Deep Learning

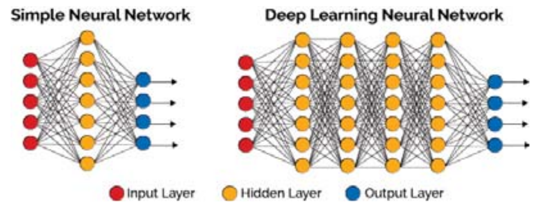
Boonserm Kijsirikul

Dept. of Computer Engineering

Chulalongkorn University

What is Deep Learning?

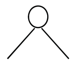
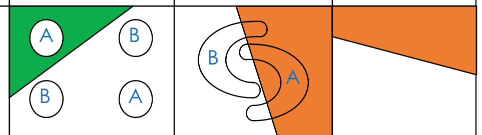
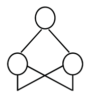
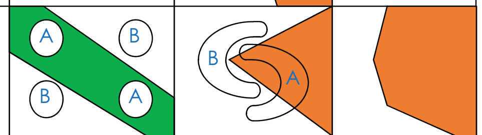
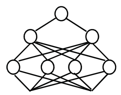
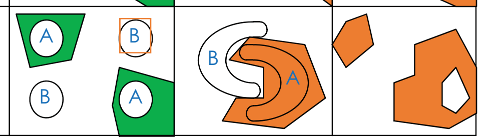
- cascade of many layers of nonlinear processing units for feature extraction and transformation
  - Each successive layer uses the output from the previous layer as input
- may be supervised or unsupervised
- applications include speech recognition, image classification, text analysis, etc.



Motivation

- Deep Architectures can be representationally efficient
  - Fewer computational units for same function
- Automatic feature extraction
  - Less human effort
- Unsupervised learning
  - Modern data sets are enormous
- Deep architectures work well!
  - vision, audio, NLP, etc.

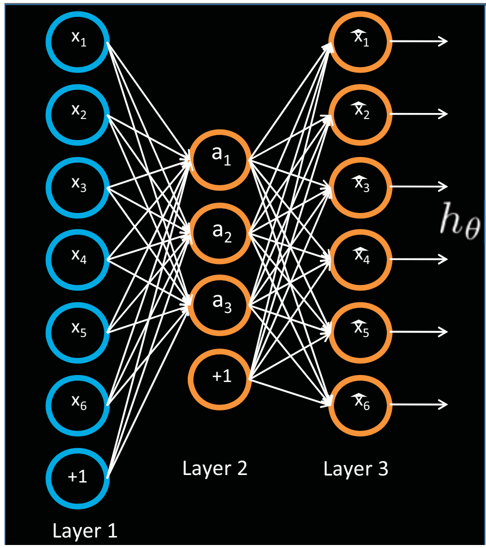
Behavior of Multilayer Neural Networks

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyper plane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Autoencoder

- An autoencoder is a neural network trained to attempt to copy its input to the output.
- Contain two parts:
  – Encoder: map the input to a hidden representation
  – Decoder: map the hidden representation to the output

Unsupervised Feature Learning with a Neural Network



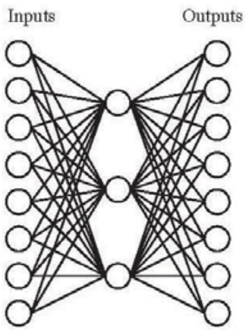
Autoencoder.
Network is trained to output the input (learn identify function).

$$h_\theta(x) \approx x$$

Trivial solution unless:
- Constrain number of units in Layer 2 (learn compressed representation), or
- Constrain Layer 2 to be **sparse**.

Compressed Representation

- Can this be learned

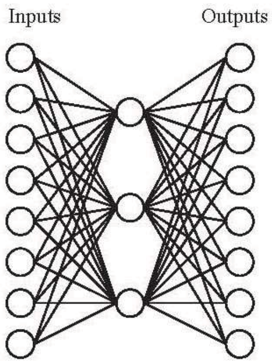


A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Compressed Representation

- Hidden units transform the input space into a new space representing "constructed" features



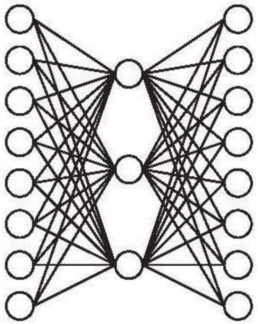
Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Compressed Representation

- Hidden units transform the input space into a new space representing "constructed" features

Inputs Outputs



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	$\overset{1}{.89}$	$\overset{0}{.04}$	$\overset{0}{.08}$	→	10000000
01000000	→	$\overset{0}{.01}$	$\overset{0}{.11}$	$\overset{1}{.88}$	→	01000000
00100000	→	$\overset{1}{.01}$	$\overset{1}{.97}$	$\overset{0}{.27}$	→	00100000
00010000	→	$\overset{1}{.99}$	$\overset{1}{.97}$	$\overset{1}{.71}$	→	00010000
00001000	→	$\overset{0}{.03}$	$\overset{0}{.05}$	$\overset{0}{.02}$	→	00001000
00000100	→	$\overset{0}{.22}$	$\overset{1}{.99}$	$\overset{1}{.99}$	→	00000100
00000010	→	$\overset{1}{.80}$	$\overset{0}{.01}$	$\overset{1}{.98}$	→	00000010
00000001	→	$\overset{1}{.60}$	$\overset{1}{.94}$	$\overset{0}{.01}$	→	00000001

Autoencoder Variants

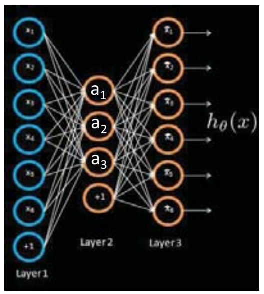
- Bottleneck: use fewer hidden units than inputs
- Sparsity: use a penalty function that encourages most hidden unit activations to be near 0
- Denoising: train to predict true input from corrupted input
- Contractive: force encoder to have small derivatives of hidden unit output as input varies
- Etc.

Sparse Autoencoder

Training a sparse autoencoder.

Given unlabeled training data $x^{(1)}, x^{(2)}, \ldots$

$$\min_{\theta} \sum_i \left(\underbrace{\left\| h_\theta(x^{(i)}) - x^{(i)} \right\|^2}_{\text{Reconstruction error term}} + \lambda \underbrace{\sum_j \left| a_j^{(i)} \right|}_{\text{sparsity}} \right)$$



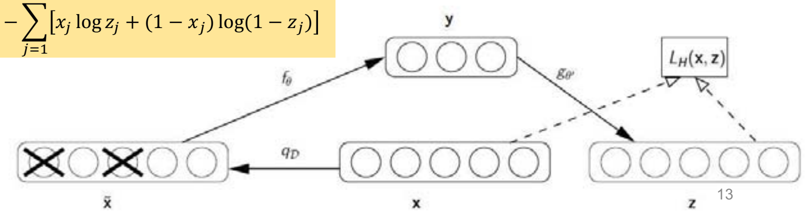
Denoising Autoencoder

- A good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input.
- The partially *corrupted* output is cleaned (de-noised).

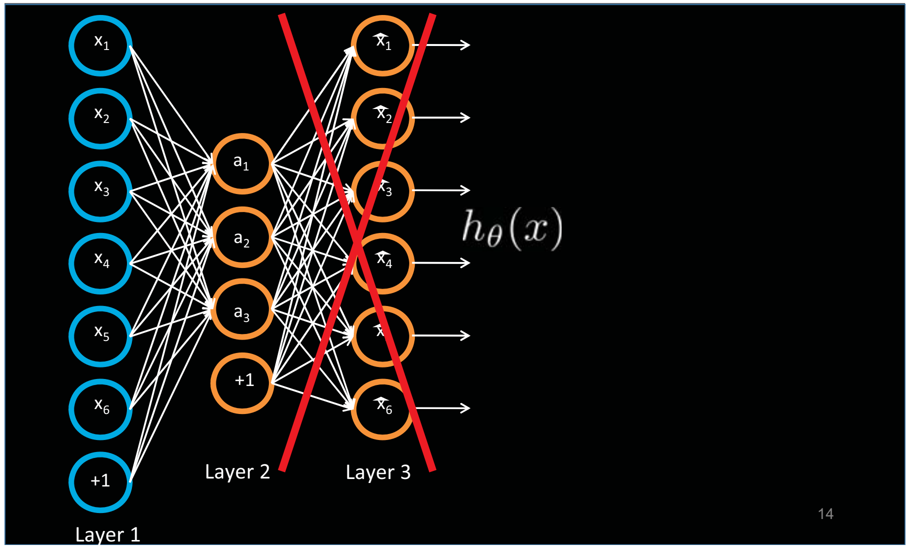
Denoising Autoencoder Procedure

1. (Corrupting) Clean input $\boldsymbol{x}$ is partially corrupted through a stochastic mapping $q_D(\widetilde{\boldsymbol{x}}|\boldsymbol{x})$, yielding corrupted input $\widetilde{\boldsymbol{x}} \sim q_D(\widetilde{\boldsymbol{x}}|\boldsymbol{x})$.
2. The corrupted input $\widetilde{\boldsymbol{x}}$ passes through a basic autoencoder and is mapped to a hidden representation $\boldsymbol{y} = f_\theta(\widetilde{\boldsymbol{x}}) = s(\mathbf{W}\widetilde{\boldsymbol{x}} + b)$.
3. From this hidden representation, we can reconstruct $\boldsymbol{z} = g_\theta(\boldsymbol{y})$.
4. Minimize the reconstruction error $L_H(\boldsymbol{x}, \boldsymbol{z})$.
   - $L_H(\boldsymbol{x}, \boldsymbol{z})$ choices: cross-entropy loss or squared error loss

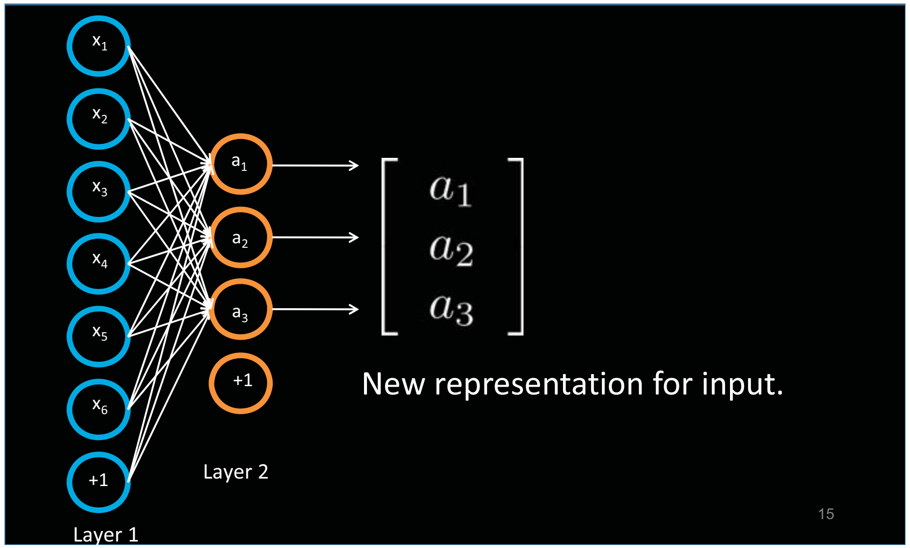
$$L_H(x,z) = -\sum_{j=1}^{d} [x_j \log z_j + (1-x_j)\log(1-z_j)]$$



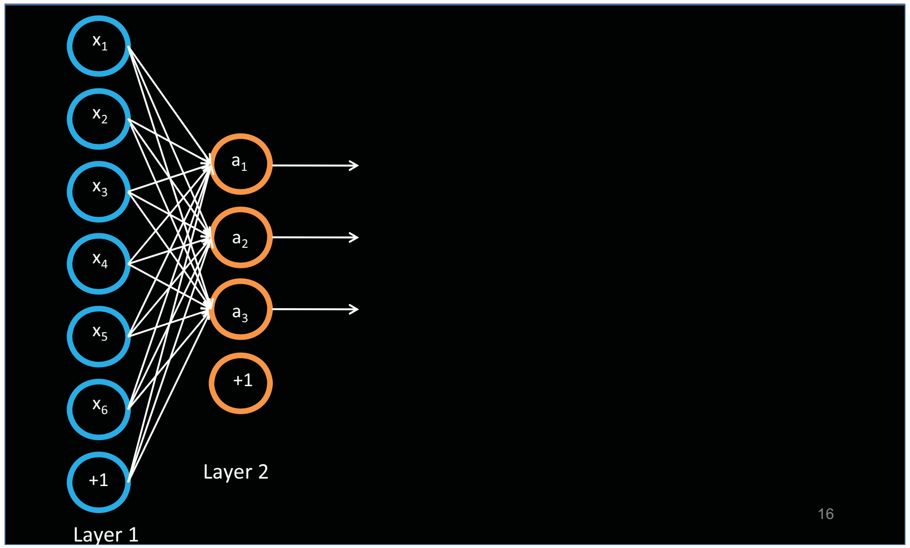
Stacking Autoencoder



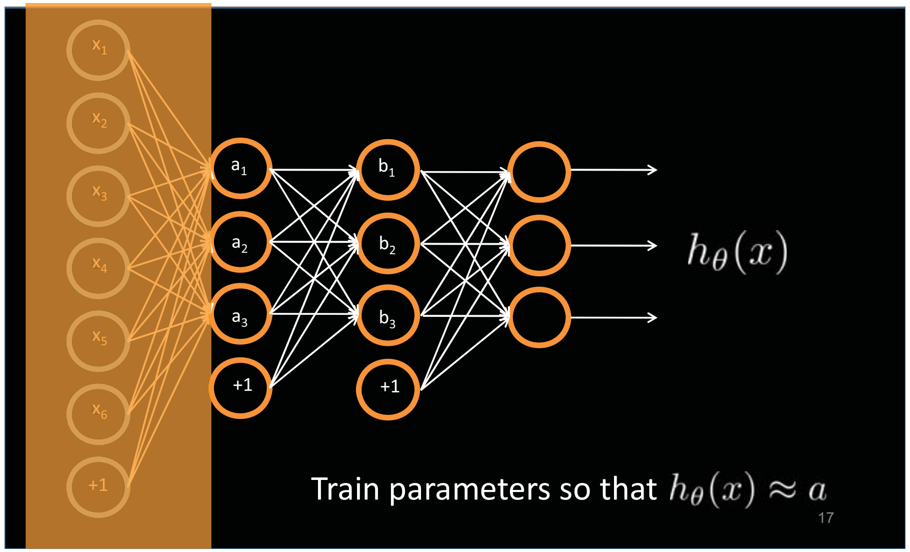
Stacking Autoencoder



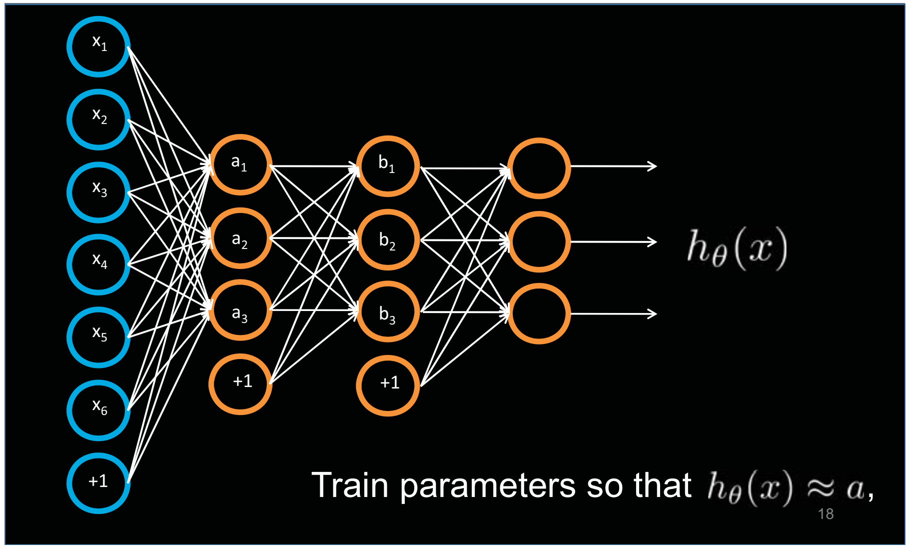
Stacking Autoencoder



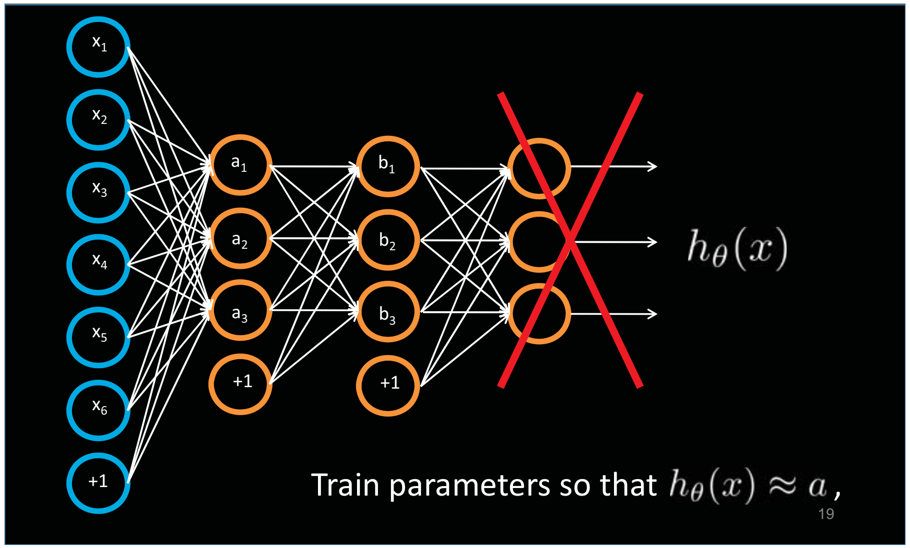
Stacking Autoencoder



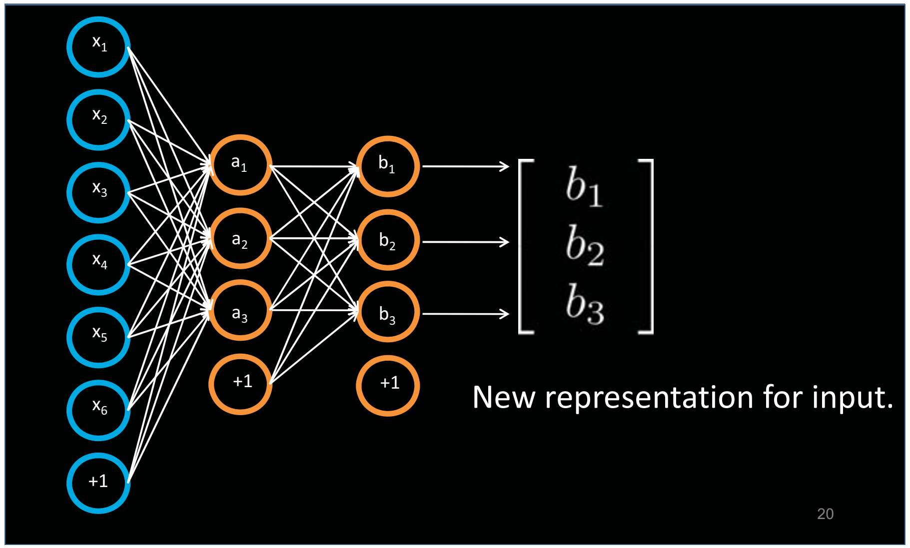
Stacking Autoencoder



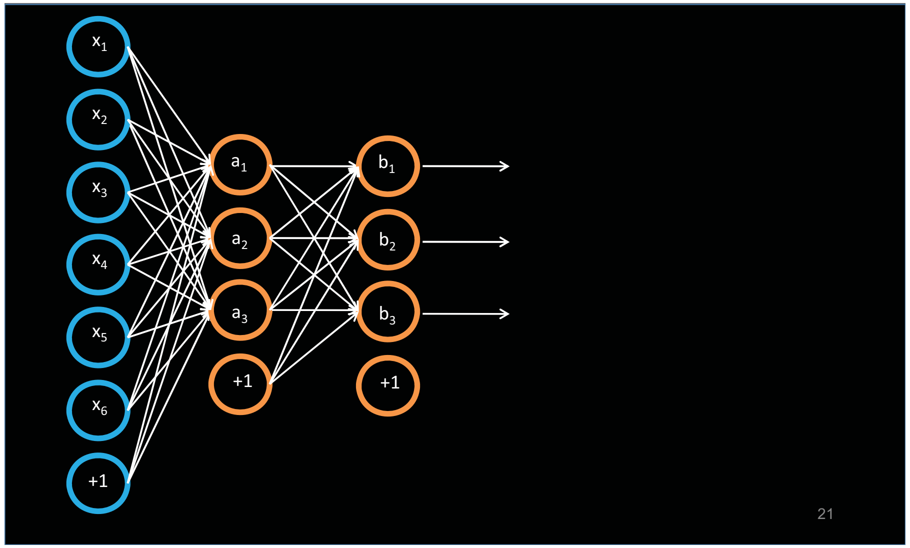
Stacking Autoencoder



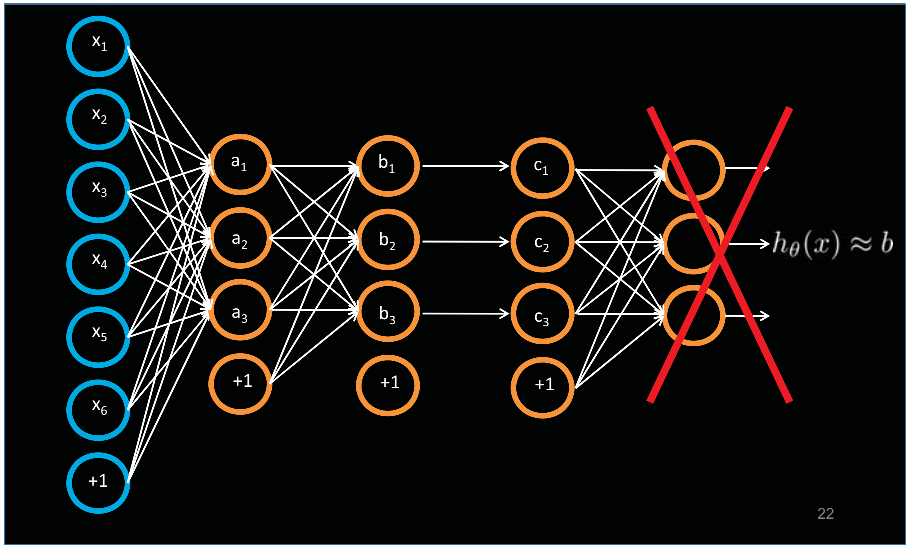
Stacking Autoencoder



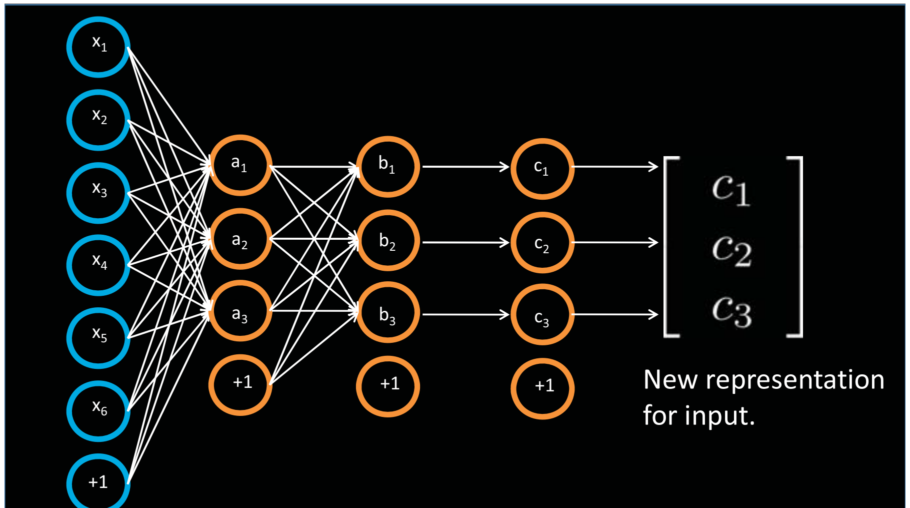
Stacking Autoencoder



Stacking Autoencoder



Stacking Autoencoder



Use [c_1, c_3, c_3] as representation to feed to learning algorithm.

Fine-Tuning

- After completion, run backpropagation on the entire network to fine-tune weights for the supervised task
- Because this backpropagation starts with good structure and weights, its credit assignment is better and so its final results are better than if we just ran backpropagation initially

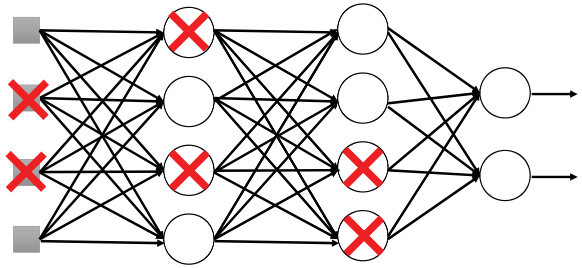
Dropout

- "Hiding" parts of the network during training
- Allows for greater multi-function learning
- Proof against overfitting
- All percentage dropouts work, even 50+%
- Build some redundancy into the hidden units
- Essentially create an "ensemble" of neural networks, but without high cost of training many deep networks

Dropout

Training:

Pick a mini-batch

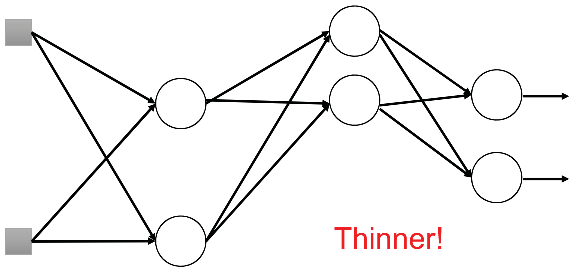


- Each time before computing the gradients
  - Each neuron has p% to dropout

Dropout

Training:

Pick a mini-batch

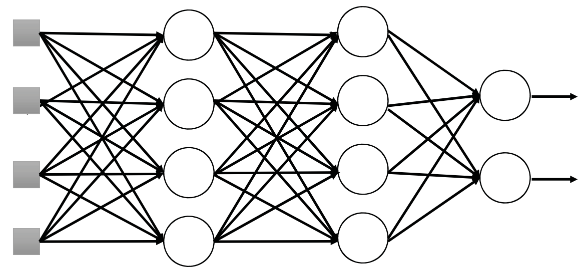


- Each time before computing the gradients
  - Each neuron has p% to dropout
    The structure of the network is changed.
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

Testing:



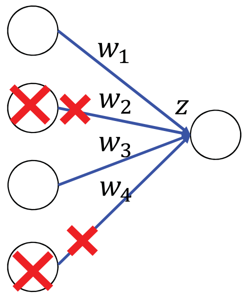
- No dropout
  - If the dropout rate at training is p%, all the weights times (1-p)%
  - Assume that the dropout rate is 50%. If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout

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

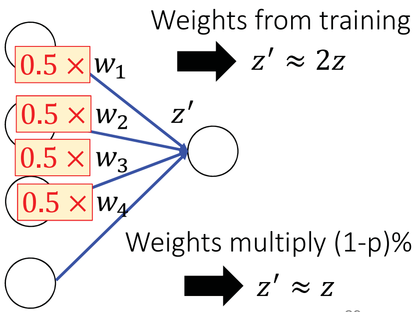
Training of Dropout

Assume dropout rate is 50%



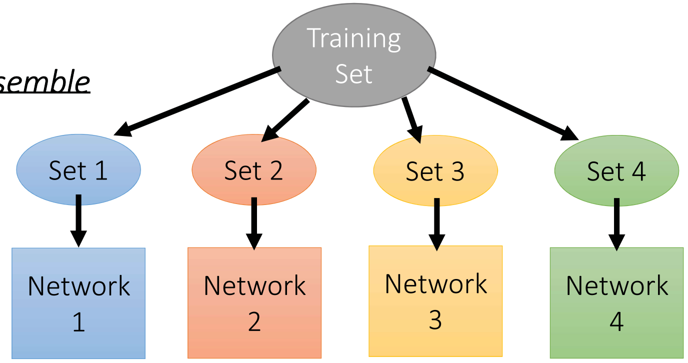
Testing of Dropout

No dropout



Dropout is a kind of ensemble

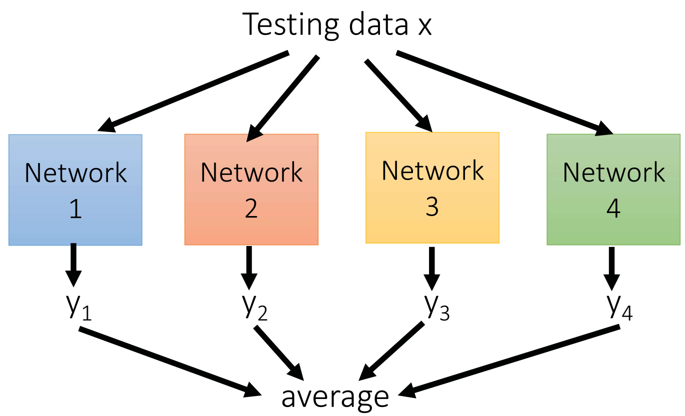
Ensemble



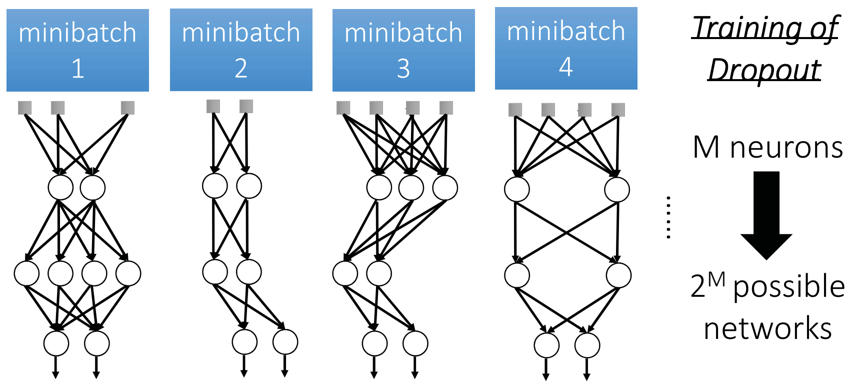
Train a bunch of networks with different structures

Dropout is a kind of ensemble

Ensemble



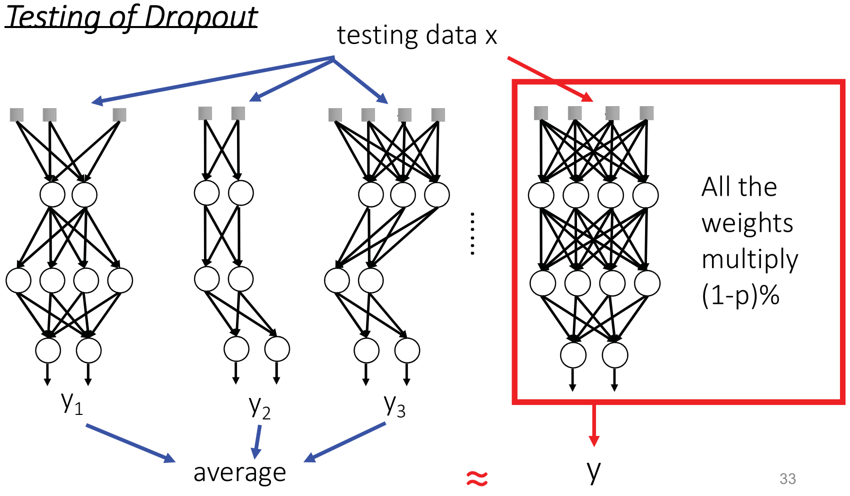
Dropout is a kind of ensemble



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

Dropout is a kind of ensemble

Testing of Dropout



Softmax

- Softmax squashes a K-dimensional output vector z of K-class classification to a K-dimensional vector $\sigma(z)$ of real values in range [0,1]

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for\ k = 1 \ldots K$$

- Example: $z = [1.0, 2.0, 3.0] \Rightarrow \sigma(z) = [0.09, 0.24, 0.67]$
- Softmax can be used as probability.
- Derivative of softmax

$$\frac{d(\sigma(z)_j)}{dz_j} = \sigma(z)_j(1 - \sigma(z)_j) \qquad (1)$$

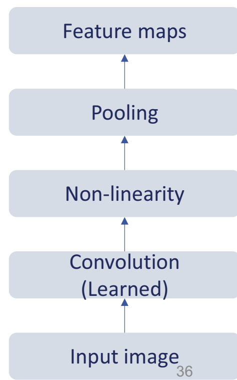
$$\frac{d(\sigma(z)_i)}{dz_j} = -\sigma(z)_i\sigma(z)_j,\ for\ i \neq j \qquad (2)$$

Convolutional Neural Networks

- inspired by mammalian visual cortex
  - The visual cortex contains a complex arrangement of cells, which are sensitive to small sub-regions of the visual field, called a receptive field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.
  - Two basic cell types:
    - Simple cells respond maximally to specific edge-like patterns within their receptive field.
    - Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

Convolutional Neural Networks

- Intuition: Neural network with specialized connectivity structure,
  - Stacking multiple layers of feature extractors
  - Low-level layers extract local features.
  - High-level layers extract learn global patterns.
- A CNN is a list of layers that transform the input data into an output class/prediction.
  - Convolve input
  - Non-linearity
  - Subsampling or Pooling
  - Fully connection
  - Output



Convolutional Neural Networks

LeNet-5

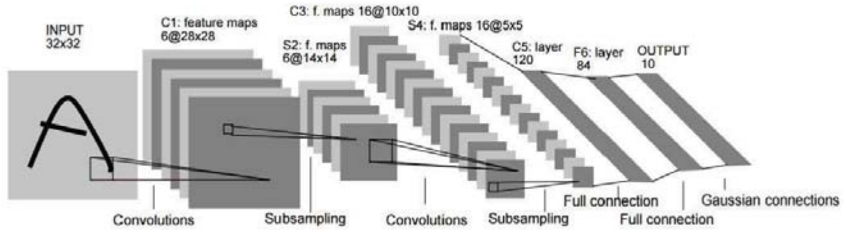


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

LeNet-5

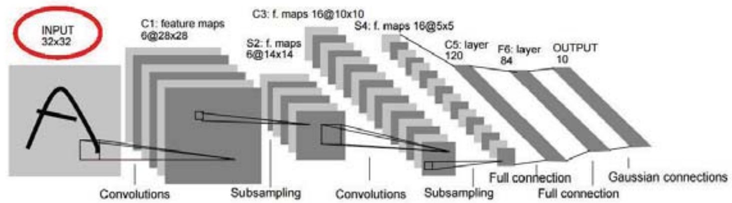


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

LeNet-5

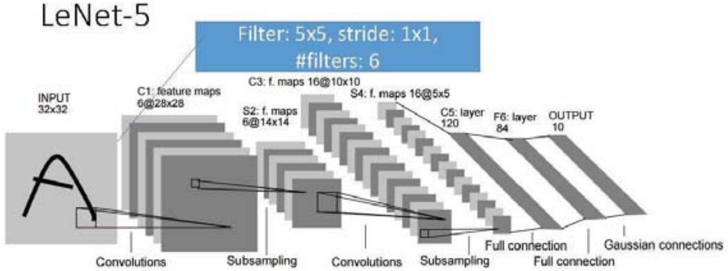


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

LeNet-5

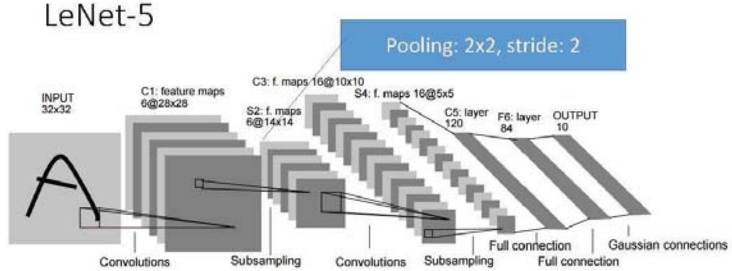


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

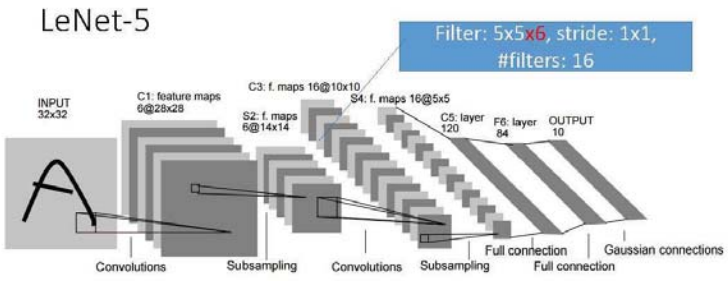


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

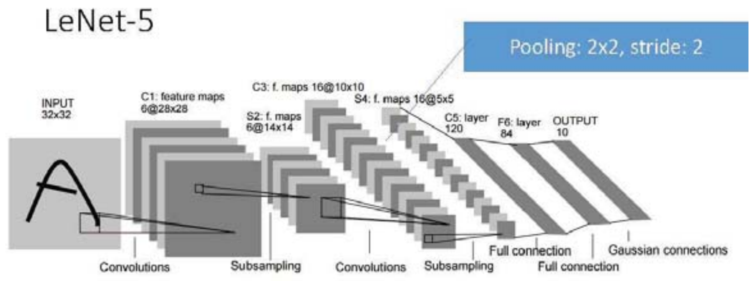


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

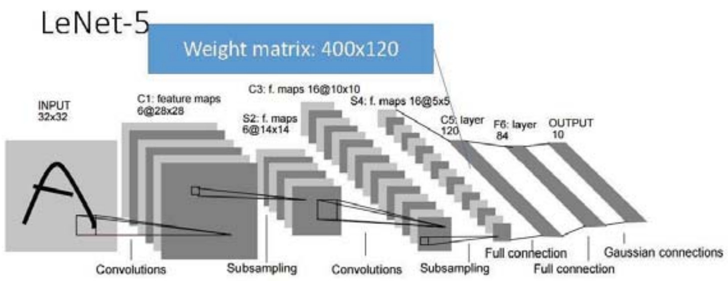


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

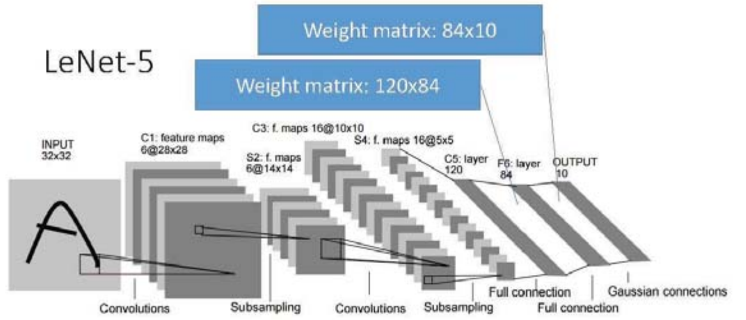


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

Big Difference from Small Shift

First 5x5 values

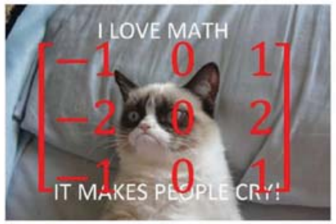
```
array([[51, 49, 51, 56, 55],
       [53, 53, 57, 61, 62],
       [67, 68, 71, 74, 75],
       [76, 77, 79, 82, 80],
       [71, 73, 76, 75, 75]], dtype=uint8)
```

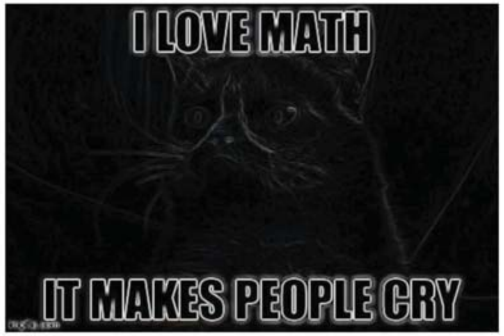
First 5x5 values

```
array([[58, 57, 57, 59, 59],
       [58, 57, 57, 58, 59],
       [59, 58, 58, 58, 58],
       [61, 61, 60, 60, 59],
       [64, 63, 62, 61, 60]], dtype=uint8)
```

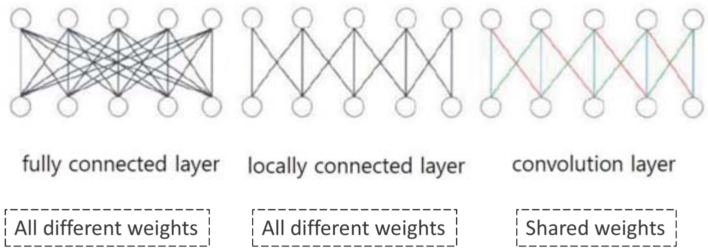

An Example of Handcrafted Features

e.g. Sobel 2-D filter





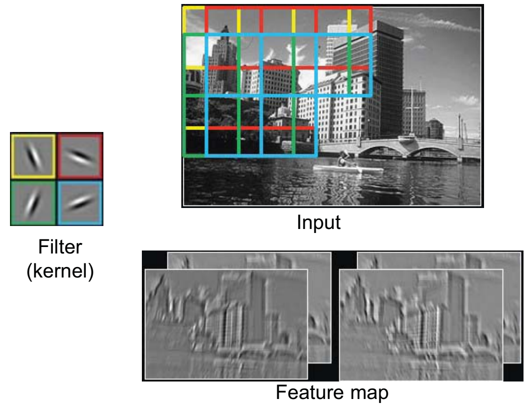
Connectivity & Weight Sharing Depends on Layer

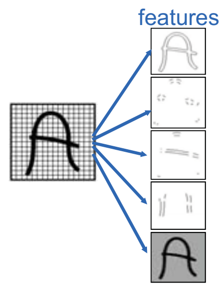


- **Convolution layer** has much smaller number of parameters by local connection and weight sharing

Convolution Layer

- Detect the same feature at different positions in the input image

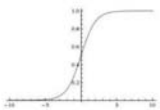




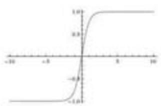
Activation Functions

Sigmoid

$\sigma(x) = 1/(1+e^{-x})$

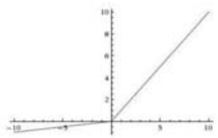


tanh tanh(x)



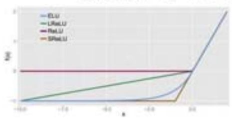
ReLU max(0,x)

Leaky ReLU

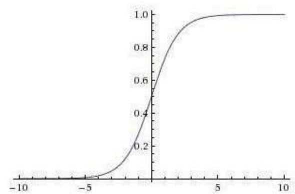


Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



Sigmoid Function



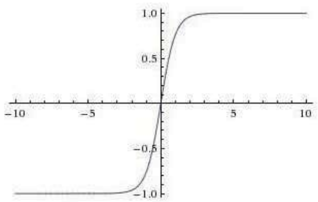
Sigmoid

$\sigma(x) = 1/(1+e^{-x})$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

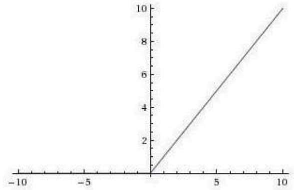
1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

tanh function



- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

ReLU Function



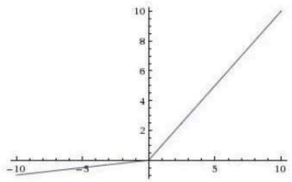
ReLU
(Rectified Linear Unit)

Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

- Not zero-centered output
- ReLU units can "die"

Leaky ReLU

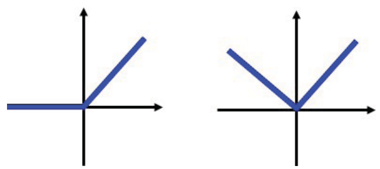


Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

Maxout



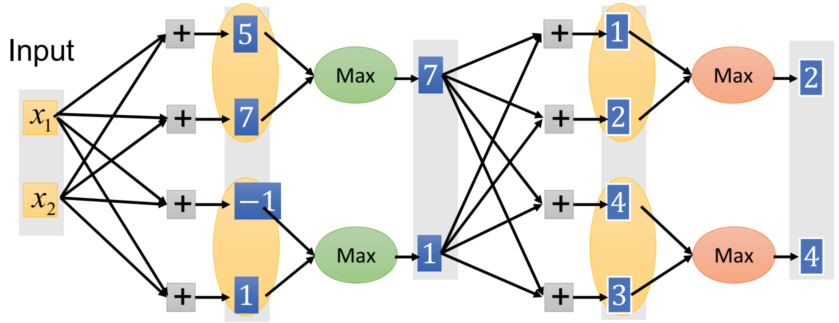
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

– Generalize ReLU and Leaky ReLU
– Does not saturates
– Does not die

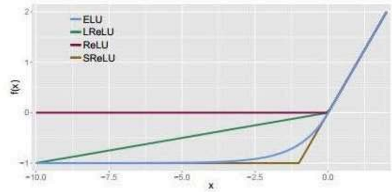
– doubles the number of parameters/neuron

Maxout



- You can have more than two elements in a group.

Exponential Linear Unit (ELU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

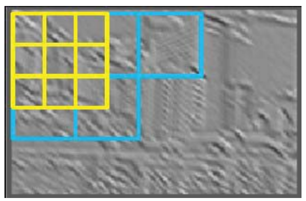
- All benefits of ReLU
- Does not die
- Closer to zero mean outputs

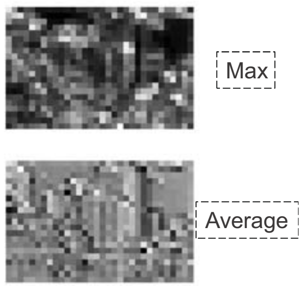
- Computation requires exp()

Sub-sampling (pooling) layer

➢ Spatial Pooling
- Average or Max

➢ Role of Pooling
- Invariance to small transformations
- reduce the effect of noises and shift or distortion





Average & Max Pooling

- Average pooling

1	4	1	6
2	1	0	9
1	2	7	1
4	1	0	0

2	4
2	2

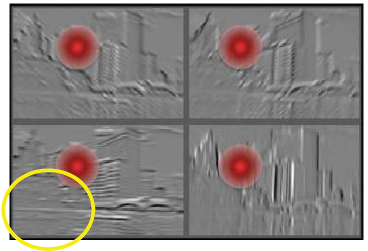
- Max pooling

1	4	1	6
2	1	0	9
1	2	7	1
4	1	0	0

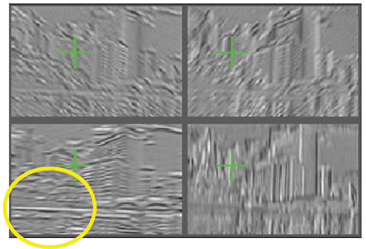
4	9
4	7

Normalization

- Contrast normalization (between/across feature map)
  - Equalizes the features map



Feature maps

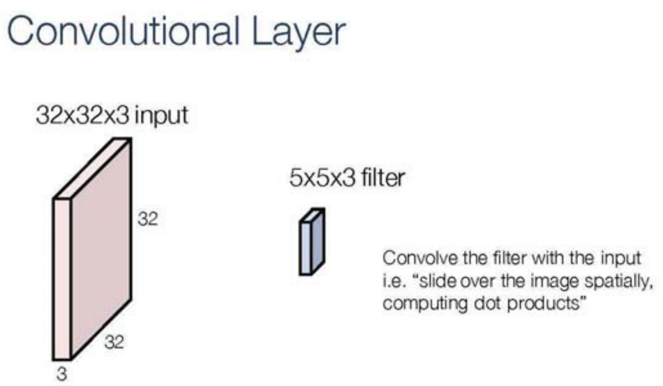


Feature maps
after contrast normalization

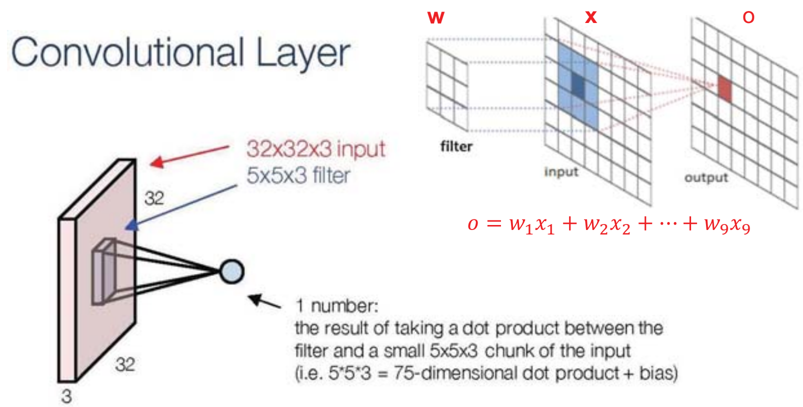
Convolutional Layer

- The core layer of CNNs
- The convolutional layer consists of a set of filters.
  - Each filter covers a spatially small portion of the input data.
- Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.
  - As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.
- Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.
- The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

Convolutional Layer



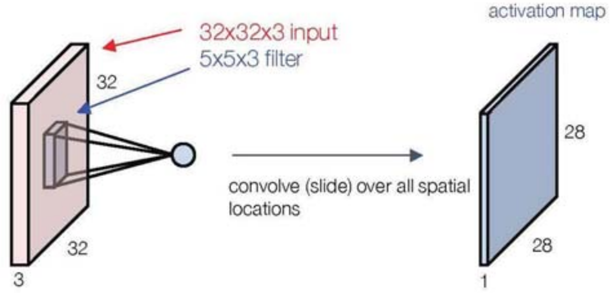
Convolutional Layer



$o = w_1x_1 + w_2x_2 + \cdots + w_9x_9$

Convolutional Layer

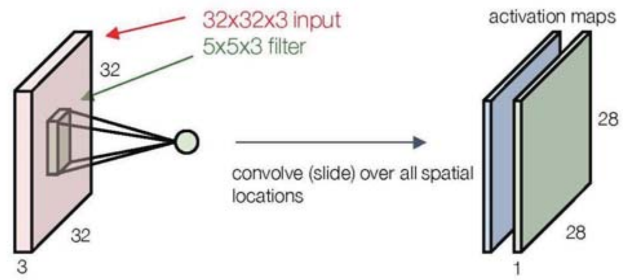
Convolutional Layer



Convolutional Layer

Convolutional Layer

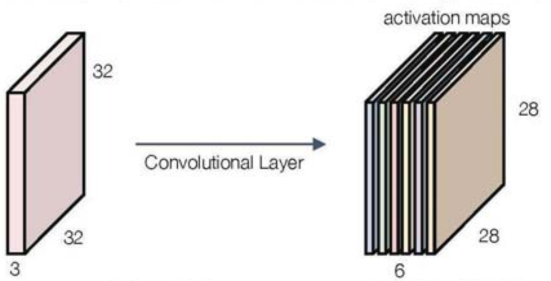
consider a second, green filter



Convolutional Layer

Convolutional Layer

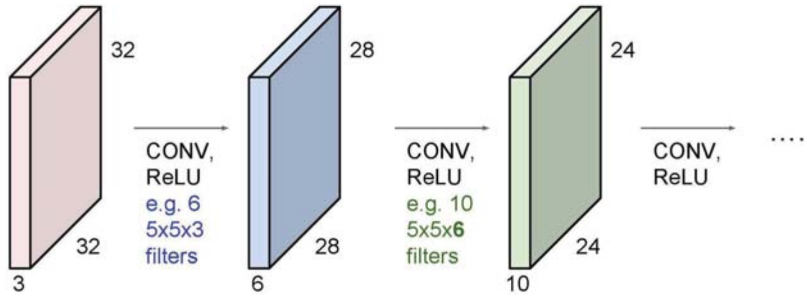
- Multiple filters produce multiple output channels
- For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get an output of size 28x28x6.

Convolutional Layer

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Average Pooling with No Padding

- no padding – input=3x3, filter=2x2, stride=1x1

1	2	3
4	5	6
7	8	9

3	

Average Pooling with No Padding

- no padding – input=3x3, filter=2x2, stride=1x1

1	2	3
4	5	6
7	8	9

3	4

Average Pooling with No Padding

- no padding – input=3x3, filter=2x2, stride=1x1

1	2	3
4	5	6
7	8	9

3	4
6	

Average Pooling with No Padding

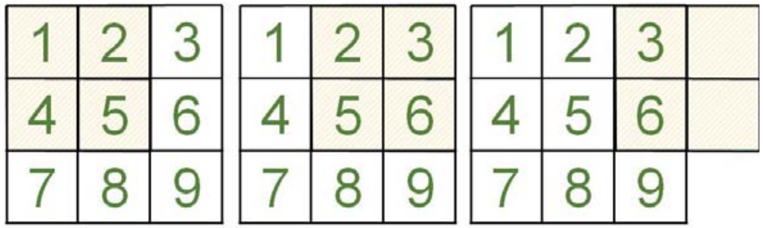
- no padding – input=3x3, filter=2x2, stride=1x1

1	2	3
4	5	6
7	8	9

3	4
6	7

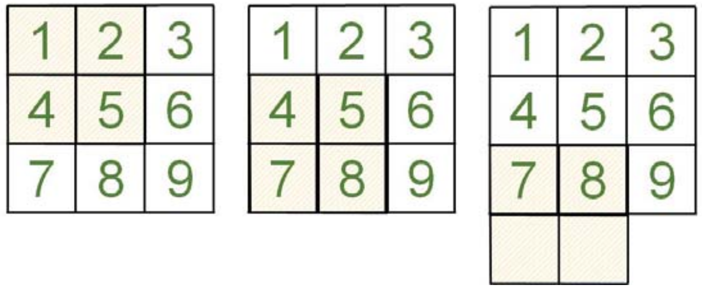
Average Pooling with Padding

- zero padding – input=3x3, filter=2x2, stride=1x1



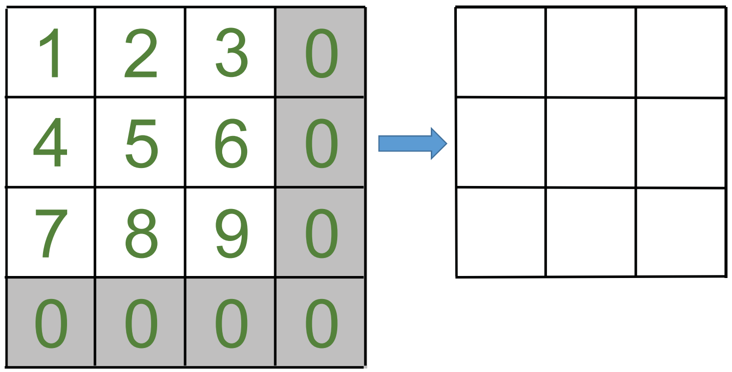
Average Pooling with Padding

- zero padding – input=3x3, filter=2x2, stride=1x1



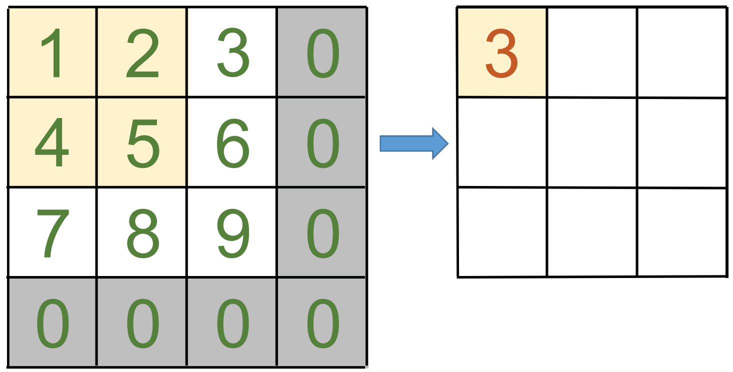
Average Pooling with Padding

- zero padding – input=3x3, filter=2x2, stride=1x1



Average Pooling with Padding

- Zero padding – input=3x3, filter=2x2, stride=1x1



Average Pooling with Padding

- zero padding – input=3x3, filter=2x2, stride=1x1

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

3	4	

Average Pooling with Padding

- zero padding – input=3x3, filter=2x2, stride=1x1

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

3	4	4.5

Average Pooling with Padding

- zero padding – input=3x3, filter=2x2, stride=1x1

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

3	4	4.5
6	7	7.5
7.5	8.5	9

Max Pooling with Padding

- zero padding – input=3x5, filter=3x3, stride=2x2

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

1	2	3	4	5		
6	7	8	9	10		
11	12	13	14	15		

Max Pooling with Padding

• zero padding – input=3x5, filter=3x3, stride=2x2

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

Max Pooling with Padding

• zero padding – input=3x5, filter=3x3, stride=2x2

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	0	0	0	0	0	0

→

Max Pooling with Padding

• zero padding – input=3x5, filter=3x3, stride=2x2

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	0	0	0	0	0	0

→

7		

Max Pooling with Padding

• zero padding – input=3x5, filter=3x3, stride=2x2

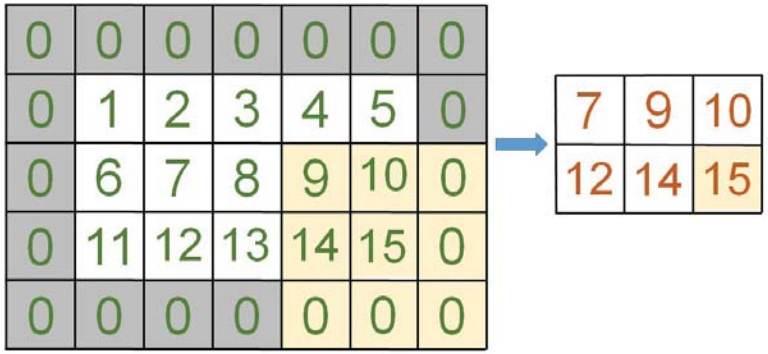
0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	0	0	0	0	0	0

→

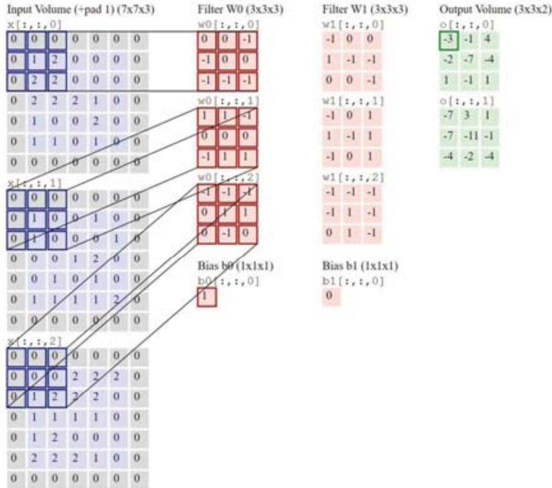
7	9	

Max Pooling with Padding

- zero padding – input=3x5, filter=3x3, stride=2x2



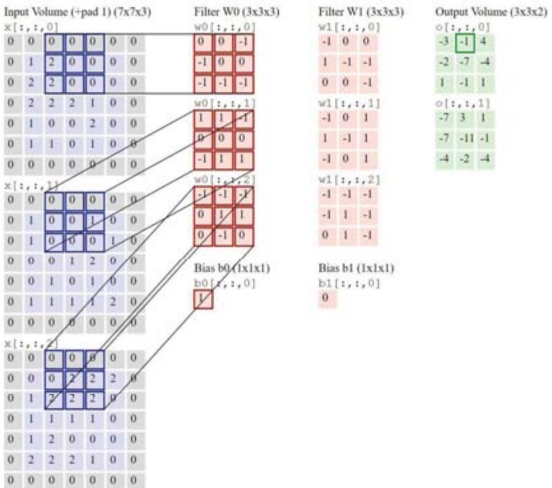
An Example of Conv Layer



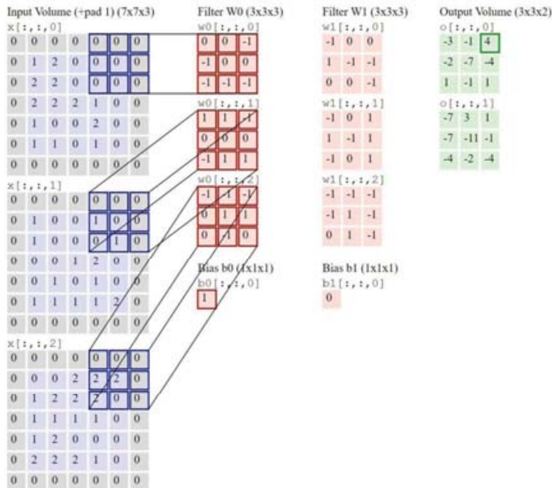
(0+0+0
+0+0+0
+0 -2 -2)+
(0+0+0
+0+0+0
+0+1+0)+
(0+0+0
+0+0+0
+0 -1+0)+
1

= -3

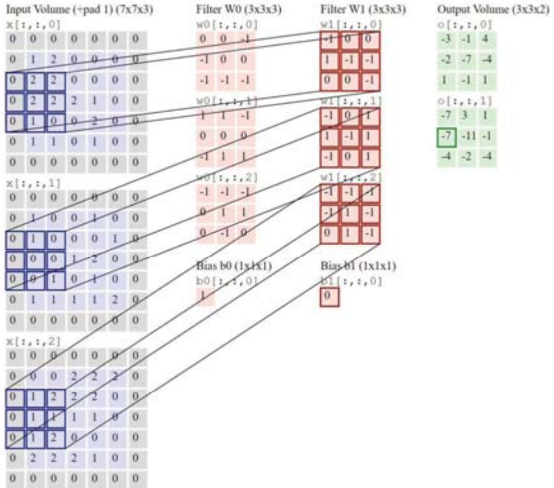
An Example of Conv Layer



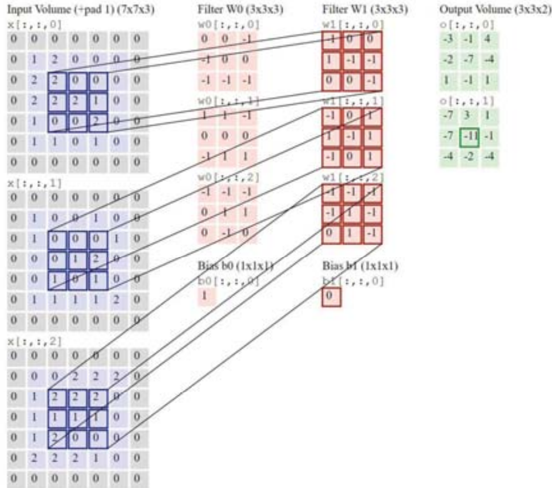
An Example of Conv Layer



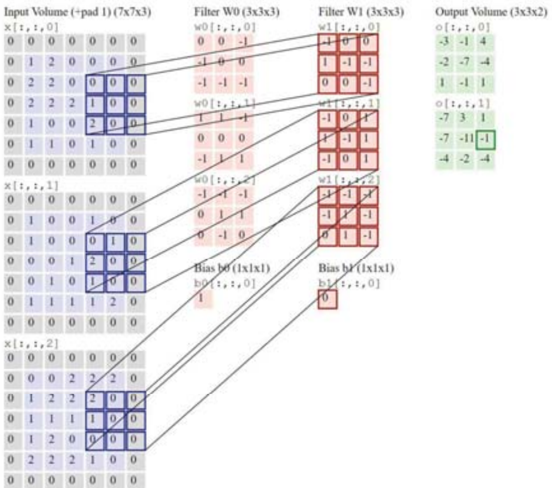
An Example of Conv Layer



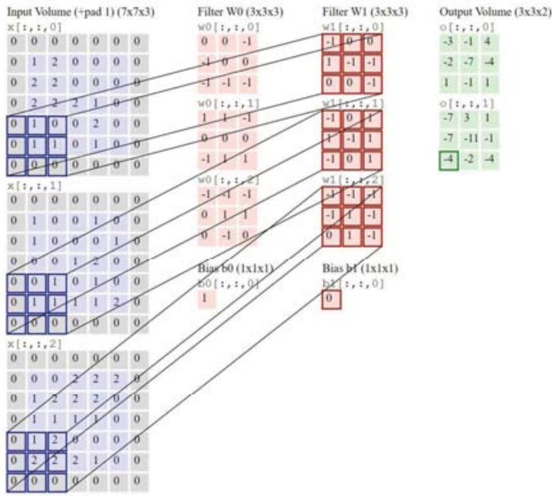
An Example of Conv Layer



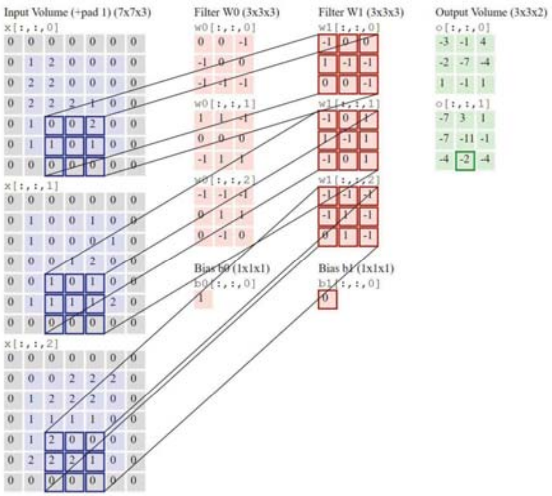
An Example of Conv Layer



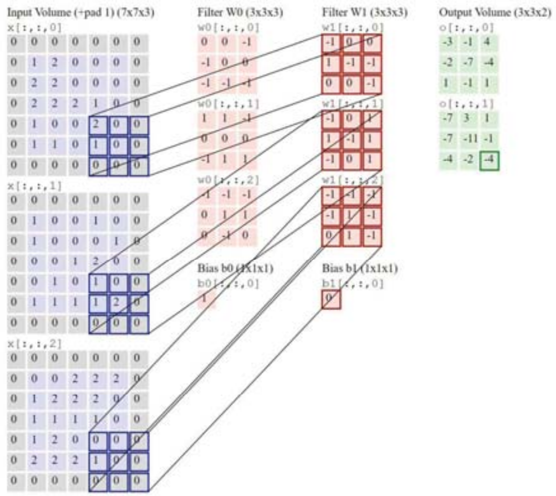
An Example of Conv Layer



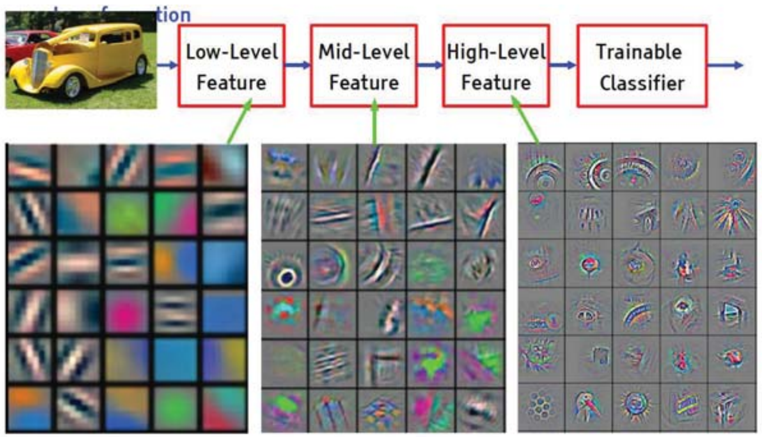
An Example of Conv Layer



An Example of Conv Layer

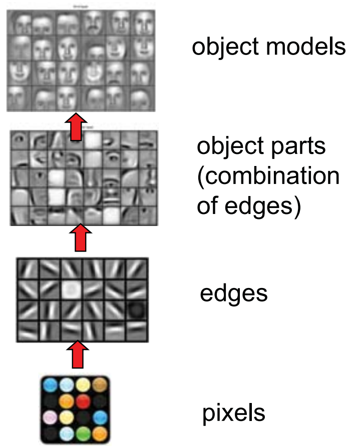


Learned Features

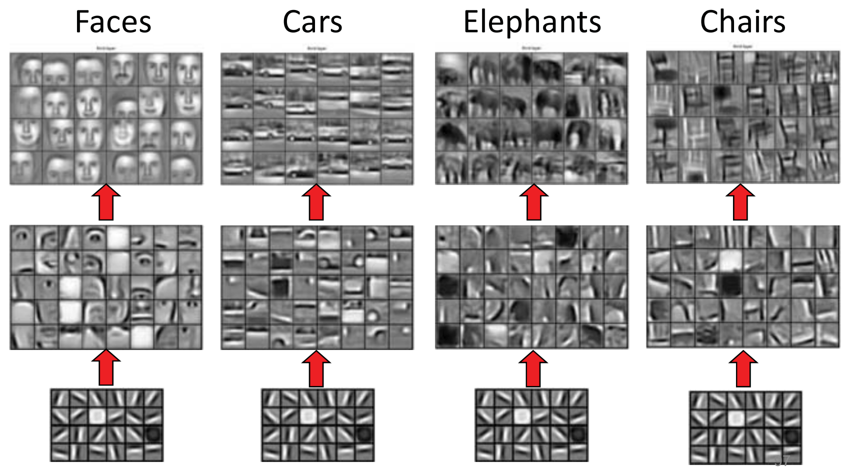


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Feature Hierarchies

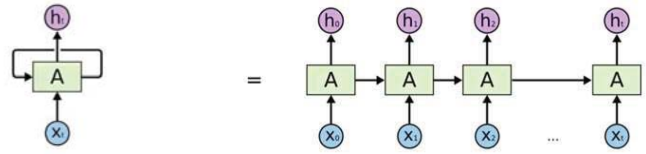


Examples of Learned Object Parts from Object Categories



Recurrent Neural Networks

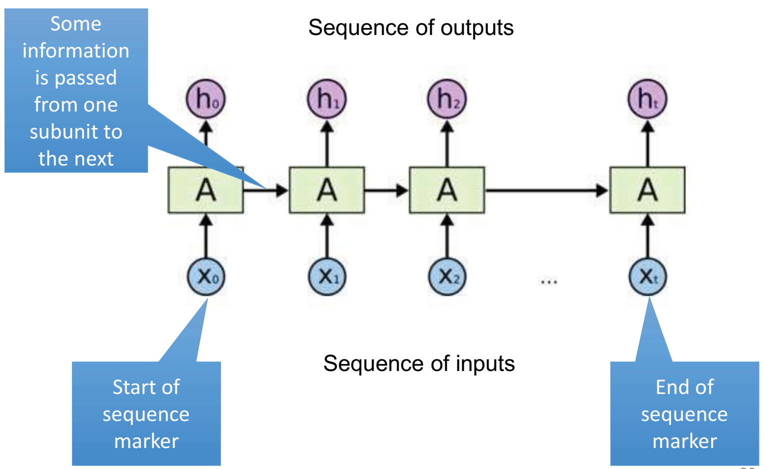
- Recurrent Neural Networks are networks with loops in them, allowing information to persist.



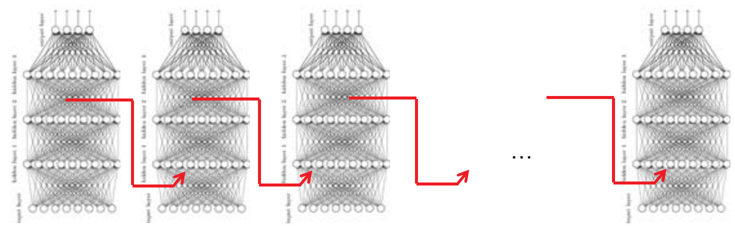
Recurrent Neural Networks have loops.

An unrolled recurrent neural network.

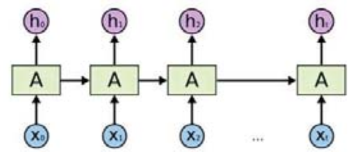
Recurrent Neural Networks



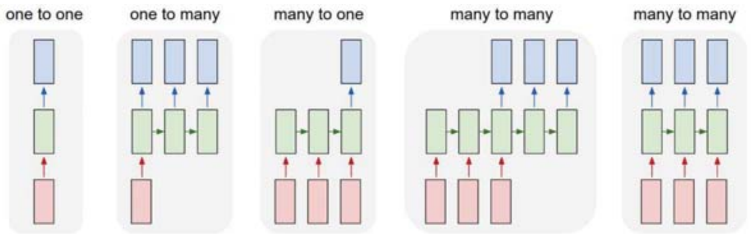
Architecture for an 1980's RNN



Problem with this: it's extremely deep and very hard to train



Examples of Recurrent Neural Networks



(1) Standard mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).

(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

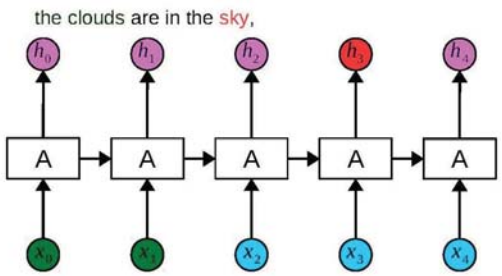
(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).

(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

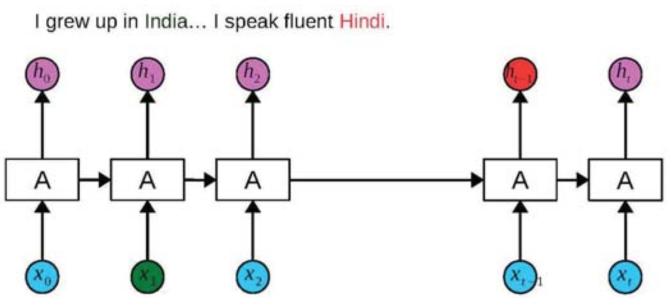
Long Short-Term Memory (LSTM)

- RNN short-term dependencies
- Language model trying to predict the next word based on the previous ones



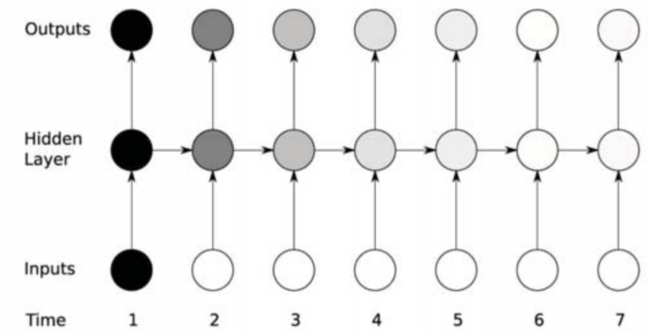
Long Short-Term Memory (LSTM)

- RNN long-term dependencies
- Language model trying to predict the next word based on the previous ones

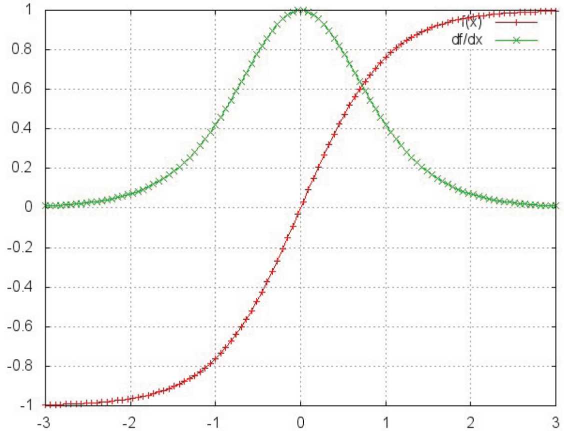


The Vanishing Gradient Problem

- The influence of a given input on the hidden layer, and thus on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections.

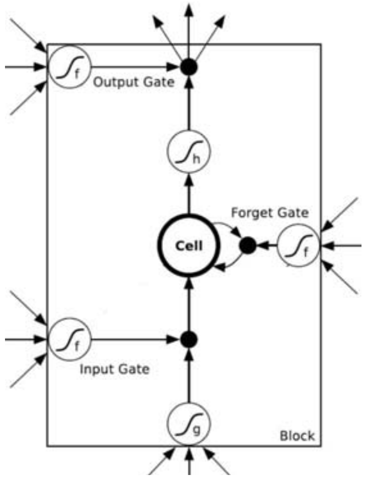


Vanishing Gradients

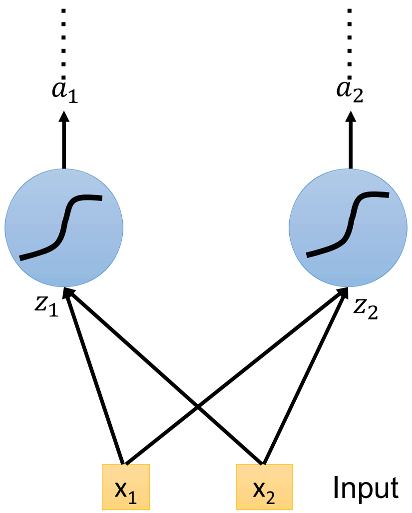


Long Short-Term Memory (LSTM)

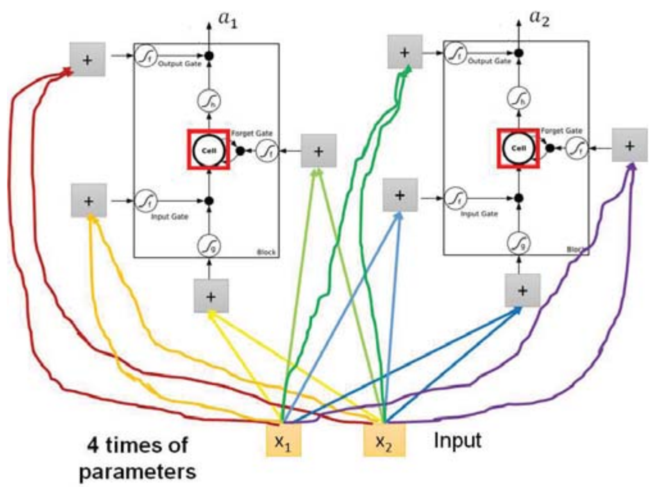
- Uses memory blocks
- A memory block contains
  - ➢ self-connected memory cell
  - ➢ 3 multiplicative units
    input, output, forget gates
- Gates allows LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem.



Simply Replace the Neurons with LSTM

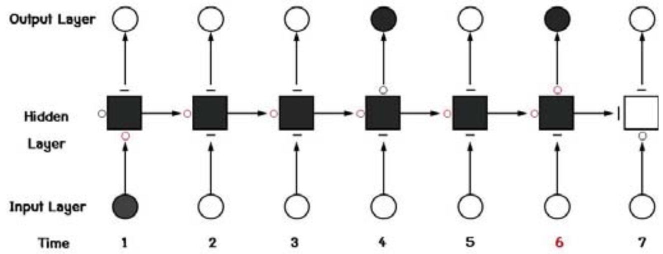


Simply Replace the Neurons with LSTM

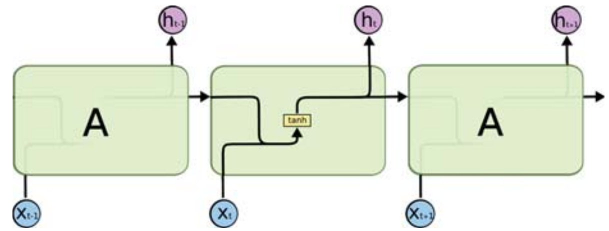


Preservation of Gradient Information

- The shading indicates their sensitivity to the inputs
- The memory cell *remembers* the first input as long as the forget gate is open and the input gate is closed.
- The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell



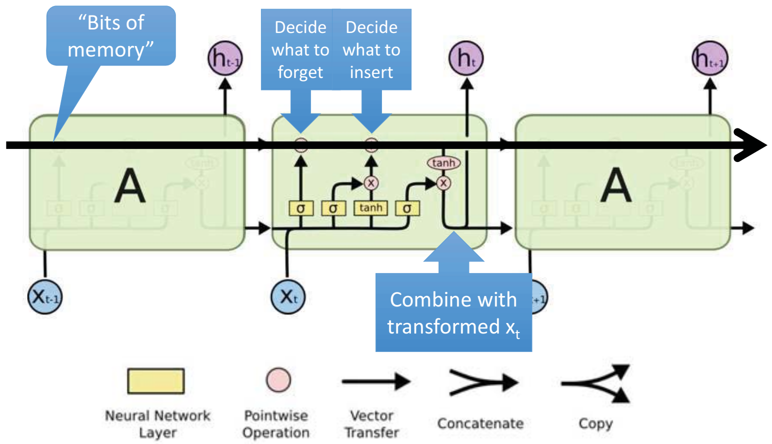
Standard RNN



- The repeating module in a standard RNN contains a single layer.
- The module will have a very simple structure

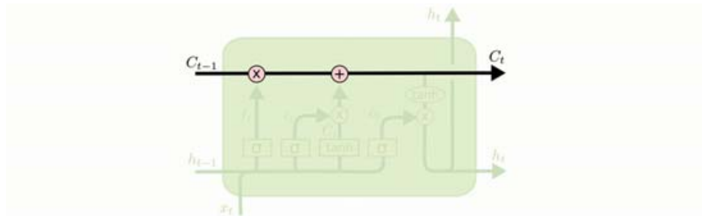
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Repeating Module in an LSTM



- The repeating module in an LSTM contains four interacting layers.

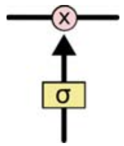
The Core Idea Behind LSTMs



- The horizontal line running through the top represents the cell state.
- The cell state runs straight down the entire chain, with only some minor linear interactions.
- It is very easy for information to just flow along it unchanged.

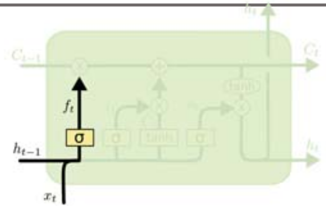
The Core Idea Behind LSTMs

- The ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates optionally let information through, composed of
  - ➢ a sigmoid neural net layer and
  - ➢ a pointwise multiplication operation.



- The sigmoid outputs numbers between zero and one, describing how much of each component should be let through.
  - zero means "let nothing through"
  - one means "let everything through"

Forget Gate

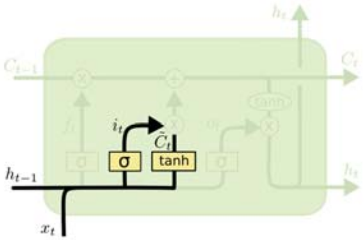


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

f_t : What part of memory to "forget" – zero means forget this bit

- First decide what information to throw away from state
- Using a sigmoid called the "forget gate"
- It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
  - ➢ 1 represents "completely keep this"
  - ➢ 0 represents "completely get rid of this."

Input Gate



i_t : What bits to insert into the next states

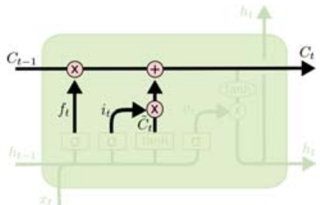
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$\tilde{C}_t$: What content to store into the next state

- Next, decide what new information to store in the state
  - ➢ a sigmoid called the "input gate" decides which values to update
  - ➢ a tanh creates a vector of new candidate values, $\tilde{C}_t$ that could be added to the state.
  - ➢ These two will be combined to create an update to the state.

Update the Cell State

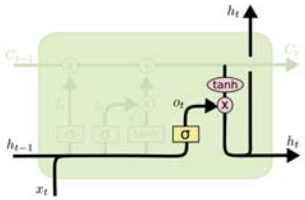


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

C_t : Next memory cell content – mixture of not-forgotten part of previous cell and insertion

- Update the old cell state, C_{t-1}, into the new cell state C_t, by
  - ➢ Multiplying C_{t-1} by f_t, forgetting the things we decided to forget earlier.
  - ➢ Then adding $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

Output Gate



o_t : What part of cell to output

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

$$h_t = o_t * \tanh\left(C_t\right)$$

$\tanh(C_t)$: maps to [-1,+1]

- Finally, decide what to output.
- The output is based on cell state.
  - First, run a sigmoid to decide what parts of the cell state to output.
  - Then, put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.