Dropout and Convolutional Neural Networks

Outline

- Deep learning algorithms without pre-training
- Dropout
- Convolutional Neural Networks
- Summary

Deep Learning without Pre-training?

- DBN and SDA with pre-training
  - To solve the vanishing gradient problem caused by backpropagation
- Learn properly with deep neural networks without pre-training, how?

Outline

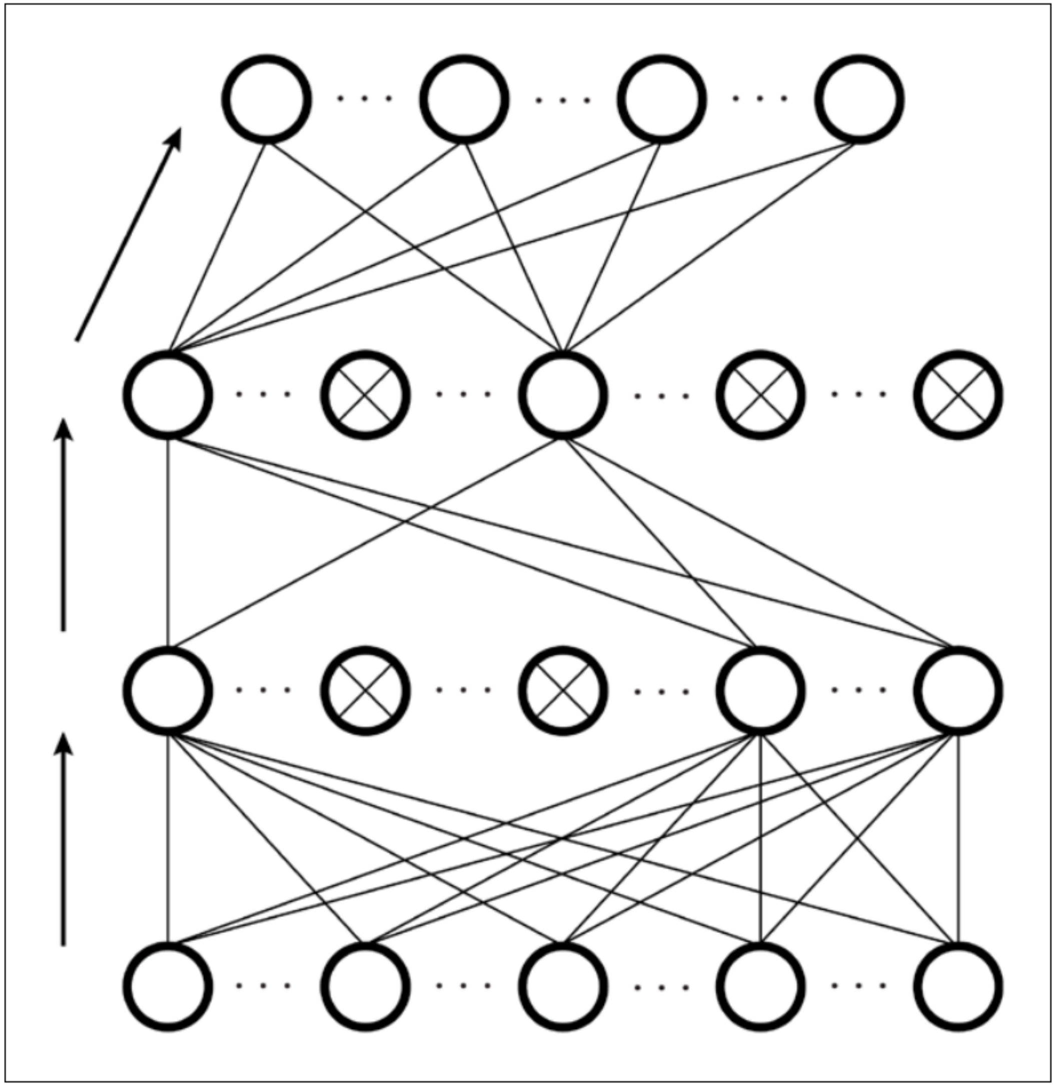
- Deep learning algorithms without pre-training
- Dropout
- Convolutional Neural Networks
- Summary

Dropout

- “If there's a problem with the network being tied densely, just force it to be sparse”
- The dropout algorithm
  - Some of the units are forcibly dropped while training
  - “Improving neural networks by preventing co adaptation of feature detectors” (Hinton, et. al. 2012, http://arxiv.org/pdf/1207.0580.pdf)
  - “Dropout: A Simple Way to Prevent Neural Networks from Overfitting” (Srivastava, et. al. 2014, https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf)

Dropout

- Dropped units are interpreted as non-existent in the network
- Change the structure of the original neural network while training
  - Adding a dropout mask (binary mask)



Dropout

- DA and dropout look similar
  - Corrupting input data in DA also adds binary masks to the data when implemented
- Two differences
  - DA applies the mask only to units in the input layer, dropout applies it to units in the hidden layer or both
  - In DA, once the corrupt input data is generated, the data will be used throughout the whole training epochs, dropout masks will be generated in each layer in each iteration

Reasonable?

- The network is well trained with dropout because it puts more weights on the existing neurons to reflect the characteristics of the input data
  - Pre-training find weights automatically
- Dropout has a single demerit
  - Requires more training epochs than other algorithms to train and optimize the model
- Improvement of the activation function: to be more sparse

Activation: Rectifier

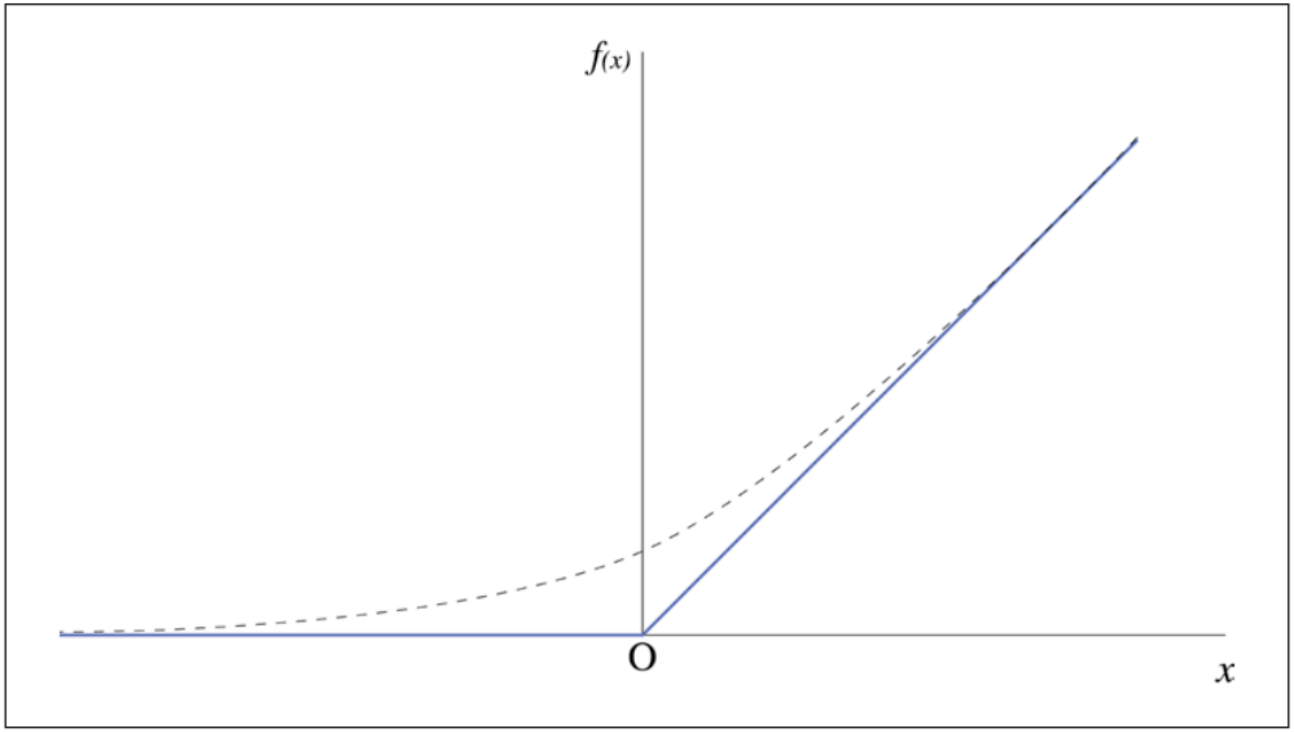
- A unit-applied rectifier is called a Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x) = \begin{cases} x & if\ x \geq 0 \\ 0 & otherwise \end{cases}$$

- broken line:

  softplus function

$$softplus(x) = in(1 + \exp(x))$$



ReLU

- Rectifier is far simpler than the sigmoid function and hyperbolic tangent
  - Time cost will reduce when calculate
- Derivative of the rectifier—which is necessary when calculating backpropagation errors—is also simple
  - Shorten the time cost
- Equation of the derivative

$$f'(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & otherwise \end{cases}$$

ReLU

- Since both the rectifier and the derivative of it are very sparse, the neural networks will be also sparse through training
- No vanishing gradient problem
  - Don't have the causal curves that the sigmoid function and hyperbolic tangent contain anymore

The Calculation

- h: ReLU activation function

$$Z_j = h\left(\sum_i w_{ji}x_i + b_j\right)$$

- With binary mask

$$Zj = h\left(\sum_i w_{ji}x_i + b_j\right)m_j$$

$$m_j \sim Bernoulli(1-p)$$

Smart Computing 智慧計算實驗室 SAIC

The Calculation

- Suppose we have $$a_j = \sum_i w_{ji} x_i + b_j$$
- Define delta, E: evaluation function

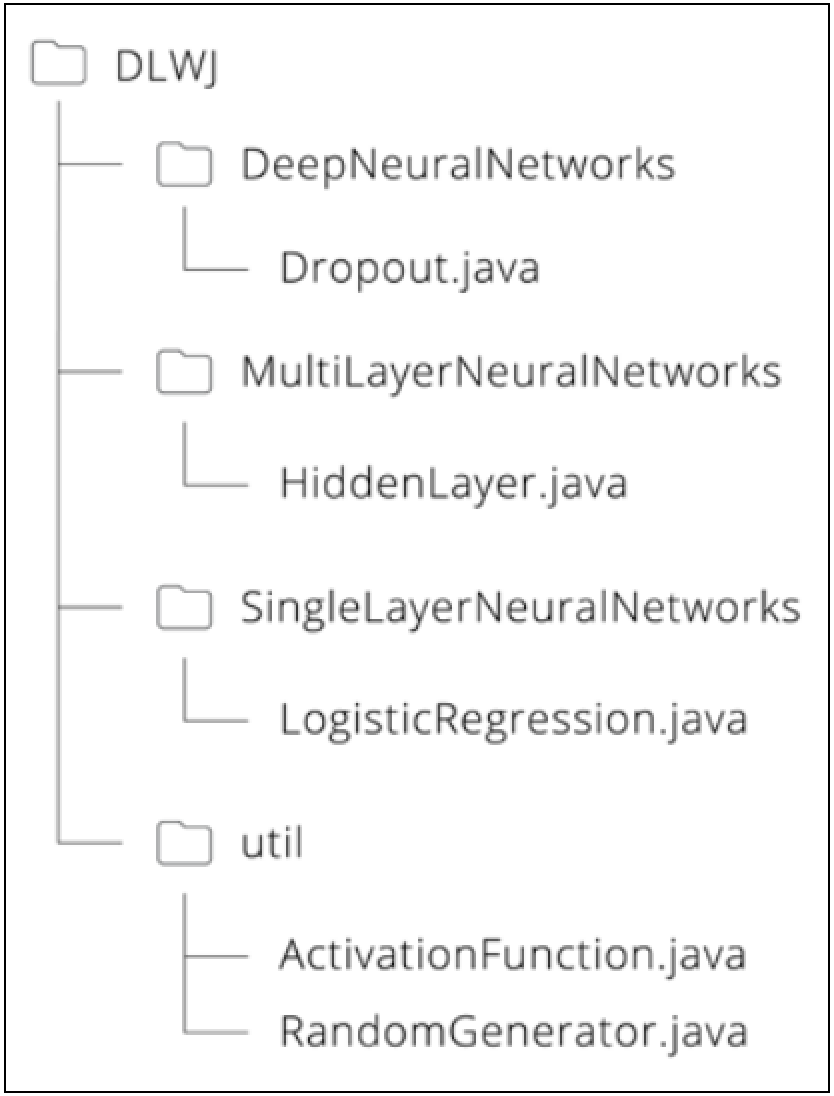
$$\delta_j := \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

$$a_k = \sum_j w_{kj} z_j + c_k$$
$$= \sum_j w_{kj} h(a_j) m_j + c_k$$

Here, the delta can be described as follows:

$$\delta_j = h'(a_j) m_j \sum_k \delta_k w_{kj}$$

Implementation



Smart Computing
SAIC
智慧計算實驗室

Outline

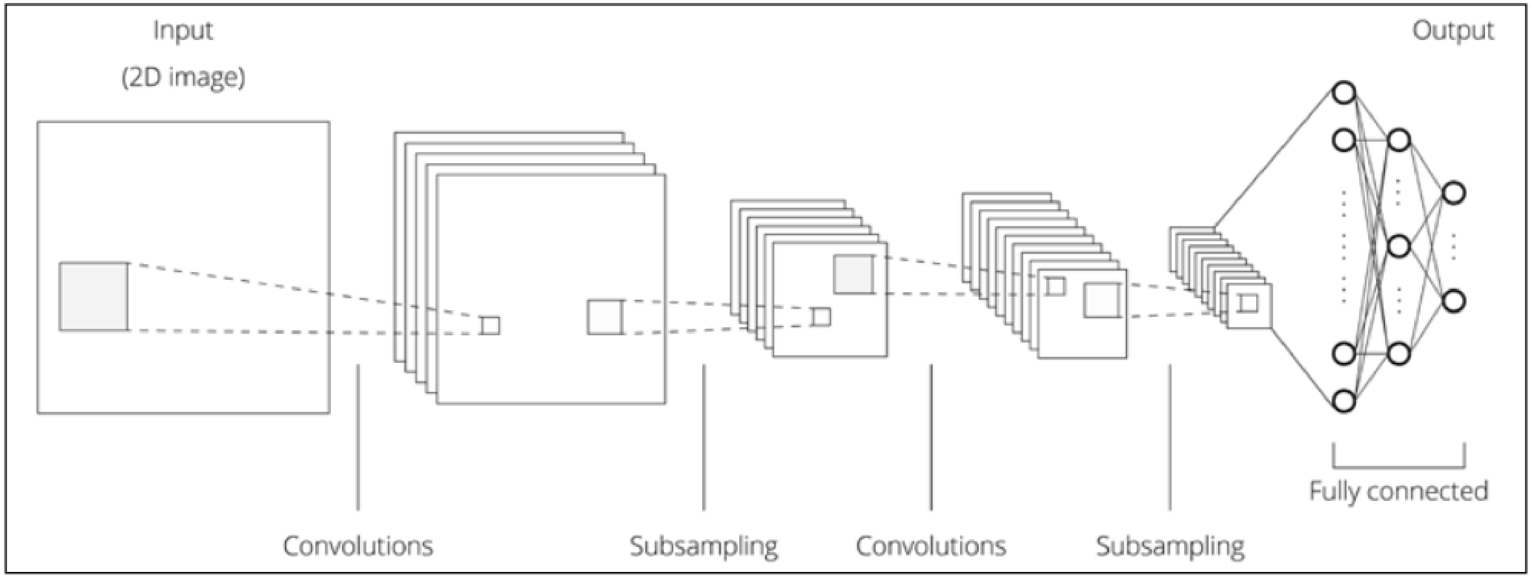
- Deep learning algorithms without pre-training
- Dropout
- Convolutional Neural Networks
- Summary

Convolutional Neural Networks, CNN

- At real-world application, data is not necessarily one-dimensional
- In CNN, features are extracted from two-dimensional input data
  - Convolutional layers
  - Pooling layers
  - Inspired by human visual areas

CNN

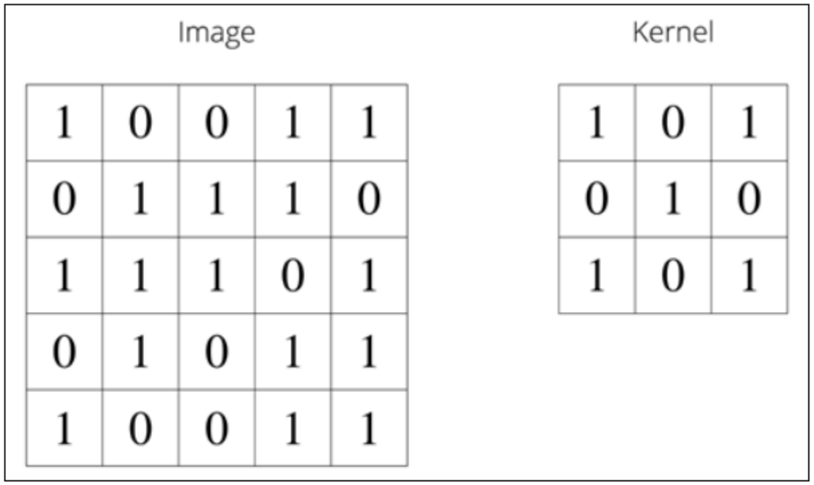
- CNN preprocessing
  - Segment the input data into several domains. This process is equivalent to a human's receptive fields
  - Extract the features from the respective domains, such as edges and position aberrations

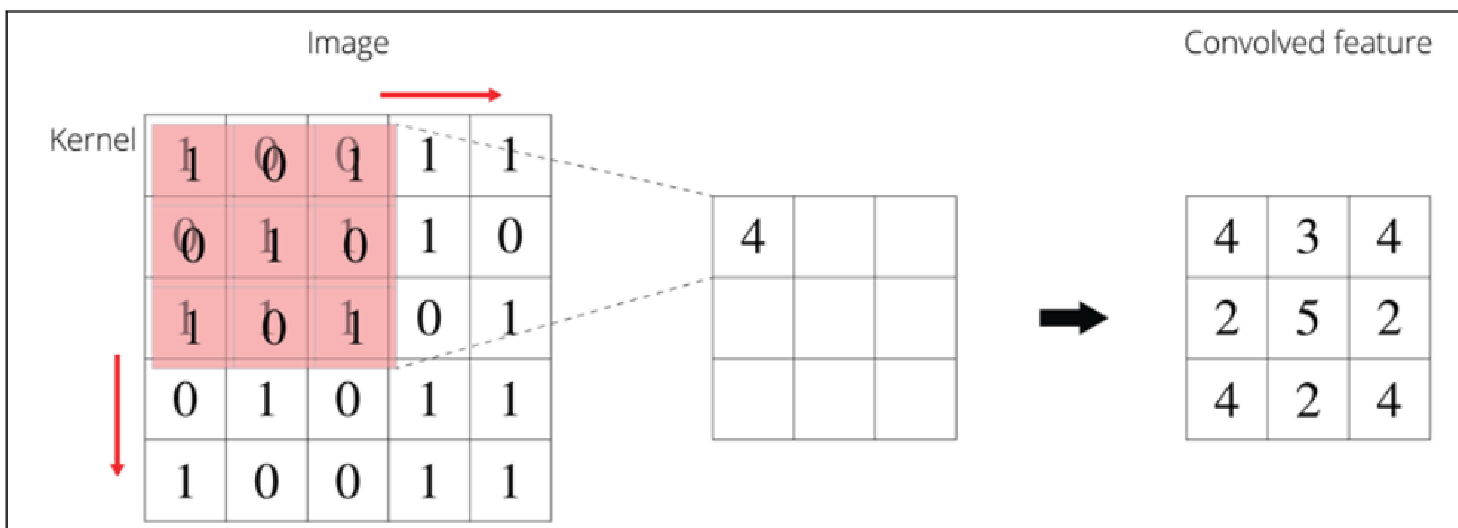


Convolution

- Convolutional layers literally perform convolution
  - Applying several filters (kernels) to the image to extract features
  - The kernel slides across the image and returns the summation of its values within the kernel as a multiplication filter
  - Convolved images are called feature maps

Convolution





Different Kernels

- Extract many kinds of features by changing kernel values
  - Left: extract edges (accentuates the color differences)
  - Right: blur image (degrade the original values)

-1	-1	-1
-1	8	-1
-1	-1	-1

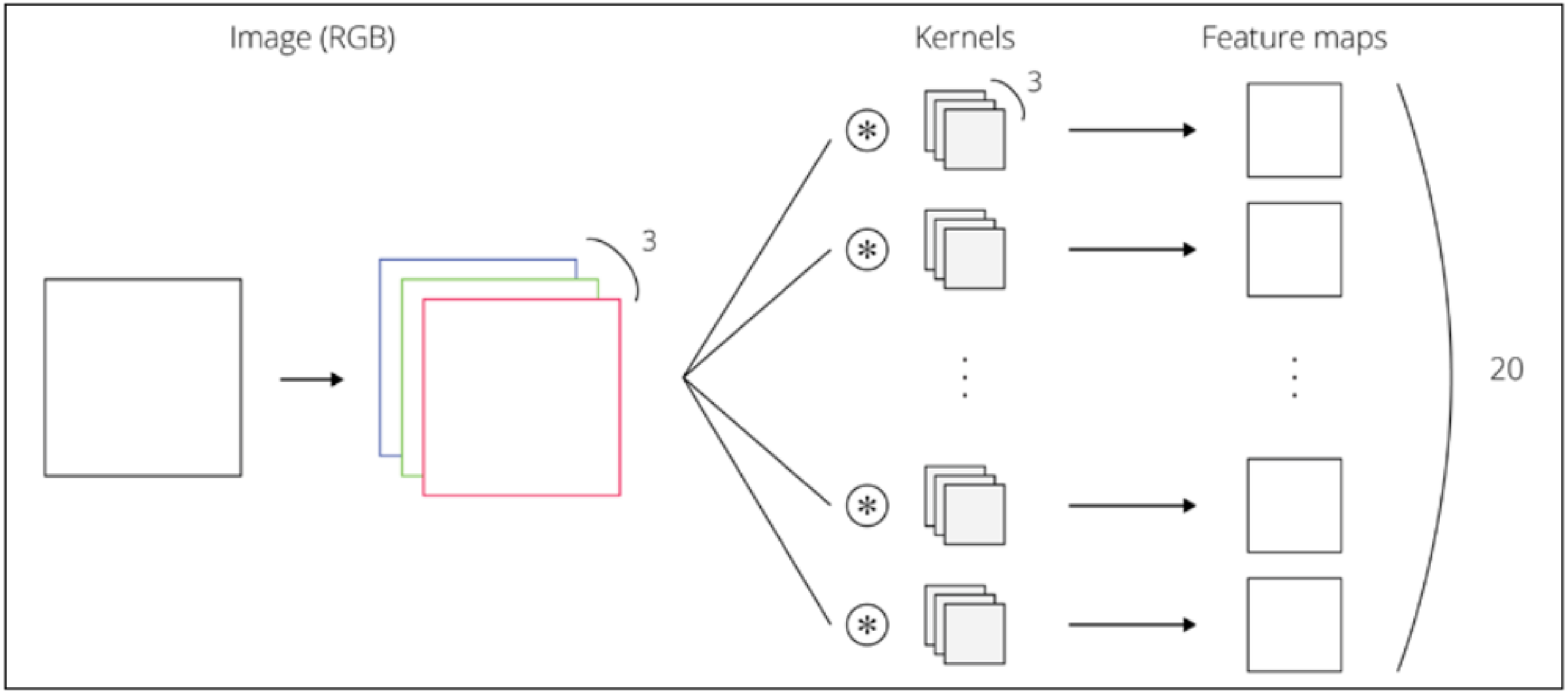
0.1	0.1	0.1
0.1	0.1	0.1
0.1	0.1	0.1

CNN

- In CNN, kernels are learned
  - Parameters trained in CNN are the weights of kernels
- Why CNN predict with higher precision rates?
  - Local receptive field
- Number of feature maps and the number of kernels are always the same

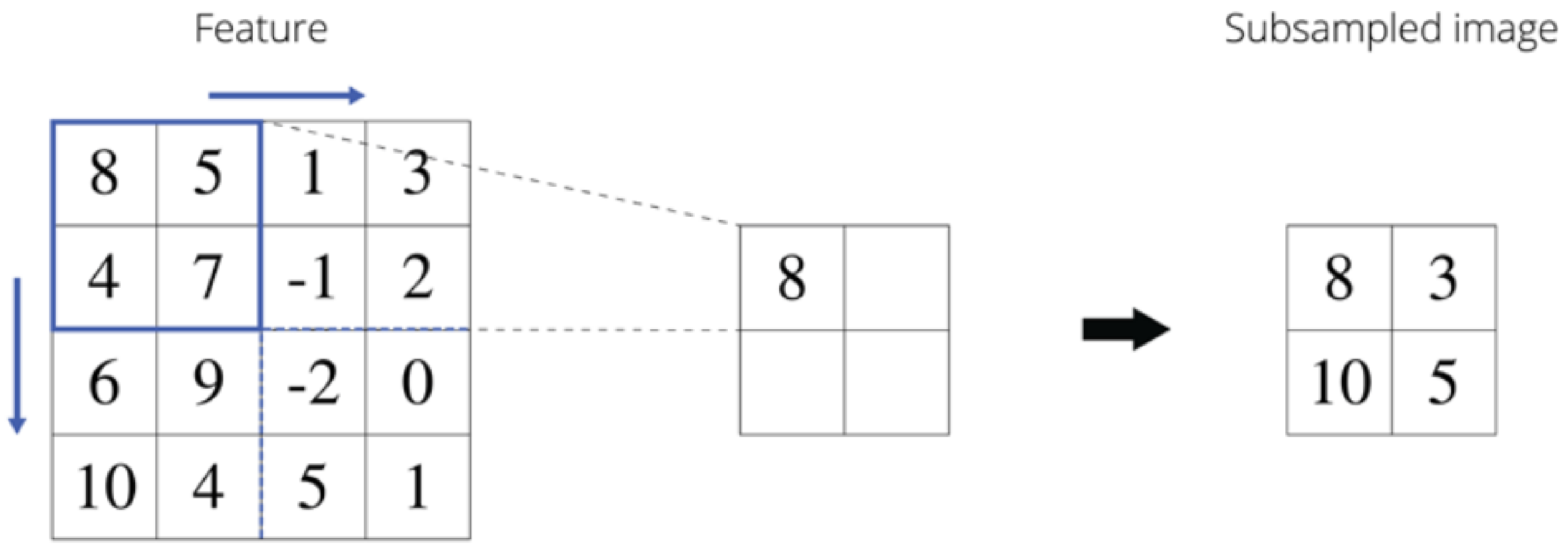
Example

- 3-channeled image, 20 kernels



Pooling

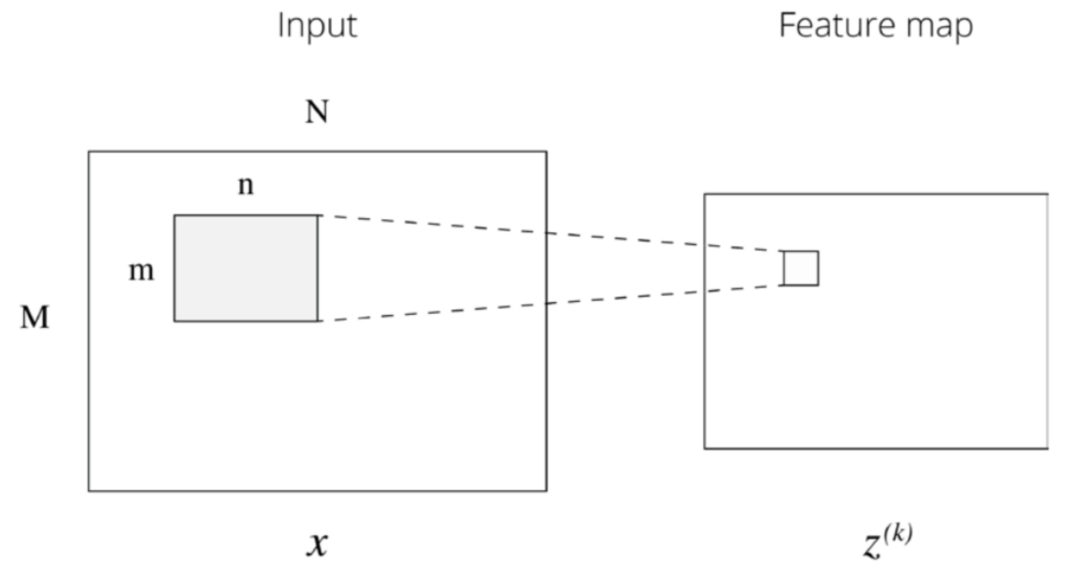
- Also called subsampling
- Most famous: max-pooling



Equations

$$z_{ij}^{(k)} = \sum_{s=0}^{m-1}\sum_{t=0}^{n-1} w_{st}^{(k)} x(i+s)(j+t)$$

$$z_{ij}^{(k)} = \sum_{c}\sum_{s=0}^{m-1}\sum_{t=0}^{n-1} w_{st}^{(k,c)} x_{(i+s)(j+t)}^{(c)}$$



$$a_{ij}^{(k)} = h\left(z_{ij}^{(k)} + b^{(k)}\right) = \max\left(0, z_{ij}^{(k)} + b^{(k)}\right)$$

$$y_{ij}^{(k)} = \max\left(a_{(l_1 i+s)(l_2 j+t)}^{(k)}\right)$$

l 1 and l 2 are the size of pooling filter

Equations

$$\frac{\partial E}{\partial a^{(k)}_{(l_1 i+s)(l_2 j+t)}} = \begin{cases} \dfrac{\partial E}{\partial y^{(k)}_{ij}} & if\ y^{(k)}_{ij} = a^{(k)}_{(l_1 i+s)(l_2 j+t)} \\ 0 & otherwise \end{cases}$$

$$\frac{\partial E}{\partial b^{(k)}} = \sum_{i=0}^{M-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial a^{(k)}_{ij}} \frac{\partial a^{(k)}_{ij}}{\partial b^{(k)}}$$

$$\delta^{(k)}_{ij} := \frac{\partial E}{\partial a^{(k)}_{ij}}$$

$$c^{(k)}_{ij} := z^{(k)}_{ij} + b^{(k)}$$

$$\frac{\partial E}{\partial b^{(k)}} = \sum_{i=0}^{M-m} \sum_{j=0}^{N-n} \delta^{(k)}_{ij} \frac{\partial a^{(k)}_{ij}}{\partial c^{(k)}_{ij}} \frac{\partial c^{(k)}_{j}}{\partial b_{(k)}}$$

$$= \sum_{i=0}^{M-m} \sum_{j=0}^{N-n} \delta^{(k)}_{ij} h'\left(c^{(k)}_{ij}\right)$$

Equations

$$\frac{\partial E}{\partial w_{st}^{(k,c)}} = \sum_{i=0}^{M-m}\sum_{j=0}^{N-n}\frac{\partial E}{\partial z_{ij}^{(k)}}\frac{\partial z_{ij}^{(k)}}{\partial w_{st}^{(k,c)}}$$

$$= \sum_{i=0}^{M-m}\sum_{j=0}^{N-n}\frac{\partial E}{\partial z_{ij}^{(k)}}\frac{\partial a_{ij}^{(k)}}{\partial z_{ij}^{(k)}}x_{(i+s)(j+t)}^{(c)}$$

$$= \sum_{i=0}^{M-m}\sum_{j=0}^{N-n}\delta_{ij}^{(k)}h'\left(c_{ij}^{(k)}\right)x_{(i+s)(j+t)}^{(c)}$$

$$\frac{\partial E}{\partial w_{ij}^{(c)}} = \sum_{k}\sum_{s=0}^{m-1}\sum_{t=0}^{n-1}\frac{\partial E}{\partial z_{(i-s)(j-t)}^{(k)}}\frac{\partial z_{(i-s)(j-t)}^{(k)}}{\partial x_{ij}^{(c)}}$$

$$= \sum_{k}\sum_{s=0}^{m-1}\sum_{t=0}^{n-1}\frac{\partial E}{\partial z_{(i-s)(j-t)}^{(k)}}w_{st}^{(k,c)}$$

Equations

$$\frac{\partial E}{\partial z_{ij}^{(c)}} = \frac{\partial E}{\partial a_{ij}^{(k)}} \frac{\partial a_{ij}^{(k)}}{\partial z_{ij}^{(k)}}$$

$$= \delta_{ij}^{(k)} \frac{\partial a_{ij}^{(k)}}{\partial c_{ij}^{(k)}} \frac{\partial c_{ij}^{(k)}}{\partial z_{ij}^{(k)}}$$

$$= \delta_{ij}^{(k)} h'\left(c_{ij}^{(k)}\right)$$

So, the error can be written as follows:

$$\frac{\partial E}{\partial x_{ij}^{(c)}} = \sum_{k} \sum_{s=0}^{m-1} \sum_{t=0}^{n-1} \delta_{(i-s)(j-t)}^{(k)} h'\left(c_{(i-s)(j-t)}^{(k)}\right) w_{st}^{(k,c)}$$

Implementation

```
DLWJ
├── DeepNeuralNetworks
│   ├── ConvolutionalNeuralNetworks.java
│   └── ConvolutionPoolingLayer.java
├── MultiLayerNeuralNetworks
│   └── HiddenLayer.java
├── SingleLayerNeuralNetworks
│   └── LogisticRegression.java
└── util
    ├── ActivationFunction.java
    └── RandomGenerator.java
```

Outline

- Deep learning algorithms without pre-training
- Dropout
- Convolutional Neural Networks
- Summary

Summary

- Two deep learning algorithms that don't require pre-training: deep neural networks with dropout and CNN
- The key to high precision rates is how we make the network sparse: dropout
- Rectifier: the activation function that can solve the problem of saturation
- CNN is the most popular algorithm for image recognition

Summary

- References
  - Deep Sparse Rectifier Neural Networks (Glorot, et. al. 2011, http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf)
  - ImageNet Classification with Deep Convolutional Neural Networks (Krizhevsky et. al. 2012, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)
  - Maxout Networks (Goodfellow et al. 2013, http://arxiv.org/pdf/1302.4389.pdf)