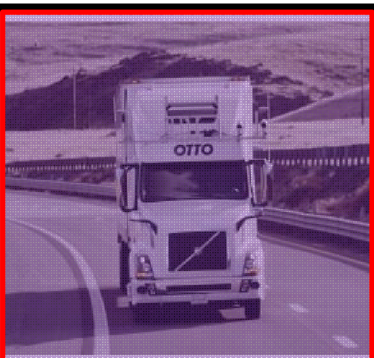
Lecture 1
Basics for Machine Learning and A Special Emphasis on CNN

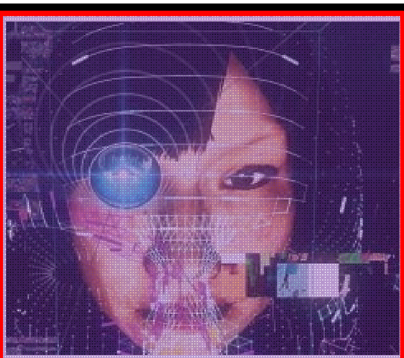
Lin ZHANG, PhD
School of Software Engineering
Tongji University
Fall 2017

10 Breakthrough Technologies 2017 (MIT Tech Review)

Reversing Paralysis



Self Driving


Paying with Your Face

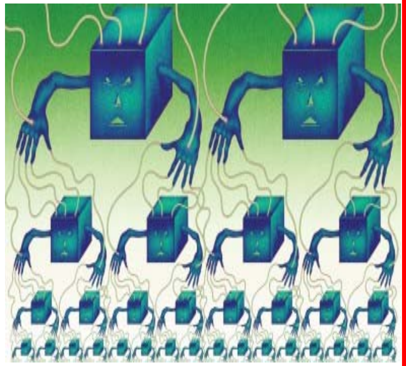
Practical Quantum Computers

The 360-Degree Selfie

Hot Solar Cells

Gene Therapy 2.0

The Cell Atlas


Botnets of Things

Reinforcement Learning

The core is machine learning

傍晚，小街路面上沁出微雨后的湿润，和煦的细风吹来，抬头看看天边的晚霞，嗯，明天又是一个好天气。走到水果摊旁，挑了个根蒂蜷缩、敲起来声音浊响的青绿西瓜，一边满心期待着皮薄肉厚瓤甜的爽落感，一边愉快地想着：这学期狠下了功夫，基础概念弄得清清楚楚，算法作业也是信手拈来，这门课成绩一定差不了！

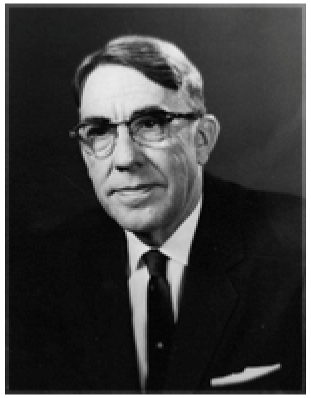
摘自《机器学习》（周志华著，2016）

Outline

- Basic concepts
- Linear model
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- DCNN for object detection

What is machine learning?

- Gives "computers the ability to learn without being explicitly programmed” (Arthur Samuel in 1959)



Arthur Lee Samuel
(December 5, 1901 – July 29, 1990)

- It explores the study and construction of algorithms that can learn from and make predictions on data
- It is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or unfeasible

[1] Samuel, Arthur L., Some Studies in Machine Learning Using the Game of Checkers, IBM Journal of Research and Development, 1959

Supervised VS Unsupervised

- Supervised learning
  - It will infer a function from labeled training data
  - The training data consists of a set of training examples
  - Each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal)
- Unsupervised learning
  - Trying to find hidden structure in unlabeled data
  - Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution
  - Such as PCA, K-means (a clustering algorithm)

About sample

- Attribute (feature), attribute value, label, and example

色泽，根蒂，敲声 $\longrightarrow$ {好瓜，坏瓜}

features labels

{青绿，蜷缩，浊响：好瓜}

feature values label value

one example

Training, testing, and validation

- Training sample and training set

A training set comprising m training samples,

$$D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$$

where $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in \chi$ is the feature vector of ith sample and $y_i \in \zeta$ is its label

By training, our aim is to find a mapping,

$$f : \chi \mapsto \zeta$$

based on D

If ζ comprises discrete values, such a prediction task is called "**classification**"; if it comprises real numbers, such a prediction task is called "**regression**"

Training, testing, and validation

- Training sample and training set
- Test set
  - A test set is a set of data that is independent of the training data, but that follows the same probability distribution as the training data
  - Used only to assess the performance of a fully specified classifier

Training, testing, and validation

- Training sample and training set
- Test set
- Validation set
  - In order to avoid overfitting, when any classification parameter needs to be adjusted, it is necessary to have a validation set; it is used for model selection
  - The training set is used to train the candidate algorithms, while the validation set is used to compare their performances and decide which one to take

Overfitting, Generalization, and Capacity

- Overfitting
  - It occurs when a statistical model describes random error or noise instead of the underlying relationship
  - It generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations
  - A model that has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data

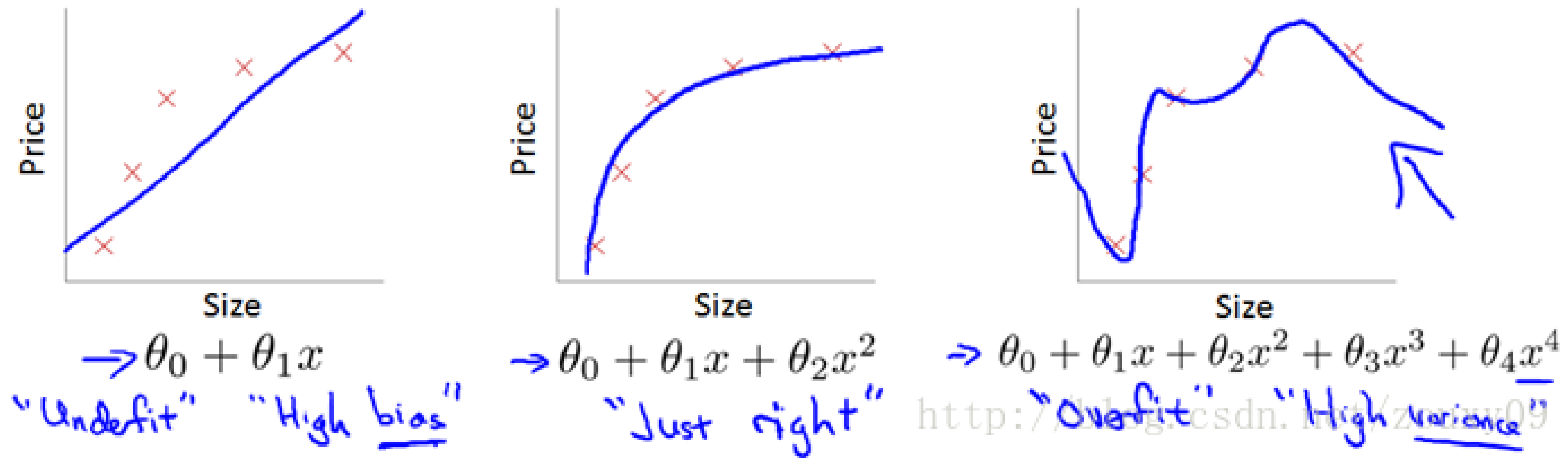
Overfitting, Generalization, and Capacity

- Overfitting
- Generalization
  - Refers to the performance of the learned model on new, previously unseen examples, such as the test set

Overfitting, Generalization, and Capacity

- Overfitting
- Generalization

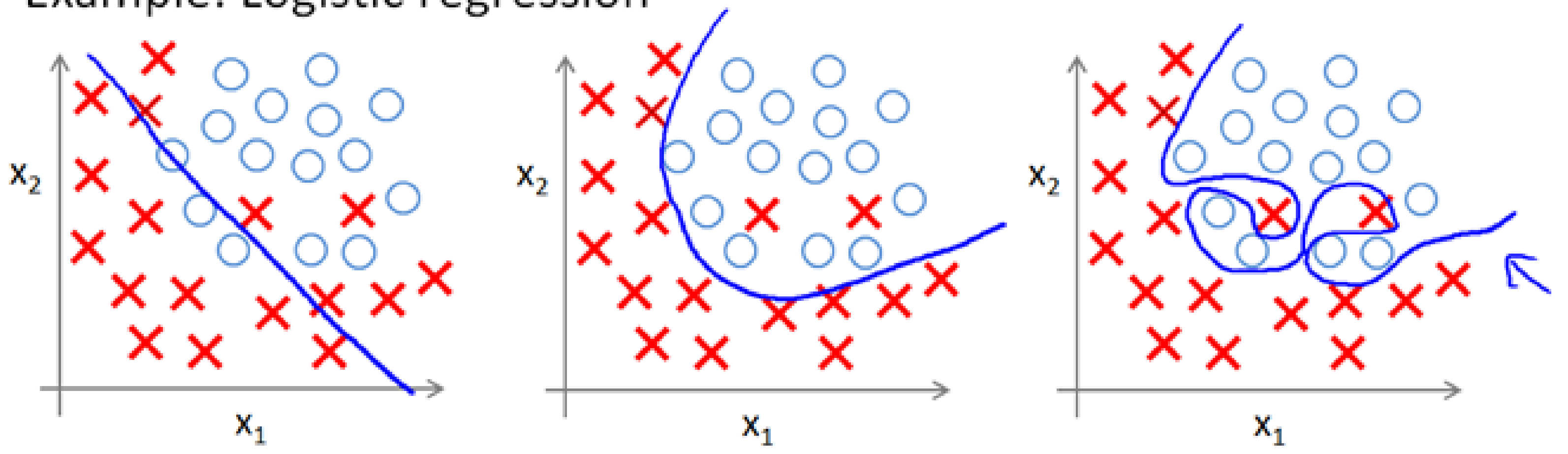
Example: Linear regression (housing prices)



Overfitting, Generalization, and Capacity

- Overfitting
- Generalization

Example: Logistic regression



$h_\theta(x) = g(\theta_0 + \underline{\theta_1 x_1 + \theta_2 x_2})$

(g = sigmoid function)

"Underfit"

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 \underline{x_1^2} + \theta_4 \underline{x_2^2} + \theta_5 \underline{x_1 x_2})$$

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 \underline{x_1^2 x_2} + \theta_4 \underline{x_1^2 x_2^2} + \theta_5 \underline{x_1^2 x_2^3} + \theta_6 \underline{x_1^3 x_2} + \ldots)$$

"Overfit"

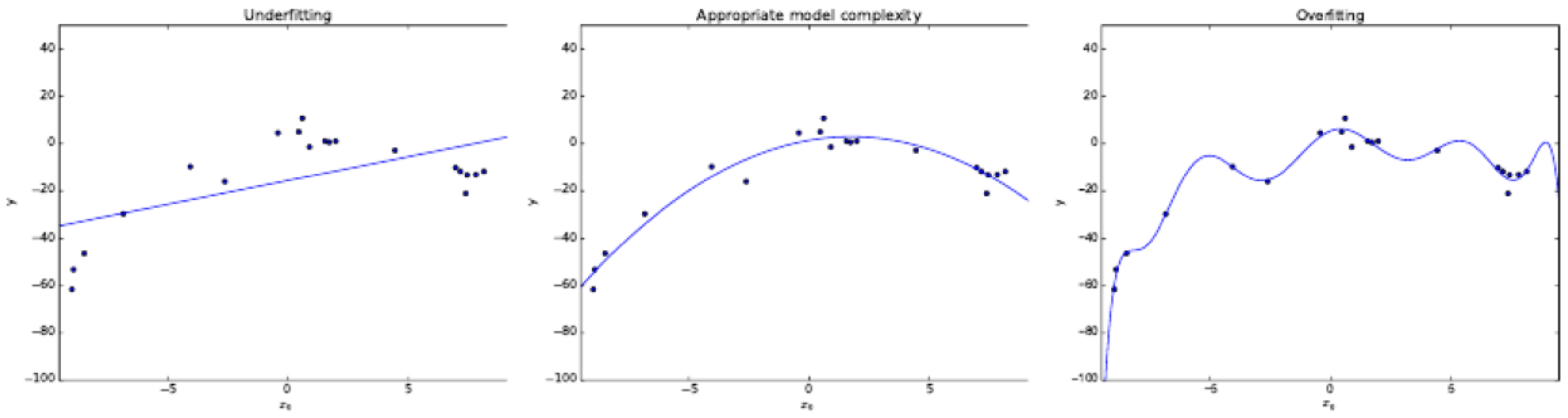
Overfitting, Generalization, and Capacity

- Overfitting
- Generalization
- Capacity
  - Measures the complexity, expressive power, richness, or flexibility of a classification algorithm
  - Ex, DCNN (deep convolutional neural networks) is powerful since its capacity is very large

$$y^* = b + \omega x, \quad y^* = b + \omega_1 x_1 + \omega_2 x_2, \quad y^* = b + \sum_{i=1}^{10} \omega_i x_i$$

$\longrightarrow$ higher capacity

Overfitting, Generalization, and Capacity



higher capacity

Performance Evaluation

Given a sample set (training, validation, or test)

$$D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_m, y_m)\}$$

To assess the performance of the learner *f*, we need to compare the prediction $f(\boldsymbol{x})$ and its ground-truth label y

For regression task, the most common performance measure is MSE (mean squared error),

$$E(f; D) = \frac{1}{m} \sum_{i=1}^{m} \left(f(\boldsymbol{x}_i) - y_i \right)^2$$

Performance Evaluation (for classification)

- Error rate
  - The ratio of the number of misclassified samples to the total number of samples

$$E(f;D) = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}\left(f(\boldsymbol{x}_i) \neq y_i\right)$$

- Accuracy
  - It is derived from the error rate

$$acc(f;D) = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}\left(f(\boldsymbol{x}_i) = y_i\right) = 1 - E(f;D)$$

Performance Evaluation (for classification)

- Precision and Recall

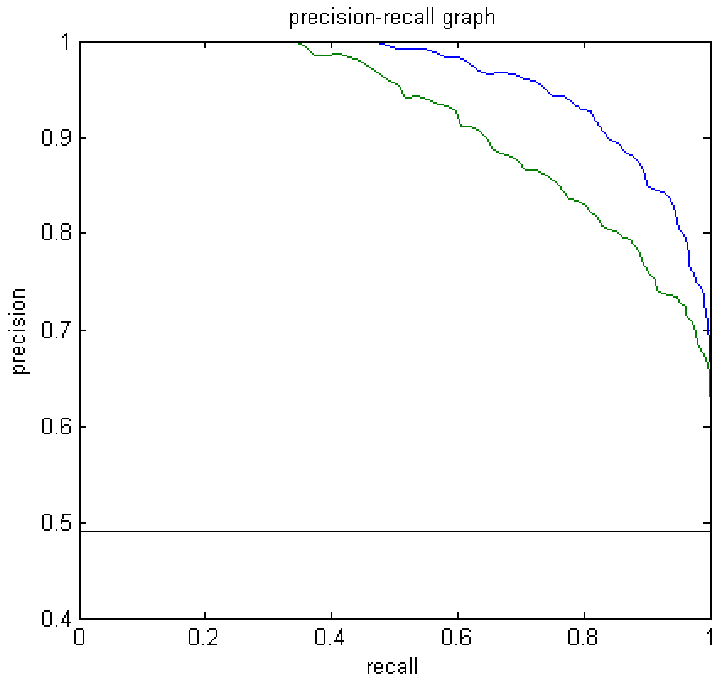
Ground truth	Prediction	
	positive	negative
positive	True Positive (TP)	False Negative (FN)
negative	False Positive (FP)	True Negative (TN)

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Performance Evaluation (for classification)

- Precision and Recall
  - Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other
  - Usually, PR-curve is not monotonic



Performance Evaluation (for classification)

- Precision-recall should be used together; it is meaningless to use only one of them
- However, in many cases, people want to know explicitly which algorithm is better; we can use F-measure

$$F_\beta = \frac{(1+\beta^2)\times P \times R}{(\beta^2 \times P) + R}$$

Performance Evaluation (for classification)

- To derive a single performance measure

Varying threshold, we can have a series (P, R) pairs,

$$(P_1, R_1), (P_2, R_2), \ldots, (P_n, R_n)$$

Then,

$$P_{macro} = \frac{1}{n}\sum_{i=1}^{n} P_i \qquad R_{macro} = \frac{1}{n}\sum_{i=1}^{n} R_i$$

$$F_{\beta-macro} = \frac{\left(1+\beta^2\right)\times P_{macro}\times R_{macro}}{\left(\beta^2\times P_{macro}\right)+R_{macro}}$$

Class-imbalance Issue

- Problem definition
  - It is the problem in machine learning where the total number of a class of data is far less than the total number of another class of data
  - This problem is extremely common in practice
- Why is it a problem?
  - Most machine learning algorithms work best when the number of instances of each classes are roughly equal
  - When the number of instances of one class far exceeds the other, problems arise

Class-imbalance Issue

- How to deal with this issue?
  - Modify the cost function
  - Under-sampling, throwing out samples from majority classes
  - Oversampling, creating new virtual samples for minority classes
    - Just duplicating the minority classes could lead the classifier to overfitting to a few examples
    - Instead, use some algorithm for oversampling, such as SMOTE (synthetic minority over-sampling techniqe)[1]

[1] N.V. Chawla *et al.*, SMOTE: Synthetic Minority Over-sampling Technique, J. Artificial Intelligence Research 16: 321-357, 2002

Class-imbalance Issue

- Minority oversampling by SMOTE[1]

Add new minority class instances by:

- For each minority class instance c
  - neighbours = Get KNN(5)
  - n = Random pick one from neighbours
  - Create a new minority class r instance using c's feature vector and the feature vector's difference of n and c multiplied by a random number
    - » i.e. r.feats = c.feats + (n.feats - c.feats) * rand(0,1)

[1] N.V. Chawla *et al.*, SMOTE: Synthetic Minority Over-sampling Technique, J. Artificial Intelligence Research 16: 321-357, 2002

Outline

- Basic concepts
- Linear model
  - Linear regression
  - Logistic regression
  - Softmax regression
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- DCNN for object detection

Linear regression

- Our goal in linear regression is to predict a target continuous value y from a vector of input values $\boldsymbol{x} \in R^d$; we use a linear function h as the model
- At the training stage, we aim to find $h(\boldsymbol{x})$ so that we have $h(\boldsymbol{x}_i) \approx y_i$ for each training sample $(\boldsymbol{x}_i, y_i)$
- We suppose that h is a linear function, so

$$h_{(\boldsymbol{\theta},b)}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} + b, \boldsymbol{\theta} \in R^{d\times 1}$$

Rewrite it,

$$\boldsymbol{\theta}^{'} = \begin{pmatrix} \boldsymbol{\theta} \\ b \end{pmatrix}, \boldsymbol{x}^{'} = \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$$

$$\boldsymbol{\theta}^T \boldsymbol{x} + b = \boldsymbol{\theta}^{'T} \boldsymbol{x}^{'} \equiv h_{\boldsymbol{\theta}^{'}}(\boldsymbol{x}^{'})$$

Later, we simply use $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}, \boldsymbol{\theta} \in R^{(d+1)\times 1}, \boldsymbol{x} \in R^{(d+1)\times 1}$

Linear regression

- Then, our task is to find a choice of $\boldsymbol{\theta}$ so that $h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ is as close as possible to y_i

The cost function can be written as,

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^T\boldsymbol{x}_i - y_i\right)^2$$

Then, the task at the training stage is to find

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^T\boldsymbol{x}_i - y_i\right)^2$$

For this special case, it has a closed-form optimal solution

Here we use a more general method, **gradient descent** method

Linear regression

- Gradient descent
  - It is a first-order optimization algorithm
  - To find a local minimum of a function, one takes steps proportional to the negative of the gradient of the function at the current point
  - One starts with a guess $\boldsymbol{\theta}_0$ for a local minimum of $J(\boldsymbol{\theta})$ and considers the sequence such that

$$\boldsymbol{\theta}_{n+1} := \boldsymbol{\theta}_n - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})_{|\boldsymbol{\theta}=\boldsymbol{\theta}_n}$$

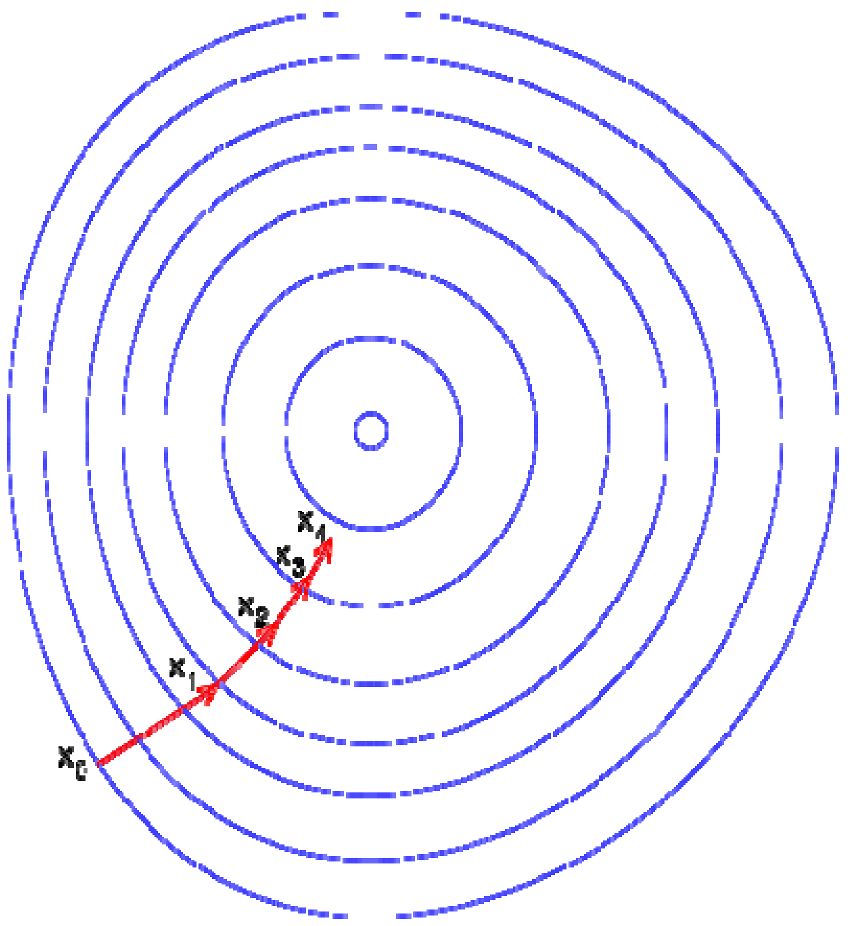
where α is called as **learning rate**

Linear regression

- Gradient descent

Linear regression

- Gradient descent



Linear regression

- Gradient descent

Repeat until convergence ($J(\boldsymbol{\theta})$ will not reduce anymore)

{

$$\boldsymbol{\theta}_{n+1} := \boldsymbol{\theta}_n - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})_{|\boldsymbol{\theta}=\boldsymbol{\theta}_n}$$

}

GD is a general optimization solution; for a specific problem, the key step is how to compute gradient

Linear regression

- Gradient of the cost function of linear regression

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^T \boldsymbol{x}_i - y_i\right)^2$$

The gradient is,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_{d+1}} \end{bmatrix}$$ where, $$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\right)x_{ij}$$

Linear regression

- Some variants of gradient descent
  - The ordinary gradient descent algorithm looks at every sample in the **entire** training set on every step; it is also called as **batch gradient descent**
  - **Stochastic gradient descent (SGD)** repeatedly run through the training set, and each time when we encounter a training sample, we update the parameters according to the gradient of the error w.r.t that single training sample only

```
Repeat until convergence
{
    for $i$ = 1 to $m$ ($m$ is the number of training samples)
    {
        $\boldsymbol{\theta}_{n+1} := \boldsymbol{\theta}_n - \alpha\left(\boldsymbol{\theta}_n^T \mathbf{x}_i - y_i\right)\mathbf{x}_i$
    }
}
```

Linear regression

- Some variants of gradient descent
  - The ordinary gradient descent algorithm looks at every sample in the **entire** training set on every step; it is also called as **batch gradient descent**
  - **Stochastic gradient descent (SGD)** repeatedly run through the training set, and each time when we encounter a training sample, we update the parameters according to the gradient of the error w.r.t that single training sample only
  - **Minibatch SGD**: it works identically to SGD, except that it uses more than one training samples to make each estimate of the gradient

Linear regression

- More concepts
  - *m* Training samples can be divided into *N* minibatches
  - When the training sweeps all the batches, we say we complete one **epoch** of training process; for a typical training process, several epochs are usually required

```
epochs = 10;
numMiniBatches = $N$;
while epochIndex< epochs && not convergent
{
    for minibatchIndex = 1 to numMiniBatches
    {
      update the model parameters based on this minibatch
    }
}
```

Outline

- Basic concepts
- Linear model
  - Linear regression
  - Logistic regression
  - Softmax regression
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- DCNN for object detection

Logistic regression

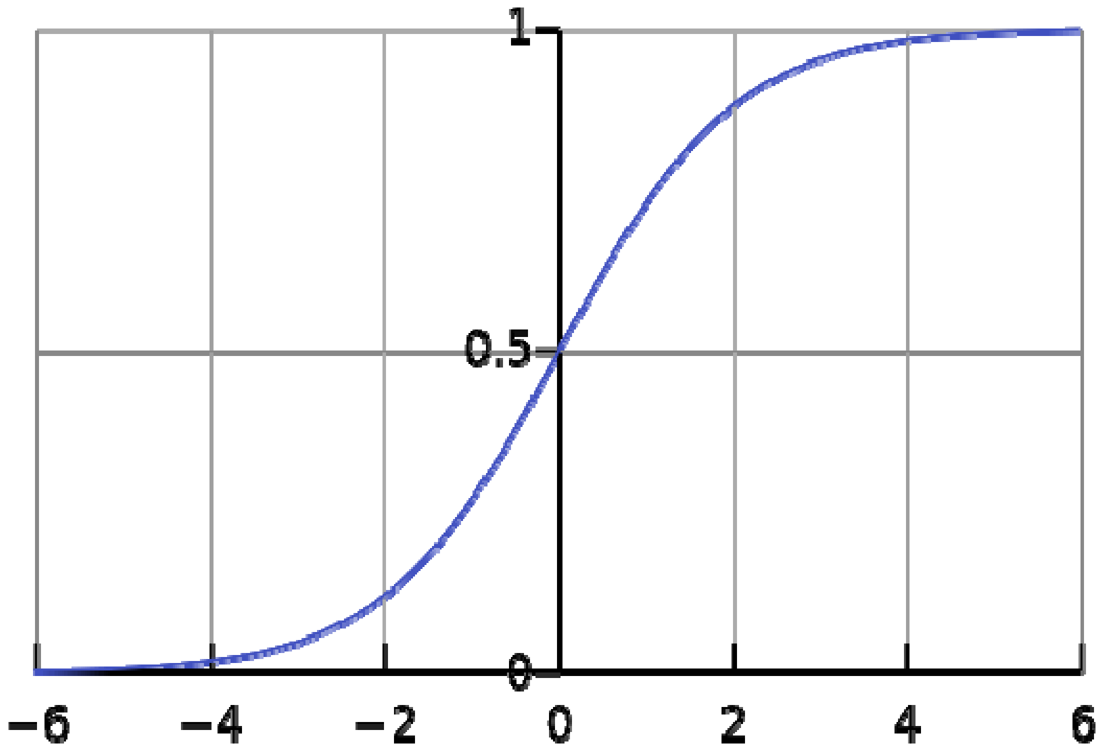
- Logistic regression is used for binary classification
- It squeezes the linear regression $\boldsymbol{\theta}^T \boldsymbol{x}$ into the range (0, 1) ; thus the prediction result can be interpreted as probability
- At the testing stage

  The probability that the testing sample $\boldsymbol{x}$ is positive is represented as $h_\theta(\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^T \boldsymbol{x})}$

  The probability that the testing sample $\boldsymbol{x}$ is negative is represented as $1\text{-}h_\theta(\boldsymbol{x})$

Function $\sigma(z) = \frac{1}{1+\exp(-z)}$ is called as **sigmoid** or **logistic** function

Logistic regression



The shape of sigmoid function

One property of the sigmoid function

$$\sigma^{'}(z) = \sigma(z)(1-\sigma(z))$$

Logistic regression

- The hypothesis model can be written neatly as

$$P(y \mid \boldsymbol{x};\boldsymbol{\theta}) = \left(h_{\boldsymbol{\theta}}(\boldsymbol{x})\right)^{y}\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x})\right)^{1-y}$$

- Our goal is to search for a value $\boldsymbol{\theta}$ so that $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is large when $\boldsymbol{x}$ belongs to “1” class and small when $\boldsymbol{x}$ belongs to “0” class

Thus, given a training set with binary labels $\left\{(\boldsymbol{x}_i, y_i): i=1,...,m\right\}$, we want to maximize,

$$\prod_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)^{y_i}\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)^{1-y_i}$$

Equivalent to maximize,

$$\sum_{i=1}^{m} y_i \log\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right) + (1-y_i)\log\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)$$

Logistic regression

- Thus, the cost function for the logistic regression is (we want to minimize),

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{m} y_i \log\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right) + (1 - y_i)\log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)$$

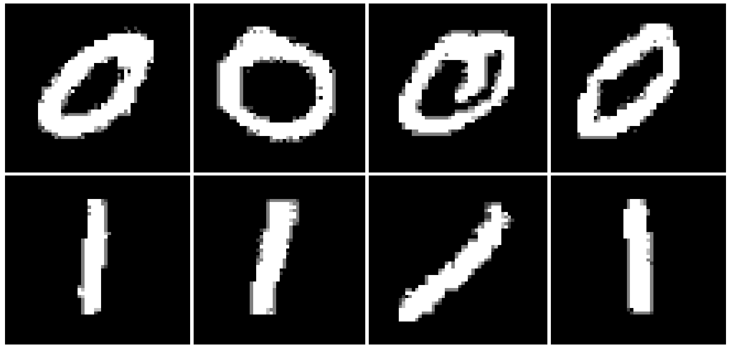
To solve it with gradient descent, gradient needs to be computed,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \boldsymbol{x}_i \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\right)$$

Assignment!

Logistic regression

- Exercise
  - Use logistic regression to perform digital classification



Outline

- Basic concepts
- Linear model
  - Linear regression
  - Logistic regression
  - Softmax regression
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- DCNN for object detection

Softmax regression

- Softmax operation
  - It squashes a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range (0, 1). The function is given by,

$$\sigma(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^{K} \exp(\mathbf{z}_k)}$$

  - Since the components of the vector $\sigma(\mathbf{z})$ sum to one and are all strictly between 0 and 1, they represent a categorical probability distribution

Softmax regression

- For multiclass classification, given a test input $\boldsymbol{x}$, we want our hypothesis to estimate $p(y=k \mid \boldsymbol{x})$ for each value k=1,2,…,K

Softmax regression

- The hypothesis should output a K-dimensional vector giving us K estimated probabilities. It takes the form,

$$h_{\phi}(\boldsymbol{x}) = \begin{bmatrix} p(y=1 \mid \boldsymbol{x};\phi) \\ p(y=2 \mid \boldsymbol{x};\phi) \\ \vdots \\ p(y=K \mid \boldsymbol{x};\phi) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp\left((\boldsymbol{\theta}_j)^T \boldsymbol{x} \right)} \begin{bmatrix} \exp\left((\boldsymbol{\theta}_1)^T \boldsymbol{x} \right) \\ \exp\left((\boldsymbol{\theta}_2)^T \boldsymbol{x} \right) \\ \vdots \\ \exp\left((\boldsymbol{\theta}_K)^T \boldsymbol{x} \right) \end{bmatrix}$$

where $\phi = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K] \in R^{(d+1)\times K}$

Softmax regression

- In softmax regression, for each training sample we have,

$$p\left(y_i = k \mid \boldsymbol{x}_i; \phi\right) = \frac{\exp\left(\left(\boldsymbol{\theta}_k\right)^T \boldsymbol{x}_i\right)}{\sum_{j=1}^{K} \exp\left(\left(\boldsymbol{\theta}_j\right)^T \boldsymbol{x}_i\right)}$$

At the training stage, we want to maximize $p\left(y_i = k \mid \boldsymbol{x}_i; \phi\right)$ for each training sample for the correct label k

Softmax regression

- Cost function for softmax regression

$$J(\phi) = -\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y_i = k\} \log \frac{\exp\left((\boldsymbol{\theta}_k)^T \boldsymbol{x}_i \right)}{\sum_{j=1}^{K} \exp\left((\boldsymbol{\theta}_j)^T \boldsymbol{x}_i \right)}$$

where 1{.} is an indicator function

- Gradient of the cost function

$$\nabla_{\boldsymbol{\theta}_k} J(\phi) = -\sum_{i=1}^{m} \left[\boldsymbol{x}_i \left(1\{y_i = k\} - p\left(y_i = k \mid \boldsymbol{x}_i; \phi \right) \right) \right]$$

Softmax regression

- Redundancy of softmax regression parameters

Subtract a fixed vector ψ from every $\boldsymbol{\theta}_j$, we have

$$p\left(y_i = k \mid \boldsymbol{x}_i; \phi\right) = \frac{\exp\left(\left(\boldsymbol{\theta}^{(k)} - \psi\right)^T \boldsymbol{x}_i\right)}{\sum_{j=1}^{K} \exp\left(\left(\boldsymbol{\theta}^{(j)} - \psi\right)^T \boldsymbol{x}_i\right)}$$

$$= \frac{\exp\left(\left(\boldsymbol{\theta}_k\right)^T \boldsymbol{x}_i\right)\exp\left(-\psi^T \boldsymbol{x}_i\right)}{\sum_{j=1}^{K} \exp\left(\left(\boldsymbol{\theta}_j\right)^T \boldsymbol{x}_i\right)\exp\left(-\psi^T \boldsymbol{x}_i\right)} = \frac{\exp\left(\left(\boldsymbol{\theta}_k\right)^T \boldsymbol{x}_i\right)}{\sum_{j=1}^{K} \exp\left(\left(\boldsymbol{\theta}_j\right)^T \boldsymbol{x}_i\right)}$$

Softmax regression

- Redundancy of softmax regression parameters
- So, in most cases, instead of optimizing $K \bullet (d+1)$ parameters, we can set $\boldsymbol{\theta}_K = 0$ and optimize only w.r.t the $(K-1) \bullet (d+1)$ remaining parameters

Cross entropy

- After the softmax operation, the output vector can be regarded as a discrete probability density function
- For multiclass classification, the ground-truth label for a training sample is usually represented in one-hot form, which can also be regarded as a density function

  For example, we have 10 classes, and the ith training sample belongs to class 7, then $y_i = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]$

- Thus, at the training stage, we want to minimize

$$\sum_i dist(h(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$$

How to define *dist*? Cross entroy is a common choice

Cross entropy

- Information entropy is defined as the average amount of information produced by a probabilistic stochastic source of data $H(X) = -\sum_i p(x_i) \log p(x_i)$
- Cross entropy can measure the difference between two distributions

$$H(p,q) = -\sum_i p(x_i) \log q(x_i)$$

- For multiclass classification, the last layer usually is a softmax layer and the loss is the 'cross entropy'

Outline

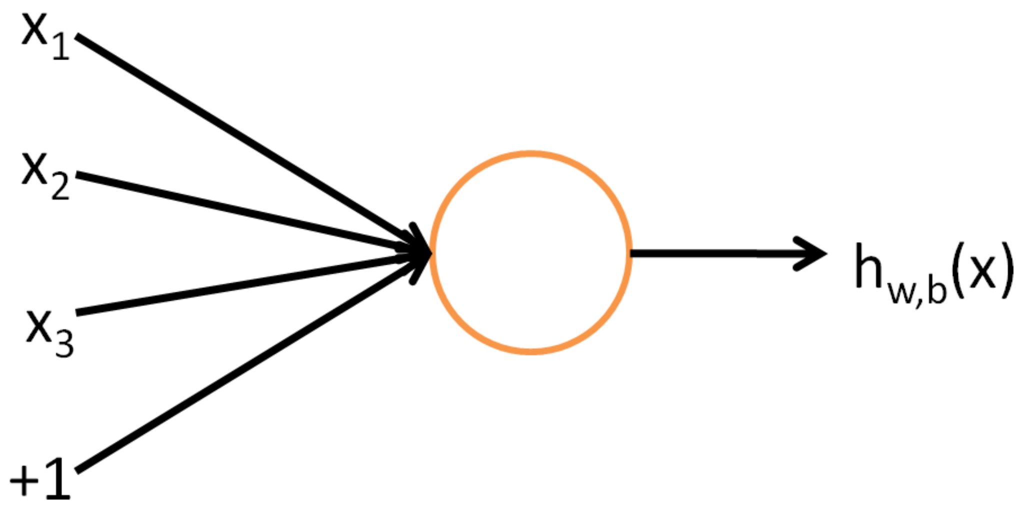
- Basic concepts
- Linear model
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- DCNN for object detection

Neural networks

- It is one way to solve a supervised learning problem given labeled training examples $\{\boldsymbol{x}_i, y_i\}(i=1,...,m)$
- Neural networks give a way of defining a complex, non-linear form of hypothesis $h_{W,b}(\boldsymbol{x})$, where W and b are the parameters we need to learn from training samples

Neural networks

- A single neuron
  - x_1, x_2, and x_3 are the inputs, +1 is the intercept term, $h_{W,b}(\boldsymbol{x})$ is the output of this neuron



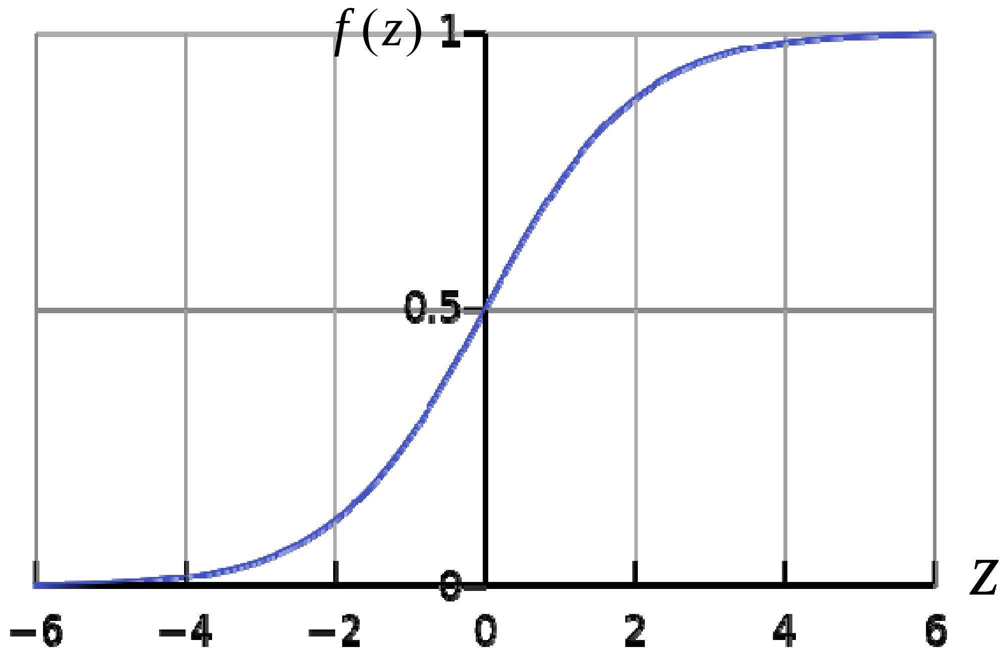
$$h_{W,b}(x) = f\left(W^T \boldsymbol{x}\right) = f\left(\sum_{i=1}^{3} W_i x_i + b\right)$$

where $f(\cdot)$ is the activation function

Neural networks

- Commonly used activation functions
  - Sigmoid function

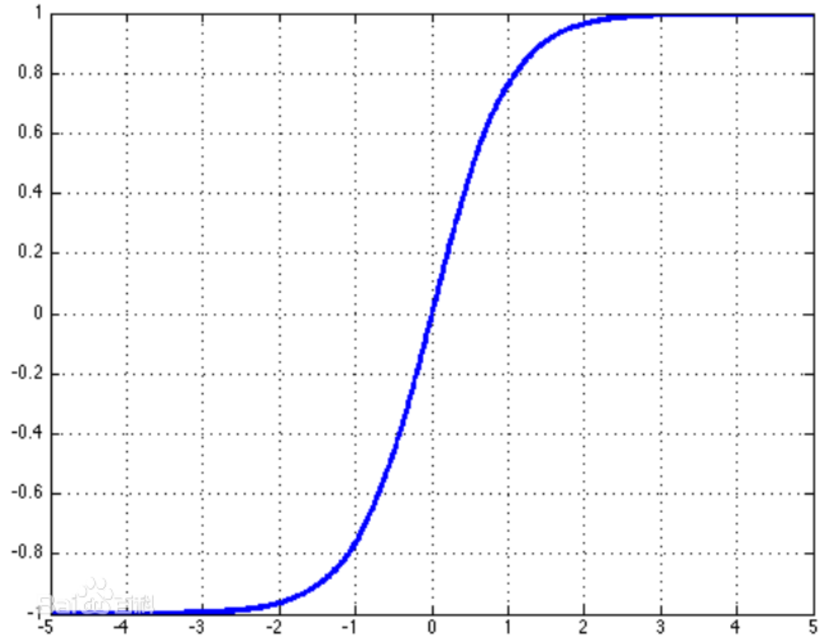
$$f(z) = \frac{1}{1+\exp(-z)}$$



Neural networks

- Commonly used activation functions
  - Tanh function

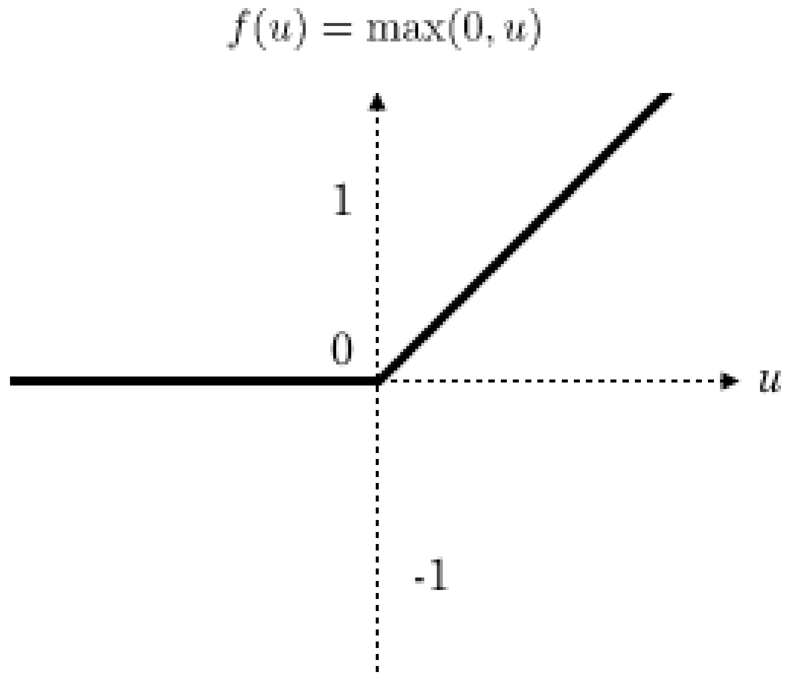
$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Neural networks

- Commonly used activation functions
  - Rectified linear unit (ReLU)

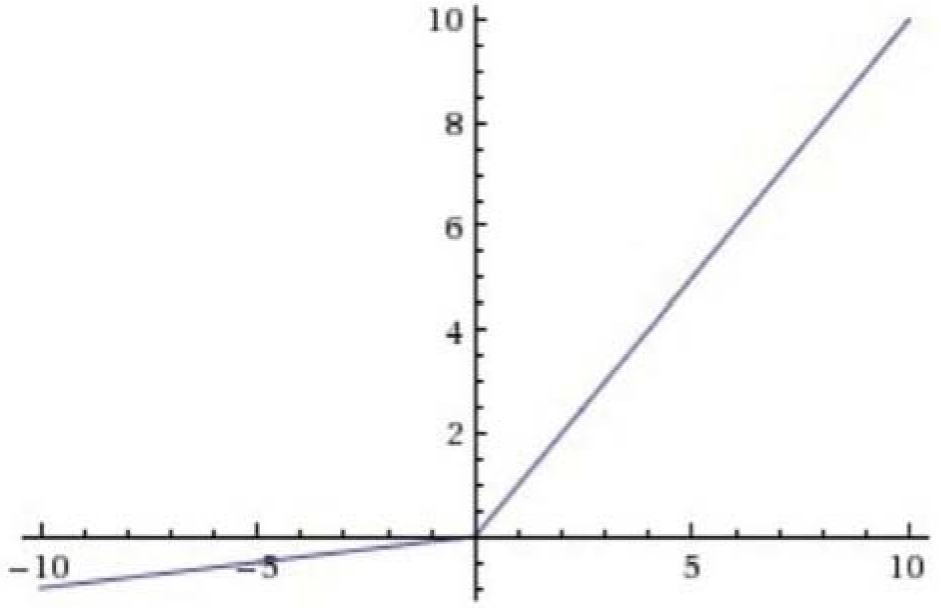
$$f(z) = \max(0, z)$$



Neural networks

- Commonly used activation functions
  - Leaky Rectified linear unit (ReLU)

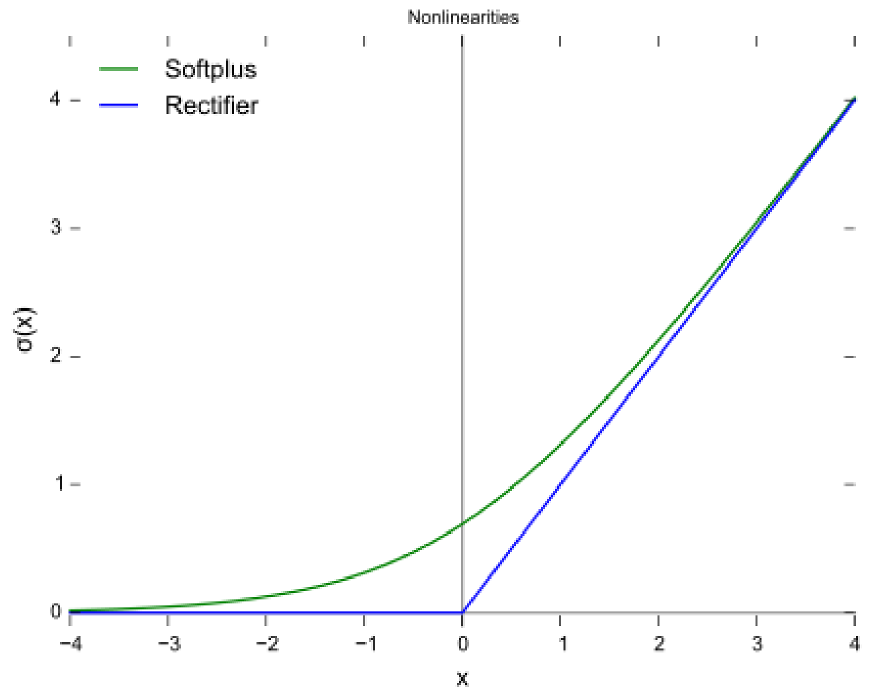
$$f(z)=\begin{cases} z, if\ \ z>0 \\ 0.01z, otherwise \end{cases}$$



Neural networks

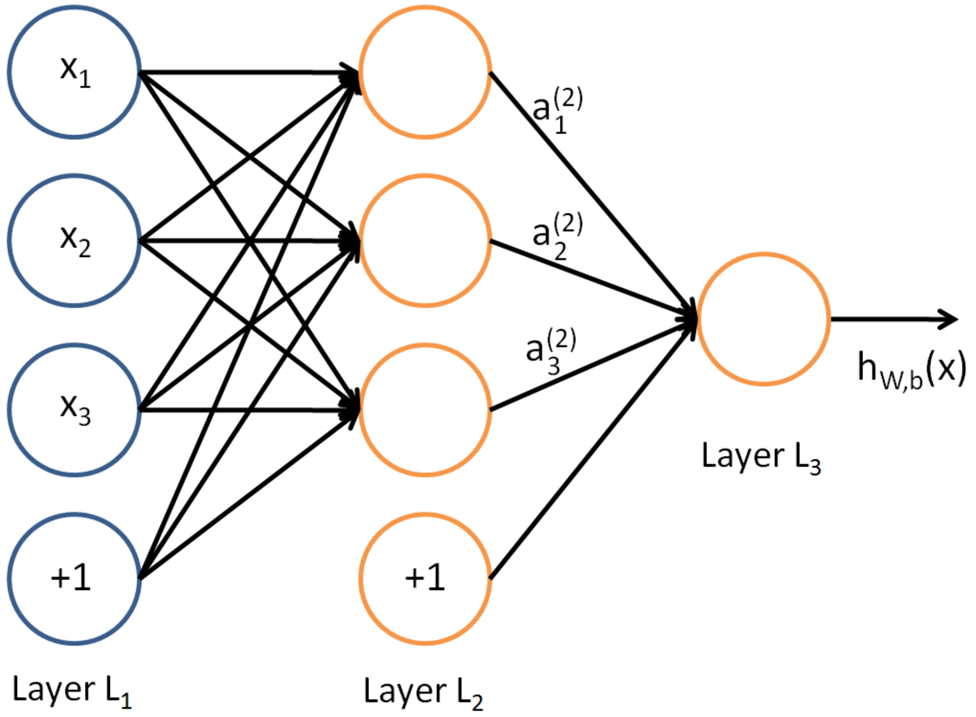
- Commonly used activation functions
  - Softplus (can be regarded as a smooth approximation to ReLU)

$$f(z) = \ln\left(1 + e^{z}\right)$$



Neural networks

- A neural network is composed by hooking together many simple neurons
- The output of a neuron can be the input of another
- Example, a three layers neural network,



Neural networks

- Terminologies about the neural network
  - The circle labeled +1 are called **bias units**
  - The leftmost layer is called the **input layer**
  - The rightmost layer is the **output layer**
  - The middle layer of nodes is called the **hidden layer**
    - » In our example, there are 3 input units, 3 hidden units, and 1 output unit
  - We denote the activation (output value) of unit i in lay l as $a_i^{(l)}$

Neural networks

$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$

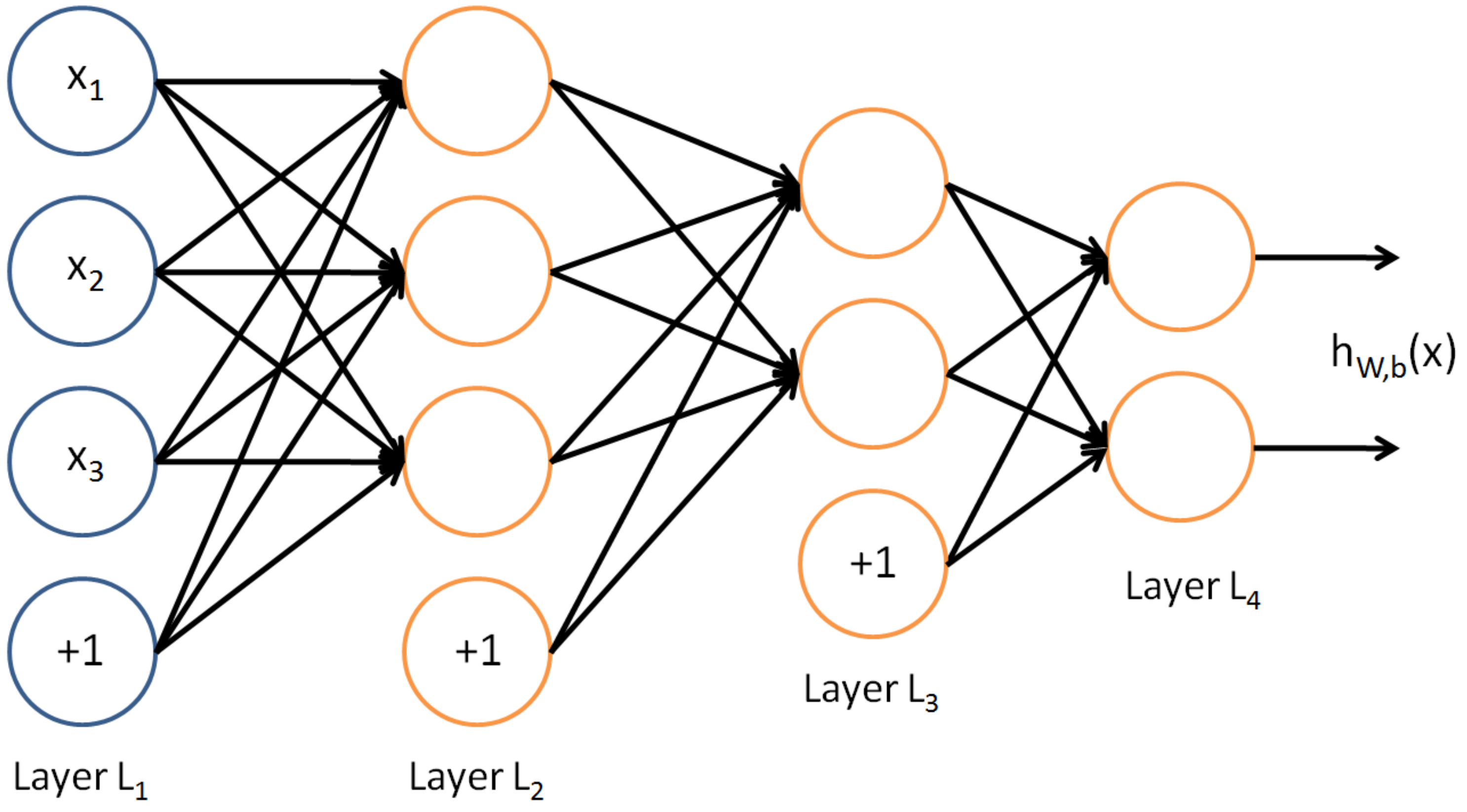
$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right)$$

$$a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}\right)$$

$$h_{W,b}(\boldsymbol{x}) = a_1^{(3)} = f\left(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{1}a_3^{(2)} + b_1^{(2)}\right)$$

Neural networks

- Neural networks can have multiple outputs
- Usually, we can add a softmax layer as the output layer to perform multiclass classification



Neural networks

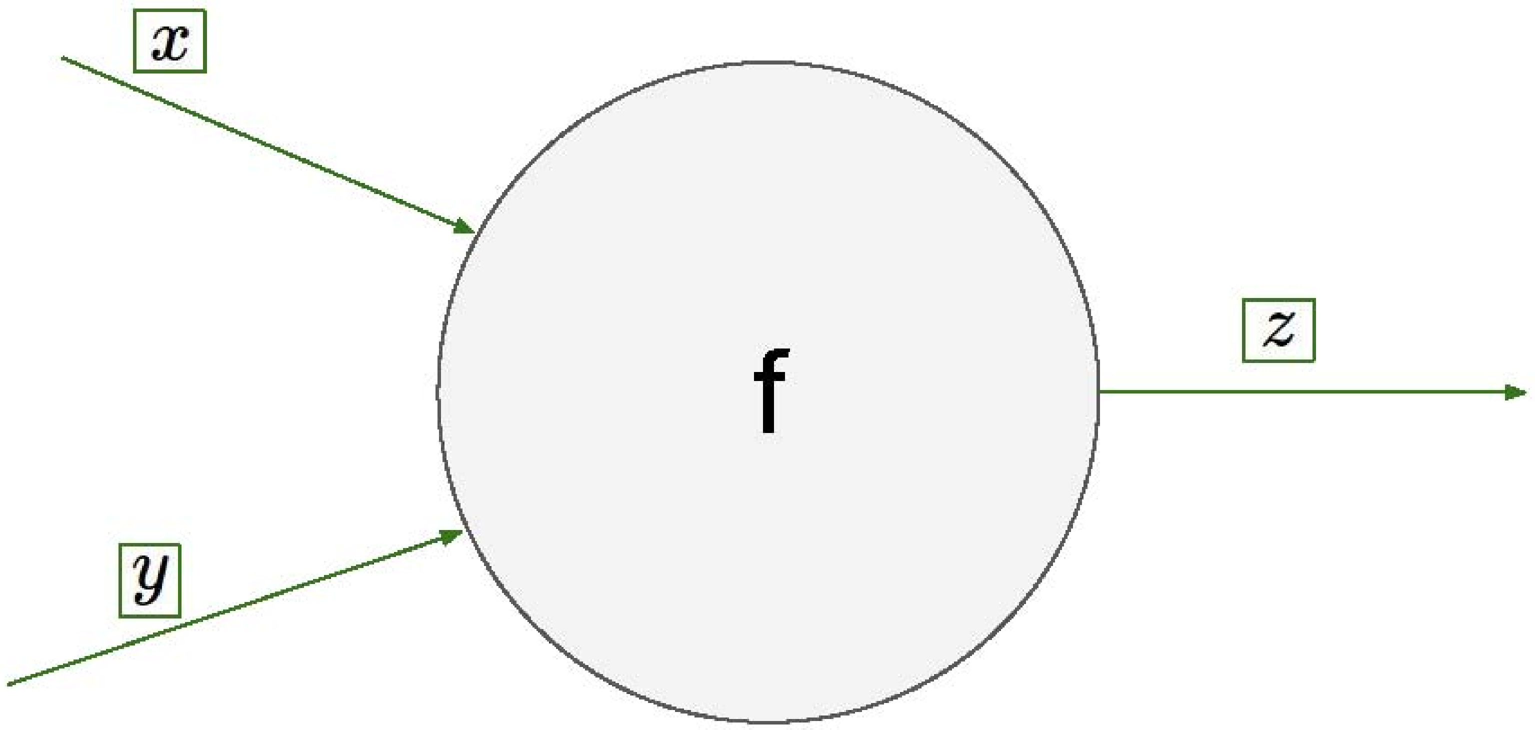
- At the testing stage, given a test input $\boldsymbol{x}$, it is straightforward to evaluate its output
- At the training stage, given a set of training samples, we need to train W and b
  - The key problem is how to compute the gradient
  - Backpropagation algorithm

Neural networks

- Backpropagation
  - A common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent
  - Its purpose is to compute the partial derivative of the loss to each parameter (weights)
  - neural nets will be very large: impractical to write down gradient formula by hand for all parameters
  - recursive application of the chain rule along a computational graph to compute the gradients of all parameters

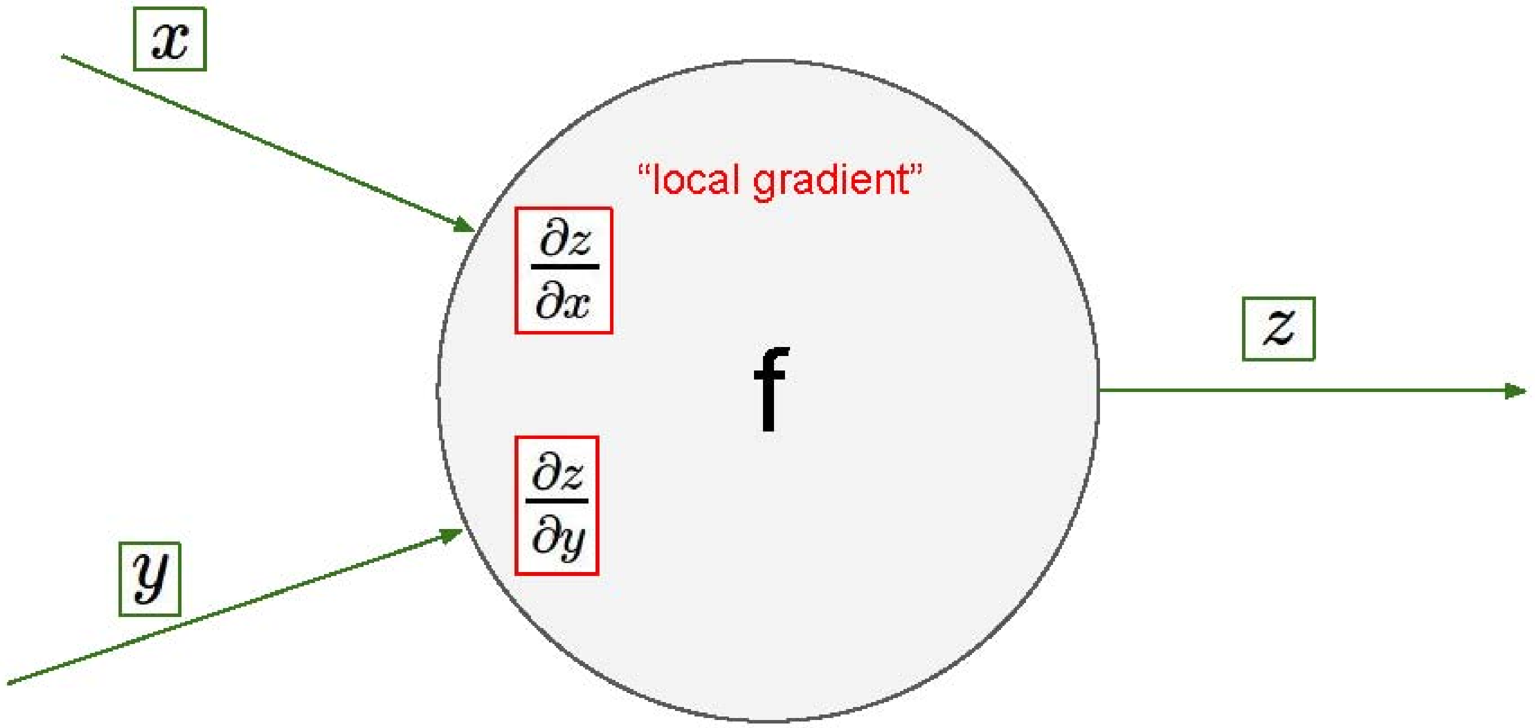
Neural networks

- Backpropagation



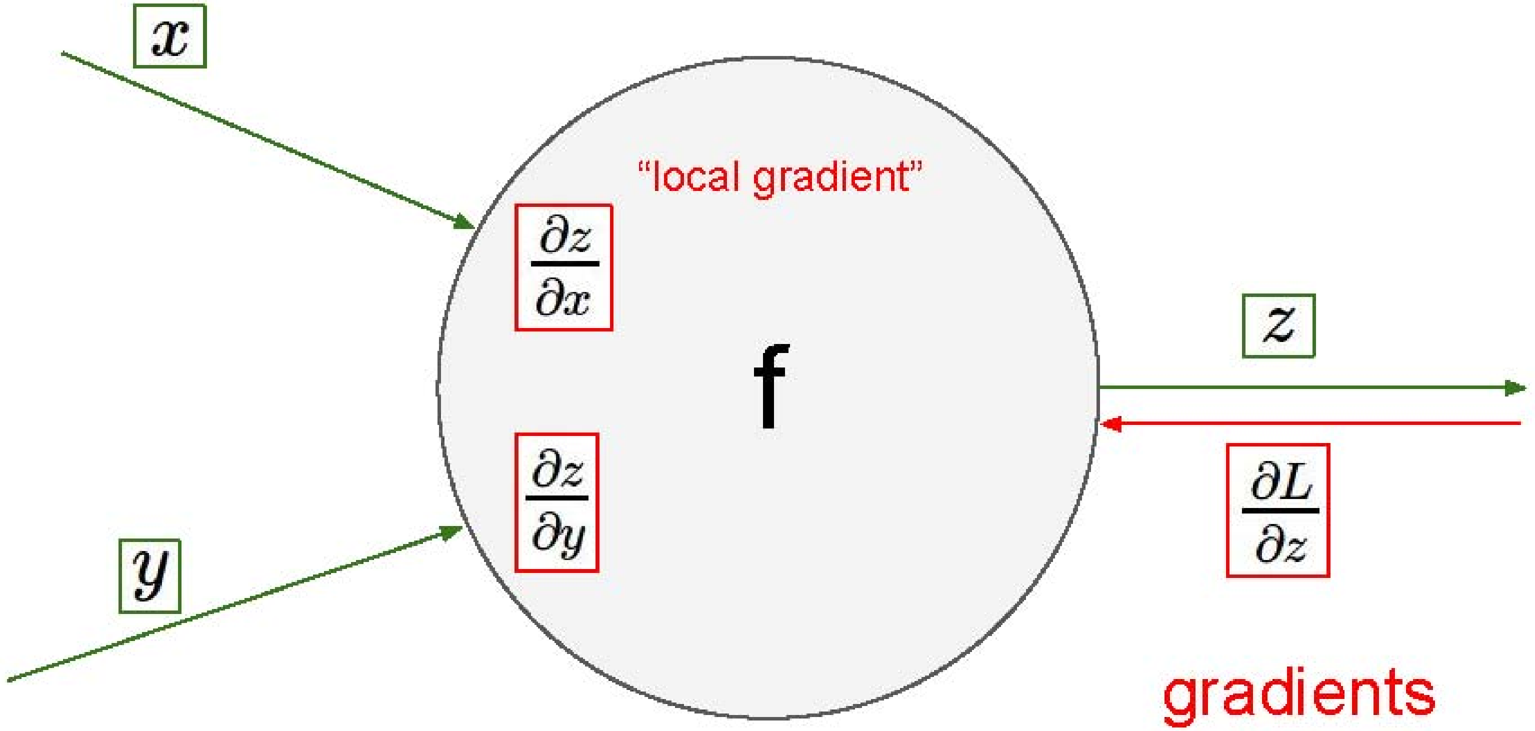
Neural networks

- Backpropagation


x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z

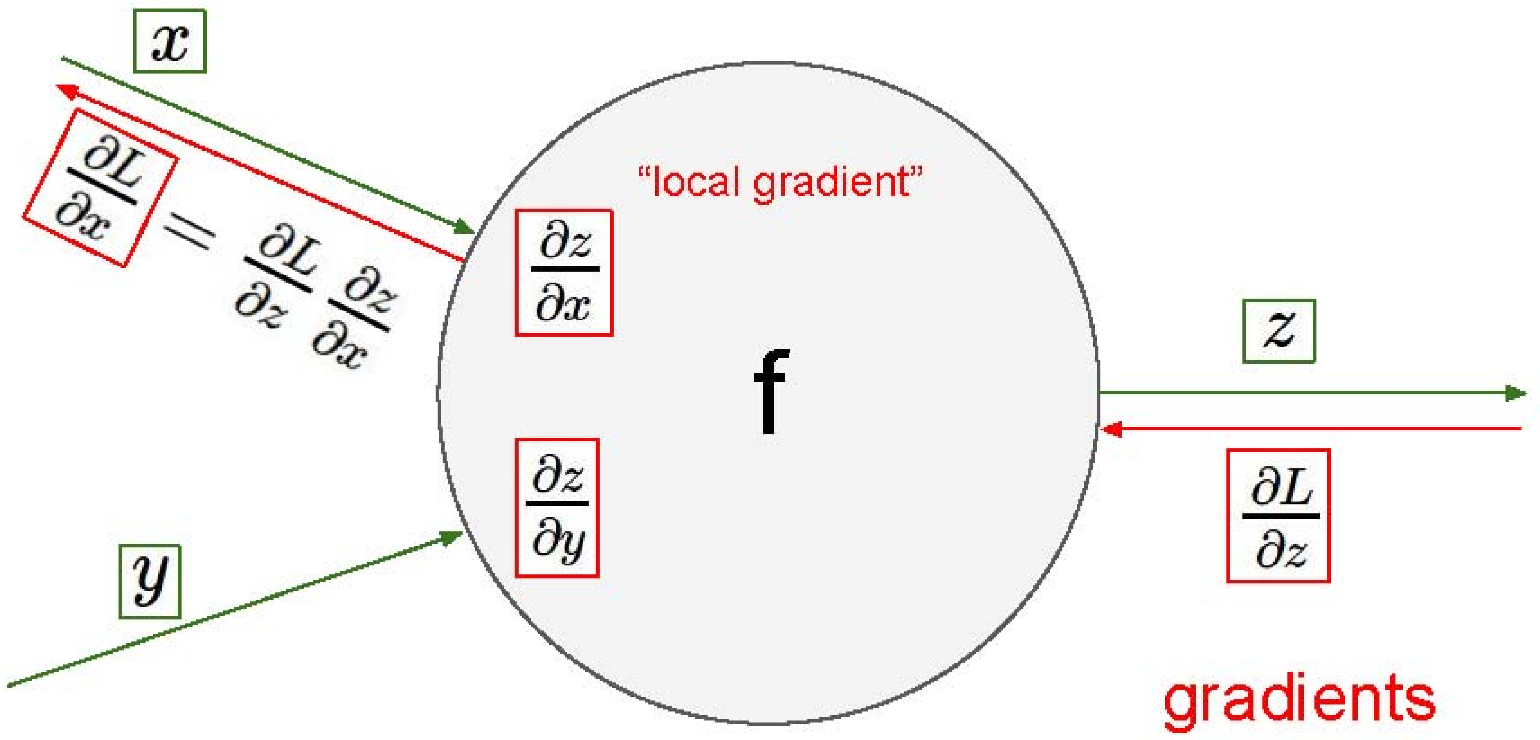
Neural networks

- Backpropagation


x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
gradients

Neural networks

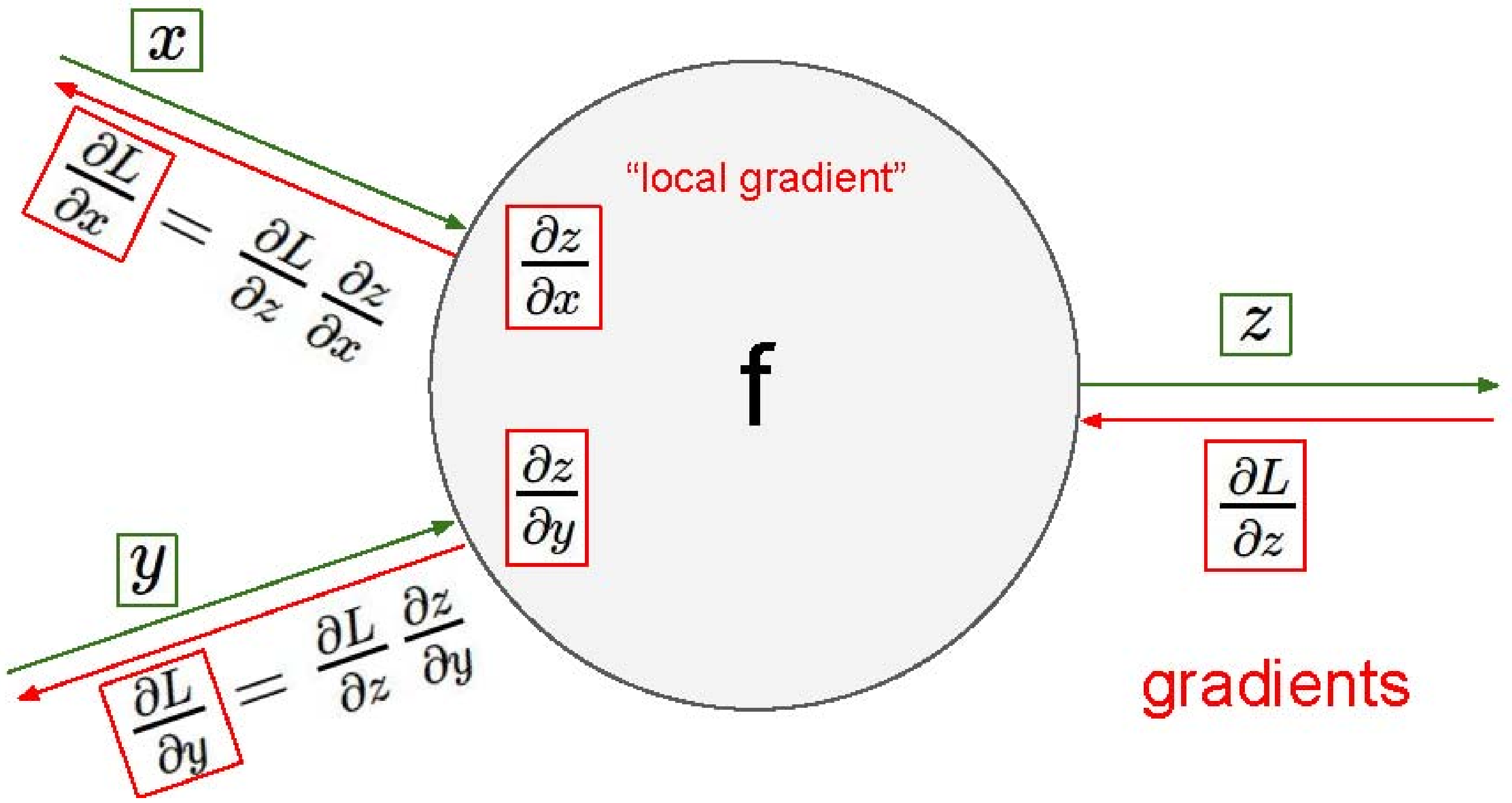
- Backpropagation



x
y
z
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
gradients

Neural networks

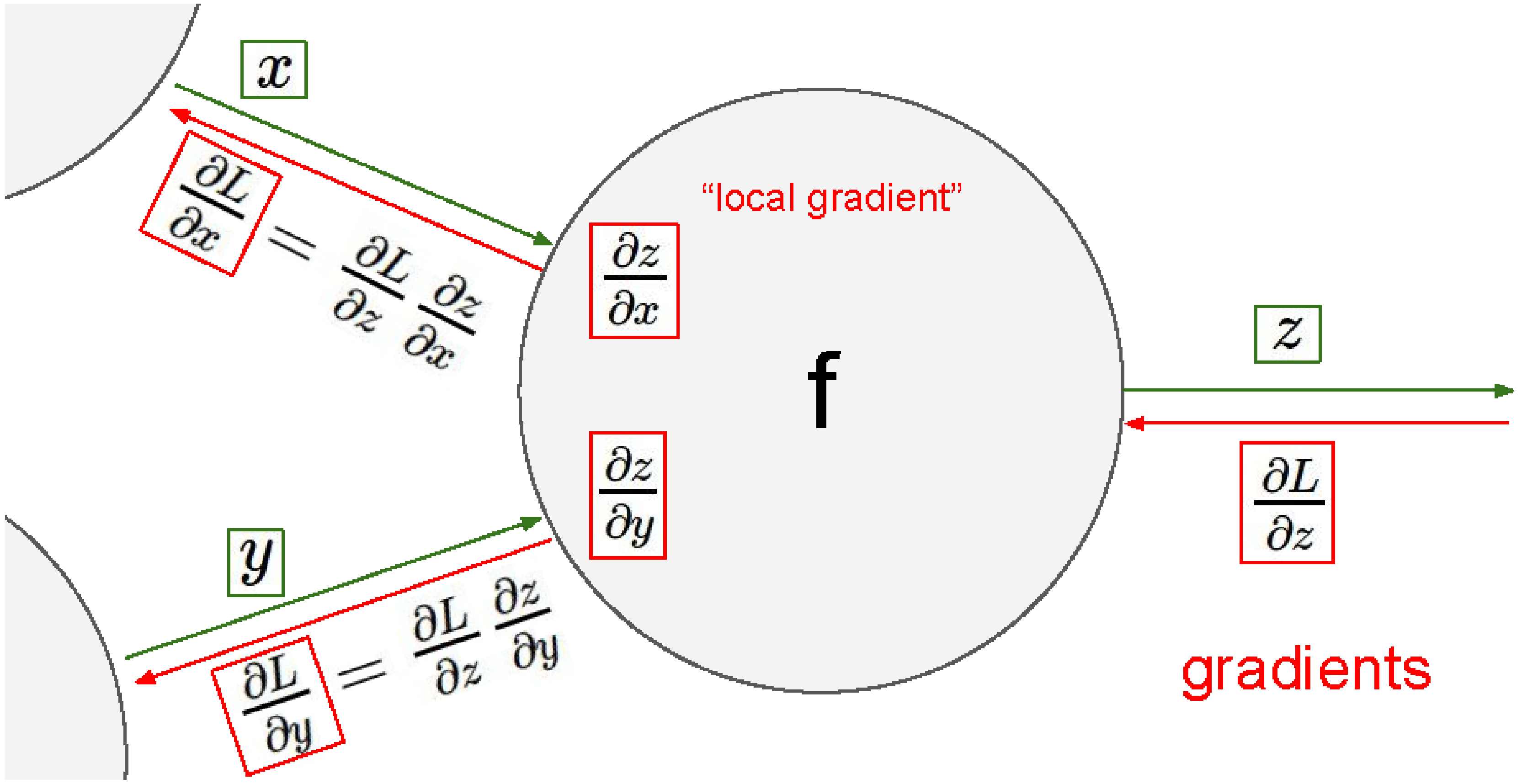
- Backpropagation



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Neural networks

- Backpropagation



x
y
z
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial z}$
gradients
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

Outline

- Basic concepts
- Linear model
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- CNN for object detection

Convolutional neural network

- Specially designed for data with grid-like structures (LeCun et al. 98)
  - 1D grid: sequential data
  - 2D grid: image
  - 3D grid: video, 3D image volume
- Beat all the existing computer vision technologies on object recognition on ImageNet challenge with a large margin in 2012

Convolutional neural network

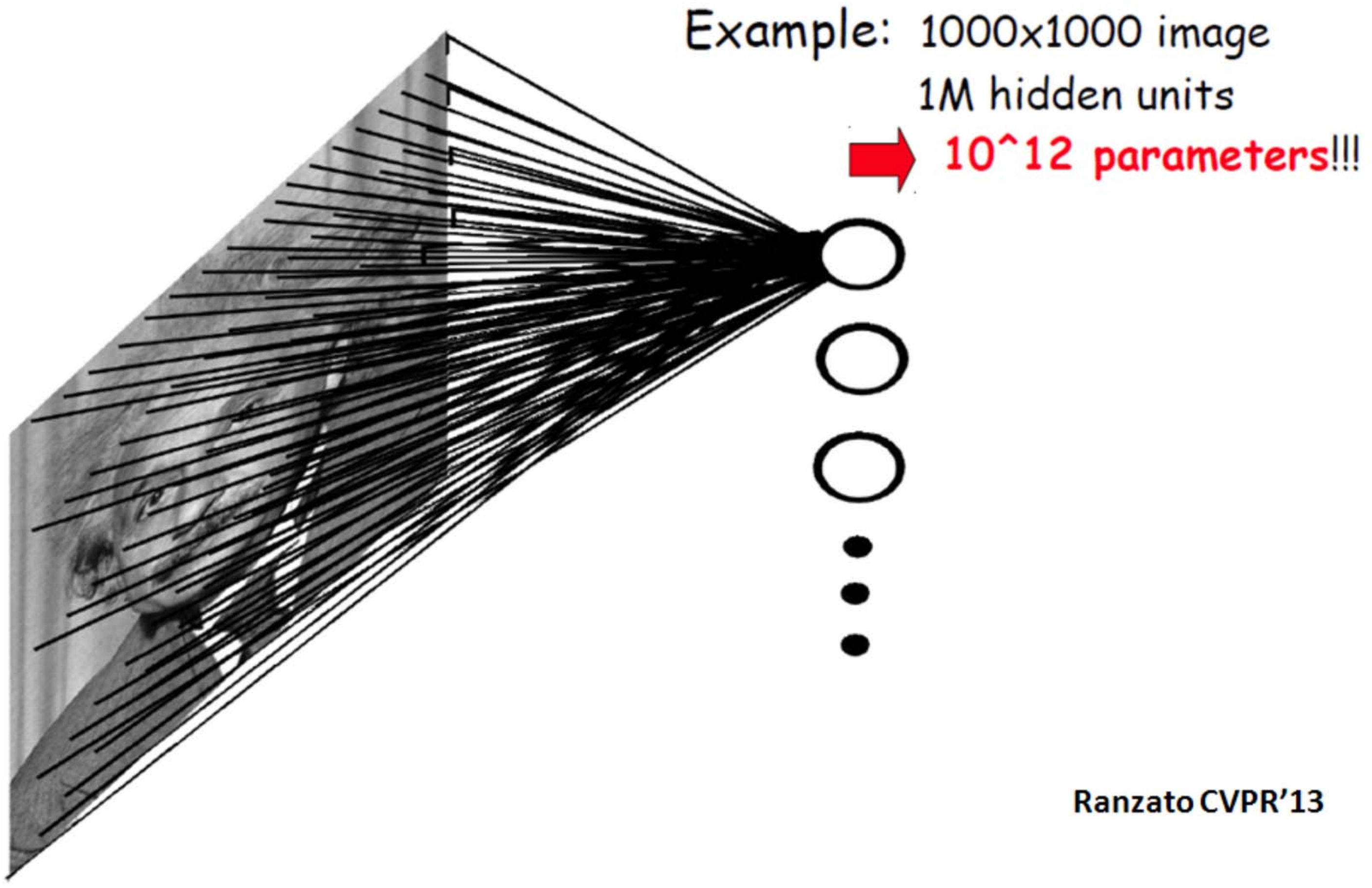
- Something you need to know about DCNN
  - Traditional model for PR: fixed/engineered features + trainable classifier
  - For DCNN: it is usually an **end-to-end** architecture; learning data representation and classifier together
  - The learned features from big datasets are **transferable**
  - For training a DCNN, usually we use a **fine-tuning** scheme
  - For training a DCNN, to avoid overfitting, **data augmentation** can be performed

Convolutional neural network

- Problems of fully connected networks
  - Every output unit interacts with every input unit
  - The number of weights grows largely with the size of the input image
  - Pixels in distance are less correlated

Convolutional neural network

- Problems of fully connected networks

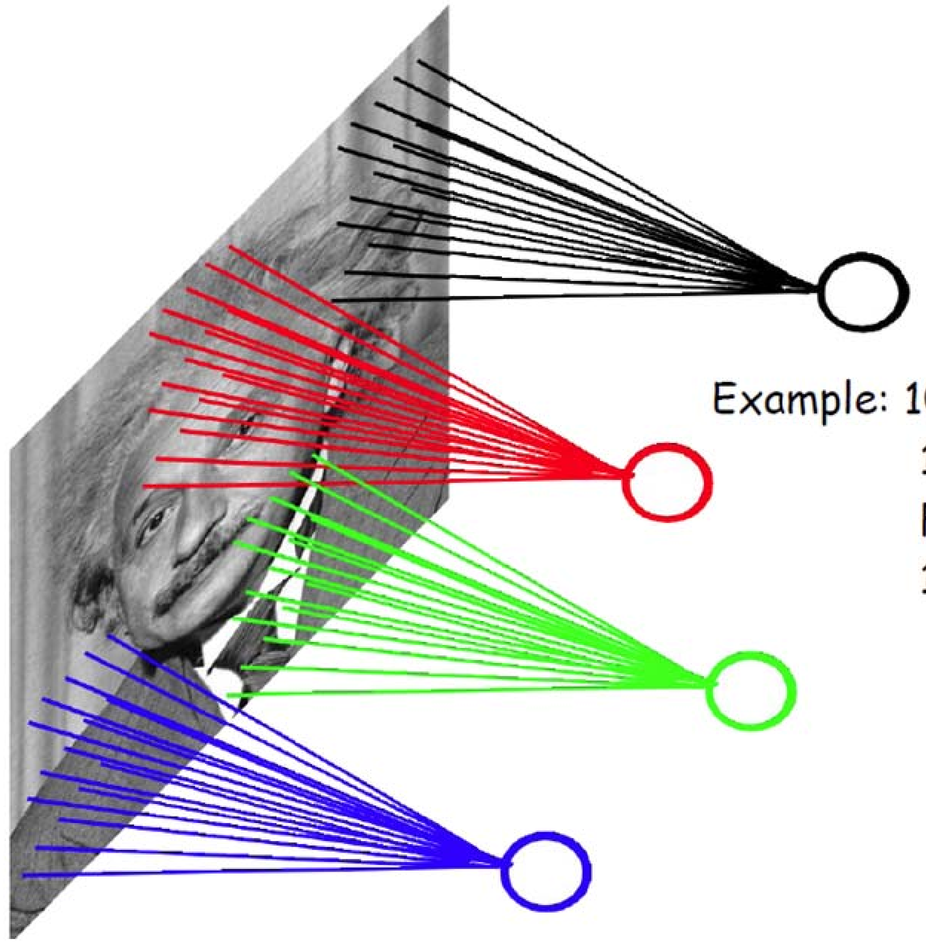


Convolutional neural network

- One simple solution is locally connected neural networks
  - Sparse connectivity: a hidden unit is only connected to a local patch (weights connected to the patch are called filter or kernel)
  - It is inspired by biological systems, where a cell is sensitive to a small sub-region of the input space, called a receptive field; Many cells are tiled to cover the entire visual field

Convolutional neural network

- One simple solution is locally connected neural networks

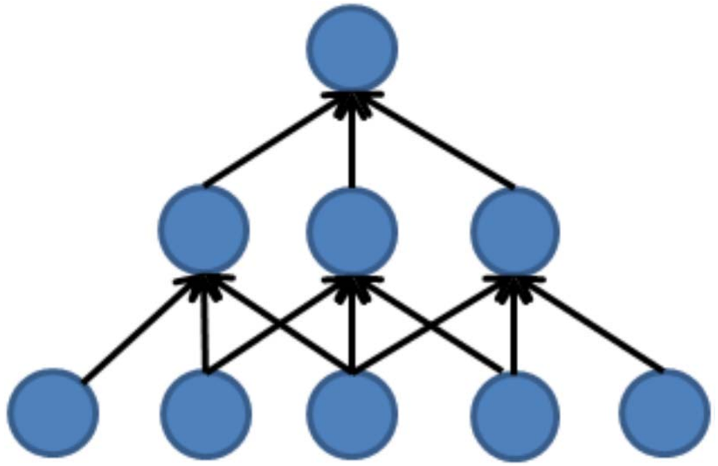


Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Ranzato CVPR'13

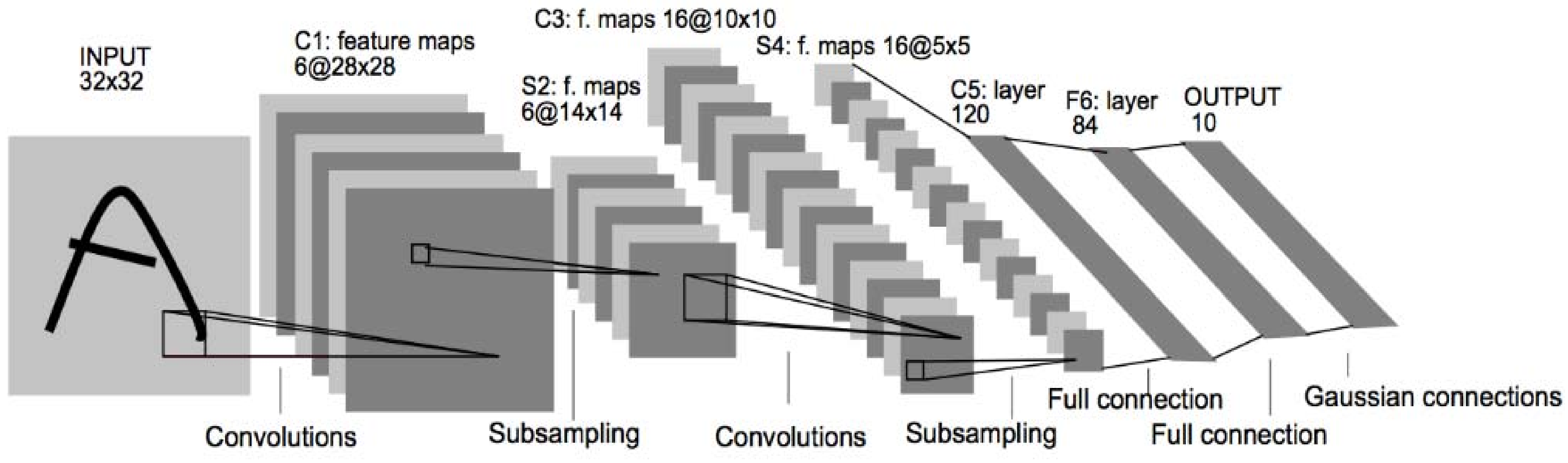
Convolutional neural network

- One simple solution is locally connected neural networks
  - The learned filter is a spatially local pattern
  - A hidden node at a higher layer has a larger receptive field in the input
  - Stacking many such layers leads to “filters” (not anymore linear) which become increasingly “global”



Convolutional neural network

- The first CNN
  - LeNet[1]



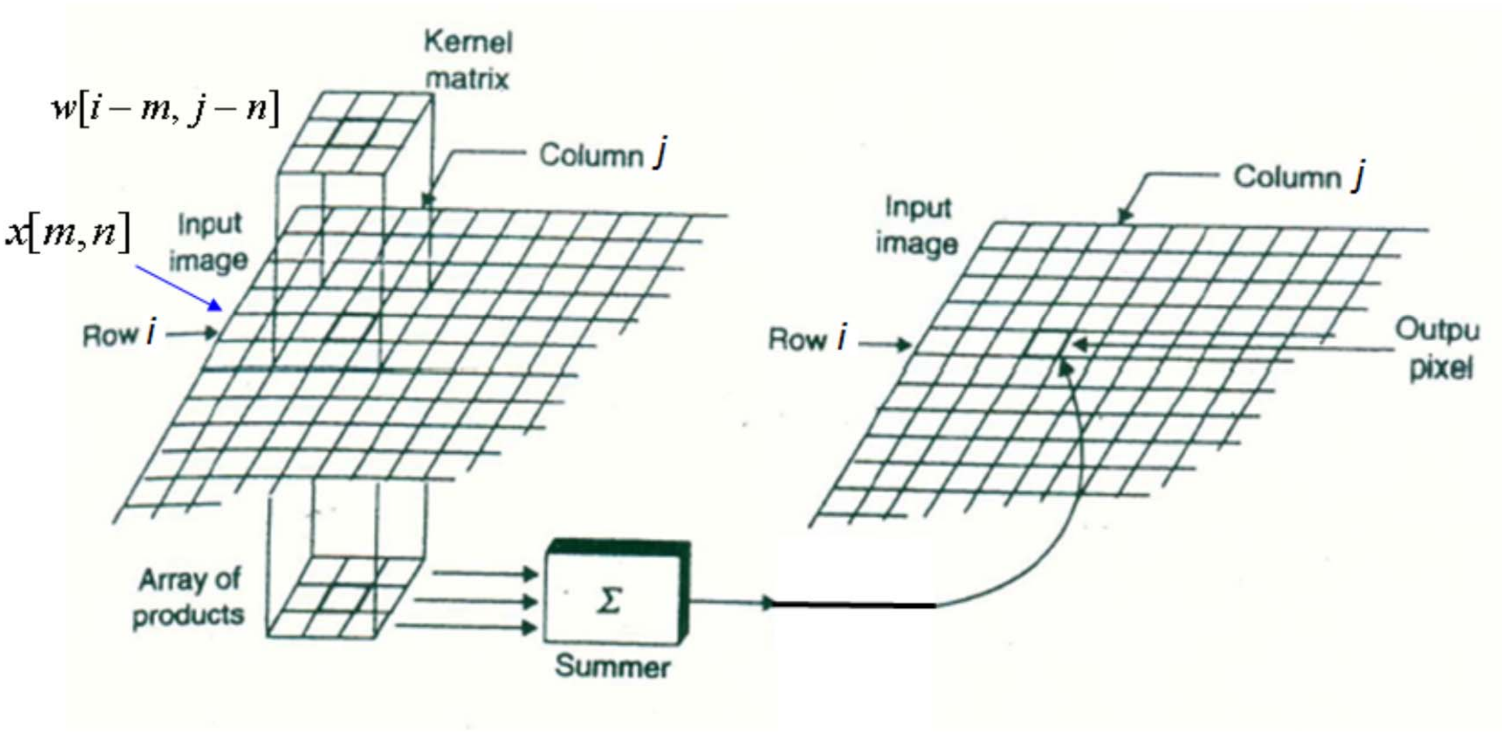
CNN called LeNet by Yann LeCun (1998)

[1] Y. LeCun et al., Gradient-based Learning Applied to Document Recognition, Proceedings of the IEEE, Vol. 86, pp. 2278-2324, 1998

Convolutional neural network

- Convolution
  - Computing the responses at hidden nodes is equivalent to convoluting the input image x with a learned filter w

$$net[i,j] = (x * w)[i,j] = \sum_m \sum_n x[m,n] w[i-m, j-n]$$



Convolutional neural network

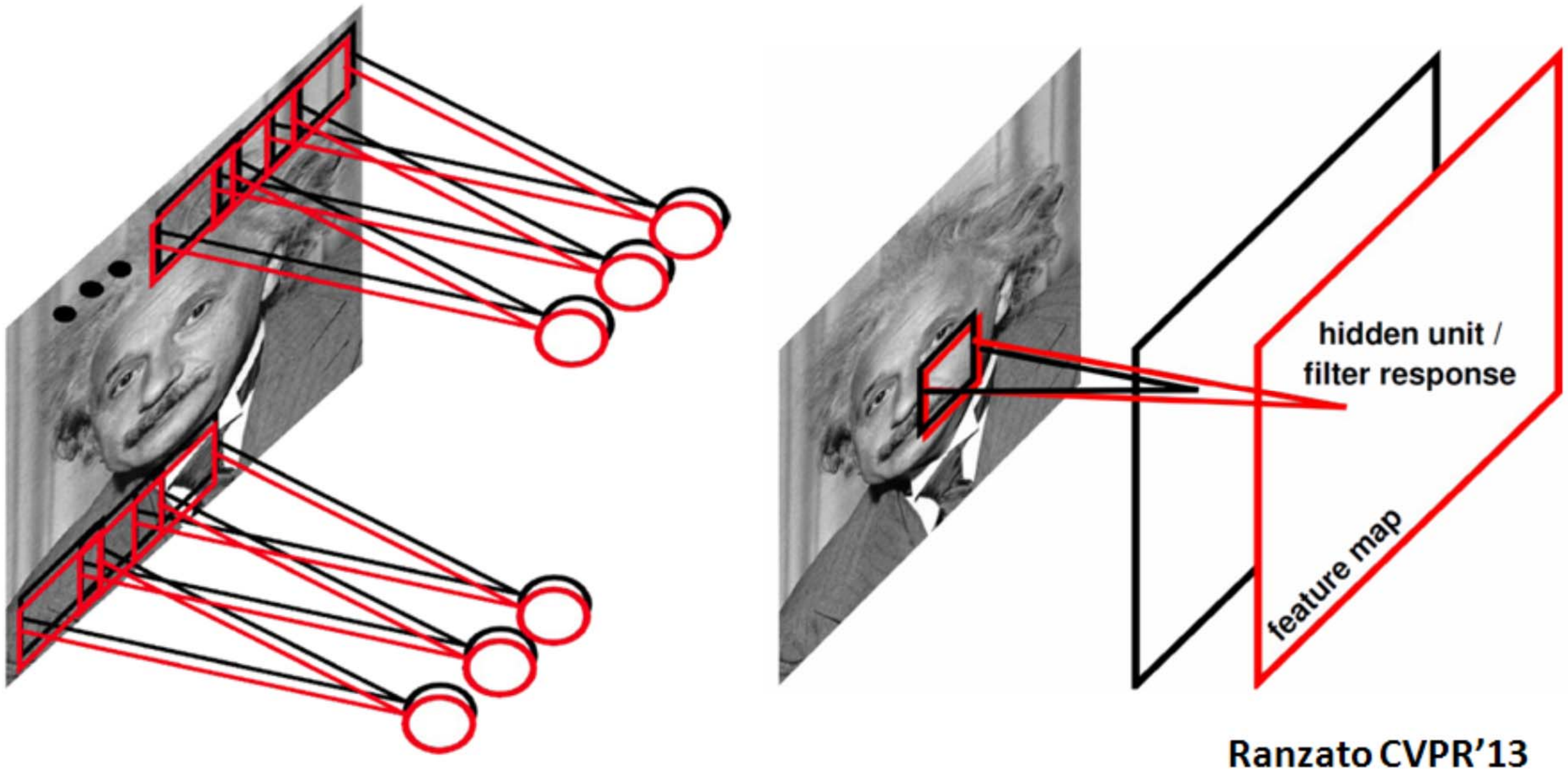
- Downsampled convolution layer (optional)
  - To reduce computational cost, we may want to skip some positions of the filter and sample only every s pixels in each direction. A downsampled convolution function is defined as

$$net(i, j) = (\mathbf{x} * \mathbf{w})[i \times s, j \times s]$$

  - s is referred as the stride of this downsampled convolution
  - Also called as strided convolution

Convolutional neural network

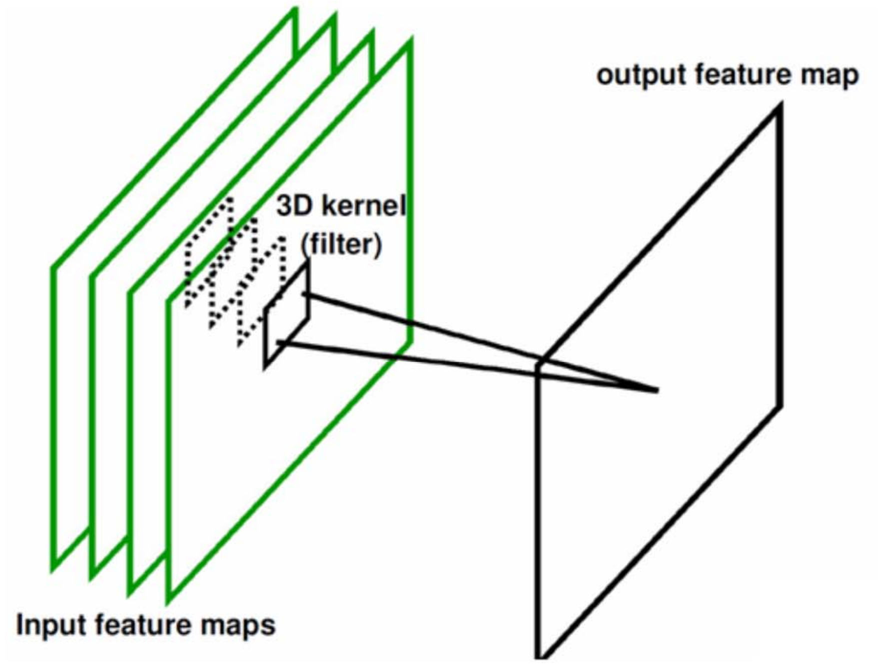
- Multiple filters
  - Multiple filters generate multiple feature maps
  - Detect the spatial distributions of multiple visual patterns

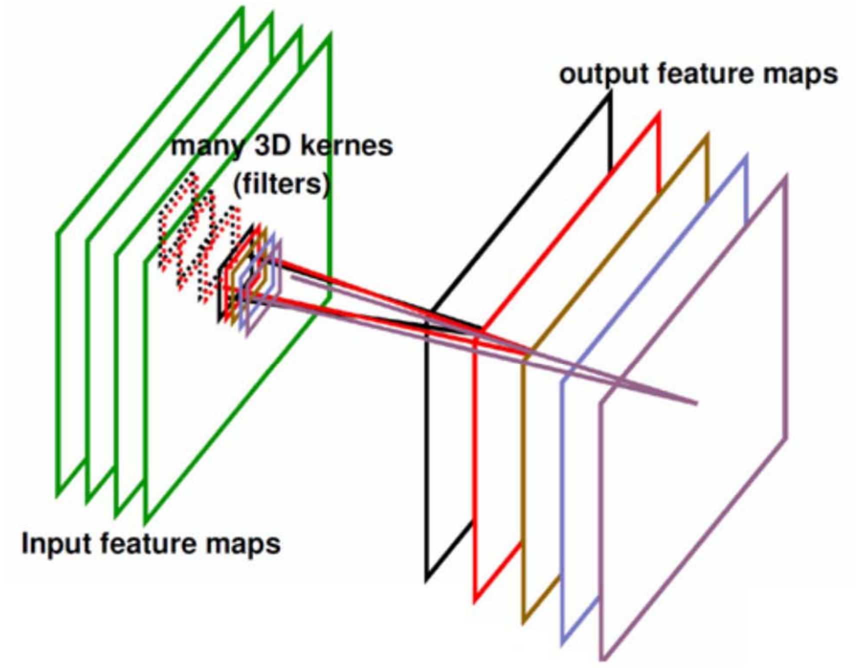


Ranzato CVPR'13

Convolutional neural network

- 3D filtering when input has multiple feature maps

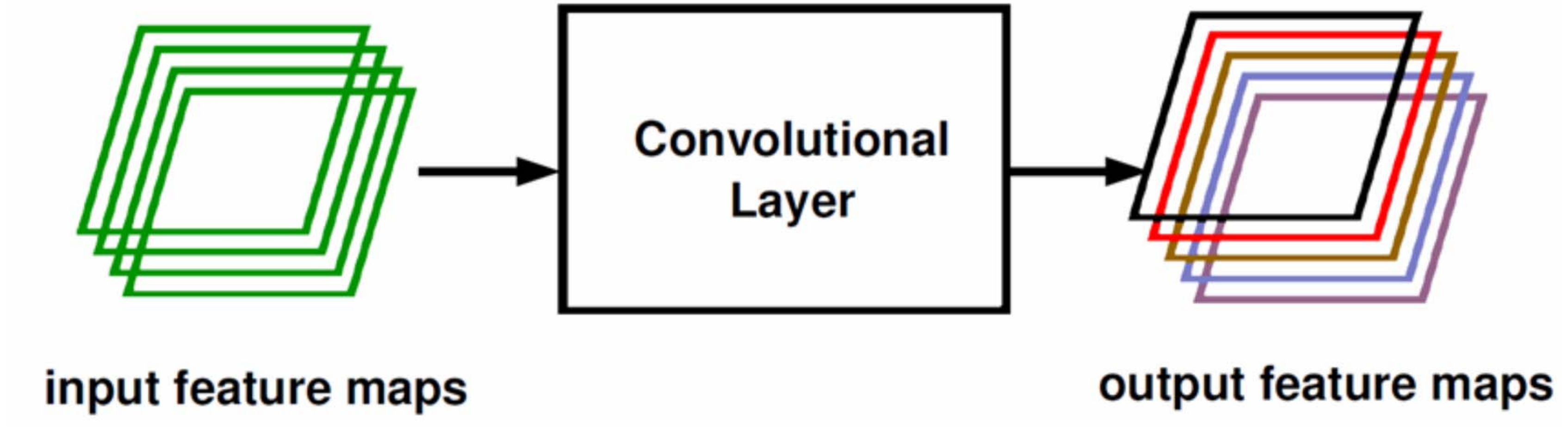




Ranzato CVPR'13

Convolutional neural network

- Convolutional layer



Ranzato CVPR'13

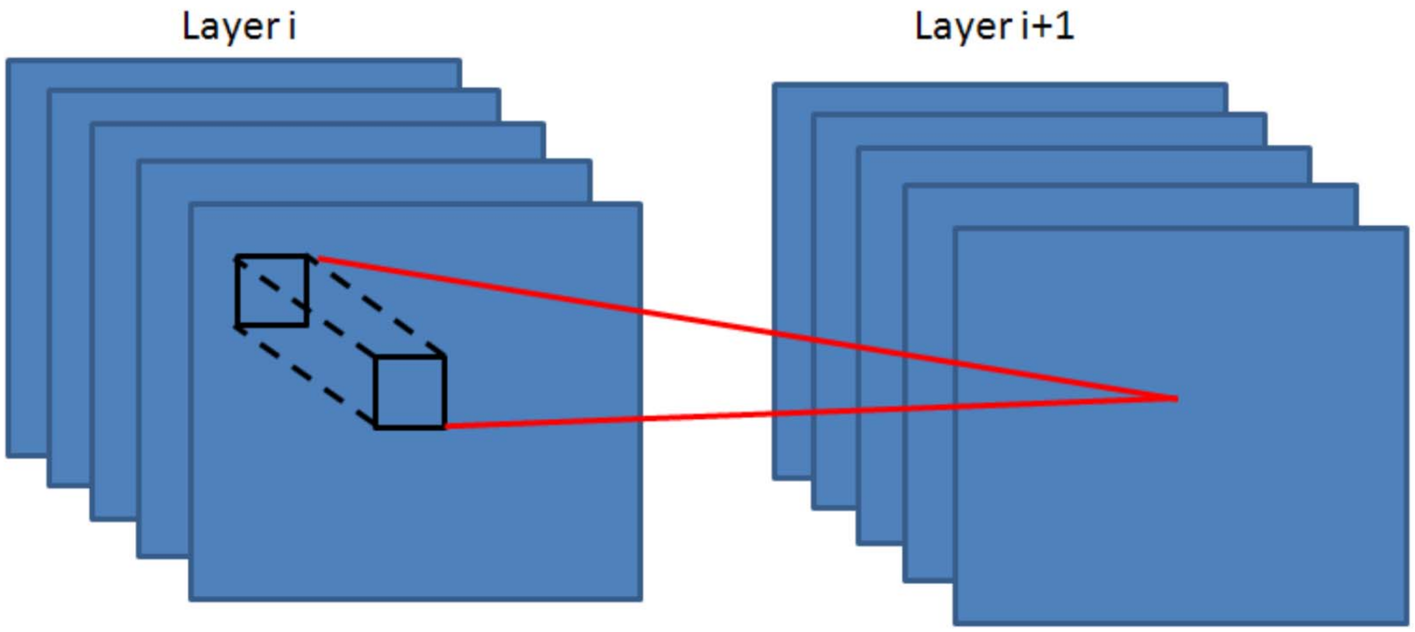
Convolutional neural network

- To the convolution responses, we then perform nonlinear activation
  - ReLU
  - Tanh
  - Sigmoid
  - Leaky ReLU
  - Softplus

Convolutional neural network

- Local contrast normalization (optional)
  - Normalization can be done within a neighborhood along both spatial and feature dimensions

$$h_{i+1,x,y,k} = \frac{h_{i,x,y,k} - m_{i,N(x,y,k)}}{\sigma_{i,N(x,y,k)}}$$

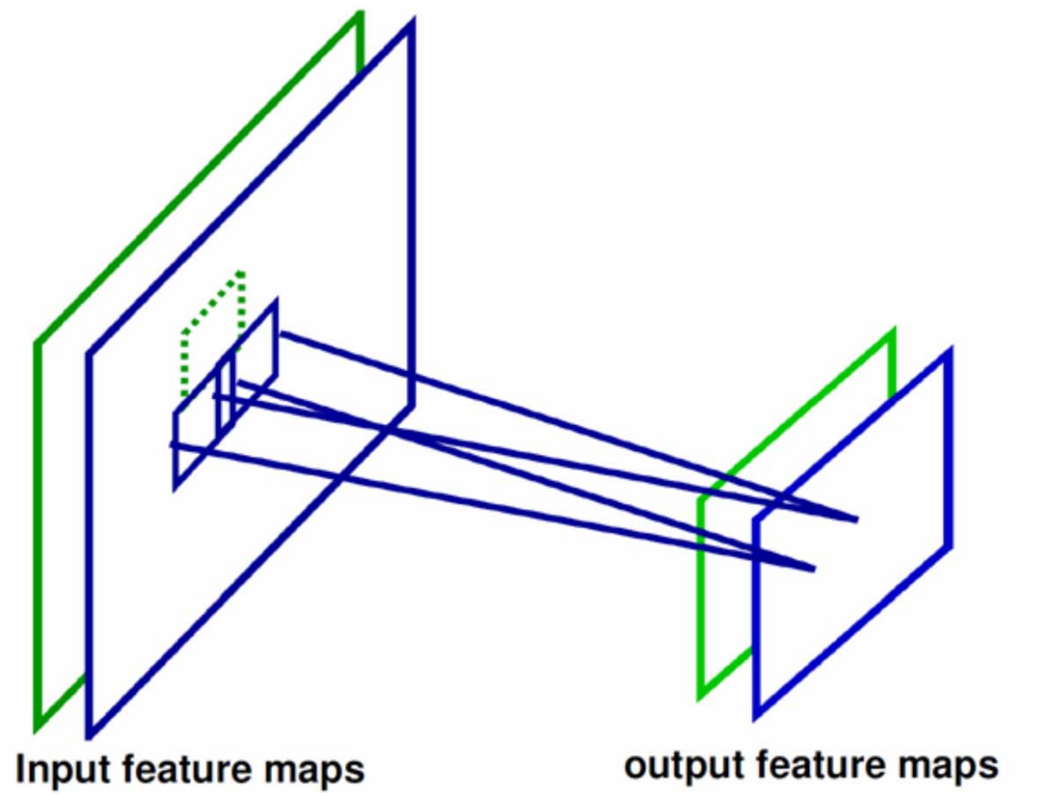


Convolutional neural network

- Then, we perform pooling
  - Max-pooling partitions the input image into a set of rectangles, and for each sub-region, outputs the maximum value
  - Non-linear down-sampling
  - The number of output maps is the same as the number of input maps, but the resolution is reduced
  - Reduce the computational complexity for upper layers and provide a form of translation invariance
  - Average pooling can also be used

Convolutional neural network

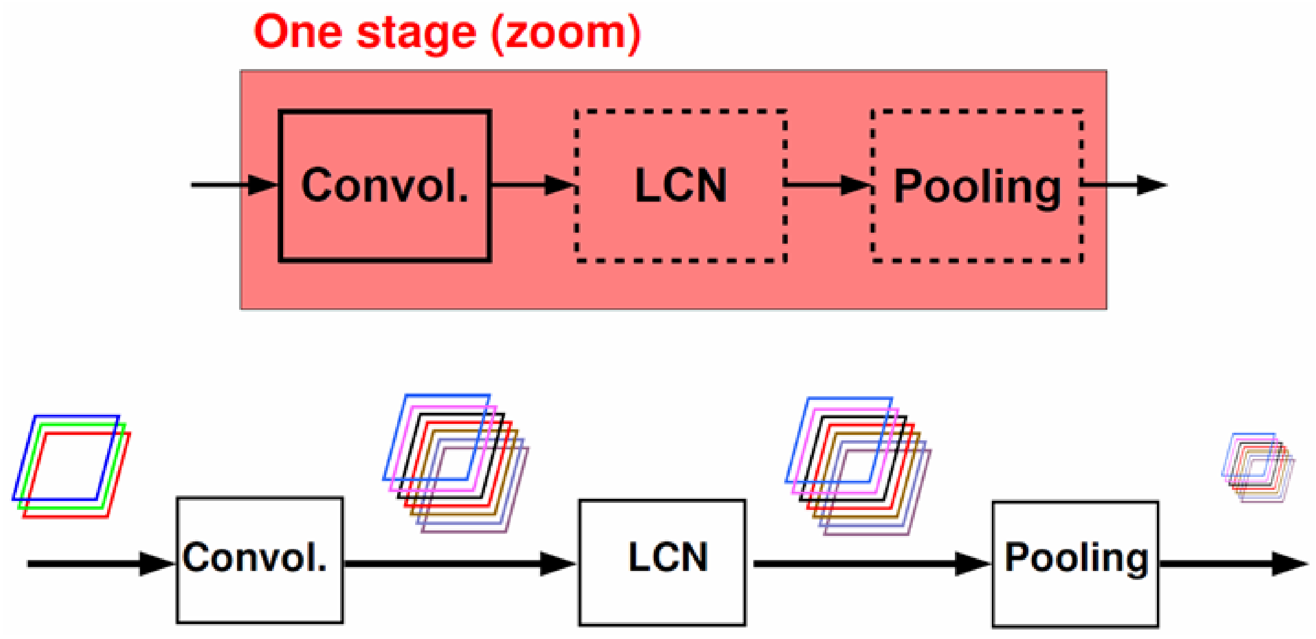
- Then, we perform pooling



Ranzato CVPR'13

Convolutional neural network

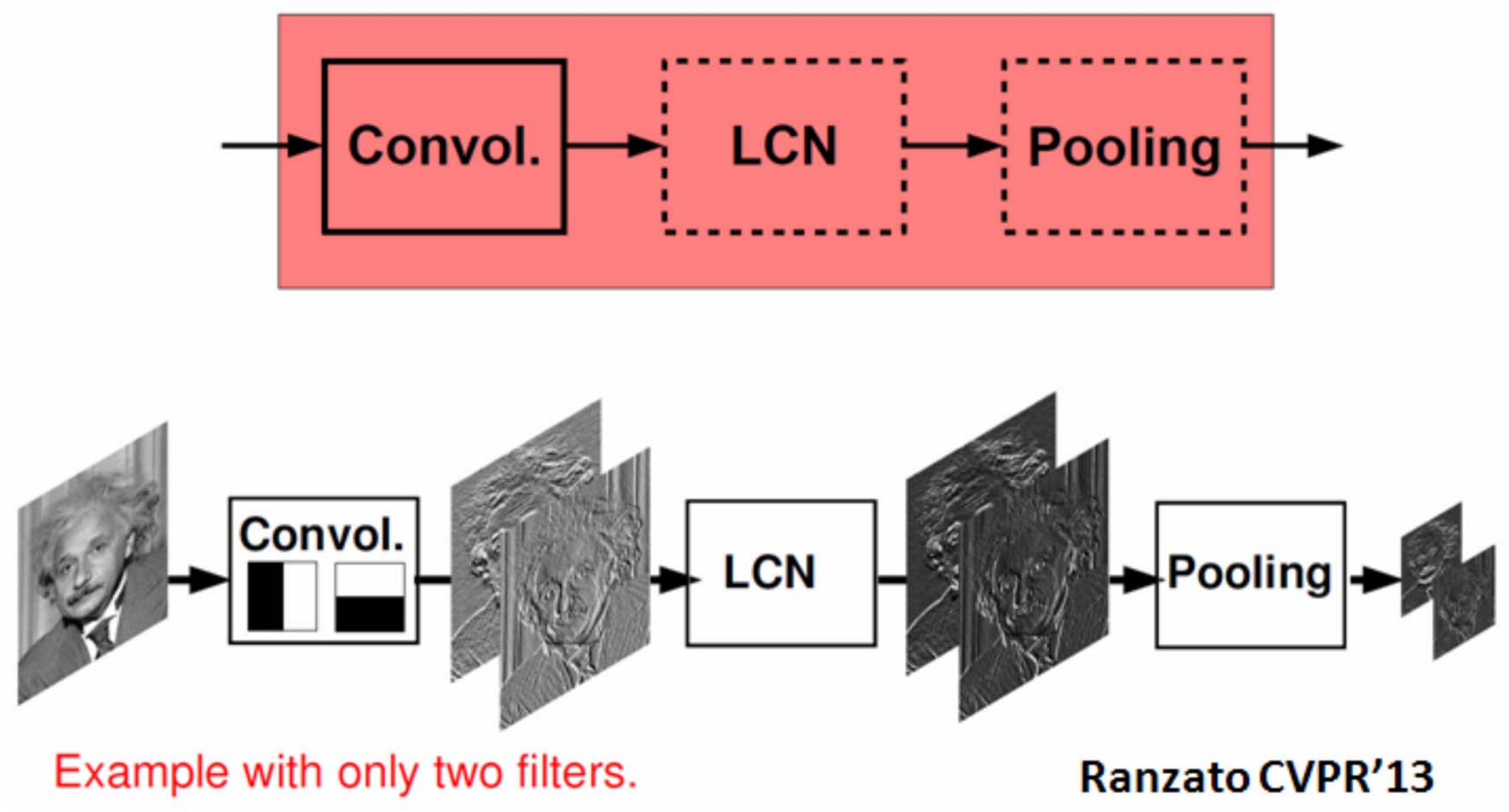
- Typical architecture of CNN
  - Convolutional layer increases the number of feature maps
  - Pooling layer decreases spatial resolution
  - LCN and pooling are optional at each stage



Ranzato CVPR'13

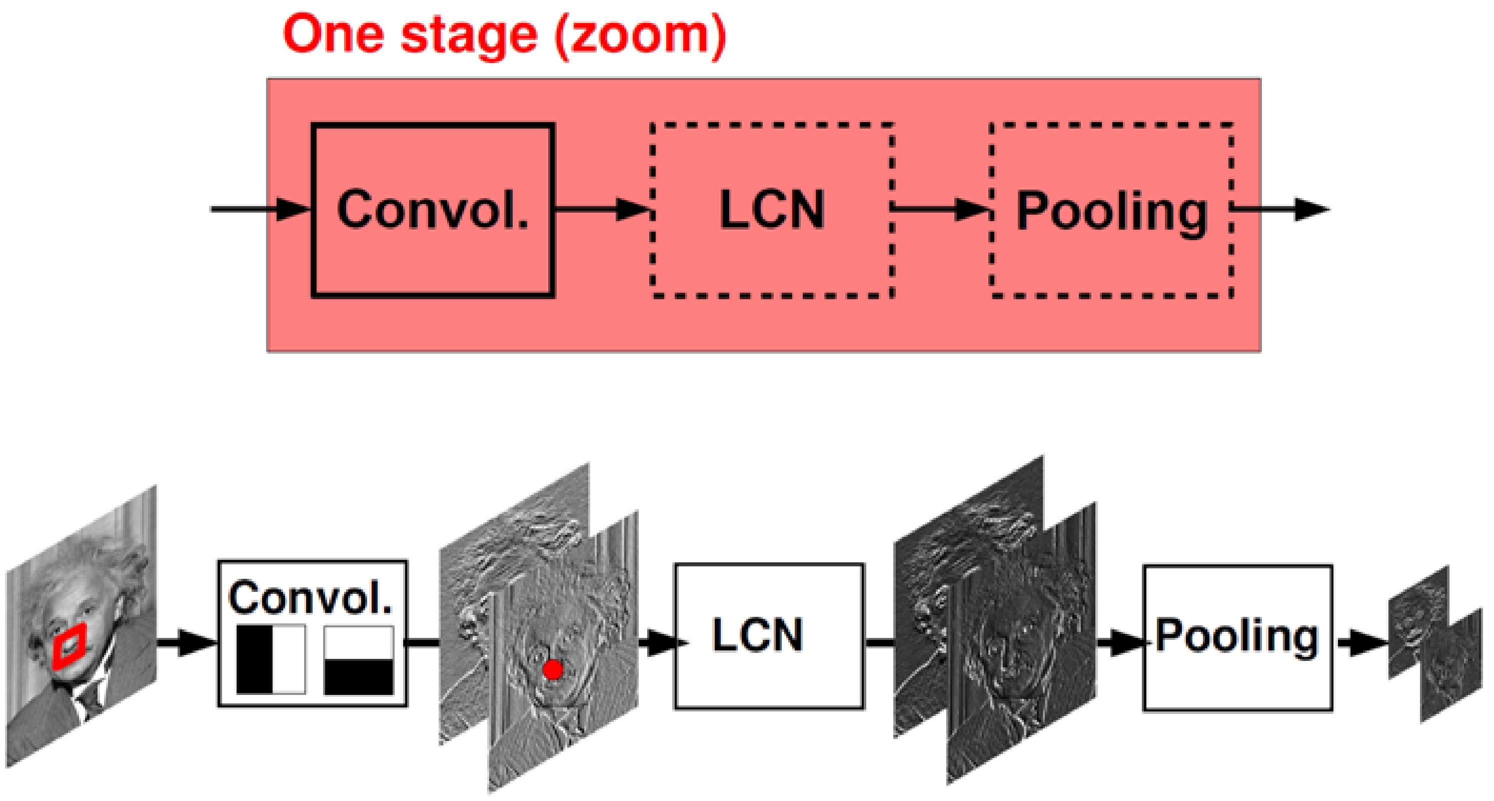
Convolutional neural network

- Typical architecture of CNN



Convolutional neural network

- Typical architecture of CNN

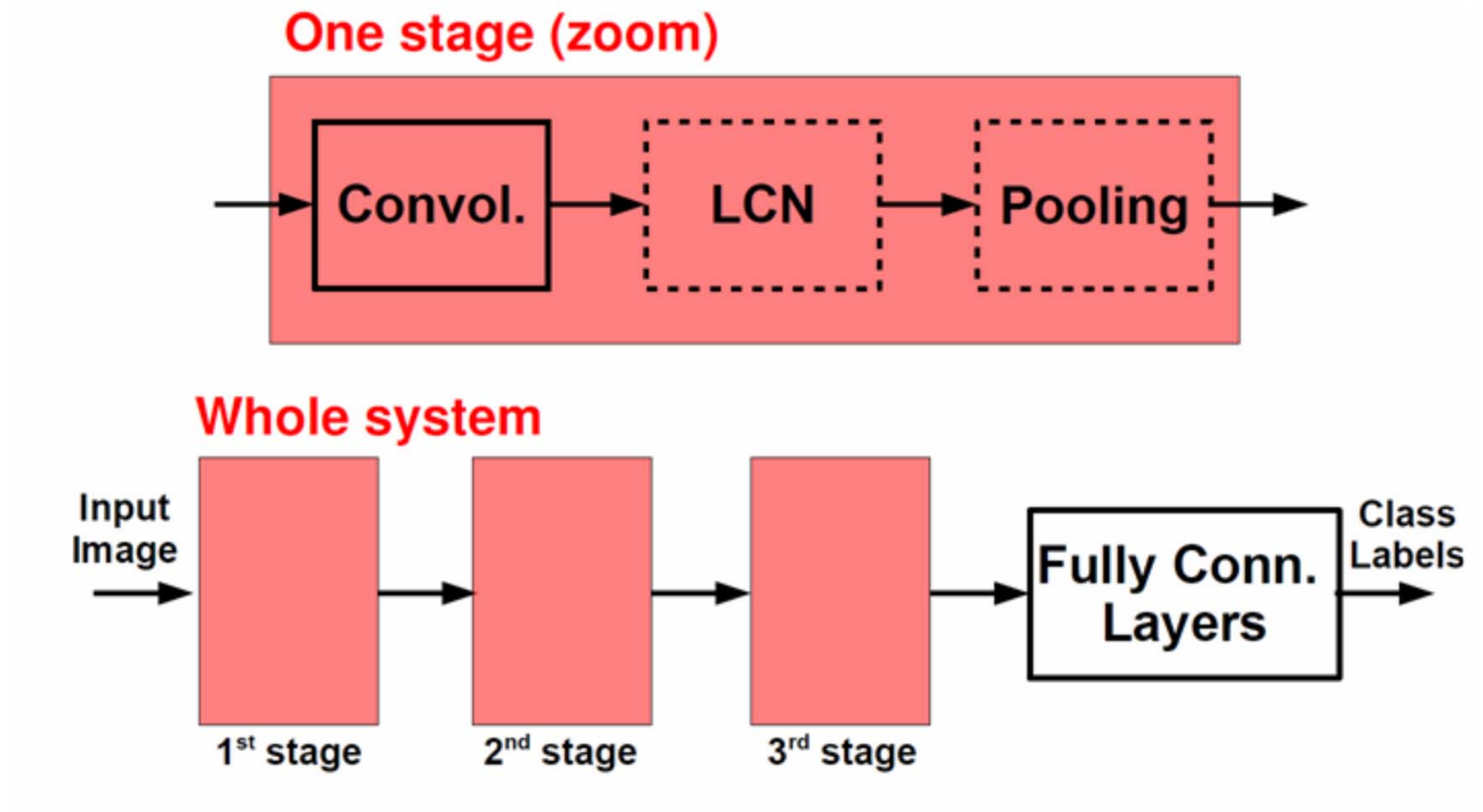


A hidden unit in the first hidden layer is influenced by a small neighborhood (equal to size of filter).

Ranzato CVPR'13

Convolutional neural network

- Typical architecture of CNN

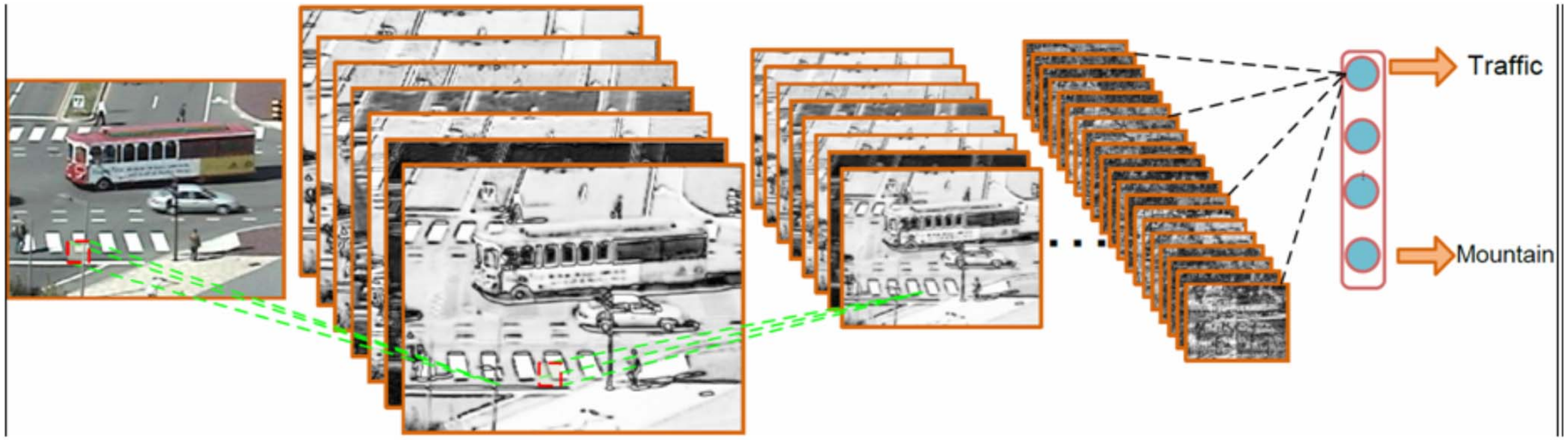


After a few stages, residual spatial resolution is very small.
We have learned a descriptor for the whole image.

Ranzato CVPR'13

Convolutional neural network

- Typical architecture of CNN



Convolution Pooling

Convolutional neural network

- Some notes about the CNN layers in most recent net architectures
  - Spatial pooling (such as max pooling) is not recommended now. It is usually replaced by a strided convolution, allowing the network to learn its own spatial downsampling
  - Fully connected layers are not recommended now; instead, the last layer is replaced by global average pooling (for classification problems, the number of feature map channels of the last layer should be the same as the number of classes

Convolutional neural network

- Example:
  - Train a digit classification model (LeNet) and then test it

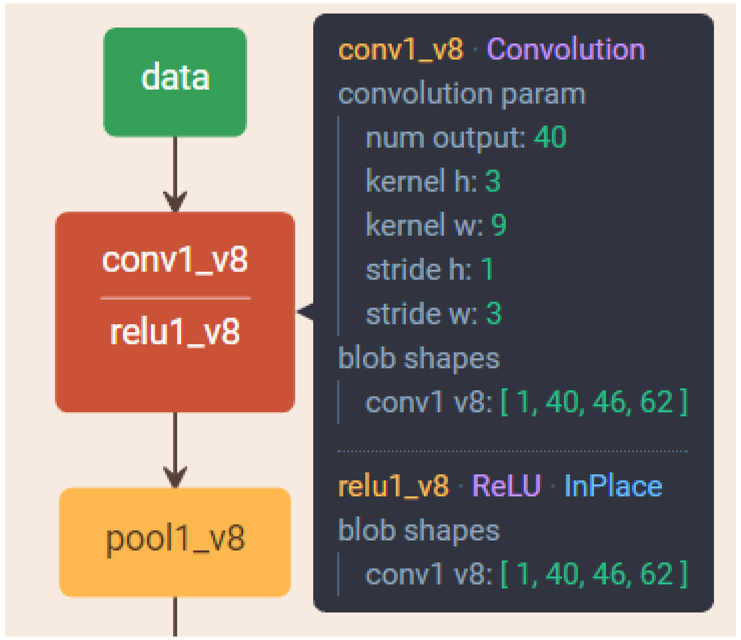
Finish this exercise in lab session

Convolutional neural network

- Opensource platforms for CNN
  - CAFFE official, http://caffe.berkeleyvision.org/
  - Tensorflow, https://www.tensorflow.org/
  - Pytorch, www.pytorch.org/
  - MatConvNet, http://www.vlfeat.org/matconvnet/
  - Theano, http://deeplearning.net/software/theano/

Convolutional neural network

- An online tool for network architecture visualization
  - http://ethereon.github.io/netscope/quickstart.html
  - Network architecture conforms to the CAFFE prototxt format
  - The parameter settings and the output dimension of each layer can be conveniently observed



conv1_v8 · Convolution
convolution param
num output: 40
kernel h: 3
kernel w: 9
stride h: 1
stride w: 3
blob shapes
conv1 v8: [1, 40, 46, 62]

relu1_v8 · ReLU · InPlace
blob shapes
conv1 v8: [1, 40, 46, 62]

Outline

- Basic concepts
- Linear model
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
  - AlexNet
  - NIN
  - GoogLeNet
  - ResNet
  - DenseNet
- CNN for object detection

AlexNet (NIPS 2012)

- AlexNet: CNN for object recognition on ImageNet challenge
  - Trained on one million images of 1000 categories collected from the web with two GPU. 2GB RAM on each GPU. 5GB of system memory
  - Training lasts for one week
  - Google and Baidu announced their new visual search engines with the same technology six months after that
  - Google observed that the accuracy of their visual search engine was doubled

[1] A. Krizhevsky *et al.*, ImageNet classification with deep convolutional neural networks, in Proc. NIPS, 2012

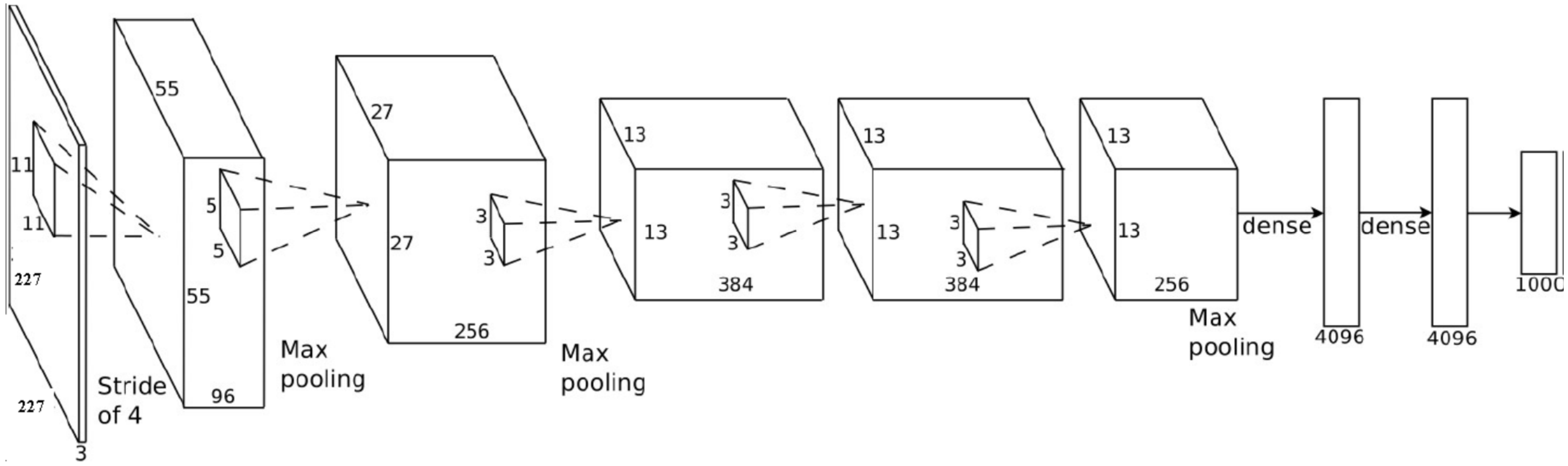
AlexNet (NIPS 2012)

- ImageNet
  - http://www.image-net.org/

1000 object classes that we recognize
poster created by Fengjun Lv using VIPBase
images courtesy of ImageNet (http://www.image-net.org/challenges/LSVRC/2010/index)

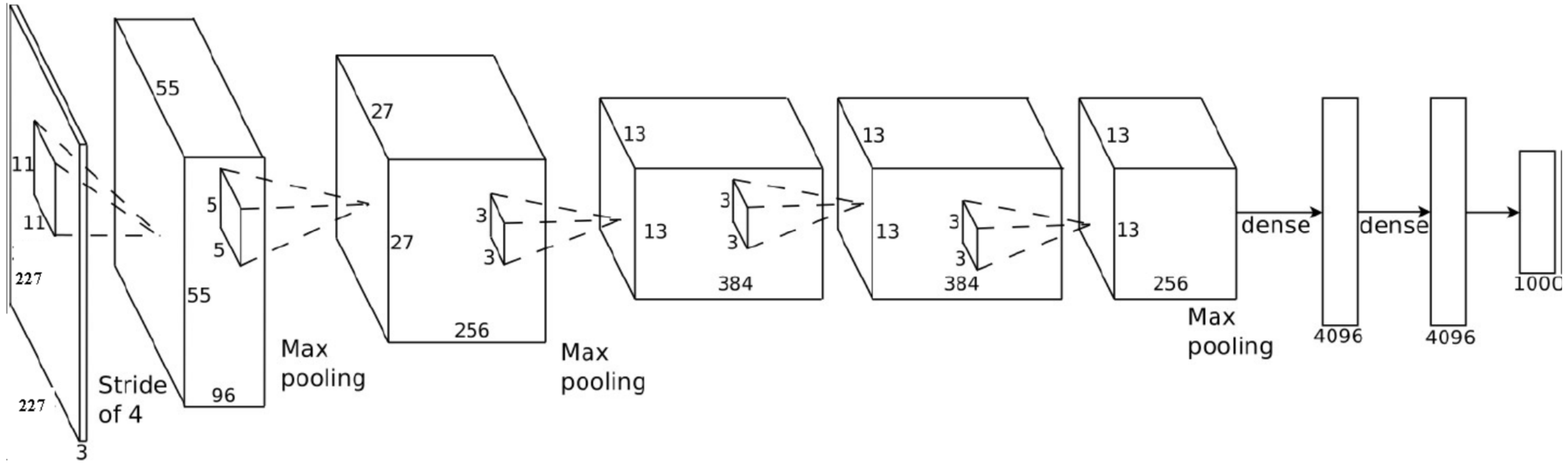
AlexNet (NIPS 2012)

- Architecture of AlexNet
  - 5 convolutional layers and 2 fully connected layers for learning features
  - Max-pooling layers follow first, second, and fifth convolutional layers



AlexNet (NIPS 2012)

- Architecture of AlexNet
  - The first time deep model is shown to be effective on large scale computer vision task
  - The first time a very large scale deep model is adopted
  - GPU is shown to be very effective on this large deep model

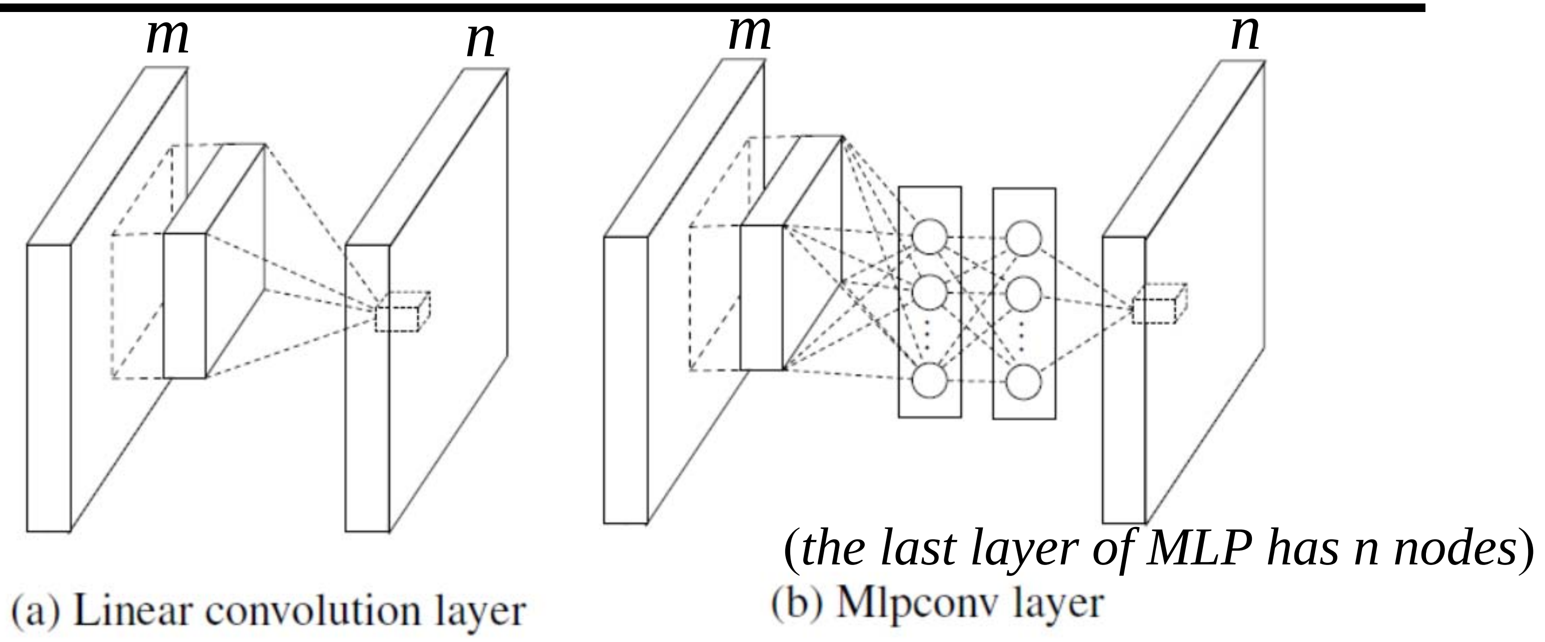


Network In Network (NIN, ICLR 2014)

- Main idea of NIN
  - Conventional convolutional layers uses linear filters followed by a nonlinear activation function to abstract the information within a receptive field
  - Instead, NIN uses micro neural networks with more complex structures to abstract the data within the receptive field
  - The feature maps are obtained by sliding the micro network over the input in a similar manner as CNN
  - Moreover, they use global average pooling over feature maps in the classification layer, which is easier to interpret and less prone to overfitting than traditional fully connected layers

[1] M. Liu *et al*., Network in network, in Proc. ICLR, 2014

Network In Network (NIN, ICLR 2014)

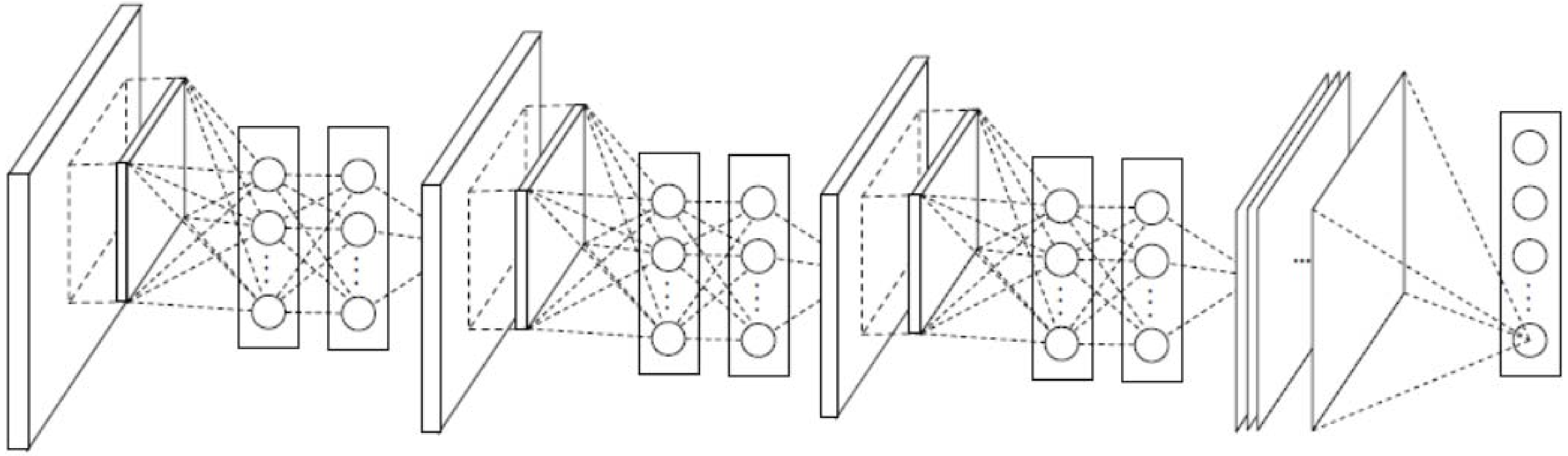


(*the last layer of MLP has n nodes*)

(a) Linear convolution layer (b) Mlpconv layer

- Comparison of linear convolution layer and mlpconv layer
  - Both the layers map the local receptive field to an output feature vector
  - The mlpconv layer maps the input local patch to the output feature vector with a multilayer perceptron (MLP) consisting of multiple fully connected layers with nonlinear activation functions

Network In Network (NIN, ICLR 2014)



The overall structure of NIN. The last layer is the global average pooling

- More about global average pooling
  - Fully connected layers are prone to overfitting
  - If there are c classes, the last MLP layer should output c feature maps, one feature map for each corresponding category of the classification task
  - Take the average of each feature map to get a c dimensional vector for softmax classification

Network In Network (NIN, ICLR 2014)

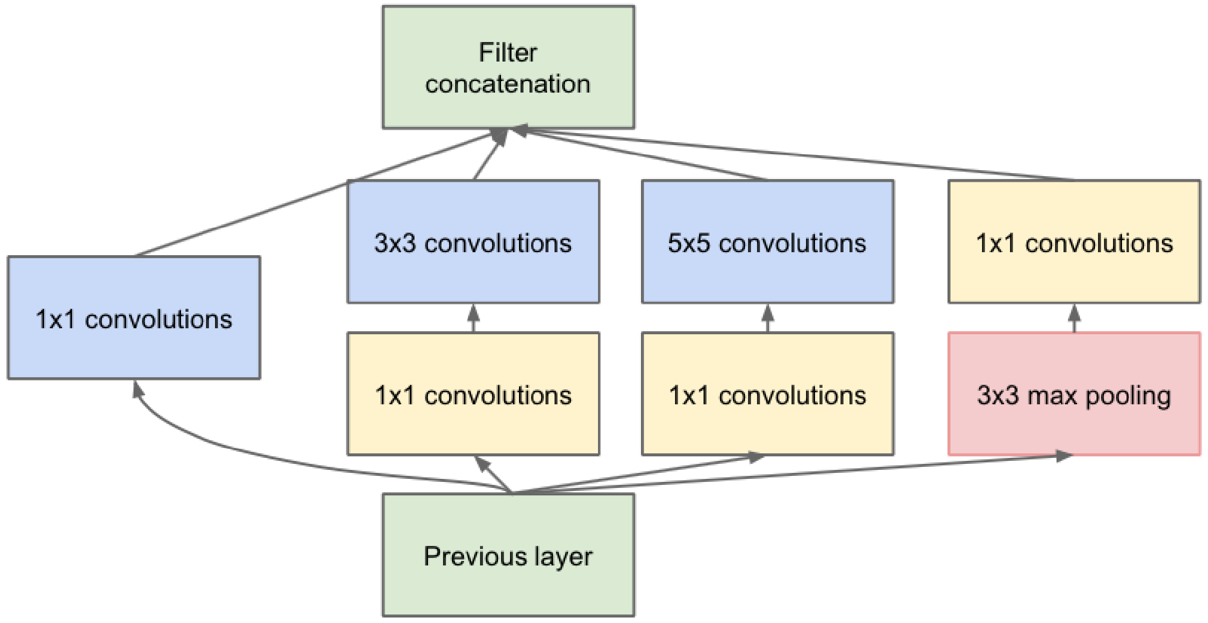
- NIN can be implemented with conventional convolutional layers

For a mlpconv layer, suppose that the input feature map is of the size $m \times m \times 32$, the expected output feature map is of the size $m \times m \times 64$, the receptive field is 5×5; the mlpconv layer has 2 hidden layers, whose node numbers are 16 and 32, respectively.

How to implement this mlpconv layer with convolutional layers?

GoogLeNet (CVPR 2015)

- Main idea: make the network deeper and wider, while keeping the number of parameters
- Inception module



[1] C. Szegedy *et al.*, Going deeper with convolutions, in Proc. CVPR, 2015

GoogLeNet (CVPR 2015)

- Many Inception modules can stack together to form a very deep network
- GoogLeNet refers to the version the authors submitted for the ILSVRC 2014 competition
  - This network consists 27 layers (including pooling layers)

ResNet (CVPR 2016 Best Paper)

- What is the problem of stacking more layers using conventional CNNs?
  - Vanishing gradient, which can hamper the convergence
  - Accuracy get saturated, and then degraded

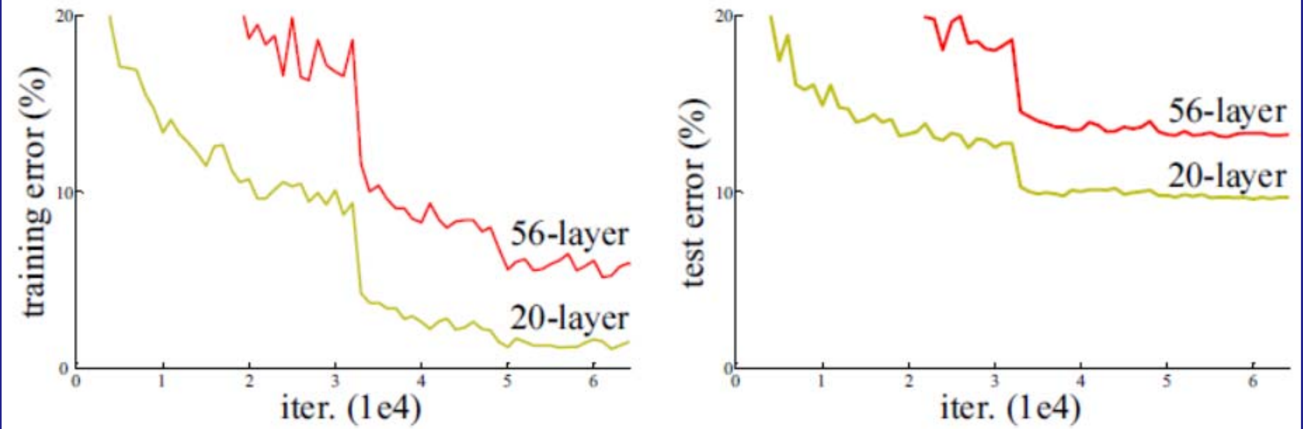


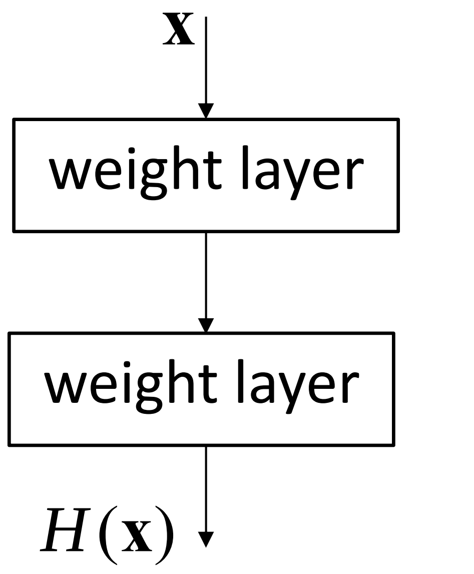
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error.

[1] K. He *et al.*, Deep residual learning for image recognition, in Proc. CVPR, 2016

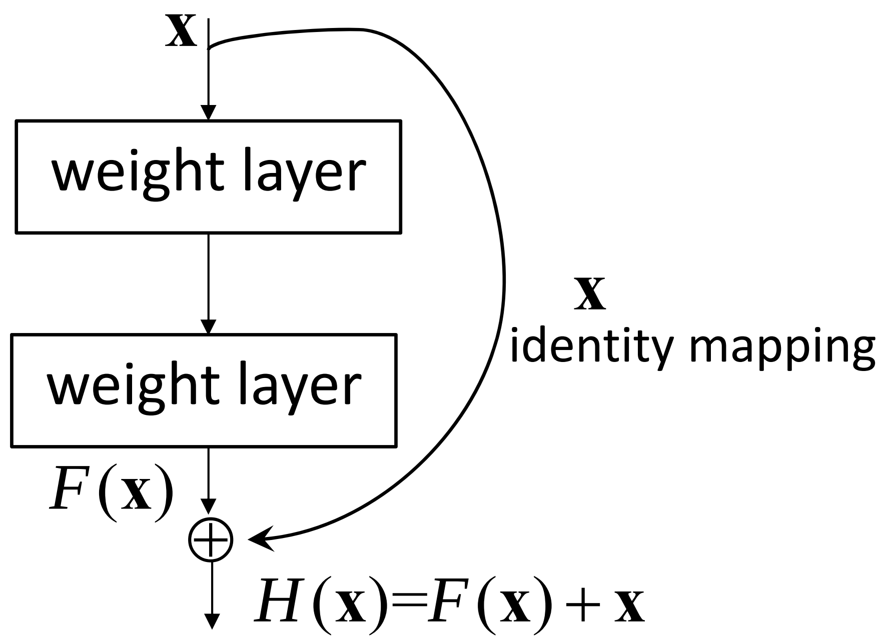
ResNet (CVPR 2016 Best Paper)

Is there any better way to design deeper networks?

Answer: Residual learning



Conventional CNN



Residual block

ResNet (CVPR 2016 Best Paper)

- It is easier to optimize the residual mapping ($F(\mathbf{x})$) than to optimize the original mapping ($H(\mathbf{x})$)
- Identity mapping is implemented by shortcut
- A residual learning block is defined as,

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

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers
$F+\mathbf{x}$ is performed by a shortcut connection and element-wise addition

Note: If the dimensions of F and $\mathbf{x}$ are not equal (usually caused by changing the numbers of input and output channels), a linear projection W_s (implemented with 1*1 convolution) is performed on $\mathbf{x}$ to match the dimensions,

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

ResNet (CVPR 2016 Best Paper)

- It is easier to optimize the residual mapping ($F(\mathbf{x})$) than to optimize the original mapping ($H(\mathbf{x})$)
- Identity mapping is implemented by shortcut
- A residual learning block is defined as,

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

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers
$F+\mathbf{x}$ is performed by a shortcut connection and element-wise addition

I highly recommend you to take a look the prototxt file of ResNet *(https://github.com/KaimingHe/deep-residual-networks)*

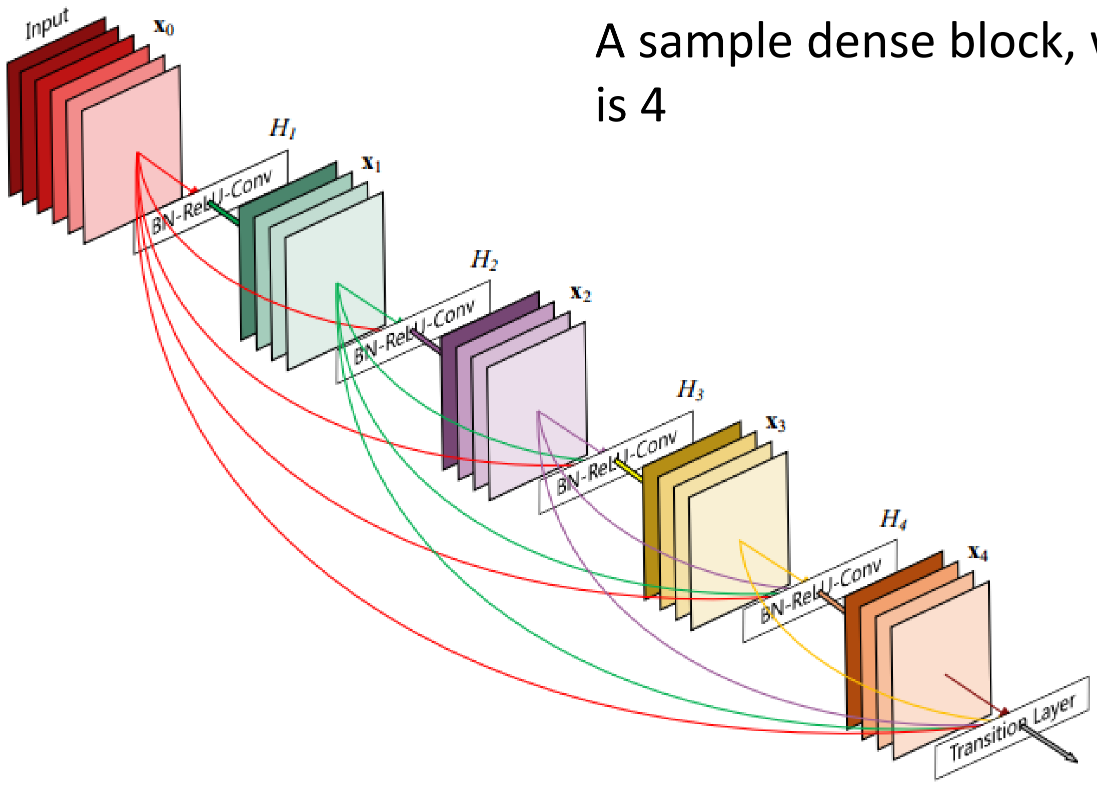
DenseNet (CVPR 2017 Best Paper)

- Highly motivated by ResNet
- A DenseNet comprises “dense blocks” and transition layers
  - Within a dense block, connect all layers with each other in a feed-forward fashion
  - In contrast to ResNet, DenseNet combine features by concatenating them
  - The number of output feature maps of each layer is set as a constant within a dense block and is called as “growth rate”
  - Between two blocks, there is a transition layer, consisting of batch normalization, 1×1 convolution, and average pooling

[1] G. Huang *et al.*, Densely connected convolutional networks, in Proc. CVPR, 2017

DenseNet (CVPR 2017 Best Paper)

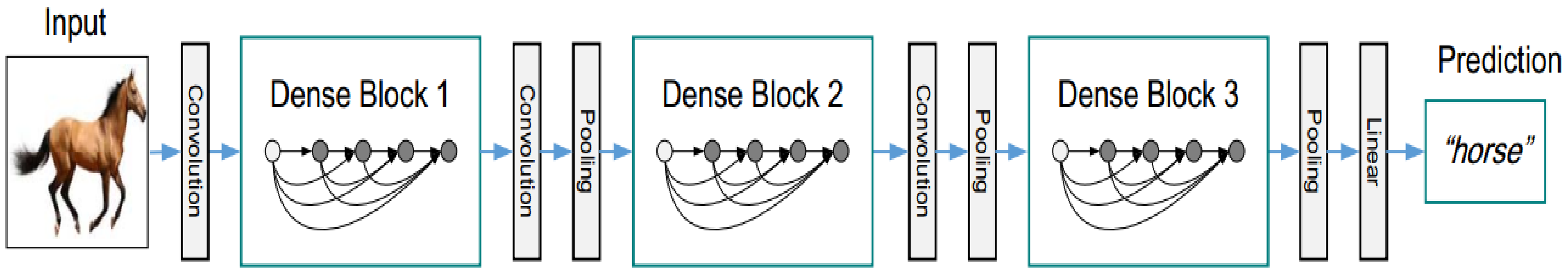
- Highly motivated by ResNet
- A DenseNet comprises "dense blocks" and transition layers



A sample dense block, whose growth rate is 4

DenseNet (CVPR 2017 Best Paper)

- Highly motivated by ResNet
- A DenseNet comprises "dense blocks" and transition layers



A sample DenseNet with three dense blocks

DenseNet (CVPR 2017 Best Paper)

- Highly motivated by ResNet
- A DenseNet comprises “dense blocks” and transition layers
- More details about DenseNet design
  - Bottleneck layers. A 1×1 convolution layer can be introduced as bottleneck layer before each 3×3 convolution to reduce the number of input feature maps, and thus to improve computational efficiency
  - Compression. If a dense block contains m feature maps, we let the following transition layer generate θm output feature maps where $0<\theta\leq1$ is referred to as the compression factor

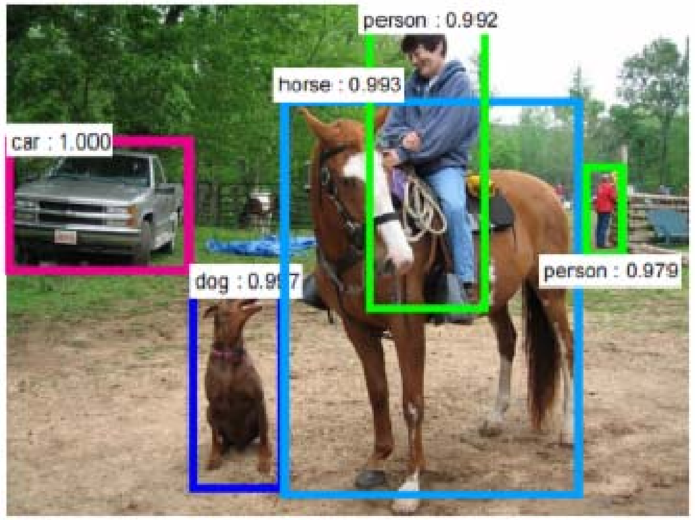
Outline

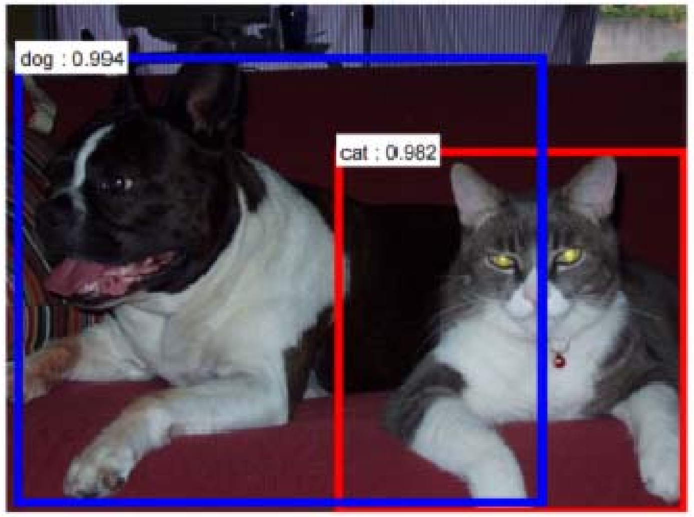
- Basic concepts
- Linear model
- Neural network
- Convolutional neural network (CNN)
- Modern CNN architectures
- CNN for object detection

Background

- Detection is different from classification
  - An image classification problem is predicting the label of an image among the predefined labels; It assumes that there is **single** object of interest in the image and it covers a significant portion of image
  - Detection is about not only finding the class of object but also localizing the extent of an object in the image; the object can be lying anywhere in the image and can be of any size (scale)

Background





Multiple objects detection

Background

- Traditional methods of detection involved using a block-wise orientation histogram (SIFT or HOG) feature which could not achieve high accuracy in standard datasets such as PASCAL VOC; these methods encode a very low level characteristics of the objects and therefore are not able to distinguish well among the different labels
- Deep learning based methods have become the state-of-the-art in object detection in image; they construct a representation in a hierarchical manner with increasing order of abstraction from lower to higher levels of neural network

Background

- Recent developments of CNN based object detectors
  - R-CNN (CVPR 2014)
  - Fast-RCNN (ICCV 2015)
  - Faster-RCNN (NIPS 2015)
  - Yolo (CVPR 2016)
  - SSD (ECCV 1016)
  - Yolov2 (CVPR 2017)

R-CNN

- Brute-force idea
  - One could perform detection by carrying out a classification on different sub-windows or patches or regions extracted from the image. The patch with high probability will not only the class of that region but also implicitly gives its location too in the image
  - One brute force method is to run classification on all the sub-windows formed by sliding different sized patches (to cover each and every location and scale) all through the image

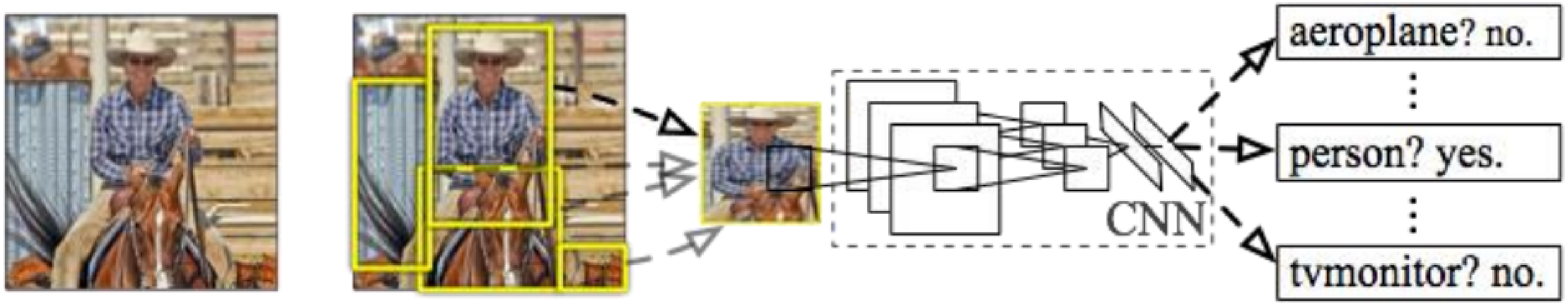
Quite Slow!!!

R-CNN

- R-CNN therefore uses an object proposal algorithm (selective search) in its pipeline which gives out a number (~2000) of TENTATIVE object locations
- These object regions are warped to fixed sized (227X227) regions and are fed to a classification convolutional network which gives the individual probability of the region belonging to background and classes

R-CNN

Region-based Convolution Networks (R-CNNs)



Input image

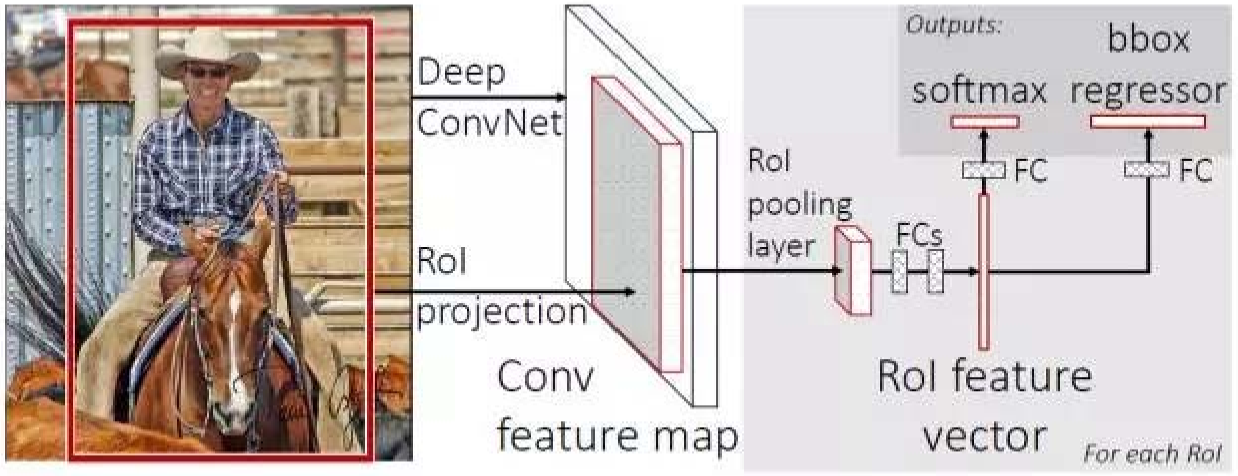
Extract region proposals (~2k / image)
e.g., selective search
[van de Sande, Uijlings et al.]

Compute CNN features on regions

Classify and refine regions

Fast-RCNN

- Compared to RCNN
  - A major change is a single network with two loss branches pertaining to soft-max classification and bounding box regression
  - This multitask objective is a salient feature of Fast-RCNN as it no longer requires training of the network independently for classification and localization



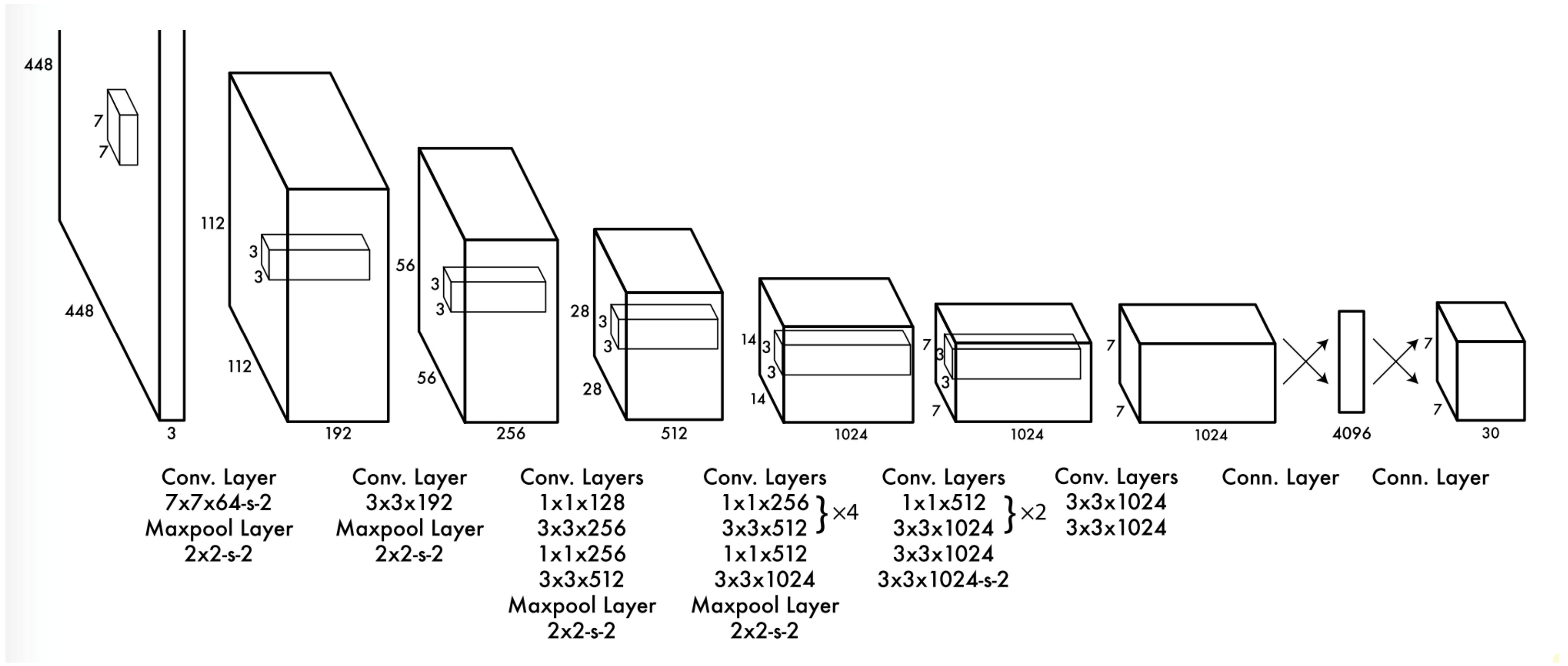
Faster-RCNN

- Compared to Fast-RCNN
  - Faster-RCNN replaces Selective Search with CNN itself for generating the region proposals (called RPN-region proposal network) which gives out tentative regions at almost negligible amount of time

Yolo (YoloV2) is a state-of-the-art method

- Yolo (You Only Look Once)
  - The major exceptional idea is that it tackles the object detection problem as a regression problem
  - A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation
  - The whole detection pipeline is a single network
  - It is extremely fast; With TitanX, it can process ~50 frames/s

Yolo (YoloV2) is a state-of-the-art method



Yolo (YoloV2) is a state-of-the-art method

- YoloV2
  - It is a quite recent extension for Yolo
  - It extends Yolo in the following aspects, batch normalization, high resolution classifier, convolutional with anchor boxes, fine-grained features, multi-scale training
  - The authors provide both Linux and Windows versions

Thanks for your attention