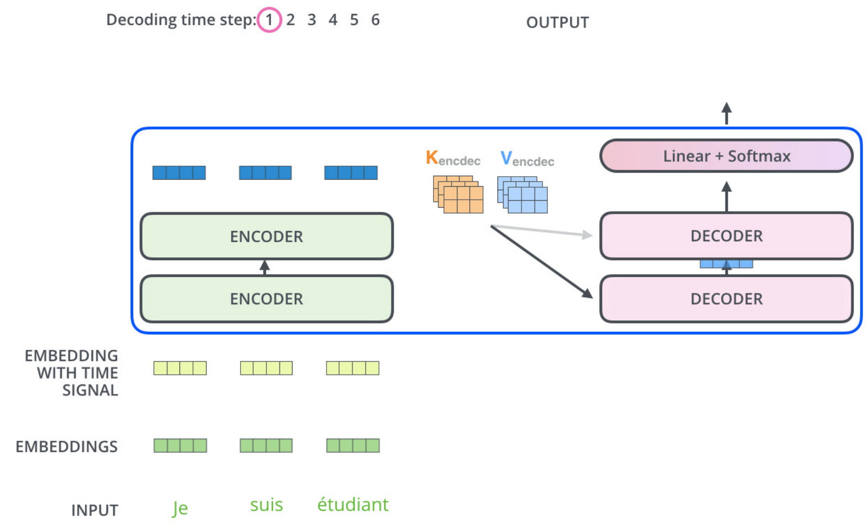


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

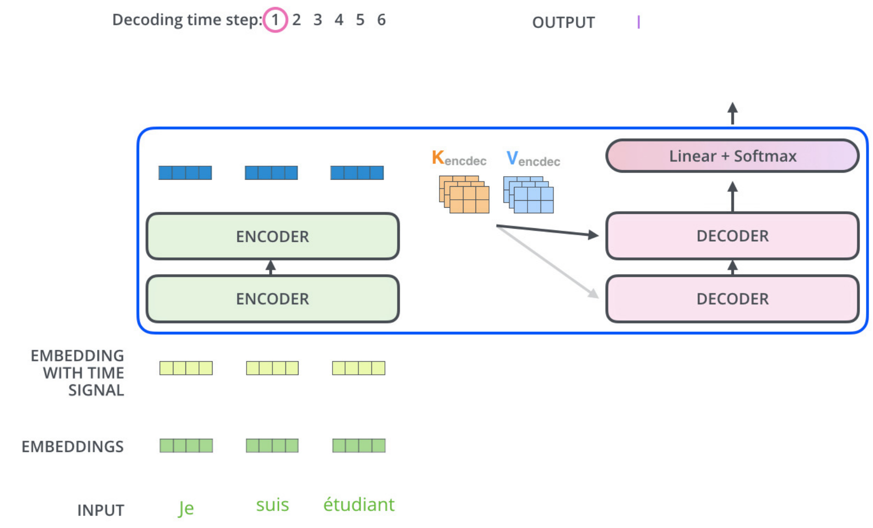


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

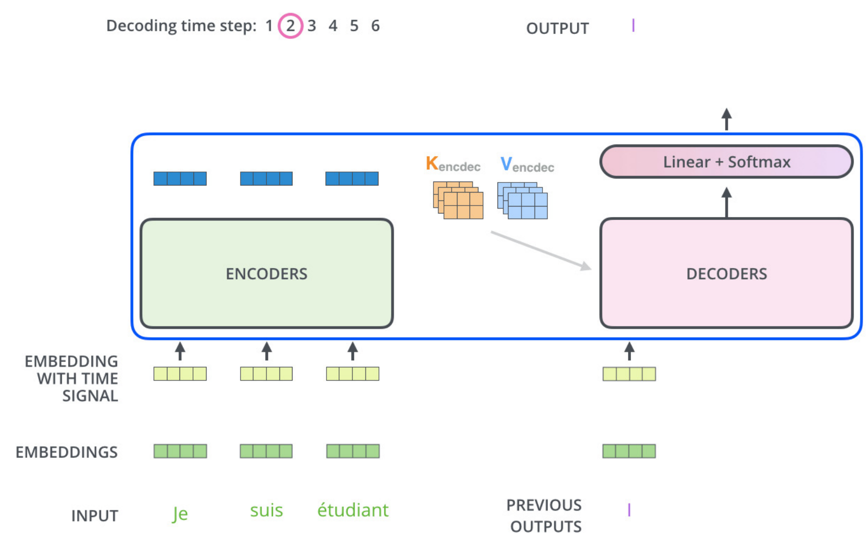


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

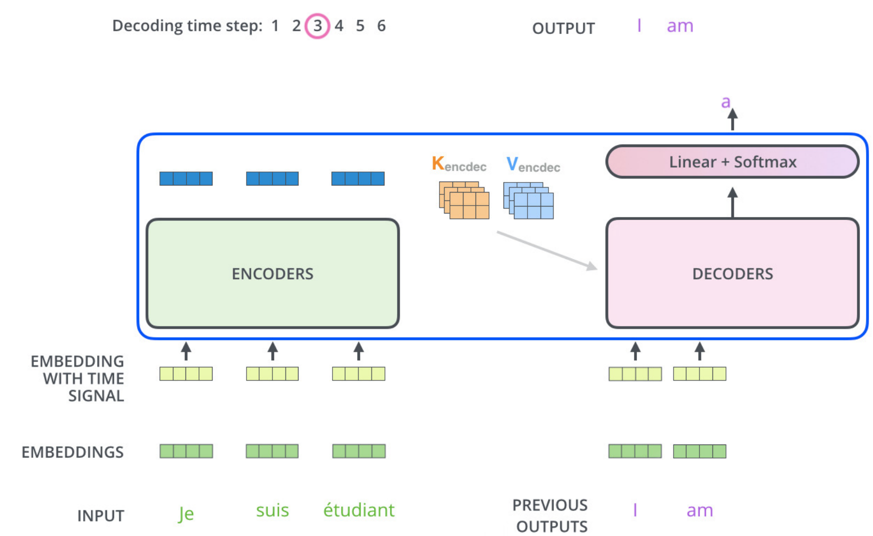


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

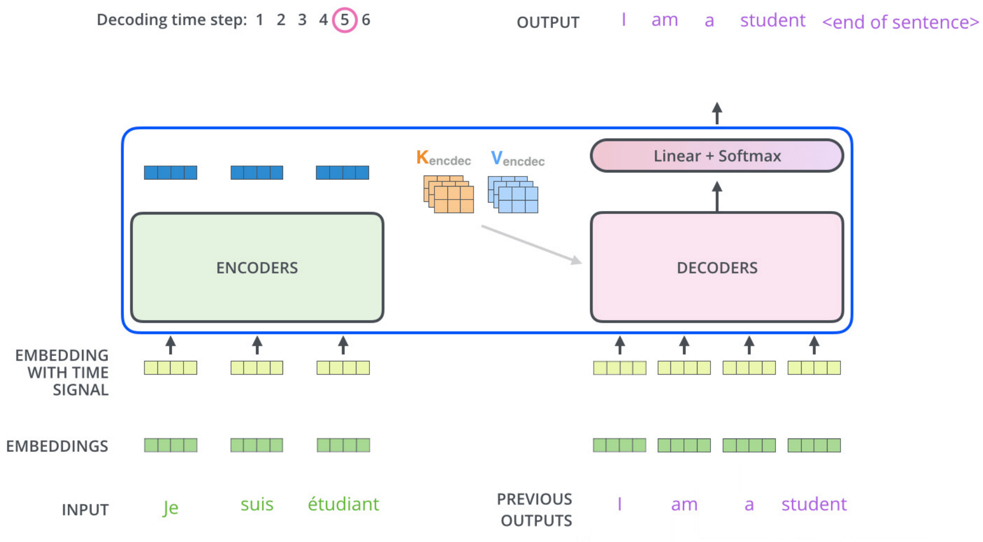


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Results on translation

Table : The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformers - Positional Encoding

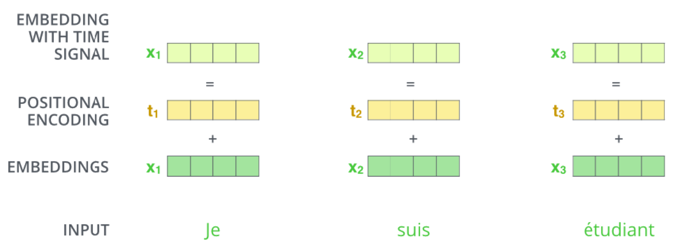


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Exercise: Check the positional encoding scheme used in the paper and understand why the scheme was chosen.

Visual Transformer

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy*,†, Lucas Beyer*, Alexander Kolesnikov*, Dirk Weissenborn*, Xiaohua Zhai*, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby*,†

*equal technical contribution, †equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

Visual Transformer

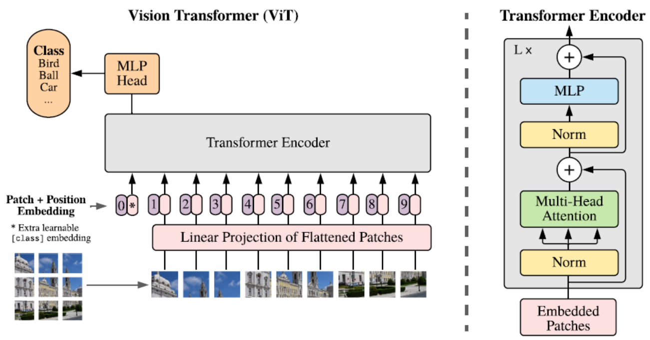


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Visual Transformer

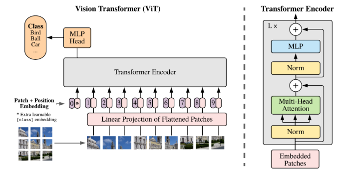


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches
2. Flatten the patches
3. Perform linear embedding of the flattened patches
4. Feed the sequence of linear embeddings (optionally added with positional encodings) to a transformer encoder
5. Pre-train the model with image labels (large data set with full supervision)
6. Fine-tune on a downstream classification task on a smaller data set.

Visual Transformer

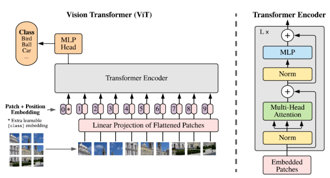


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Input: 2D image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ where (H, W) is the resolution of image in terms of height and width, and C denotes the number of channels.
- Input: 2D patch size P. Hence each patch is square of shape (P, P).
- Output: N patches denoted x_p, where j-th patch $x_p^j \in \mathbb{R}^{P \times P \times C}$.

Visual Transformer

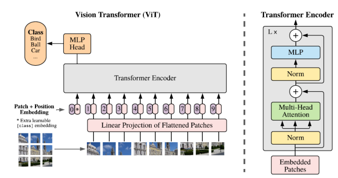


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Input: 2D image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ where (H, W) is the resolution of image in terms of height and width, and C denotes the number of channels.
- Input: 2D patch size P. Hence each patch is square of shape (P, P).
- Output: N patches denoted x_p, where j-th patch $x_p^j \in \mathbb{R}^{P \times P \times C}$.
- (Note: $N = HW/P^2$, hence we assume that both H and W are divisible by P)

Visual Transformer

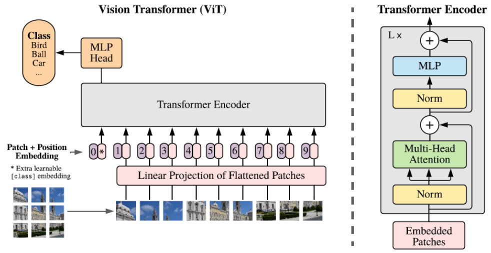


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- A learnable embedding z_0^0 is attached to sequence of embeddings at 0-th encoder block.
- The state z_L^0 at L-th block serves as representation of class label y.
- During pre-training and fine-tuning, a classification head is attached to z_L^0.
- Classification head is a MLP with single hidden layer at pre-training and with no hidden layer at fine-tuning.

Visual Transformer

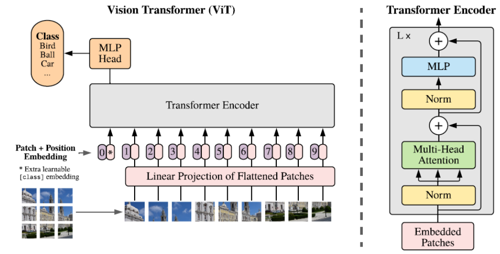


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- 1D positional encoding used.
- 2D positional encoding did not give much advantage.

Visual Transformer

$$
\begin{aligned}
\mathbf{z}_0 &= [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E}; \cdots ;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, && \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D}\\
\mathbf{z}'_\ell &= \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, && \ell = 1\ldots L\\
\mathbf{z}_\ell &= \mathrm{MLP}(\mathrm{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, && \ell = 1\ldots L\\
\mathbf{y} &= \mathrm{LN}(\mathbf{z}_L^0)
\end{aligned}
$$

Layer Normalization:

$$
\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}
$$

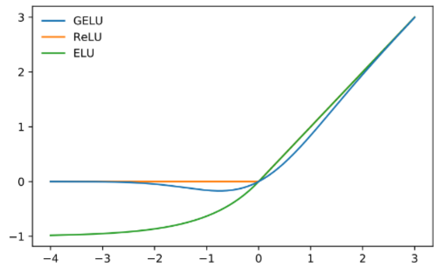
J. L. Ba, J. R. Kiros and G. E. Hinton. Layer normalization. https://arxiv.org/pdf/1607.06450.pdf

Visual Transformer

$$
\begin{aligned}
\mathbf{z}_0 &= [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E}; \cdots ;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, && \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D}\\
\mathbf{z}'_\ell &= \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, && \ell = 1\ldots L\\
\mathbf{z}_\ell &= \mathrm{MLP}(\mathrm{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, && \ell = 1\ldots L\\
\mathbf{y} &= \mathrm{LN}(\mathbf{z}_L^0)
\end{aligned}
$$

- MLP uses two layers with GeLU non-linearity.
- $GeLU(x) = xP(X \le x)$ for a random variable X. Typically $X \sim \mathcal{N}(0, 1)$.

Visual Transformer



- $ReLU(x) = \max\{0, x\}$.
- $ELU(x; \alpha) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases}$
- $GeLU(x) = x\Phi(x)$ where $\Phi(x)$ is the cdf associated with $\mathcal{N}(0, 1)$.

Visual Transformer

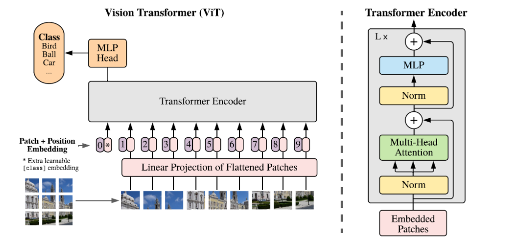


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Inductive bias in convolutions due to local two-dimensional neighborhood structure, and translation equivariance **not available** in visual transformer.
- MLP might be useful to capture local information.
- Self-attention is generally global.
- Positional encodings do not capture 2D positional information, hence spatial relations should be learned from scratch.

Visual Transformer

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

Visual Transformer

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** $\pm$ 0.04	87.76 $\pm$ 0.03	85.30 $\pm$ 0.02	87.54 $\pm$ 0.02	88.4/88.5*
ImageNet ReaL	**90.72** $\pm$ 0.05	90.54 $\pm$ 0.03	88.62 $\pm$ 0.05	90.54	90.55
CIFAR-10	**99.50** $\pm$ 0.06	99.42 $\pm$ 0.03	99.15 $\pm$ 0.03	99.37 $\pm$ 0.06	–
CIFAR-100	**94.55** $\pm$ 0.04	93.90 $\pm$ 0.05	93.25 $\pm$ 0.05	93.51 $\pm$ 0.08	–
Oxford-IIIT Pets	**97.56** $\pm$ 0.03	97.32 $\pm$ 0.11	94.67 $\pm$ 0.15	96.62 $\pm$ 0.23	–
Oxford Flowers-102	99.68 $\pm$ 0.02	**99.74** $\pm$ 0.00	99.61 $\pm$ 0.02	99.63 $\pm$ 0.03	–
VTAB (19 tasks)	**77.63** $\pm$ 0.23	76.28 $\pm$ 0.46	72.72 $\pm$ 0.21	76.29 $\pm$ 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. *Slightly improved 88.5% result reported in Touvron et al. (2020).

Note: JFT data set is an inhouse data set of Google with more than billion images.

Visual Transformer

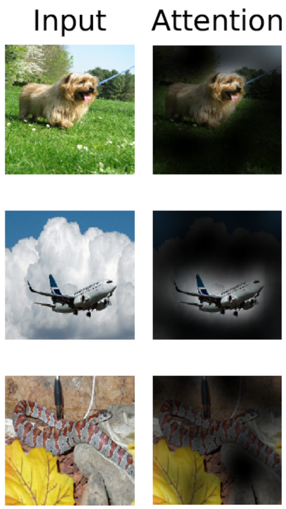
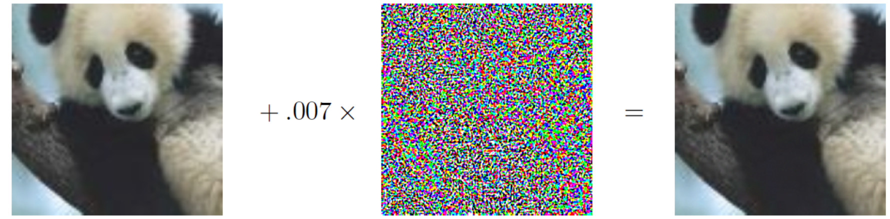


Figure : Representative examples of attention from the output token to the input space.

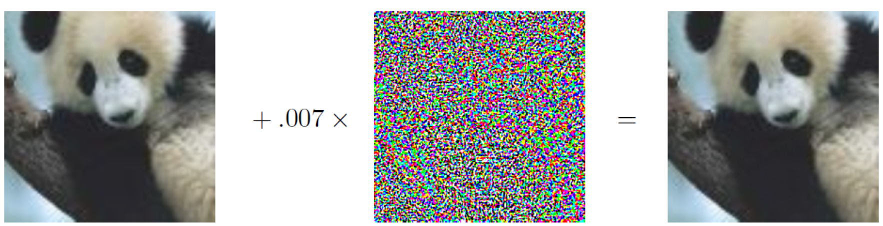
Adversarial Training



Note:

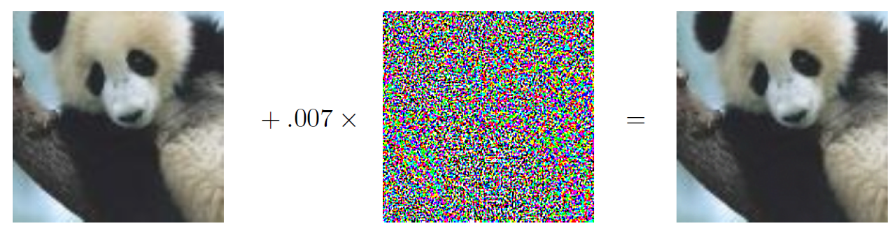
- $J(\boldsymbol{\theta}, \mathbf{x}, y)$ is the loss function.
- $\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)$ is the gradient of loss function with respect to the variations in input $\mathbf{x}$.

Adversarial Training



- Generating adversarial images: By using a perturbation $\mathbf{x} + \boldsymbol{\eta}$ where $\boldsymbol{\eta} = \epsilon\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$.

Adversarial Training



- Generating adversarial images: By using a perturbation $\mathbf{x} + \boldsymbol{\eta}$ where $\boldsymbol{\eta} = \epsilon\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$.
- This is called fast gradient sign method of generating adversarial examples.

Adversarial Training

- Adversarial training done using original images and adversarial images using an adversarial objective function:

$$\bar{J}(\boldsymbol{\theta}, \mathbf{x}, y) = \alpha J(\boldsymbol{\theta}, \mathbf{x}, y) + (1-\alpha) J(\boldsymbol{\theta}, \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)), y).$$

- $\alpha = 0.5$ was used in experiments.
- Adversarial training helped in better regularization.
- Adversarial training did not reach zero error rate on adversarial images.
- Larger models with more units in hidden layers, early stopping on adversarial validation set error, helped to decrease error rate from 89.4% on adversarial examples to 17.9%.

Adversarial Training

Other Experiments:

- Control experiments were performed by adding random uniform noise $\pm\epsilon$ to each pixel of an image to generate adversarial samples.
- This model achieved 86.2% error rate on the adversarial examples created by adding random uniform noise.
- But the model achieved 90.4% error rate on adversarial examples created by fast gradient sign method.

Adversarial Training

Other Experiments:

- Control experiments were performed by generating adversarial examples using small rotations or by addition of scaled gradient.
- Such experiments lead to smooth objective functions.
- However training on such adversarial examples was found to be ineffective.

Adversarial Training

Other Experiments:

- Experiments were performed by perturbing the activations of hidden layers (but not the output layer).
- When hidden layers have sigmoid activations, perturbing hidden layer activations helped to improve regularization.
- When hidden layer activations have ReLU type activation functions, the perturbations did not give much improvements.
- Exercise: Check why last layer was not considered for perturbations!

Adversarial Training

Observations:

- An adversarial example generated for one model is misclassified by other models too.

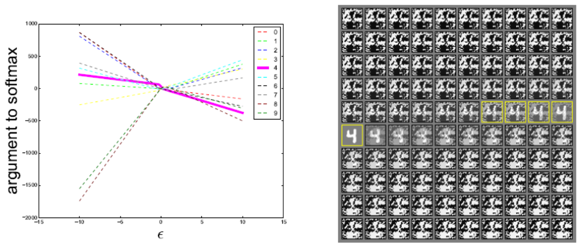


Figure 4: By tracing out different values of ϵ, we can see that adversarial examples occur reliably for almost any sufficiently large value of ϵ provided that we move in the correct direction. Correct classifications occur only on a thin manifold where $\boldsymbol{x}$ occurs in the data. Most of $\mathbb{R}^n$ consists of adversarial examples and *rubbish class examples* (see the appendix). This plot was made from a naively trained maxout network. Left) A plot showing the argument to the softmax layer for each of the 10 MNIST classes as we vary ϵ on a single input example. The correct class is 4. We see that the unnormalized log probabilities for each class are conspicuously piecewise linear with ϵ and that the wrong classifications are stable across a wide region of ϵ values. Moreover, the predictions become very extreme as we increase ϵ enough to move into the regime of rubbish inputs. Right) The inputs used to generate the curve (upper left = negative ϵ, lower right = positive ϵ, yellow boxes indicate correctly classified inputs).

Adversarial Training

Observations:

- When other models misclassify an adversarial example, the class labels given by different models are almost the same.
- One hypothesis to validate this observation: most neural networks try to approximate to a reference linear model on the training set.

Adversarial Training

Optimization problem:

- Adversarial training can be posed as the following training problem:

$$\min_{\theta} \sum_{i=1}^{N} \max_{\delta \in \Delta} \ell(f_\theta(x^i + \delta), y^i)$$

where $\Delta = \{\vartheta : \|\vartheta\|_\infty \leq \epsilon\}$ for some fixed $\epsilon > 0$.

Note: FGSM can be seen to consider an inaccurate approximate solution to the inner problem as $\delta^* = \epsilon \text{sign}(\nabla_x \ell(f(x), y))$.

Adversarial Training

PGD Method:

Algorithm 1 PGD adversarial training for T epochs, given some radius ϵ, adversarial step size α and N PGD steps and a dataset of size M for a network f_θ

```
for t = 1...T do
  for i = 1...M do
    // Perform PGD adversarial attack
    δ = 0 // or randomly initialized
    for j = 1...N do
      δ = δ + α · sign(∇_δ ℓ(f_θ(x_i + δ), y_i))
      δ = max(min(δ, ε), −ε)
    end for
    θ = θ − ∇_θ ℓ(f_θ(x_i + δ), y_i) // Update model weights with some optimizer, e.g. SGD
  end for
end for
```

Adversarial Training

Optimization problem:

- Adversarial training can be posed as the following training problem:

$$\min_{\theta} \sum_{i=1}^{N} \max_{\delta \in \Delta} \ell(f_\theta(x^i + \delta), y^i)$$

where $\Delta = \{\vartheta : \|\vartheta\|_\infty \leq \epsilon\}$ for some fixed $\epsilon > 0$.

- Landscape of local maxima of the inner maximization problem was investigated to check the behavior.
- Done by restarting PGD from different points in the ℓ_∞ balls around the data points.
- Existence of multiple maxima; in each case, the loss plateaus quickly.
- Exercise: How to check for distinct maxima?

Adversarial Training

Loss values for different restarts:

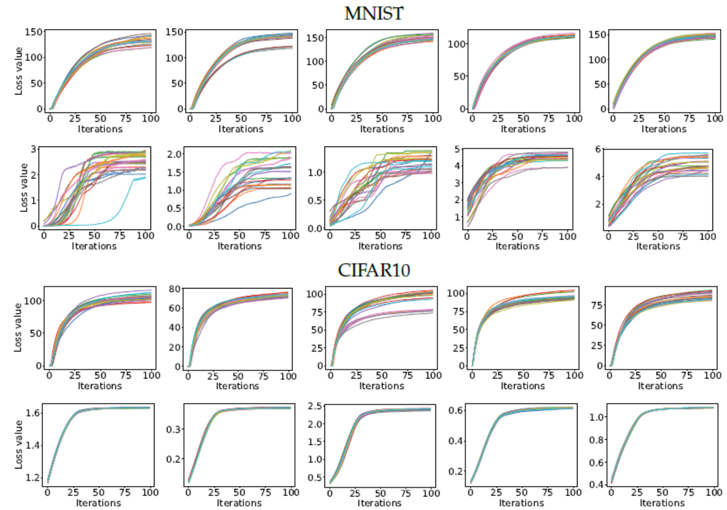


Figure : Loss function value over PGD iterations for 20 random restarts on random examples. The 1st and 3rd rows correspond to standard networks, while the 2nd and 4th to adversarially trained ones.

Adversarial Training

Loss value concentration over multiple restarts:

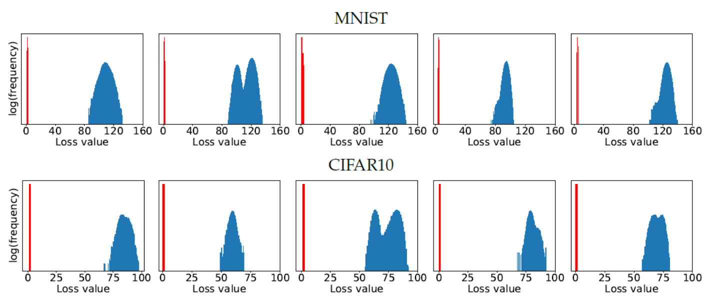
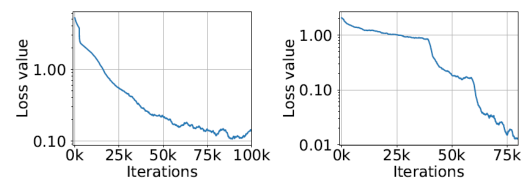


Figure : Values of the local maxima given by the cross-entropy loss for five examples from the MNIST and CIFAR10 evaluation datasets. For each example, we start projected gradient descent (PGD) from 10^5 uniformly random points in the ℓ_∞-ball around the example and iterate PGD until the loss plateaus. The blue histogram corresponds to the loss on a standard network, while the red histogram corresponds to the adversarially trained counterpart. The loss is significantly smaller for the adversarially trained networks, and the final loss values are very concentrated without any outliers.

The average loss of final iterate over 10^5 restarts follows a concentrated distribution without extreme outliers.

Adversarial Training

Loss behavior on adversarial examples:



(a) MNIST (b) CIFAR10

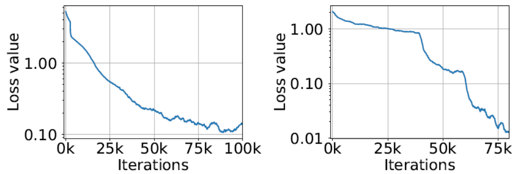
Figure : Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (2.1), thus producing an increasingly robust classifier.

MNIST:

- Training was done on CNN with two conv layers with 32 and 64 filters respectively, followed by 2×2 max-pooling and fully connected layer of size 1024.
- $\epsilon = 0.3$
- 40 iterations of PGD with step size 0.01.

Adversarial Training

Loss behavior on adversarial examples:



(a) MNIST (b) CIFAR10

Figure : Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (2.1), thus producing an increasingly robust classifier.

CIFAR:

- Training was done on Resnet.
- $\epsilon = 8$
- 40 iterations of PGD with step size 0.02.

Adversarial Training

Types of adversarial attakcs:

- Black box attacks: when the adversary has no knowledge of the architecture used for training.
- White box attacks: when the adversary has complete knowledge of the architecture used for training.

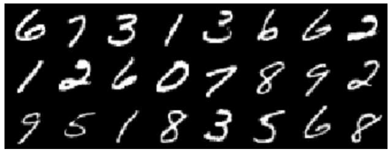
Adversarial Training

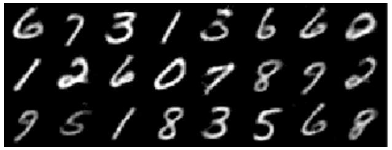
Types of adversarial examples:

- Non-targeted: add perturbation to an image so that the network predicts a different class label other than the original class label.
- Targeted: add perturbation to an image so that the network predicts a particular class label of the adversary's choice different from the original class label.

Introduction to the problem setup

What do we mean by modeling the unknown distribution $P(X)$?





Introduction to the problem setup

Problem setup:

- Input: Data set $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$ denotes the i-th data sample or data point.
- $\mathcal{X}$ is an appropriate input space.
- Examples of $\mathcal{X}$:
  - Set of images (*e.g.* digits, faces, animals, etc.) or videos.
  - Set of vector-valued data or matrix-valued data or tensor-valued data.
  - Set of natural language sentences.
  - Set of documents (*e.g* newspaper articles, books).
  - Set of software programs.

Introduction to the problem setup

What do we mean by modeling the unknown distribution $P(X)$?

- For realizations x of X which are similar to the data samples in $D = \{x^i\}_{i=1}^N$, we want $P_X(x)$ to take higher values (*e.g.* $P_X(x) \approx 1$).
- For realizations x of X which are **not** similar to the data samples in $D = \{x^i\}_{i=1}^N$, we want $P_X(x)$ to take smaller values (*e.g.* $P_X(x) \approx 0$).

Introduction to the problem setup

Problem setup:

- Input: Data set $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$ denotes the i-th data sample or data point.
- Assumption: The data samples are generated from some unknown probability distribution denoted by $P(X)$ (also denoted by $P_X(x)$).
  - Note: X is a random variable and x is a realization of X.
- Aim: To model the unknown distribution $P(X)$ using the observed data samples $D = \{x^i\}_{i=1}^N$.

Introduction to the problem setup

Several ways to model $P(X)$:

- Density estimation techniques
  - Kernel density estimation
  - Spectral density estimation
  - Non-parametric density estimation
- Histogram fitting
- Generative models

Generative models

Recall:

- Input: Data set $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$ denotes the i-th data sample or data point.
- Assumption: The data samples are generated from some unknown probability distribution denoted by $P_X(x)$.

Generative models

Recall:

- Input: Data set $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$ denotes the i-th data sample or data point.
- Assumption: The data samples are generated from some unknown probability distribution denoted by $P_X(x)$.

Generative Model:

- A machine learning model of the unknown $P_X(x)$, given the data $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$.

Generative models

- Input: Data set $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$ denotes the i-th data sample or data point.
- Modeling $P_X(x)$ is usually a complicated task.
  - Suppose that $x \in \mathbb{R}^d$ or $x \in \mathbb{R}^{m \times n}$ (that is, x is vector-valued or matrix-valued).

Generative models

- Marginal distribution:

$$P_X(x) = \int P_{(X,Z)}(x,z)dz$$
$$= \int P_{X|Z}(x|z)P_Z(z)dz$$

Requirements:

- A suitable choice for **prior** distribution $P_Z(z)$.
- How to model the **conditional** distribution $P_{X|Z}(x|z)$?

Catch:

- Computing the integral is generally **intractable**.
- Need to use computationally intensive Markov-Chain Monte Carlo techniques to estimate the integral.

Generative models

- Input: Data set $D = \{x^i\}_{i=1}^N$, where $x^i \in \mathcal{X}$ denotes the i-th data sample or data point.
- Modeling $P_X(x)$ is usually a complicated task.
  - Suppose that $x \in \mathbb{R}^d$ or $x \in \mathbb{R}^{m \times n}$ (that is, x is vector-valued or matrix-valued).
  - Correlations between various components of x might be difficult to model.
- To model $P_X(x)$ we use a trick:
  - Marginal distribution:

$$P_X(x) = \int P_{(X,Z)}(x,z)dz$$
$$= \int P_{X|Z}(x|z)P_Z(z)dz \text{ (By definition of conditional probability)}$$

Generative models

A different approach:

- True posterior:

$$P_{Z|X}(z|x) = \frac{P_{X|Z}(x|z)P_Z(z)}{P_X(x)} \text{ (By Bayes' Theorem \& law of total proabability)}$$

can be used to model $P_X(x)$.

Generative models

- Marginal distribution:

$$P_X(x) = \int P_{(X,Z)}(x,z)dz$$
$$= \int P_{X|Z}(x|z)P_Z(z)dz$$

- **Why do we need this marginal distribution?**
  - We introduce a new random variable Z.
  - Z is called a latent variable.
  - Z is usually under our control.
  - By suitable choice of Z with a known **prior** distribution $P_Z(z)$ we can try to model $P_X(x)$ if we can effectively model the **conditional** distribution $P_{X|Z}(x|z)$.

Recap of Variational Bayes

- Thus we have:

$$KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x))$$
$$= E_{Z\sim Q}\left[\log Q_{Z|X}(z|x) - \log P_{X|Z}(x|z) - \log P_Z(z)\right] + \log P_X(x)$$

- Rearranging, we get:

$$\log P_X(x) - KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x))$$
$$= E_{Z\sim Q}\left[\log P_Z(z) - \log Q_{Z|X}(z|x) + \log P_{X|Z}(x|z)\right]$$
$$= E_{Z\sim Q}\left[-\log \frac{Q_{Z|X}(z|x)}{P_Z(z)} + \log P_{X|Z}(x|z)\right]$$
$$= E_{Z\sim Q}\left[\log P_{X|Z}(x|z)\right] - E_{Z\sim Q}\left[\log \frac{Q_{Z|X}(z|x)}{P_Z(z)}\right]$$
$$= E_{Z\sim Q}\left[\log P_{X|Z}(x|z)\right] - KL(Q_{Z|X}(z|x)||P_Z(z))$$

Recap of Variational Bayes

- True posterior:

$$P_{Z|X}(z|x) = \frac{P_{X|Z}(x|z)P_Z(z)}{P_X(x)}$$

  can be used to model $P_X(x)$.
- Computing $P_{Z|X}(z|x)$ is intractable in general.
- Use a customized distribution $Q_{Z|X}(z|x)$ (called recognition model) to approximate $P_{Z|X}(z|x)$.

Recap of Variational Bayes

Recall our idea:

- Use a customized distribution $Q_{Z|X}(z|x)$ (called recognition model) to approximate $P_{Z|X}(z|x)$.

Objective:

$$\log P_X(x) - KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x)) = E_{Z\sim Q}\left[\log P_{X|Z}(x|z)\right] - KL(Q_{Z|X}(z|x)||P_Z(z))$$

Aim: To maximize the objective.

Note:

- $\log P_X(x)$ denotes the log likelihood, which we wanted to model.
- $KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x))$ denotes the dissimilarity between the recognition distribution Q and the true posterior $P_{Z|X}(z|x)$.
- The KL term acts like a regularizer.

Recap of Variational Bayes

- Error of approximation between $P_{Z|X}$ and $Q_{Z|X}$ can be computed using the Kullback-Leibler (or KL)-Divergence:

$$KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x))$$
$$= \int \log \frac{Q_{Z|X}(z|x)}{P_{Z|X}(z|x)} Q_{Z|X}(z|x)dz$$
$$= E_{Z\sim Q}\left[\log Q_{Z|X}(z|x) - \log P_{Z|X}(z|x)\right]$$
$$= E_{Z\sim Q}\left[\log Q_{Z|X}(z|x) - \log \frac{P_{X|Z}(x|z)P_Z(z)}{P_X(x)}\right]$$
$$= E_{Z\sim Q}\left[\log Q_{Z|X}(z|x) - \log P_{X|Z}(x|z) - \log P_Z(z)\right] + \log P_X(x)$$

Recap of Variational Bayes

Recall: Our aim is to maximize objective:

$$\log P_X(x) - KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x)) = E_{Z\sim Q}\left[\log P_{X|Z}(x|z)\right] - KL(Q_{Z|X}(z|x)||P_Z(z))$$

In the presence of a dataset D, our objective would become:

$$\begin{aligned}\max\ & E_{X\sim D}\left[\log P_X(x) - KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x))\right] \\ &= E_{X\sim D}\left[E_{Z\sim Q}\left[\log P_{X|Z}(x|z)\right] - KL(Q_{Z|X}(z|x)||P_Z(z))\right]\end{aligned}$$

For a sample x^i from D, the corresponding objective term is:

$$\mathcal{L}(\theta,\phi;x^i) = E_{Z\sim Q}\left[\log P_{X|Z}(x^i|z)\right] - KL(Q_{Z|X}(z|x^i)||P_Z(z))$$

- For data set $D = \{x^i\}_{i=1}^N$ and a randomly chosen minibatch $\mathcal{B}$ of size M, we can find $\mathcal{L}(\theta,\phi;\mathcal{B}) = \frac{M}{N}\sum_{x\in\mathcal{B}}\mathcal{L}(\theta,\phi;x)$.

Recap of Variational Bayes

To maximize Objective:

$$\log P_X(x) - KL(Q_{Z|X}(z|x)||P_{Z|X}(z|x)) = E_{Z\sim Q}\left[\log P_{X|Z}(x|z)\right] - KL(Q_{Z|X}(z|x)||P_Z(z))$$

- We parametrize P using θ.
- We parametrize Q using ϕ.

Thus we get:

$$\begin{aligned}\log P_X(x;\theta) - KL(Q_{Z|X}(z|x;\phi)||P_{Z|X}(z|x;\theta)) = E_{Z\sim Q}&\left[\log P_{X|Z}(x|z;\theta)\right] \\ &- KL(Q_{Z|X}(z|x;\phi)||P_Z(z;\theta))\end{aligned}$$

VAE

- Major assumption: $P_Z(z) \approx \mathcal{N}(0,I)$.
- We assume $Q_{Z|X}(z|x^i;\phi) = \mathcal{N}(z;\mu^i,(\sigma^i)^2 I)$, where $\mathcal{N}(z;\mu^i,(\sigma^i)^2 I)$ denotes the normal distribution with mean $\mu^i = \mu(x^i)$ and covariance matrix $(\sigma^i)^2 I$ with $(\sigma^i)^2 = \sigma^2(x^i)$.
- We can adopt reparametrization trick using $G(x^i,\epsilon^l;\phi) = \mu^i + \sigma^i \odot \epsilon^l$ where $\epsilon^l \sim \mathcal{N}(0,I)$ and $\odot$ denotes the elementwise multiplication.
- Then we can write:

$$\mathcal{L}(\theta,\phi;x^i) \approx \frac{1}{2}\sum_{j=1}^{d}\left(1 + \log((\sigma_j^i)^2) - (\mu_j^i)^2 - (\sigma_j^i)^2\right) + E_{Z\sim Q}\left[\log P_{X|Z}(x^i|z)\right]$$

Recap of Variational Bayes

Coding theory perspective:

- $Q_{Z|X}(z|x;\phi)$ is called a probabilistic encoder since given a sample x, Q encodes it into a distribution.
- $P_{X|Z}(x|z;\theta)$ is called a probabilistic decoder since given a latent variable z, P produces a distribution over corresponding values of x.
- Hence the methodology is called auto-encoding variational Bayes (AEVB).

Generative models - Variational Bayes Approach

Requirement:

- Given an approximate posterior $Q_{Z|X}(z|x)$, we need to sample $Z \sim Q$.

Caveat:

- The approximate posterior $Q_{Z|X}(z|x)$ might not be differentiable.

Reparametrization trick for sampling Z

- Assume a differentiable $G(\epsilon, x; \phi)$ and sample $Z \sim G$.
- **Note:** We have now introduced a new variable ϵ.
- **Assumption:** $\epsilon \sim p(\epsilon)$.

VAE

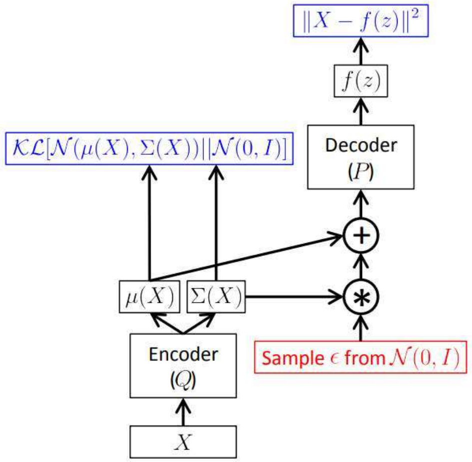
- Hence the overall loss becomes:

$$\mathcal{L}(\theta, \phi; x^i) \approx \frac{1}{2}\sum_{i=1}^{d}\left(1 + \log((\sigma_j^i)^2) - (\mu_j^i)^2 - (\sigma_j^i)^2\right) + \frac{1}{L}\sum_{l=1}^{L}\left(\log P_{X|Z}(x^i|z^{i,l})\right)$$

where $z^{i,l} \sim G(\epsilon^{i,l}, x^i; \phi)$ and $\epsilon^{i,l} \sim p(\epsilon)$.

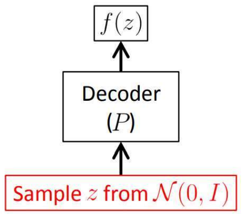
VAE

Training a VAE:



VAE

Testing VAE:



- Remove the encoder
- Sample $z \sim \mathcal{N}(0, I)$.
- Generate sample using decoder.