# 大数据分析

Scalable Machine Learning

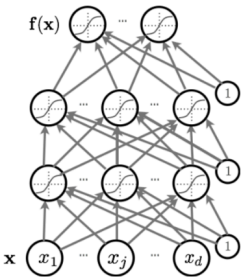
刘盛华


## Outline

- **Feedforward Neural Networks**
- **Deep learning in the past 50th years**
- **Some popular neural networks**

## Feedforward Neural Networks

- Definition of Neural Networks
  - Forward propagation
  - Types of units
  - Capacity of neural networks
- How to train neural nets:
  - Loss function
  - Backpropagation with gradient descent
- More recent techniques:
  - Dropout
  - Batch normalization
  - Unsupervised Pre-training



# Artificial Neuron

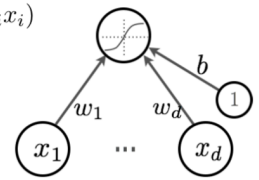
• Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

• Neuron output activation:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

where
- $\mathbf{W}$ are the weights (parameters)
- $b$ is the bias term
- $g(\cdot)$ is called the activation function

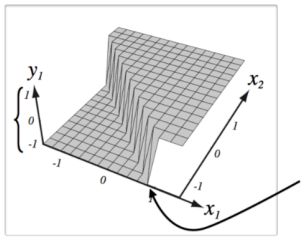



# Artificial Neuron

• Output activation of the neuron:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

Range is determined by $g(\cdot)$




(from Pascal Vincent's slides)

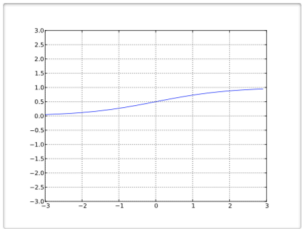
Bias only changes the position of the riff

# Activation Function

• Sigmoid activation function:

- Squashes the neuron's output between 0 and 1
- Always positive
- Bounded
- Strictly Increasing

$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

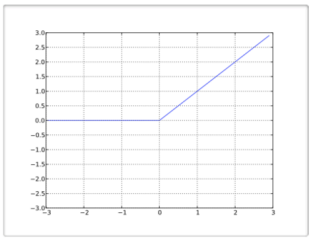


# Activation Function

• Rectified linear (ReLU) activation function:

- Bounded below by 0 (always non-negative)
- Tends to produce units with sparse activities
- Not upper bounded
- Strictly increasing

$$g(a) = \text{reclin}(a) = \max(0, a)$$



倾向于产生稀疏活动的单元

# Single Hidden Layer Neural Net

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

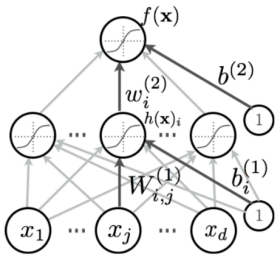
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

Output activation function




# Multilayer Neural Net

- Consider a network with L hidden layers.
  - layer pre-activation for k>0

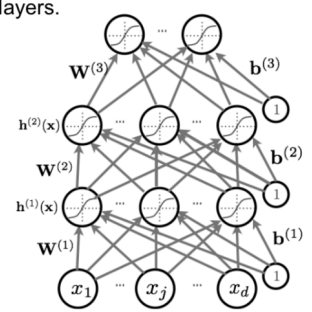
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation (k=L+1):

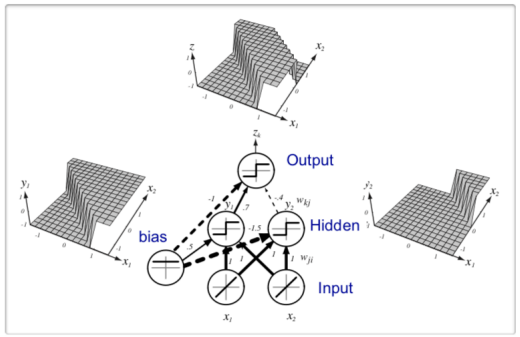
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$




$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

# Capacity of Neural Nets

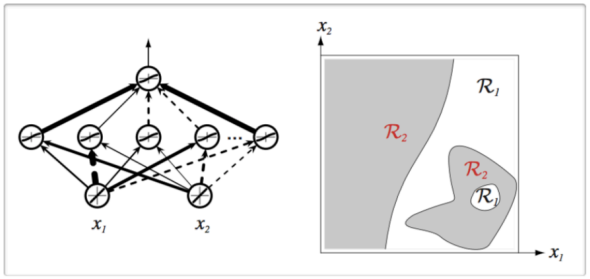
- Consider a single layer neural network




(from Pascal Vincent's slides)

# Capacity of Neural Nets

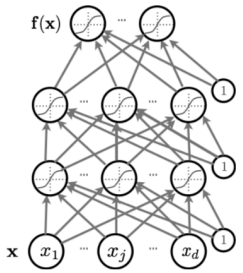
- Consider a single layer neural network




(from Pascal Vincent's slides)

# Feedforward Neural Networks

- How neural networks predict f(x) given an input x:
  - Forward propagation
  - Types of units
  - Capacity of neural networks
- How to train neural nets:
  - Loss function
  - Backpropagation with gradient descent
- More recent techniques:
  - Dropout
  - Batch normalization
  - Unsupervised Pre-training




# Training

- Empirical Risk Minimization:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T}\sum_t \underbrace{l(f(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)})}_{\text{Loss function}} + \underbrace{\lambda\Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

- To train a neural net, we need:
  - Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)})$
  - A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$ , $\nabla_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta})$

# Stochastic Gradient Descend

- Perform updates after seeing each example:
  - Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)},\mathbf{b}^{(1)},\ldots,\mathbf{W}^{(L+1)},\mathbf{b}^{(L+1)}\}$
  - For t=1:T
    - for each training example $(\mathbf{x}^{(t)},y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)}) - \lambda\nabla_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\,\Delta$$

Training epoch = Iteration of all examples

- To train a neural net, we need:
  - Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)})$
  - A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$ , $\nabla_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta})$

# Weight Decay 权重衰减

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T}\sum_t l(f(\mathbf{x}^{(t)};\boldsymbol{\theta}),y^{(t)}) + \lambda\Omega(\boldsymbol{\theta})$$

- L2 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k\sum_i\sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

- L1 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k\sum_i\sum_j |W_{i,j}^{(k)}|$$

- Only applies to weights, not biases (weigh decay)

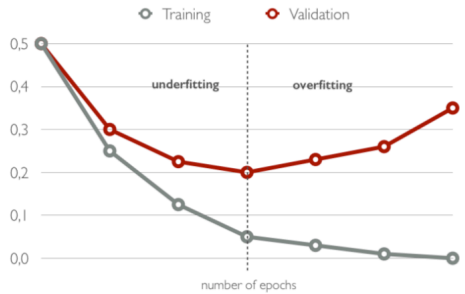
# Model Selection

- Training Protocol:
  - Train your model on the Training Set $\mathcal{D}^{\text{train}}$
  - For model selection, use Validation Set $\mathcal{D}^{\text{valid}}$
    - Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
  - Estimate generalization performance using the Test Set $\mathcal{D}^{\text{test}}$
- Generalization is the behavior of the model on **unseen examples**.

泛化

# Early Stopping

- To select the number of epochs, stop training when validation set error increases (with some look ahead).




# Mini-batch, Momentum

- Make updates based on a mini-batch of examples (instead of a single example):
  - the gradient is the average regularized loss for that mini-batch
  - can give a more accurate estimate of the gradient
  - can leverage matrix/matrix operations, which are more efficient
- Momentum: Can use an exponential average of previous gradients:

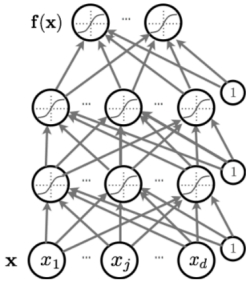
$$\overline{\nabla}_{\boldsymbol{\theta}}^{(t)} = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\boldsymbol{\theta}}^{(t-1)}$$

  - can get pass plateaus more quickly, by "gaining momentum"

通过增加动量，可以快速越过高原

# Feedforward Neural Networks

- How neural networks predict f(x) given an input x:
  - Forward propagation
  - Types of units
  - Capacity of neural networks
- How to train neural nets:
  - Loss function
  - Backpropagation with gradient descent
- More recent techniques:
  - Dropout
  - Batch normalization

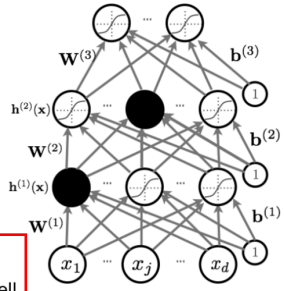


# Best Practice

- Given a dataset D, pick a model so that:
  - You can achieve 0 training error– Overfit on the training set
- Regularize the model (e.g. using Dropout).
- SGD with momentum, batch-normalization, and dropout usually works very well.

# Dropout

- Key idea: Cripple neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful
- Could use a different dropout probability, but 0.5 usually works well




(Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, JMLR 2014)

# Dropout

- Use random binary masks $m^{(k)}$
  - layer pre-activation for k>0

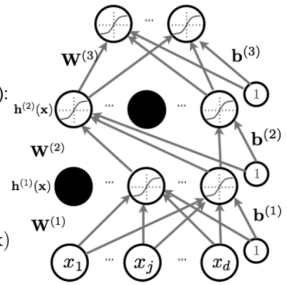
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k=1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x})) \odot \mathbf{m}^{(k)}$$

  - Output activation (k=L+1)

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$




(Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, JMLR 2014)

# Dropout at Test Time

- At test time, we replace the masks by their expectation 测试时使用mask的期望值
  - This is simply the constant vector 0.5 if dropout probability is 0.5
  - For single hidden layer: equivalent to taking the geometric average of all neural networks, with all possible binary masks
- Can be combined with unsupervised pre-training
- Beats regular backpropagation on many datasets
- Ensemble: Can be viewed as a geometric average of exponential number of networks.

整合：可以看作指数个网络的几何平均

# Batch Normalization

加速训练

- Normalizing the inputs will speed up training (Lecun et al. 1998)
  - could normalization be useful at the level of the hidden layers?
- Batch normalization is an attempt to do that (Ioffe and Szegedy, 2015)
  - each unit's pre-activation is normalized (mean subtraction, stddev division)
  - during training, mean and stddev is computed for each minibatch
  - backpropagation takes into account the normalization
  - at test time, the global mean / stddev is used

  - (Ioffe and Szegedy, ICML 2015)

# Batch Normalization

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Learned linear transformation to adapt to non-linear activation function ($\gamma$ and $\beta$ are trained)

(Ioffe and Szegedy, ICML 2015)

# Batch Normalization

- Why normalize the pre-activation?
  - can help keep the pre-activation in a non-saturating regime (though the linear transform $y_i \leftarrow \gamma \hat{x}_i + \beta$ could cancel this effect)

- Use the global mean and stddev at test time.
  - removes the stochasticity of the mean and stddev 去除随机性
  - requires a final phase where, from the first to the last hidden layer
    - propagate all training data to that layer
    - compute and store the global mean and stddev of each unit
  - for early stopping, could use a running average

(Ioffe and Szegedy, ICML 2015)

# Outline

- **Deep Neural Networks**
- **Deep learning in the past 50th years**

## What have happened in the past 50th years ?

1957 Perceptron
1975 BP
1980 RNN
1990 CNN
1997 LSTM RNN
2012 CNN

1963 Linear SVM
1992 Kernel SVM
2006 DNN
2013 RNN

Born | Milestone | Ice Age | Renaissance

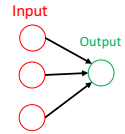
## What have happened in the past 50th years ?

1957 Perceptron
1975 BP
1980 RNN
1990 CNN
1997 LSTM RNN
2012 CNN

1963 Linear SVM
1992 Kernel SVM
2006 DNN
2013 RNN

Born with drawbacks.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$




Perceptron

Marvin Minsky's doubt:
1. Basic perceptrons were incapable of processing the exclusive-or circuit.
2. Computers didn't have enough processing power to effectively handle the long run time required by large neural networks.

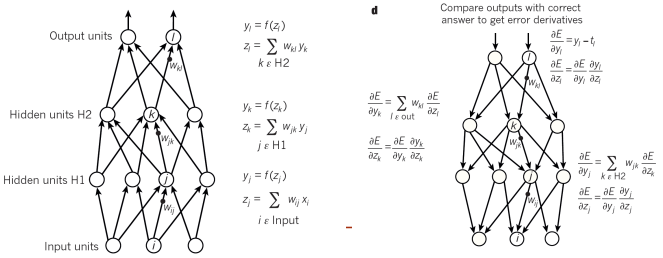
~~基本的感知器无法处理异或电路。~~
2. 计算机没有足够的处理能力来有效地处理大型神经网络所需的长时间运行

## What have happened in the past 50th years ?

1957 Perceptron
1975 BP
1980 RNN
1990 CNN
1997 LSTM RNN
2012 CNN

1963 Linear SVM
1992 Kernel SVM
2006 DNN
2013 RNN

Backpropagation: make multi-layer neural network can be really useful.

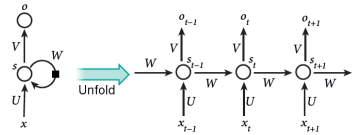



## What have happened in the past 50th years ?

1957 Perceptron
1975 BP
1980 RNN
1990 CNN
1997 LSTM RNN
2012 CNN

1963 Linear SVM
1992 Kernel SVM
2006 DNN
2013 RNN

Data is coming in sequence.
Recurrent: Set hidden units that implicitly contains history information.



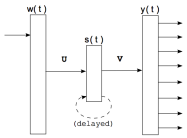

$x$: input $o$: output $s$: hidden state $U, V, W$: weights

## What have happened in the past 50th years ?

1957 Perceptron · 1975 BP · 1980 RNN · 1990 CNN · 1997 LSTM RNN · 2012 CNN

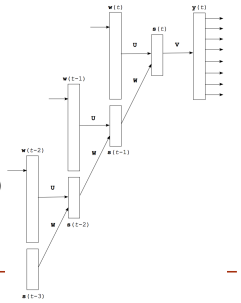
1963 Linear SVM · 1992 Kernel SVM · 2006 DNN · 2013 RNN

The recurrent net is just a layered net that keeps reusing the same weights. ——Hinton




$$s(t) = f(Uw(t) + Ws(t-1) + b)$$

每一步是一个简单的非线性变换

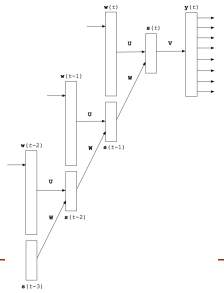


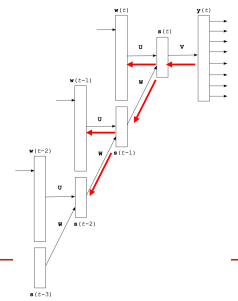
## What have happened in the past 50th years ?

1957 Perceptron · 1975 BP · 1980 RNN · 1990 CNN · 1997 LSTM RNN · 2012 CNN

1963 Linear SVM · 1992 Kernel SVM · 2006 DNN · 2013 RNN

BP gives the possibility to learn the recurrent structure ! BP给出了学习循环结构的可能性!





loss

## What have happened in the past 50th years ?

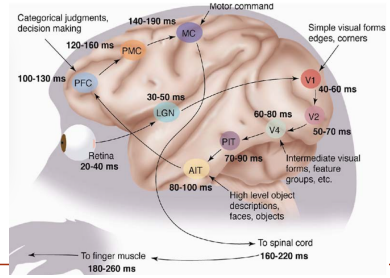
1957 Perceptron · 1975 BP · 1980 RNN · 1990 CNN · 1997 LSTM RNN · 2012 CNN

1963 Linear SVM · 1992 Kernel SVM · 2006 DNN · 2013 RNN

Basic idea of CNN:

- Local connection
- Shared weights
- Pooling
- The use of many layers

局部连接
分享权重
池化
多层



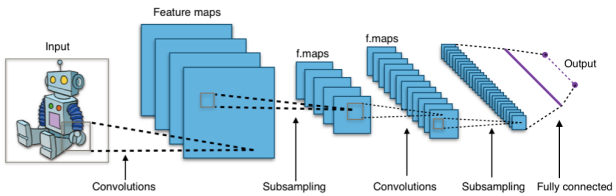

Inspired by visual neuroscience

## What have happened in the past 50th years ?

1957 Perceptron · 1975 BP · 1980 RNN · 1990 CNN · 1997 LSTM RNN · 2012 CNN

1963 Linear SVM · 1992 Kernel SVM · 2006 DNN · 2013 RNN

**Each f.map with one identical filter bank**

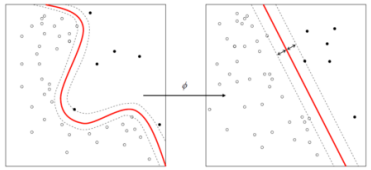



**Local connection with filter bank to detect local conjunction**

**Pooling to merge semantically similar features into one**

## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN



$$k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{V}}.$$

Kernel trick brings nonlinear
But mystery nonlinear…

$$\text{minimize } \frac{1}{n}\sum_{i=1}^{n}\zeta_i + \lambda\|w\|^2$$

$$\text{subject to } y_i(x_i \cdot w + b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0, \text{ for all } i.$$

Primal

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

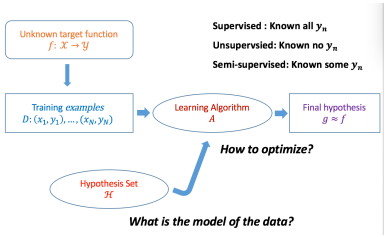
Dual

## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN

Dark days for neural networks.
Yes, we have a powerful model;
but, we do not have a useful learning algorithm !

- BP do not works well for deep architecture
- Too many computation costs

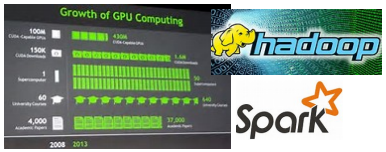



## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN

Here comes renaissance.


Big data



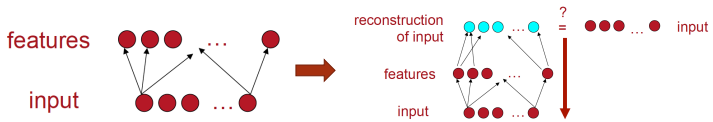

Computing Power

***Enough training dataset*** ***Complex optimization could be possible***

## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CI · 1997 · 2006 DNN · 2012 CNN · 2013 RNN

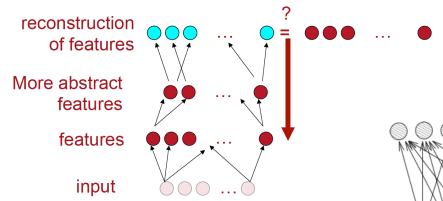

***Renaissance of Deep Neural Network***
深度神经网络的复兴
Unsupervised pre-training + fine tune



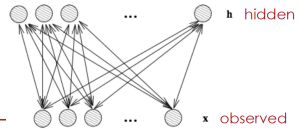

Construct a layer by supervised method. (Auto-encoder here)

## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN



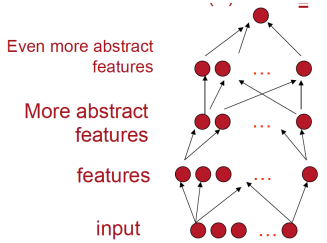

Construction methods:
- Auto-encoder
- RBM




Layer-wise construction

## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN



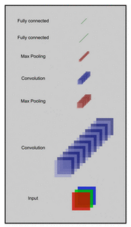

***Backpropagation***

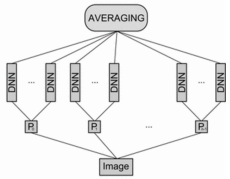
Breakthrough in speech recognition tasks !

Fine-tune

## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN






(a) DNN structure (b) P: preprocessing layer

Breakthrough in MNIST task.

| Method | Paper | Error rate[%] |
|---|---|---|
| CNN | [32] | 0.40 |
| CNN | [26] | 0.39 |
| MLP | [5] | 0.35 |
| CNN committee | [6] | 0.27 |
| MCDNN | this | **0.23** |

Why works?
- Deep
- ReLU units
- Tricks in optimization

[ CVPR 2012 ]

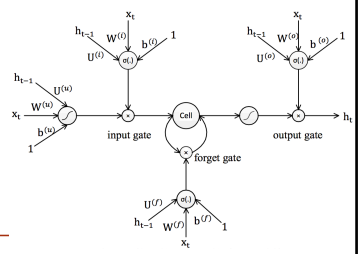
## What have happened in the past 50th years ?

1957 Perceptron · 1963 Linear SVM · 1975 BP · 1980 RNN · 1990 CNN · 1992 Kernel SVM · 1997 LSTM RNN · 2006 DNN · 2012 CNN · 2013 RNN

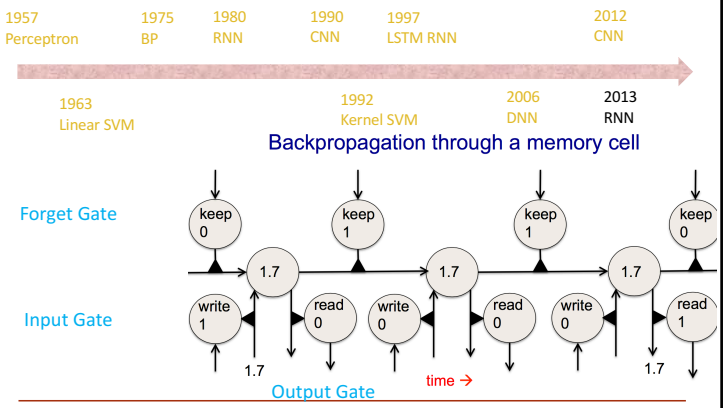
Drawback of RNN: can not hold long-term memory

| 思想 | 通过gate控制信息的读取和写入 |
|---|---|
| Input gate | 当前信息有意义的时候写入 |
| Forget gate | 过去信息没有意义的时候遗忘 |
| Output gate | 需要信息的时候读取 |
| 如何决定gate | 通过当前输入和历史信息决定 |

Breakthrough in NLP area !



## What have happened in the past 50th years ?



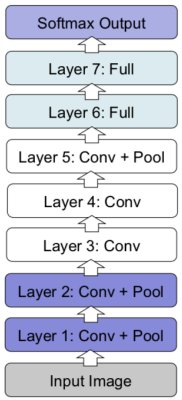

通过gate控制，神经元中1.7这个值被保存了多步

## Outline

- Feedforward Neural Networks
- Deep learning in the past 50th years
- Some popular neural networks

## AlexNet

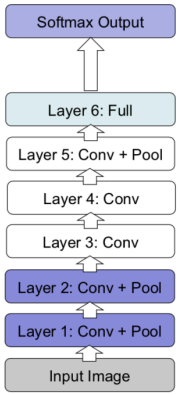
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error




[From Rob Fergus' CIFAR 2016 tutorial]

## AlexNet

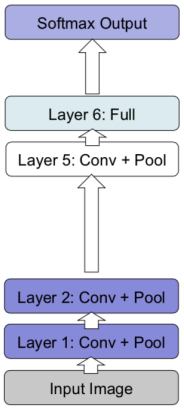
- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!




[From Rob Fergus' CIFAR 2016 tutorial]

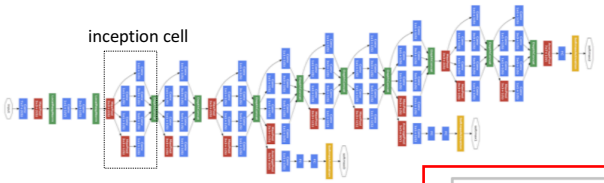
## AlexNet

- Let us remove upper feature extractor layers and fully connected:
  - Layers 3,4, 6 and 7
- Drop ~50 million parameters
- **33.5 drop in performance!**
- Depth of the network is the key.

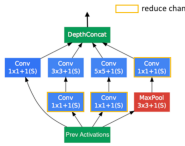



[From Rob Fergus' CIFAR 2016 tutorial]

## GoogLeNet



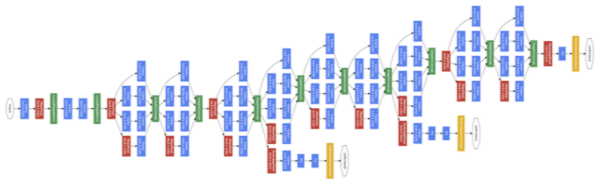

- 24 layer model that uses so-called inception module.




Convolution
Pooling
Softmax
Other

(Szegedy et al., Going Deep with Convolutions, 2014)

## GoogLeNet

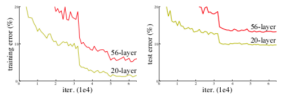


- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

(Szegedy et al., Going Deep with Convolutions, 2014)

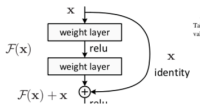
## Residual Networks

Really, really deep convnets do not train well, E.g. CIFAR10:




Key idea: introduce "pass through" into each layer

Thus only residual now needs to be learned




| method | top-1 err. | top-5 err. |
|---|---|---|
| VGG [41] (ILSVRC'14) | - | 8.43[†] |
| 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).

With ensembling, 3.57% top-5 test error on ImageNet

(He, Zhang, Ren, Sun, CVPR 2016)

## Future?

- DL has big break through in visual, speech, NLP areas

那么关系数据呢

- How about the relational data ?
  - Network Embedding
  - Recommendation
  - Heterogeneous network matching

网络嵌入
推荐
异构网络匹配

# Questions?