SE125 Machine Learning

Convolutional Neural Networks Part II

Yue Ding

School of Software, Shanghai Jiao Tong University
dingyue@sjtu.edu.cn

Convolutional Neural Networks

- 课程难度:

- 掌握程度:

References and Acknowledgement

- Stanford CS231n: Convolutional Neural Networks for Visual Recognition
- PIERIAN DATA, Convolutional Neural Networks
- Deep Learning Turtorial, Hung-yi Lee
- www.towardsdatascience.com

Representation Learning

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Original

$$\begin{bmatrix} & -1 & \\ -1 & & -1 \\ & -1 & \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$$

Blur

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

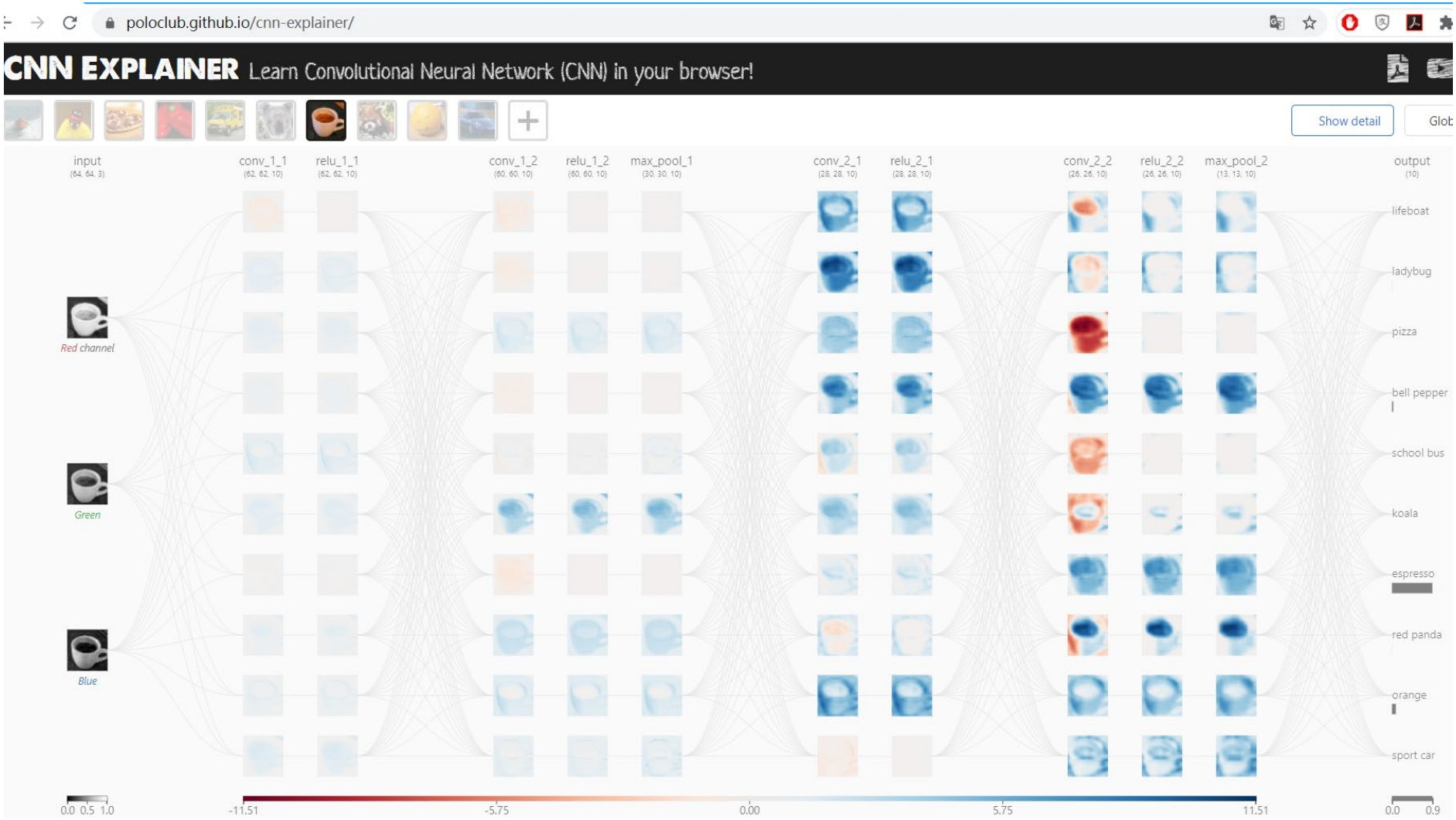
Edge *extraction*

Representation Learning

- The recognition process of image data is generally considered to be the process of Representation Learning.
- Starting from the received original pixels, the low-level features such as edges and corners are gradually extracted, then to the middle-level features such as texture, and then to the head, object parts, etc.
- High-level features, and the final network layer learns classification logic based on these learned abstract feature representations.

  - Matthew D. Zeiler, Rob Fergus: Visualizing and Understanding Convolutional Networks. ECCV (1) 2014: 818-833
  - https://poloclub.github.io/cnn-explainer/

CNN Explainer



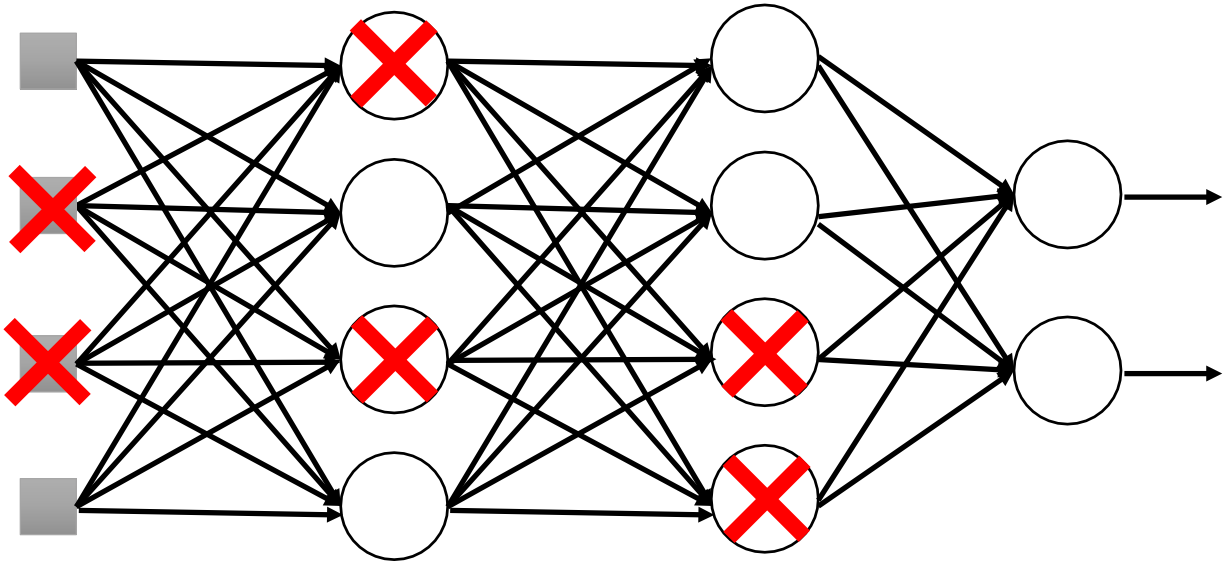
Dropout

- ***When a large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data. This "overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case.***

- Improving neural networks by preventing co-adaptation of feature detectors. *Hinton, Geoffrey E， Srivastava, Nitish， Krizhevsky, Alex， Sutskever, Ilya， Salakhutdinov, Ruslan R.* http://dx.doi.org/10.9774/GLEAF.978-1-909493-38-4_2, 2012

Dropout

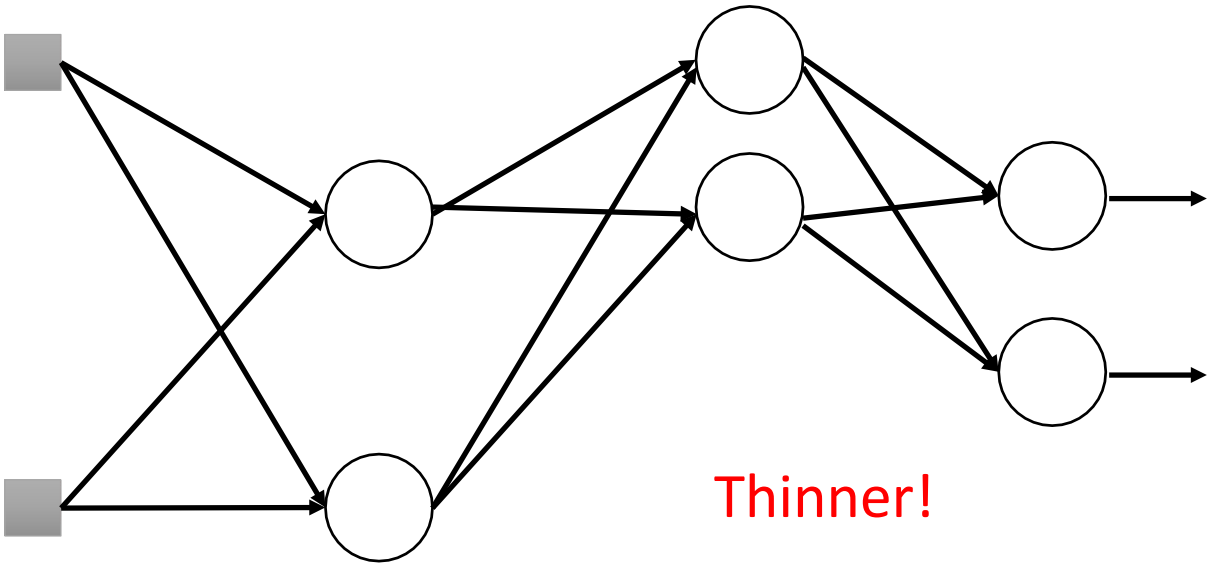
Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

Training:



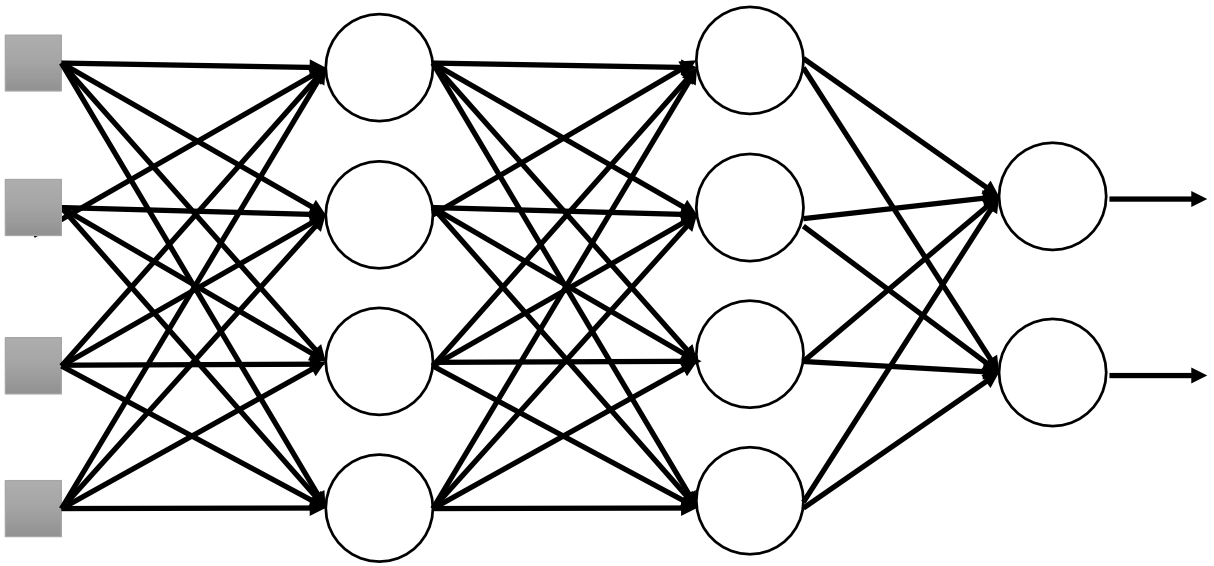
- **Each time before updating the parameters**
  - Each neuron has p% to dropout

    The structure of the network is changed.
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

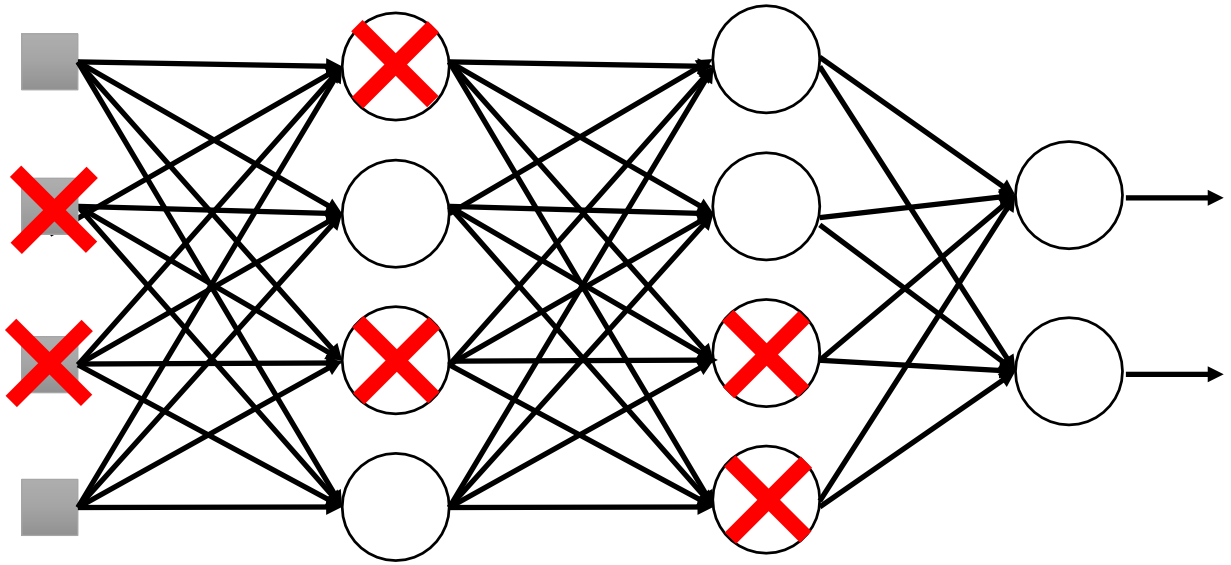
Dropout

Testing:



- **No dropout**
  - If the dropout rate at training is p%, all the weights times (1-p)%
  - Assume that the dropout rate is 50%. If a weight $w = 1$ by training, set $ww = 0.5$ for testing.

Dropout - Intuitive Reason



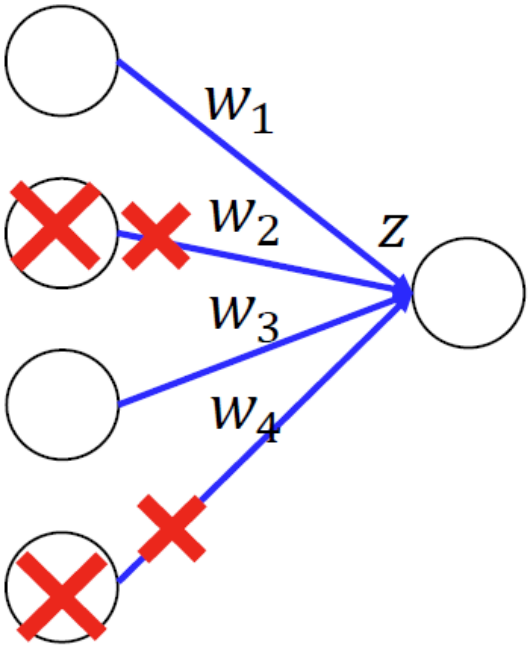
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

- Why the weights should multiply (1-p)% (dropout rate) when testing?

Training of Dropout

Assume dropout rate is 50%

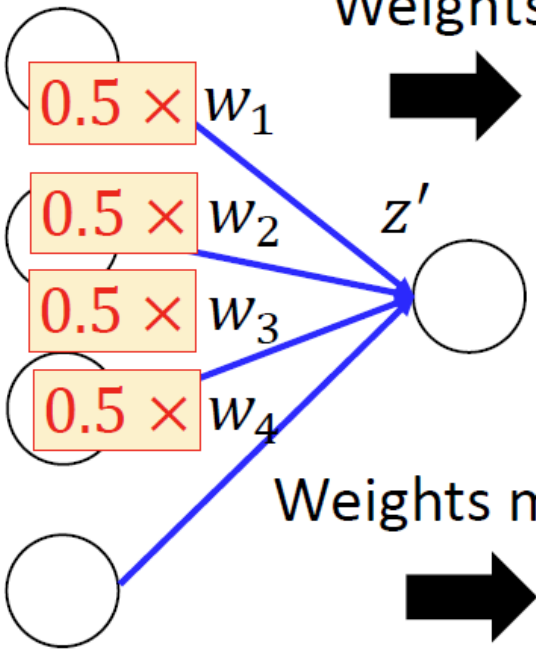


Testing of Dropout

No dropout

Weights from training

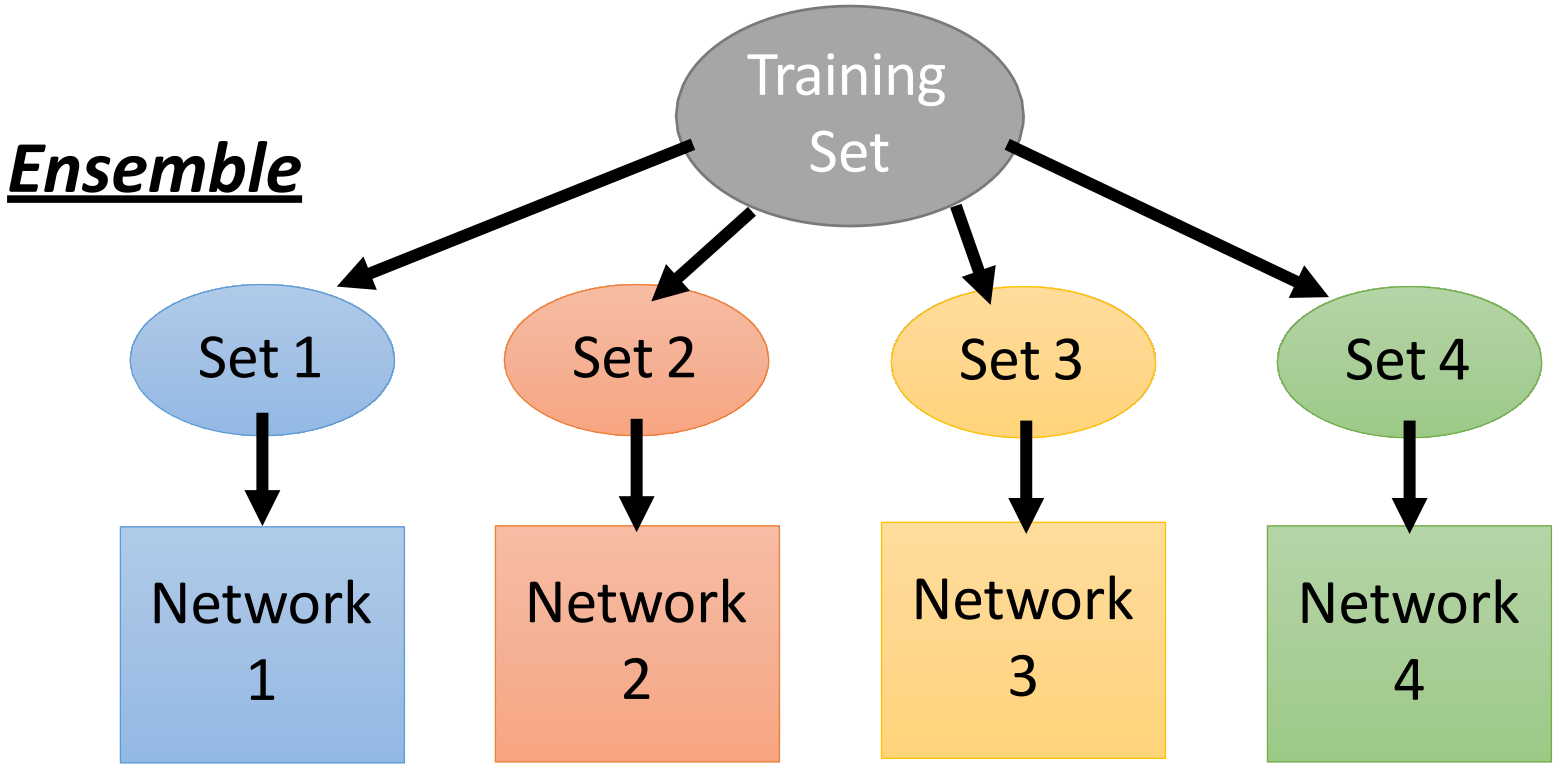
$z' \approx 2z$



Weights multiply (1-p)%

$z' \approx z$

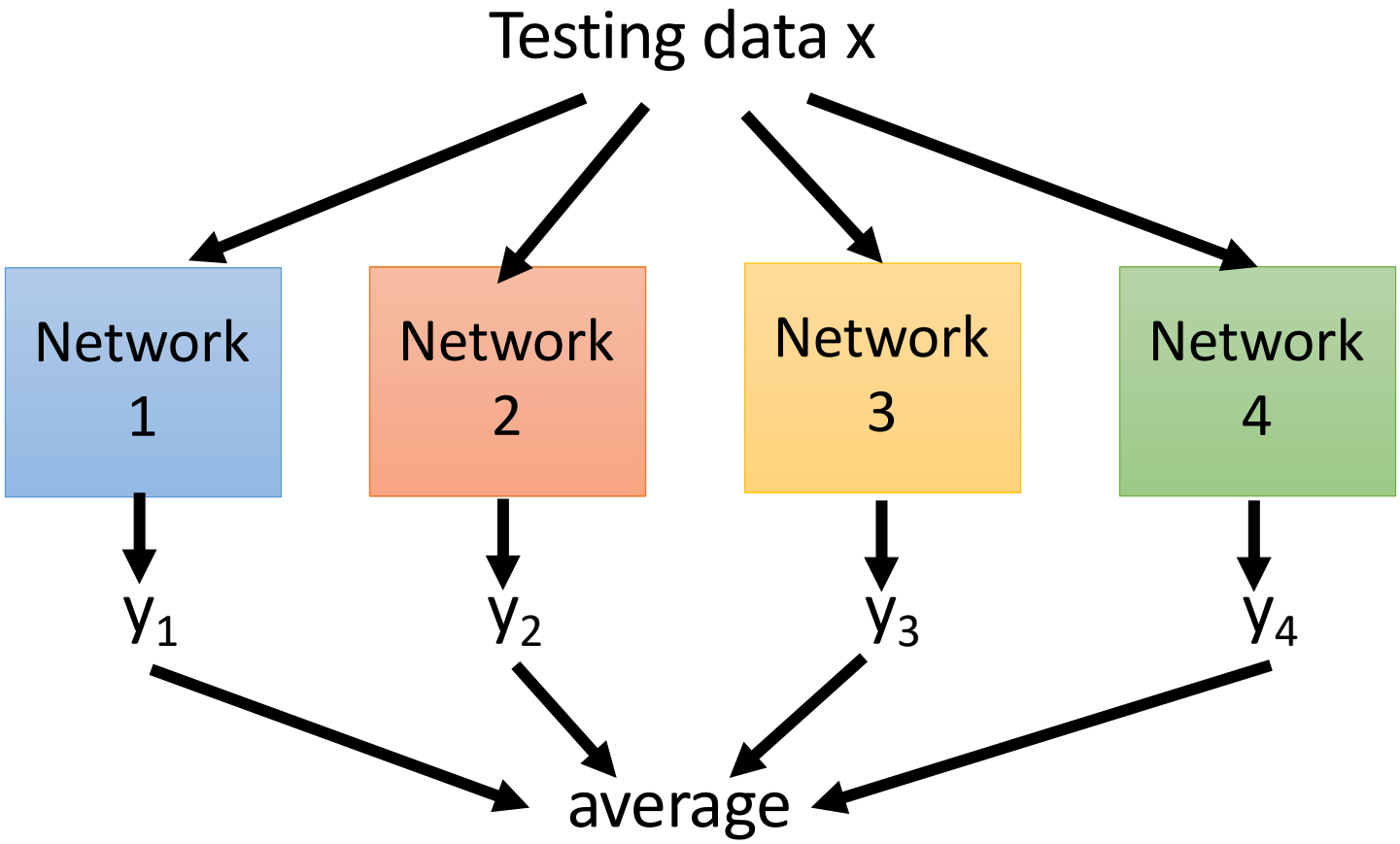
Dropout is a kind of ensemble



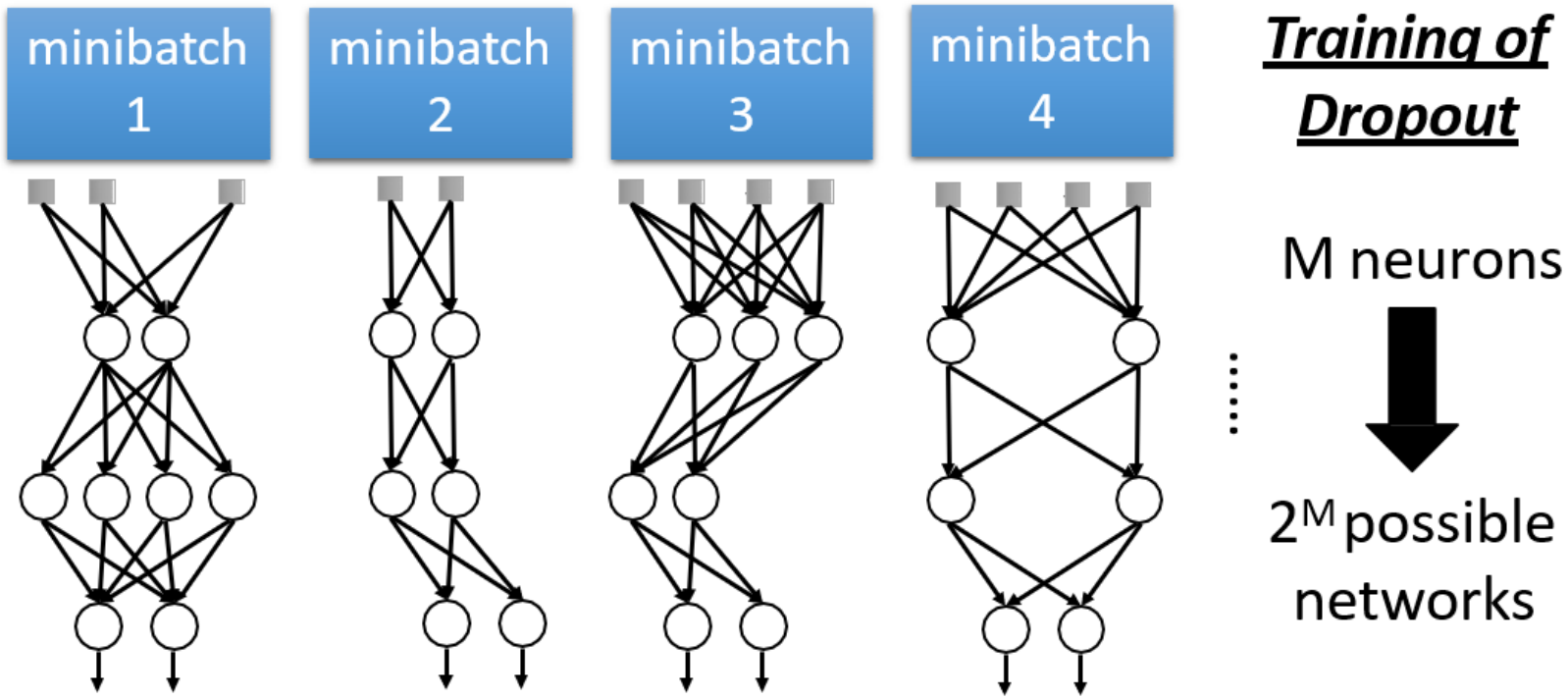
Train a bunch of networks with different structures

Dropout is a kind of ensemble

Ensemble

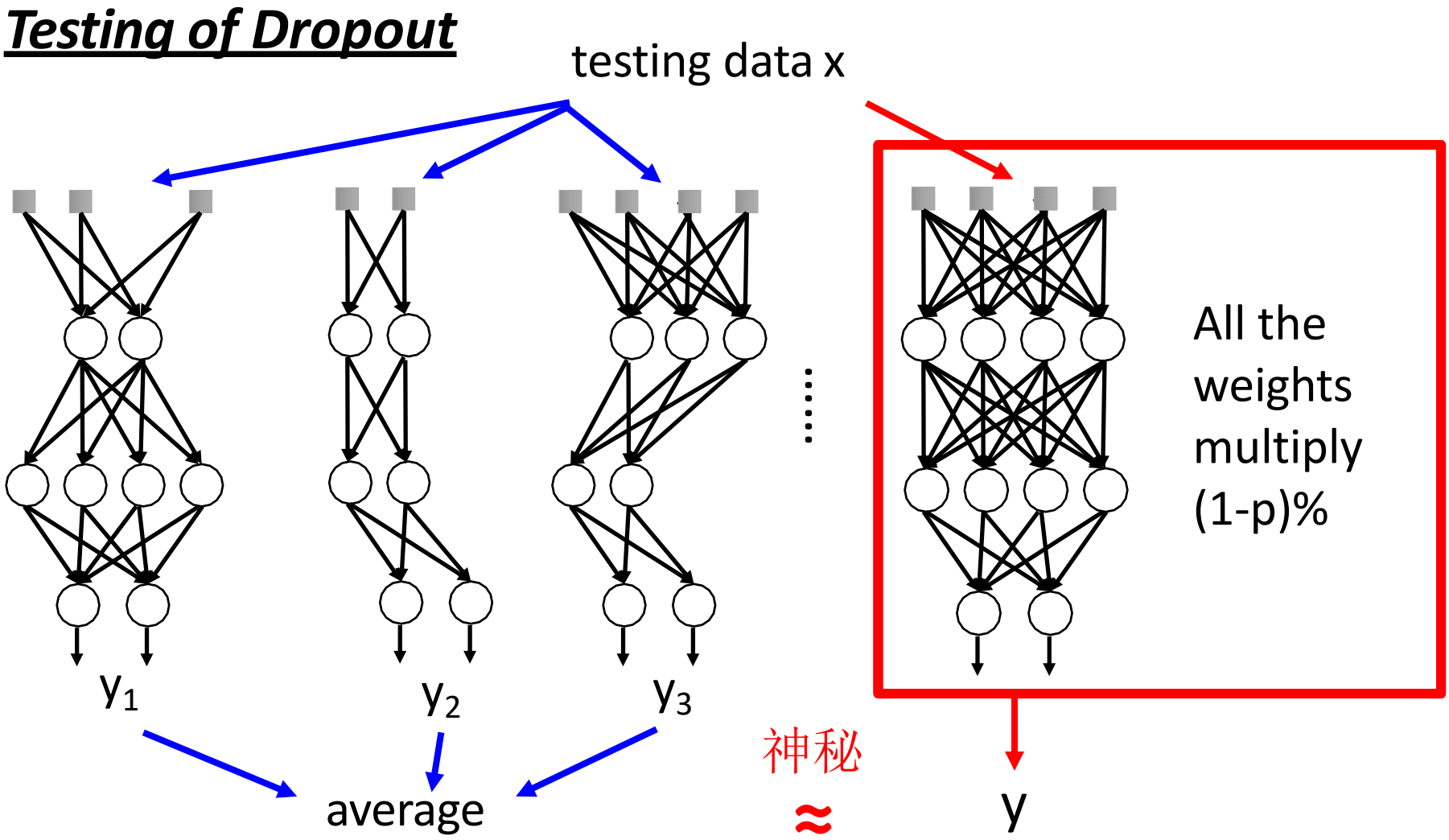


Dropout is a kind of ensemble



- Using one mini-batch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.

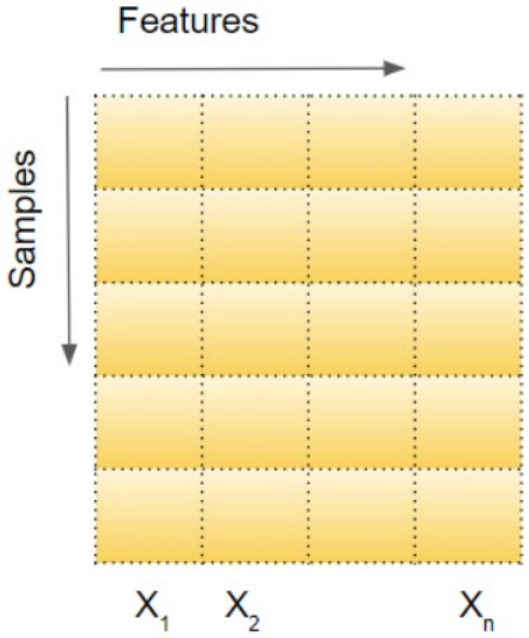


Batch Normalization

- Batch Norm is a neural network layer, it often gets added as part of a Linear or Convolutional block and helps to stabilize the network during training.

Batch Normalization

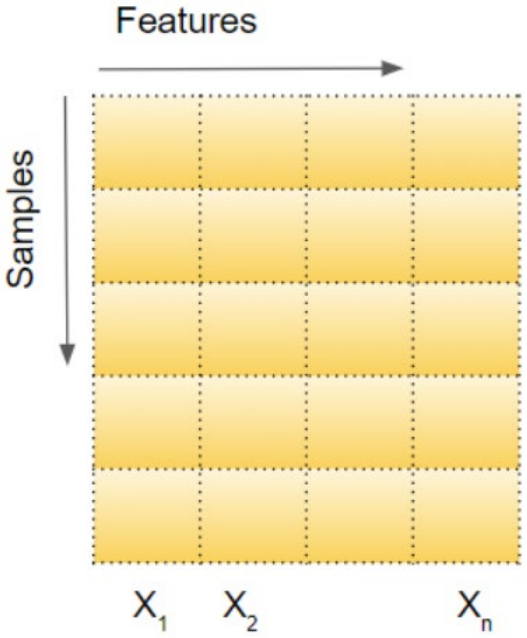
- Normalizing Input Data



$$X_i = \frac{X_i - Mean_i}{StdDev_i}$$

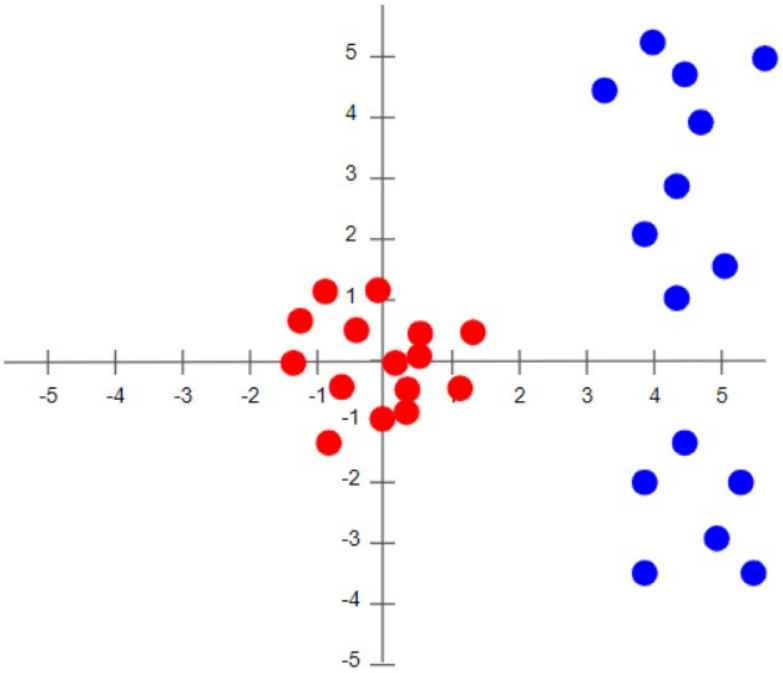
Batch Normalization

- Normalizing Input Data



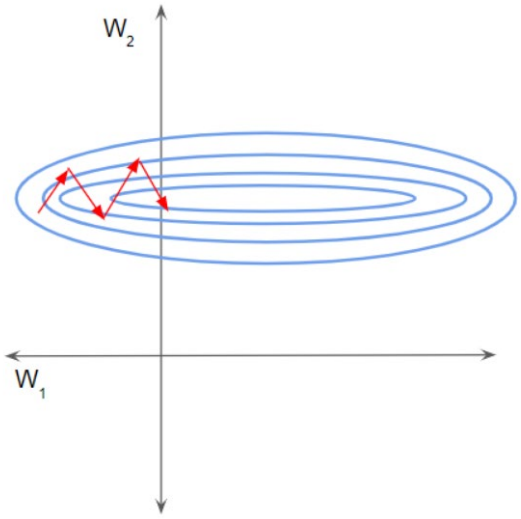
$$X_i = \frac{X_i - Mean_i}{StdDev_i}$$

- The effect of normalizing data



Batch Normalization

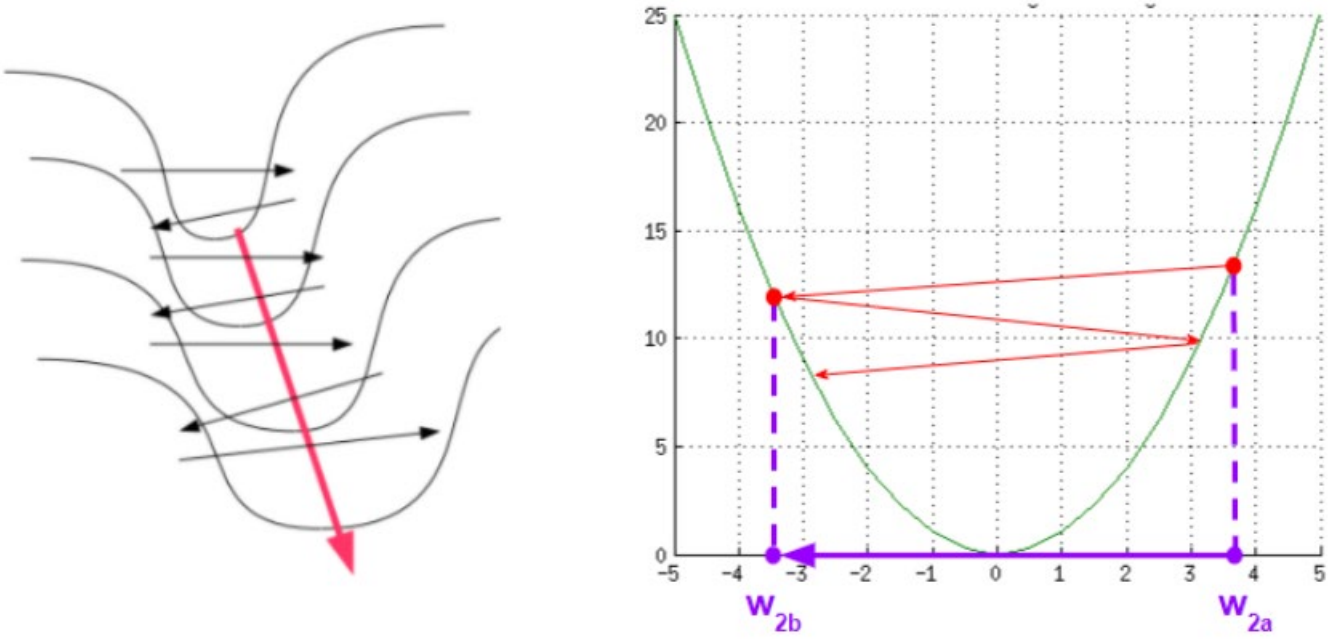
- What happens without normalization
  - An example with just two features that are on drastically different scales.
  - The network output is a linear combination of each feature vector.
  - The gradient descent trajectory



Features on different scales take longer to reach the minimum

Batch Normalization

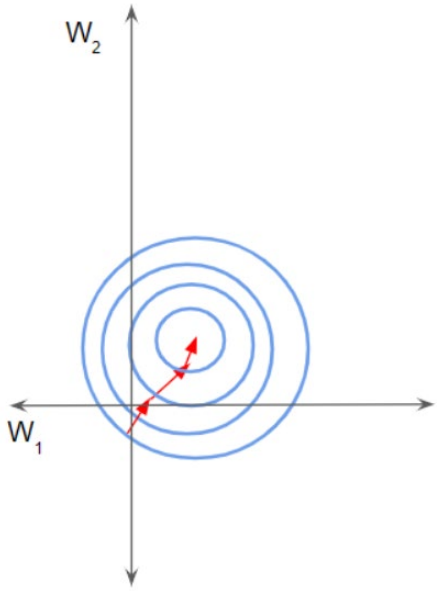
- We can decompose the gradient along the two dimensions. It is steep along one dimension and much more gentle along the other dimension.



A narrow valley causes gradient descent to bounce from one slope to the other

Batch Normalization

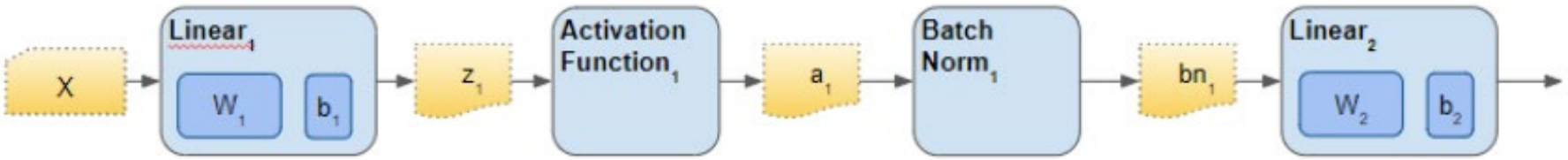
- Instead, if the features are on the same scale, the loss landscape is more uniform like a bowl. Gradient descent can then proceed smoothly down to the minimum.



Normalized data helps the network converge faster

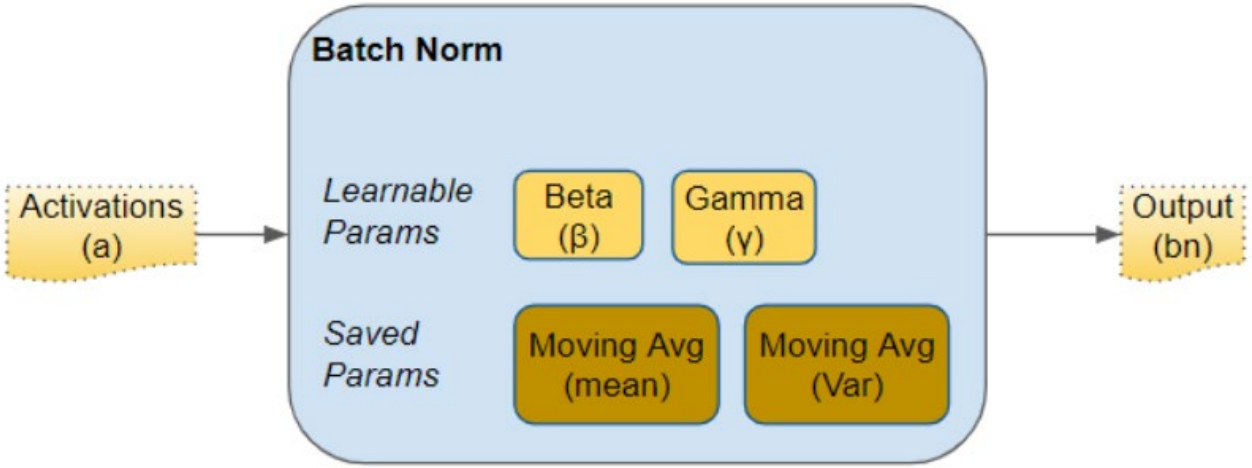
Batch Normalization

- How does BN work?
  - Batch Norm is just another network layer that gets inserted between a hidden layer and the next hidden layer.



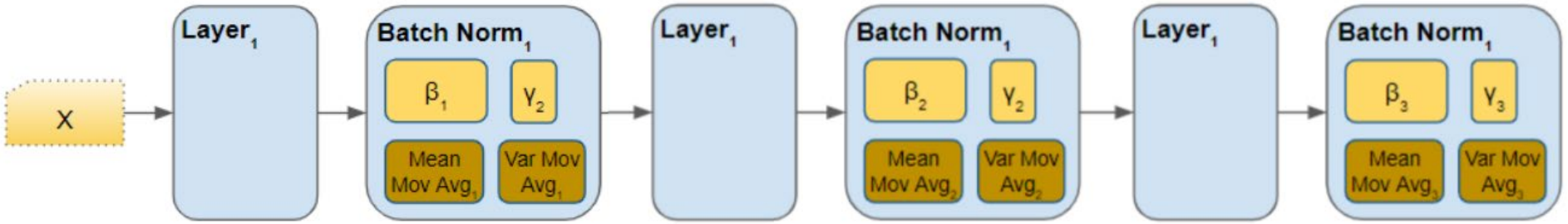
Batch Normalization

- Just like the parameters (eg. weights, bias) of any network layer, a BN layer also has parameters of its own:
  - Two learnable parameters called Beta and Gamma.
  - Two non-learnable parameters (Mean Moving Average and Variance Moving Average) are saved as part of the ‘state’ of the Batch Norm layer.

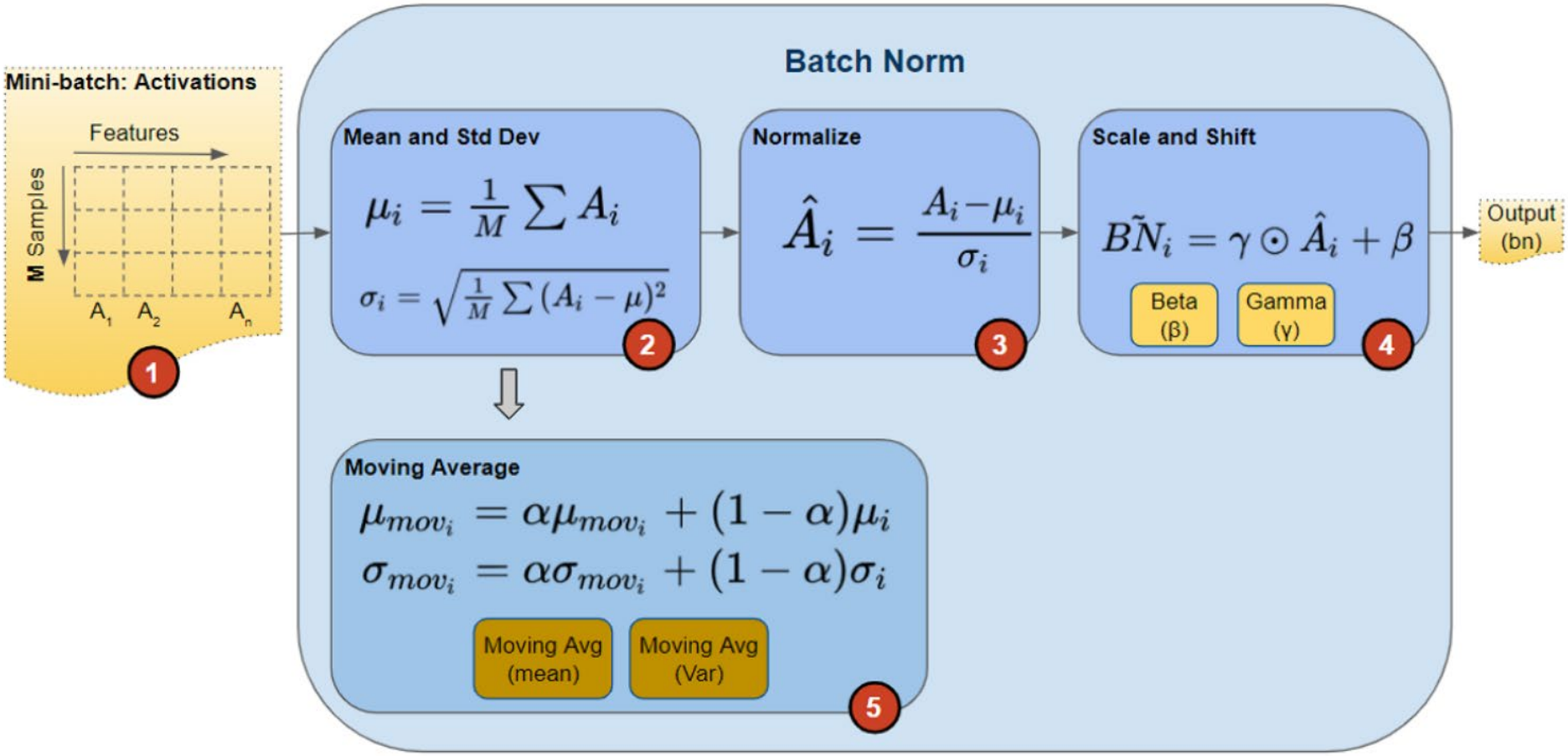


Batch Normalization

- These parameters are in per BN layer.

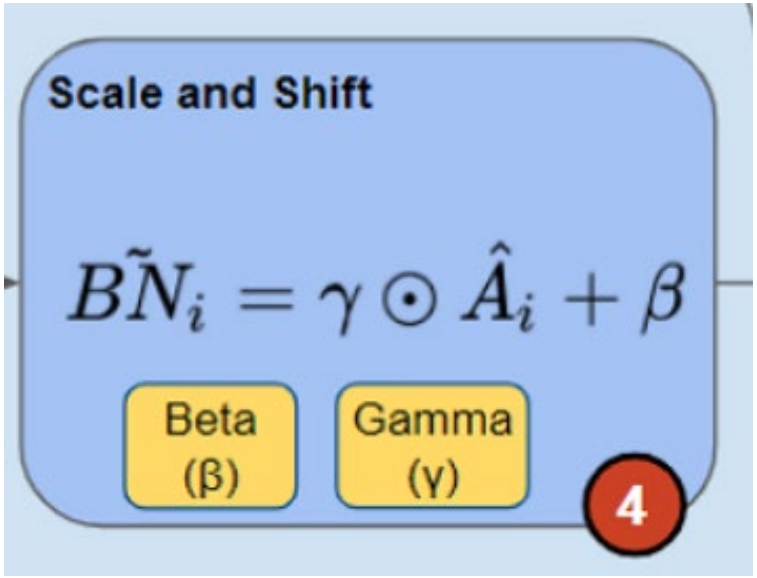


Batch Normalization



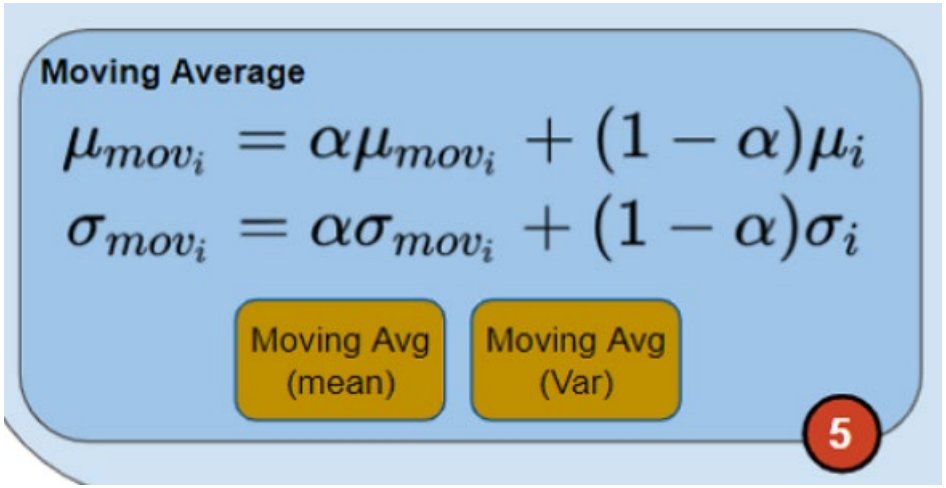
Batch Normalization

- Scale and Shift
  - Unlike the input layer, which requires all normalized values to have zero mean and unit variance, BN allows its values to be shifted (to a different mean) and scaled (to a different variance)



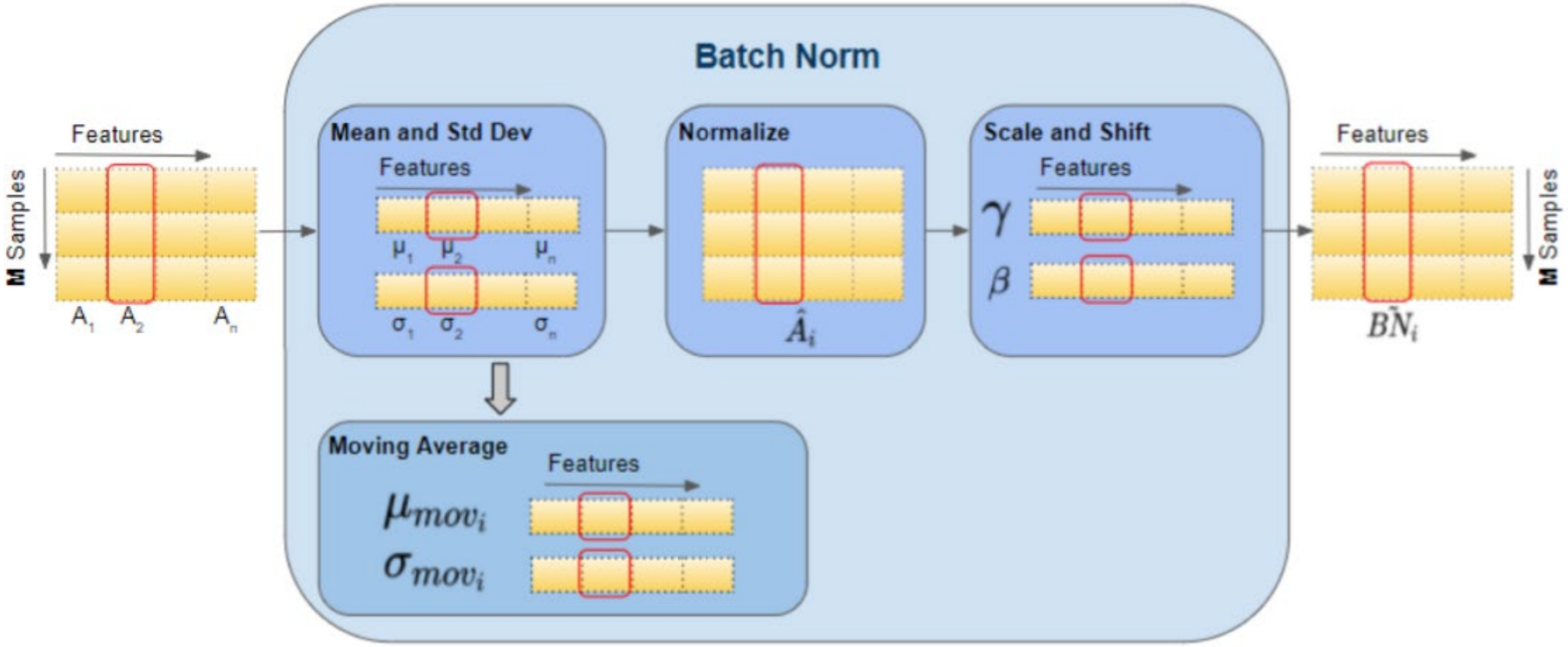
Batch Normalization

- Moving Average
  - BN keeps a running count of the Exponential Moving Average (EMA) of the mean and variance.
  - During training, it simply calculates this EMA but does not do anything with it. At the end of training, it simply saves this value as part of the layer's state, for use during the Inference phase.



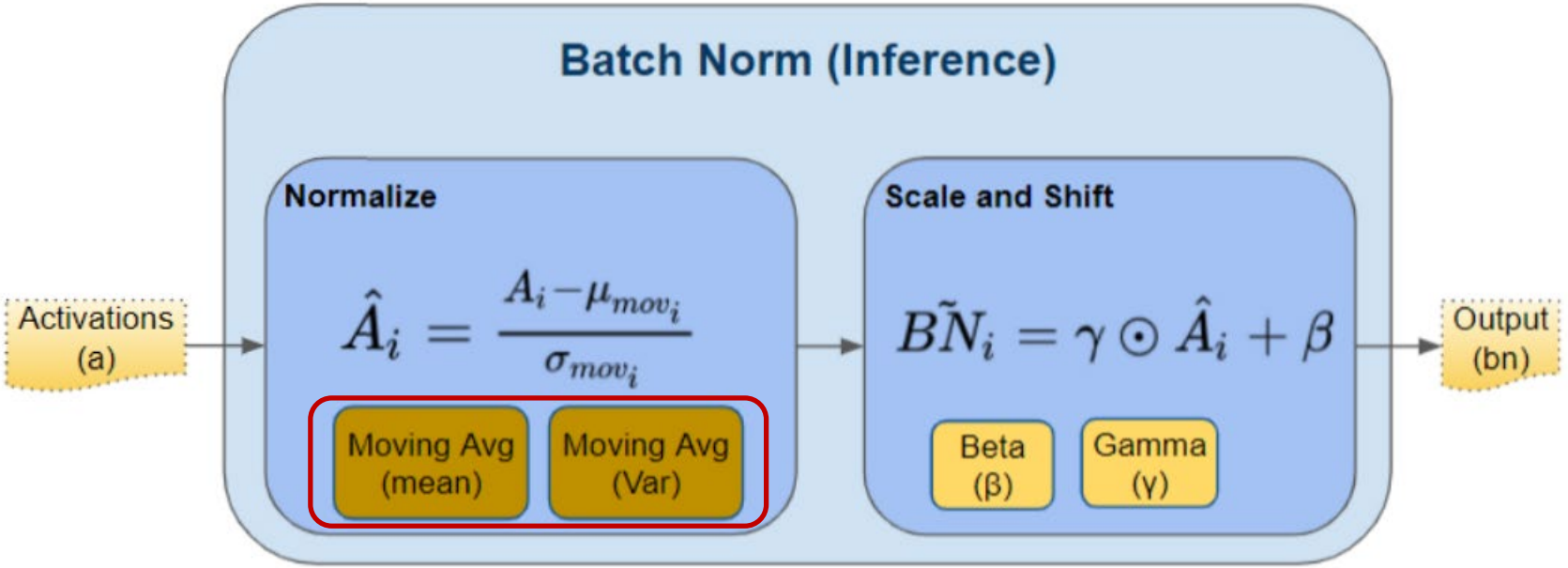
Batch Normalization

- Shapes on BN vectors



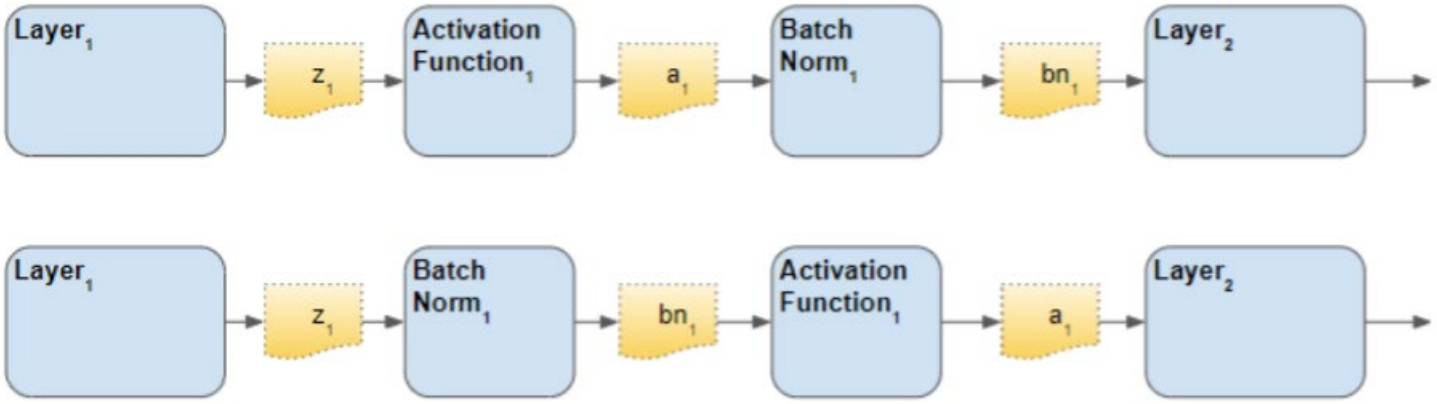
Batch Normalization

- BN during inference
  - During training, BN starts by calculating the mean and variance for a mini-batch. However, during inference, we have a single sample, not a mini-batch. How do we obtain the mean and variance in that case?



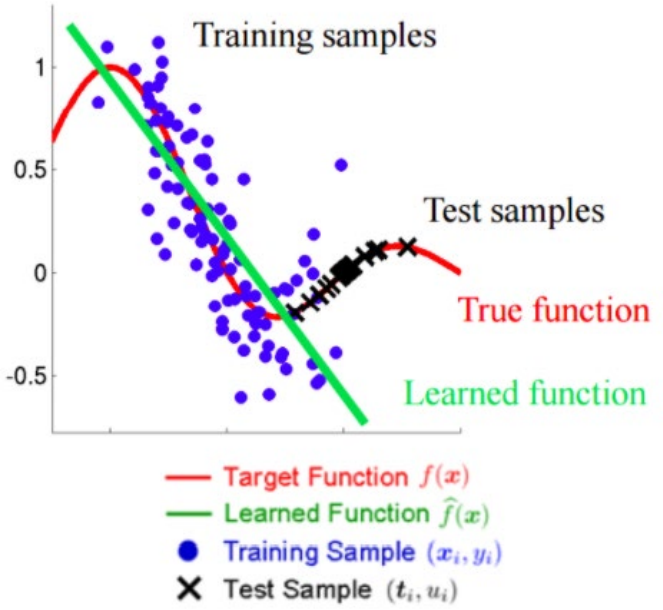
Batch Normalization

- There are two opinions for where the BN layer should be placed in the architecture — before and after activation.
  - The original paper placed it before
  - You will find both options frequently mentioned in the literature. Some say ‘after’ gives better results.



Batch Normalization

- Why does BN work?
  - Theory 1 — Internal Covariate Shift (ICS)
    - *The learning function tries to fit the training data, but the distribution of training and test is different, so predicting using this learned function will definitely give wrong predictions.*

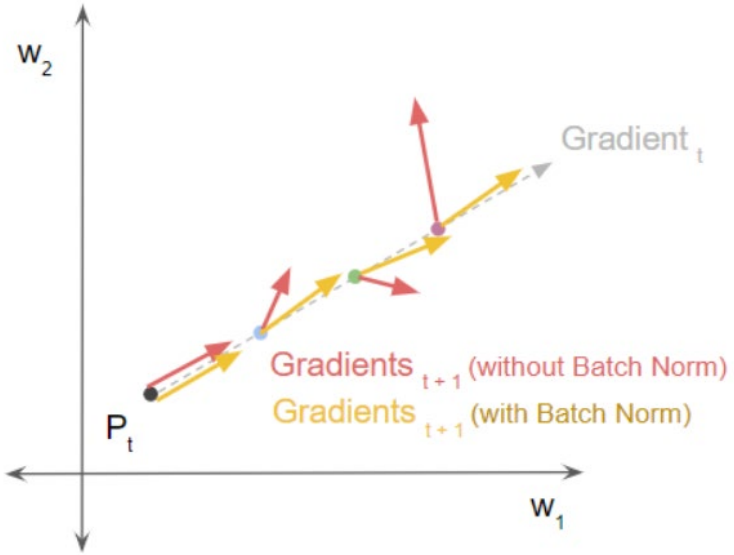


Batch Normalization

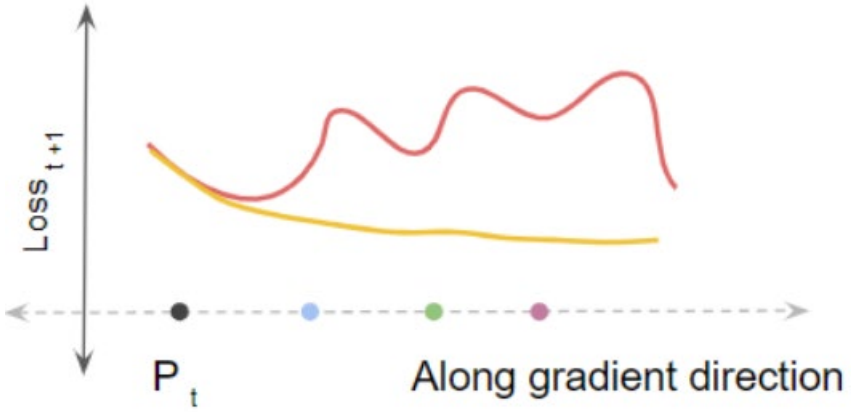
- But, a recent study reveals that such distributional stability of layer inputs has little to do with the success of BatchNorm. **From an optimization viewpoint, BatchNorm might not be even reducing that shift**.
  - *Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, Aleksander Madry: How Does Batch Normalization Help Optimization? NeurIPS 2018: 2488-2498*

Batch Normalization

- Why does BN work?
  - Theory 2 — Loss and Gradient Smoothening



Gradients with Batch Norm are smoother



Loss with Batch declines smoothly

Batch Normalization

- This smoothing effect is not unique to BatchNorm. In fact, several other natural normalization strategies have similar impact and result in a comparable performance gain.

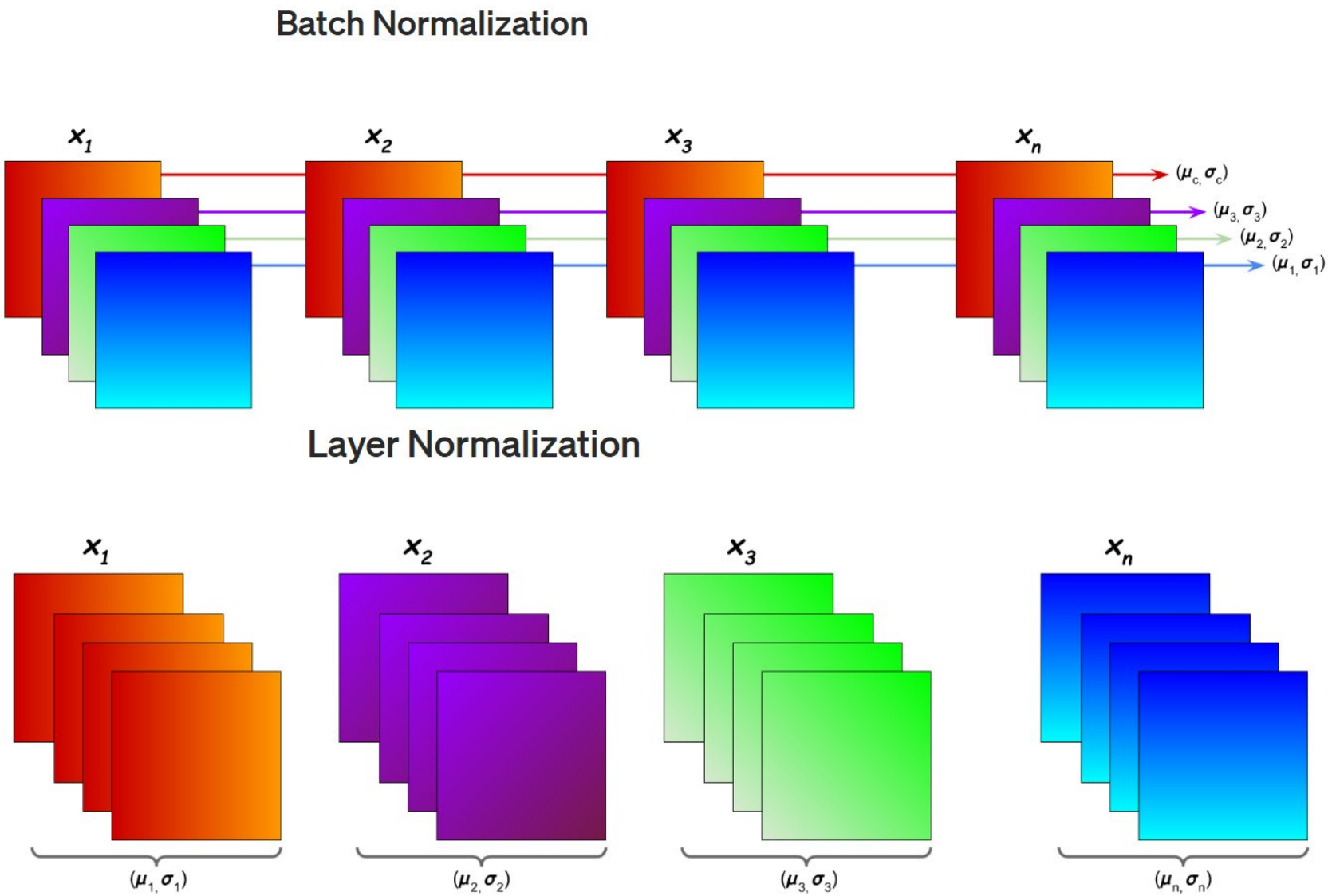
Batch Normalization

- Advantages of BN
  - It allows the model to converge faster and speed up training.
  - BN lets you use higher learning rates.
  - BN also reduces the dependence of gradients on the initial weight values.

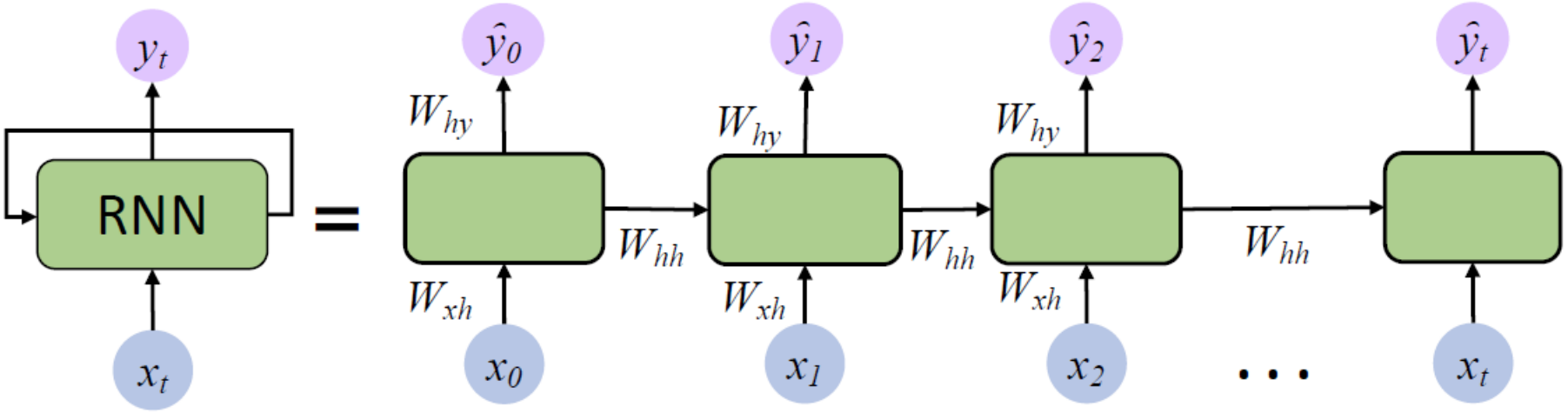
Batch Normalization

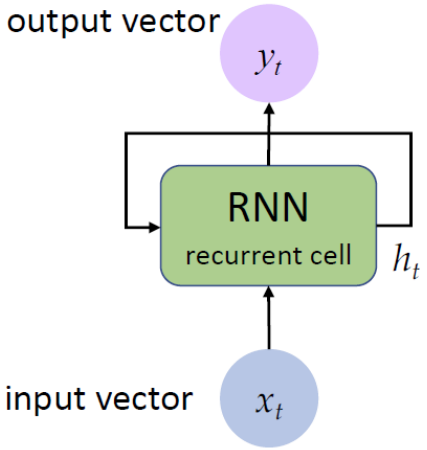
- When is BN not applicable?
  - BN doesn't work well with smaller batch sizes.
    - That results in too much noise in the mean and variance of each mini-batch.
  - BN is not used with recurrent networks.
    - Activations after each timestep have different distributions, making it impractical to apply BN to it.

Layer Norm (扩展内容)



Recap: RNN





Output Vector

$$y_t = \boldsymbol{W}_{hy}^T h_t$$

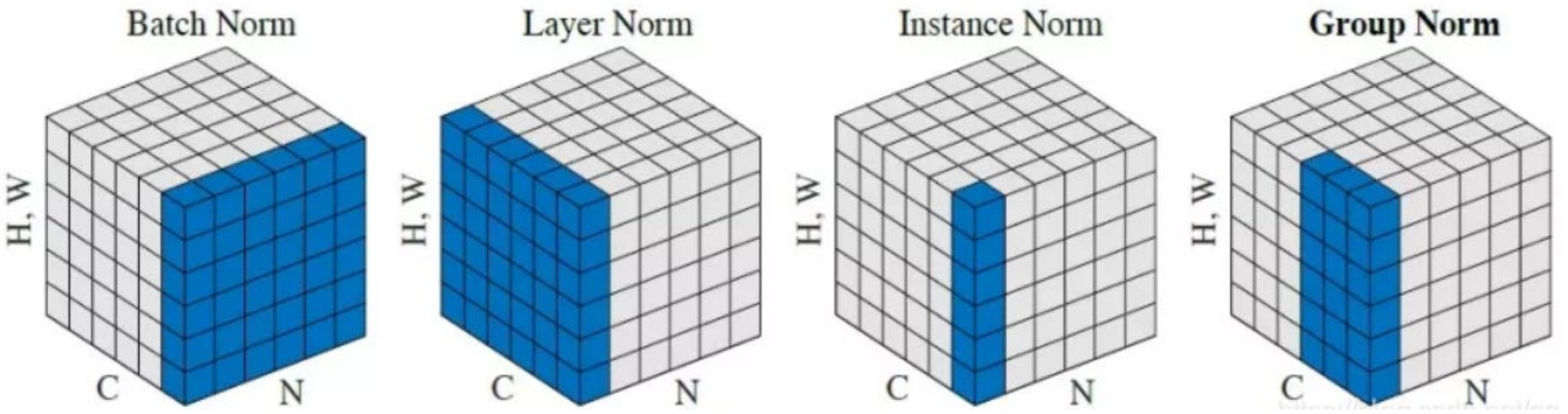
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^T h_{t-1} + \boldsymbol{W}_{xh}^T x_t)$$

Input Vector

$$x_t$$

Different Normalization Layers in Deep Learning （扩展内容）



Residual Networks (ResNet)

- There is a maximum threshold for depth with the traditional CNN model

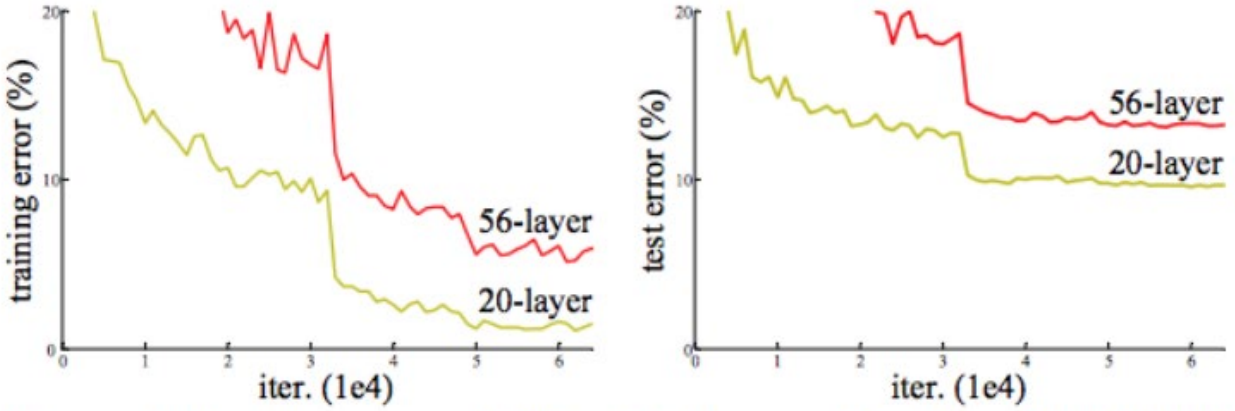


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

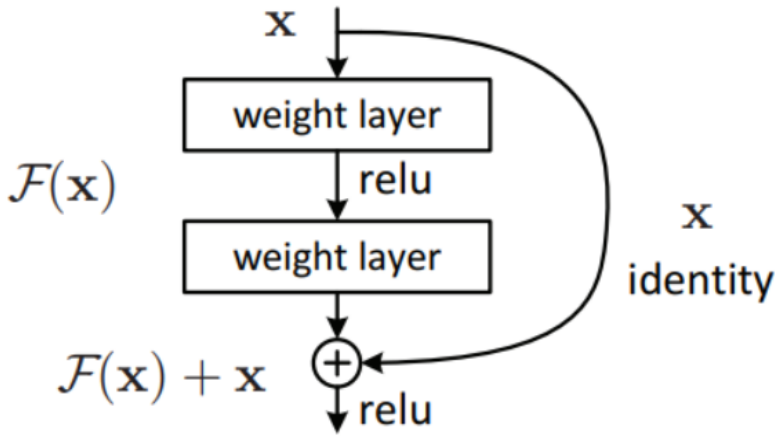
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. 2015.

Residual Networks (ResNet)

- Evidence shows that the best ImageNet models using convolutional and fully-connected layers typically contain between 16 and 30 layers.
  - The failure of the 56-layer CNN could be blamed on the optimization function, initialization of the network, or the famous vanishing/exploding gradient problem.

Residual Networks (ResNet)

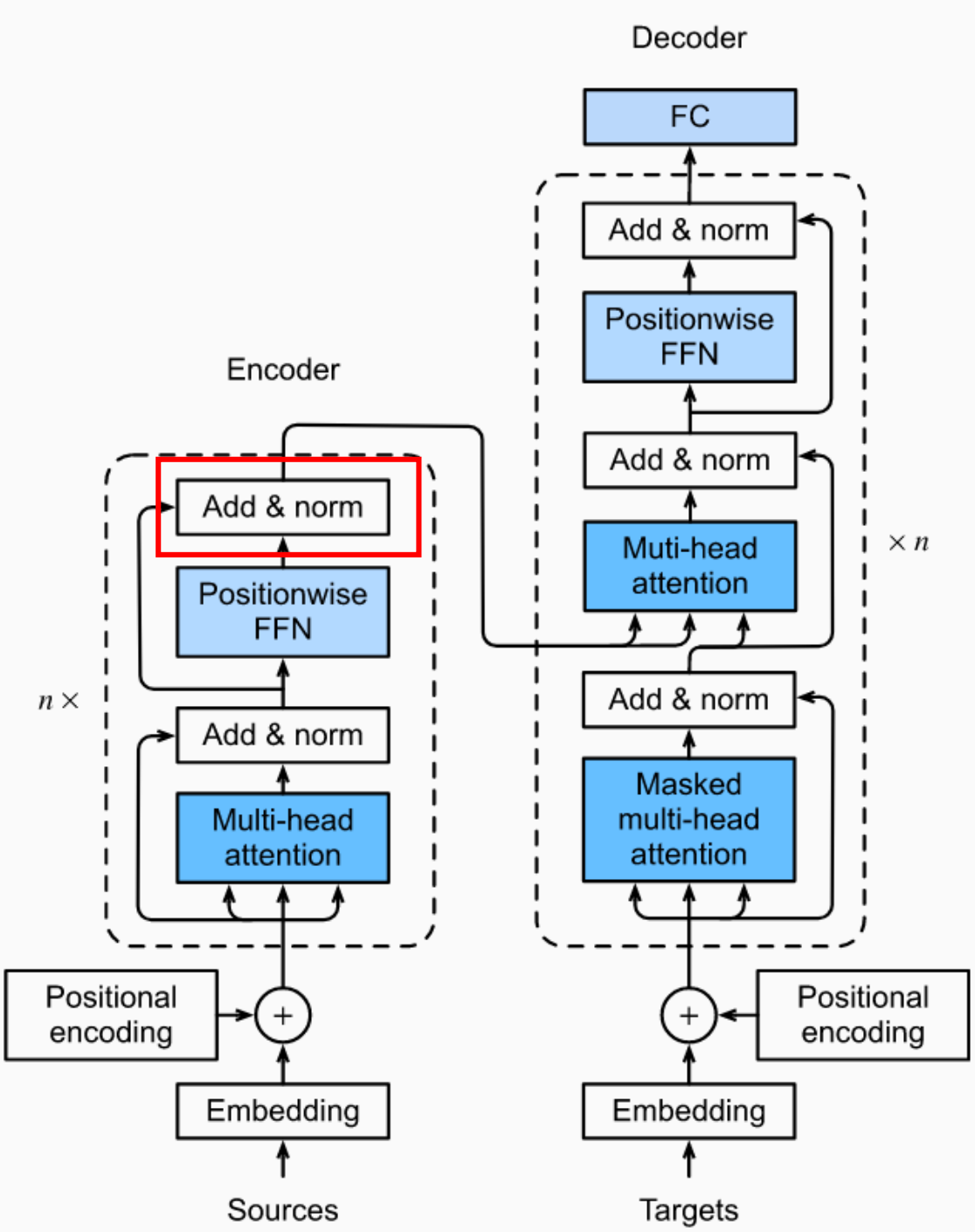
- The problem of training very deep networks has been alleviated with the introduction of a new neural network layer — **The Residual Block.**
  - The *skip connection* skips training from a few layers and connects directly to the output



$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

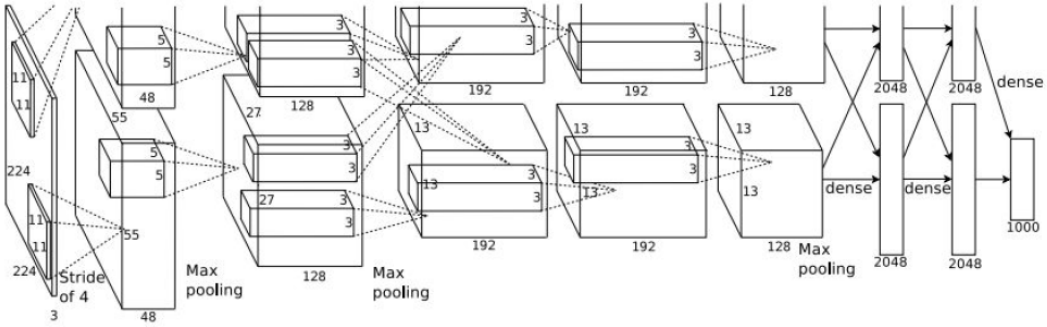
Equation used when F(x) and x have a different dimensionality such as 32x32 and 30x30. This Ws term can be implemented with 1x1 convolutions, this introduces additional parameters to the model.

- Transformer
  - Residual connection
  - Layer Norm



Case Study: AlexNet

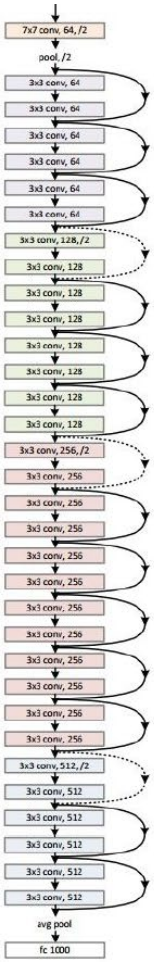
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%



Case Study: ResNet *[He et al., 2015]*

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

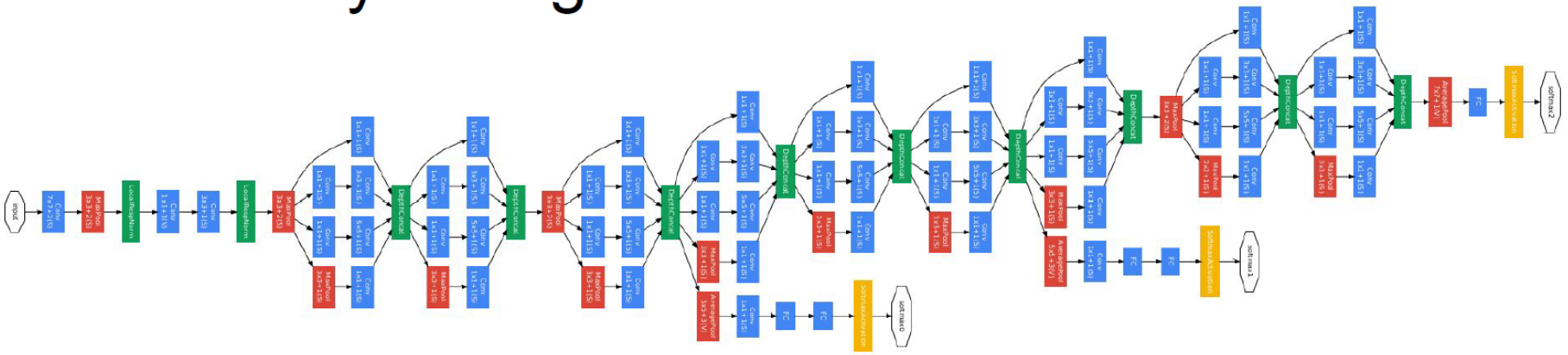
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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					

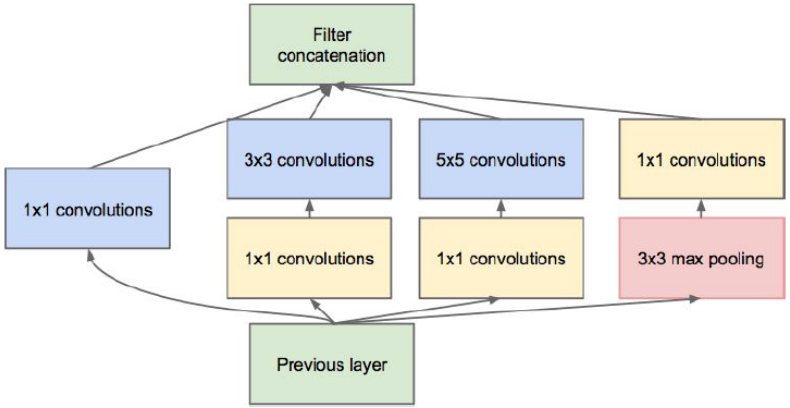
Table 2: **Number of parameters** (in millions).

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

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: 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								

Fun features:

- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Principles/Conventions to build a CNN architecture

- *The basic principle followed in building a convolutional neural network is to '***keep the feature space wide and shallow in the initial stages of the network, and the make it narrower and deeper towards the end.'**

Principles/Conventions to build a CNN architecture

- 1.Always ***start by using smaller filters*** is to collect as much local information as possible, and then ***gradually increase the filter width*** to reduce the generated feature space width to represent more global, high-level and representative information

Principles/Conventions to build a CNN architecture

- 2. Following the principle, ***the number of channels should be low in the beginning*** such that it detects low-level features which are combined to form many complex shapes(by increasing the number of channels) which help distinguish between classes.

Principles/Conventions to build a CNN architecture

- 3. ***General filter sizes used are 3x3, 5x5 and 7x7*** for the convolutional layer for a moderate or small-sized images and ***for Max-Pooling parameters we use 2x2 or 3x3 filter sizes with a stride of 2***. Larger filter sizes and strides may be used to shrink a large image to a moderate size and then go further with the convention stated.

Principles/Conventions to build a CNN architecture

- 4. ***Try using padding*** = same when you feel the border's of the image might be important or just to help elongate your network architecture as padding keeps the dimensions same even after the convolution operation and therefore you can perform more convolutions without shrinking size.

Principles/Conventions to build a CNN architecture

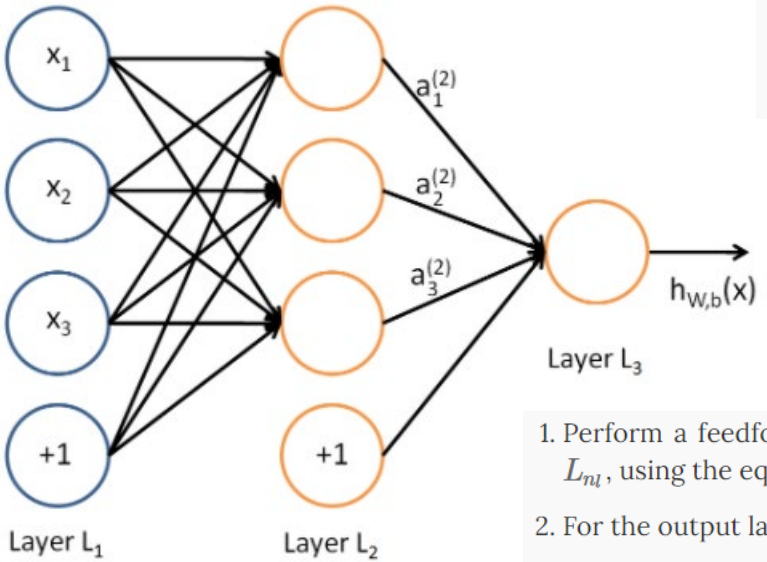
- 5. Keep adding layers until you over-fit. As once we achieved a considerable accuracy in our validation set we can use ***regularization components*** like l1/l2 regularization, dropout, batch norm, data augmentation etc. to reduce over-fitting。

Principles/Conventions to build a CNN architecture

- 6. Always use classic networks like LeNet, AlexNet, VGG-16, VGG-19 etc. as an inspiration while building the architectures for your models. By inspiration I mean follow the trend used in the architectures for example trend in the layers Conv-Pool-Conv-Pool or Conv-Conv-Pool-Conv-Conv-Pool or trend in the Number of channels 32–64–128 or 32–32-64–64 or trend in filter sizes, Max-pooling parameters etc.

Back Propagation in CNN （了解）

- Recap: BP in MLP



$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

$$J(W,b;x,y) = \frac{1}{2}\|h_{W,b}(x) - y\|^2$$

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, up to the output layer L_{n_l}, using the equations defining the forward propagation steps
2. For the output layer (layer n_l), set

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$, set

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. Compute the desired partial derivatives:

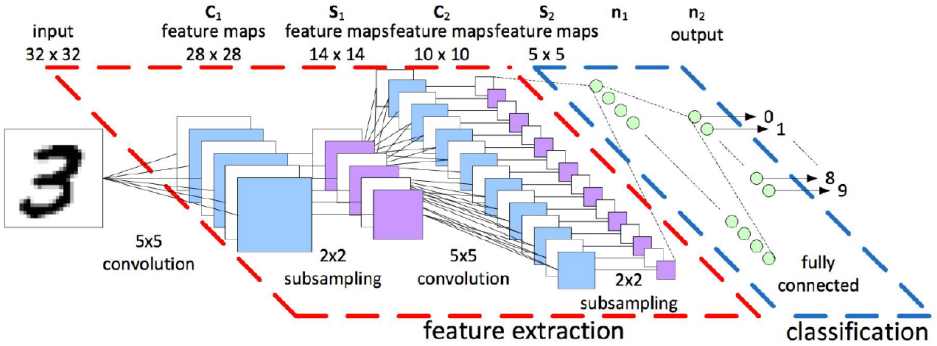
$$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)}(a^{(l)})^T,$$
$$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}.$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Back Propagation in CNN （了解）

- If the *l-th* layer is densely connected to the *(l+1)-st* layer, then the error for the *l-th* layer is computed as

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$



- The gradients are

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$
$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

(Exactly the same as MLP)

http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

Back Propagation in CNN （了解）

- If the *l-th* layer is a convolutional and subsampling layer then the error is propagated through as

$$\delta_k^{(l)} = \text{upsample}\left((W_k^{(l)})^T \delta_k^{(l+1)}\right) \bullet f'(z_k^{(l)})$$

- Where *k* indexes the filter number and $f'(z_k^{(l)})$ is the derivative of the activation function. The upsample operation has to propagate the error through the pooling layer by calculating the error w.r.t to each unit incoming to the pooling layer.

$$\delta_k^l = \begin{pmatrix} 2 & 8 \\ 4 & 6 \end{pmatrix} \xrightarrow{upsample(\delta_k^l)} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 \end{pmatrix}$$

http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

Back Propagation in CNN （了解）

- Finally, to calculate the gradient w.r.t to the filter maps, we rely on the border handling convolution operation again and flip the error matrix $\delta_k^{(l)}$ the same way we flip the filters in the convolutional layer.

$$\nabla_{W_k^{(l)}} J(W, b; x, y) = \sum_{i=1}^{m} (a_i^{(l)}) * \mathbf{rot90}(\delta_k^{(l+1)}, 2)$$

$$\nabla_{b_k^{(l)}} J(W, b; x, y) = \sum_{a,b} (\delta_k^{(l+1)})_{a,b}.$$

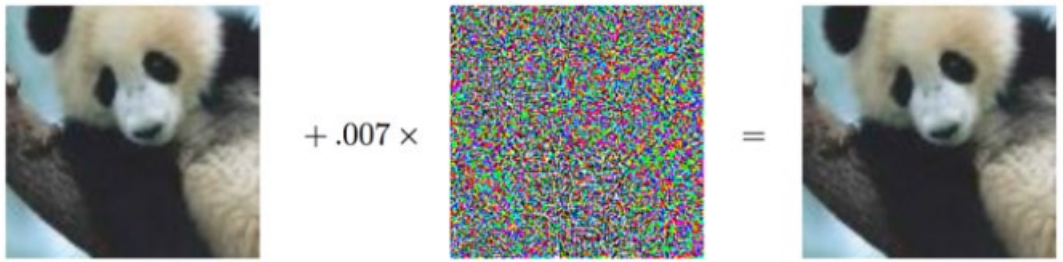
$$ROT180(w_{x,y}^{l+1}) = w_{-x,-y}^{l+1}$$

- Where $a^{(l)}$ is the input to the *l-th* layer, and $a^{(1)}$ is the input image. The operation $(a_i^{(l)}) * \delta_k^{(l+1)}$ is the "valid" convolution between *i-th* input in the *l-th* layer and the error w.r.t. the *k-th* filter.

http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/
More references for BP in CNN:
https://grzegorzgwardys.wordpress.com/2016/04/22/8/
https://zhuanlan.zhihu.com/p/61898234

Adversarial Attacks (扩展内容)

- It’s easy to attain **high confidence** in the incorrect classification of an adversarial example

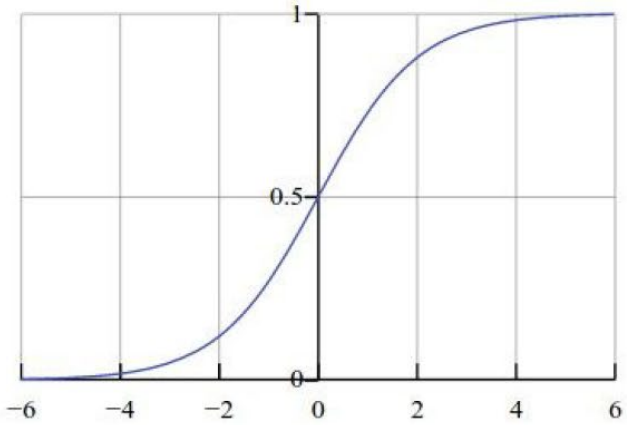


It’s easy to attain **high confidence** in the incorrect classification of an adversarial example. Source: Explaining and Harnessing Adversarial Examples, Goodfellow et al, ICLR 2015.

Adversarial Attacks (扩展内容)

Lets fool a binary linear classifier:
(logistic regression)

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$



Since the probabilities of class 1 and 0 sum to one, the probability for class 0 is $P(y=0 \mid x; w, b) = 1 - P(y=1 \mid x; w, b)$. Hence, an example is classified as a positive example (y = 1) if $\sigma(w^T x + b) > 0.5$, or equivalently if the score $w^T x + b > 0$.

Adversarial Attacks (扩展内容)

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1

← input example

← weights

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Adversarial Attacks (扩展内容)

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1

← input example

← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Adversarial Attacks (扩展内容)

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	?	?	?	?	?	?	?	?	?	?

← input example

← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Adversarial Attacks (扩展内容)

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

← input example

← weights

class 1 score before:

-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3

=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2

=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88

i.e. we improved the class 1 probability from 5% to 88%

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Adversarial Attacks (扩展内容)

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

← input example

← weights

class 1 score before:

-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3

=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2

=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88

i.e. we improved the class 1 probability from 5% to 88%

This was only with 10 input dimensions. A 224x224 input image has 150,528.

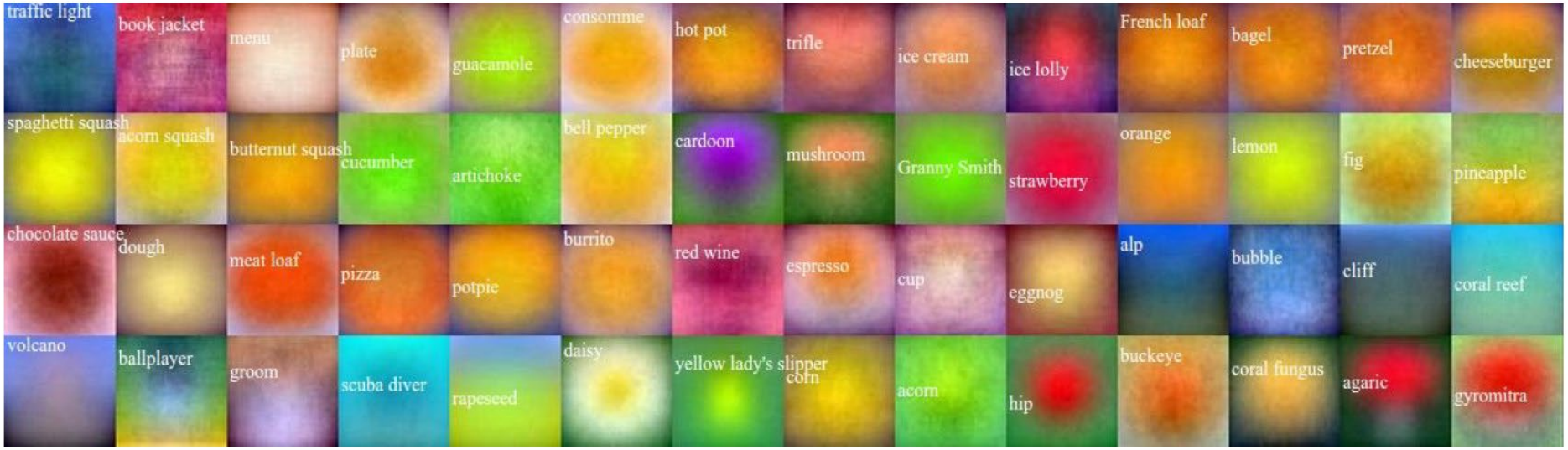
(It's significantly easier with more numbers, need smaller nudge for each)

Adversarial Attacks (扩展内容)

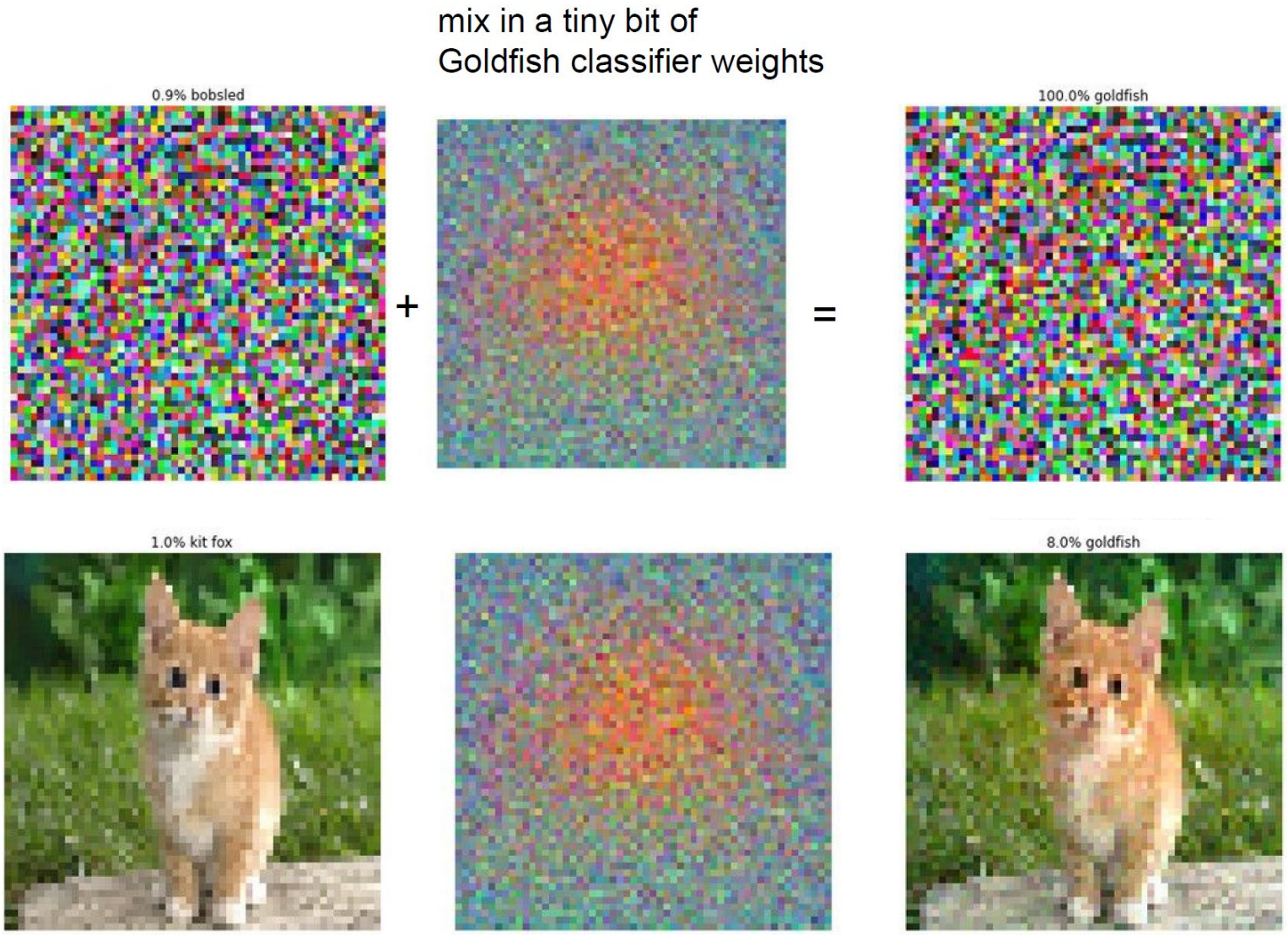
Blog post: Breaking Linear Classifiers on ImageNet

Recall CIFAR-10 linear classifiers:

ImageNet classifiers:

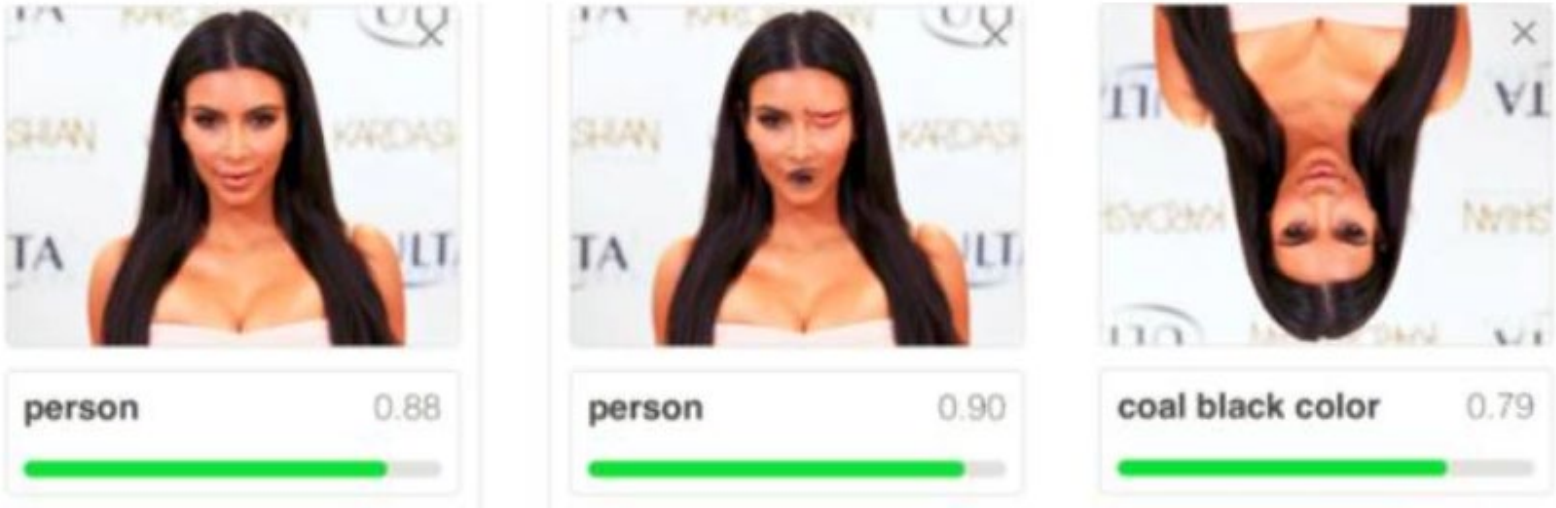


Adversarial Attacks (扩展内容)



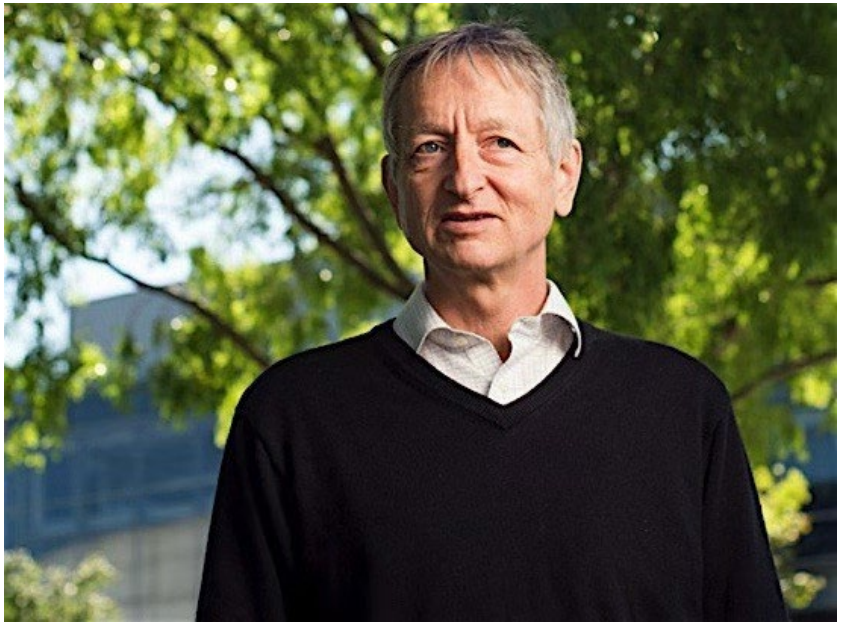
Capsule Networks （扩展内容）

- CNNs have important drawbacks
  - Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects.



Capsule Networks （扩展内容）

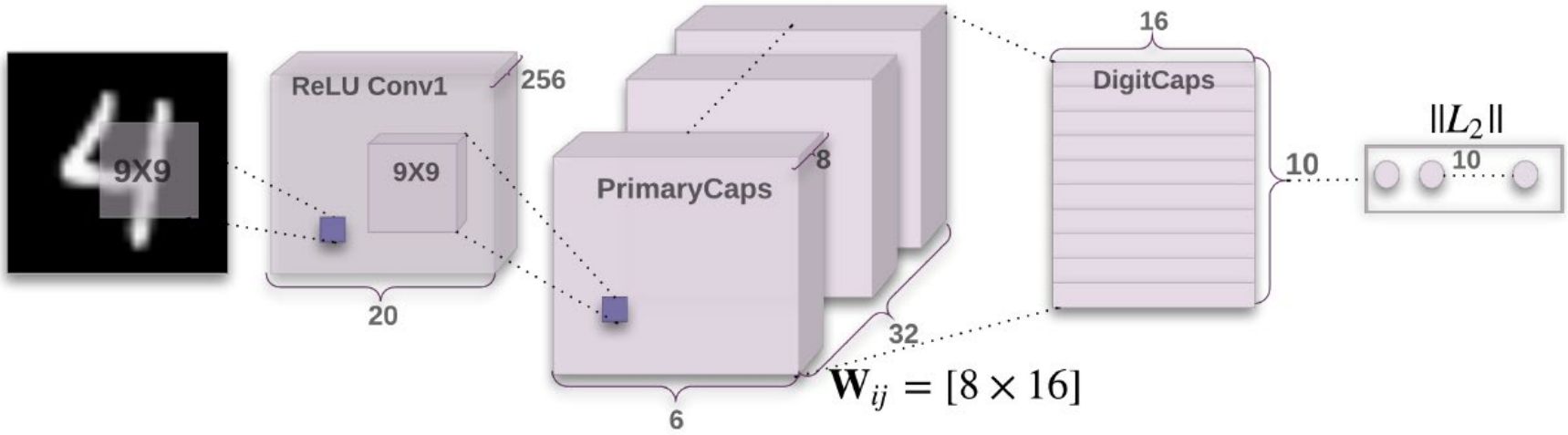
- CNNs have important drawbacks
  - “The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.” – *Geoffrey Hinton*



Capsule Networks （扩展内容）

- Sara Sabour, Nicholas Frosst, Geoffrey E. Hinton: Dynamic Routing Between Capsules.

  NIPS 2017: 3859-3869
- Key idea
  - Scalar-in, scalar-out → vector-in, vector-out
  - Max-pooling → routing-by-agreement



Summary

- Why CNN?
- Elements in CNN
  - Convolution and filters
  - Stride
  - Padding
  - Pooling
  - Flatten
  - Dropout
  - Batch normalization
  - Residual
- Understand CNN architectures