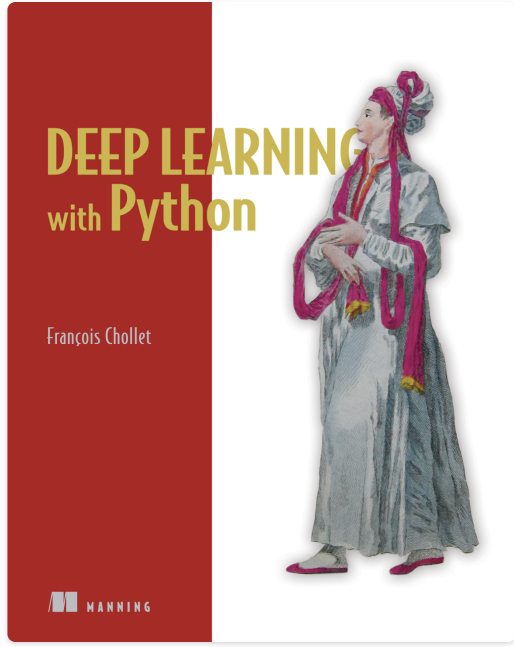
Deep Learning: An Introduction

bifnudozhao@tencent.com

Outline

- Machine Learning
- Deep Learning
- How Deep Learning Works
  - layers
  - model
  - loss function
  - optimizer
- Training
  - Evaluation of Models
  - Overfitting and Underfitting
- Deep Learning Networks



Deep Learning with Python

Machine Learning

Machine learning can be viewed as a new paradigm of programming.

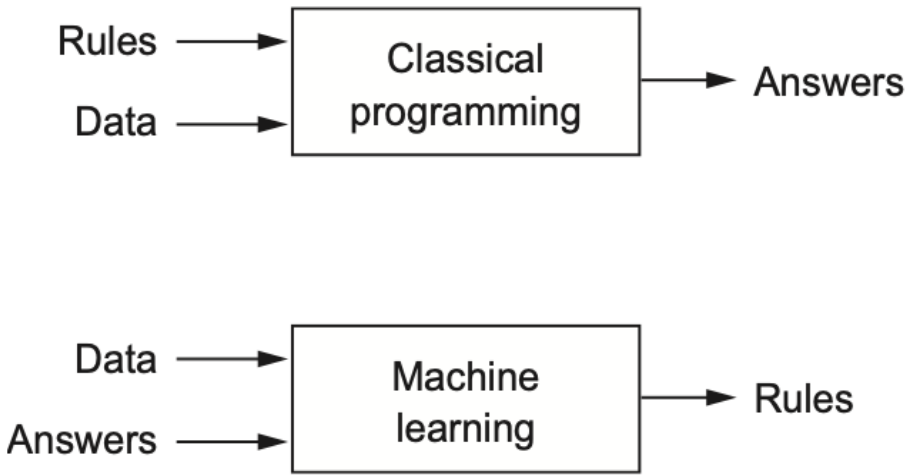


Figure 1.2 Machine learning: a new programming paradigm

A machine-learning system is trained rather than explicitly programmed.

Machine Learning

Unlike statistics, machine learning tends to deal with large, complex datasets for which classical statistical analysis such as Bayesian analysis would be impractical. As a result, machine learning, and especially deep learning, exhibits comparatively little mathematical theory —— maybe too little —— and is engineering oriented.

To do machine learning, we need three things:

- Input data points.
- Examples of the expected output.
- A way to measure whether the algorithm is doing a good job.

The central problem in machine learning and deep learning is to **meaningfully transform data**: in other words, to learn useful representations of the input data at hand.

Deep Learning

Deep learning place a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations.

In deep learning, these layered representations are (almost always) learned via models called **neural networks**. Deep learning is a **mathematical framework** for learning representations from data.

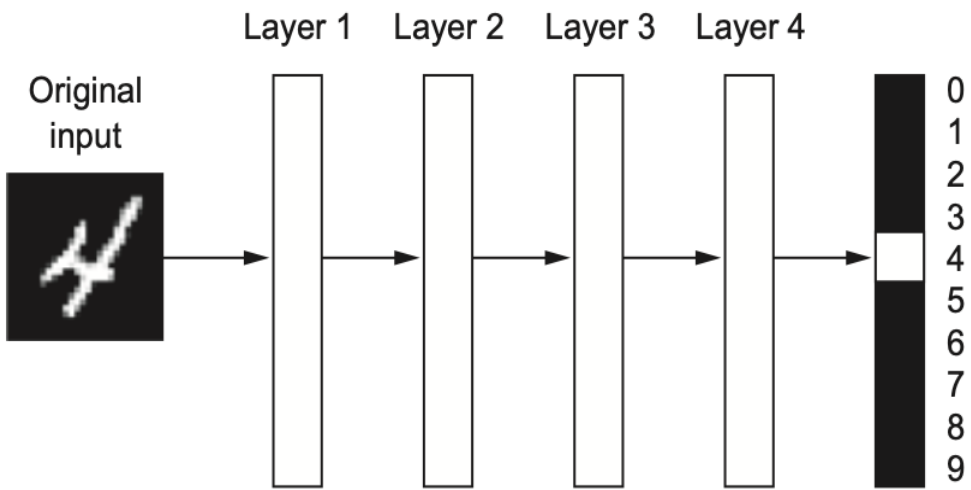


Figure 1.5 A deep neural network for digit classification

Deep Learning

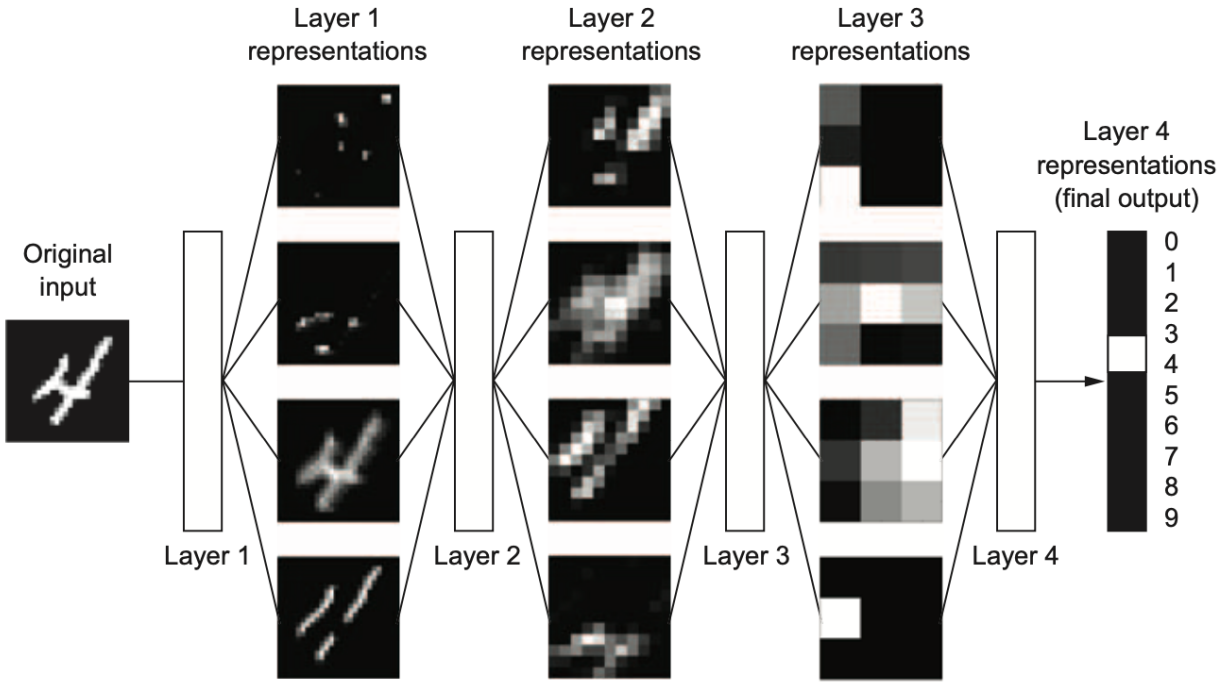


Figure 1.6 Deep representations learned by a digit-classification model

How Deep Learning Works - Parameterization

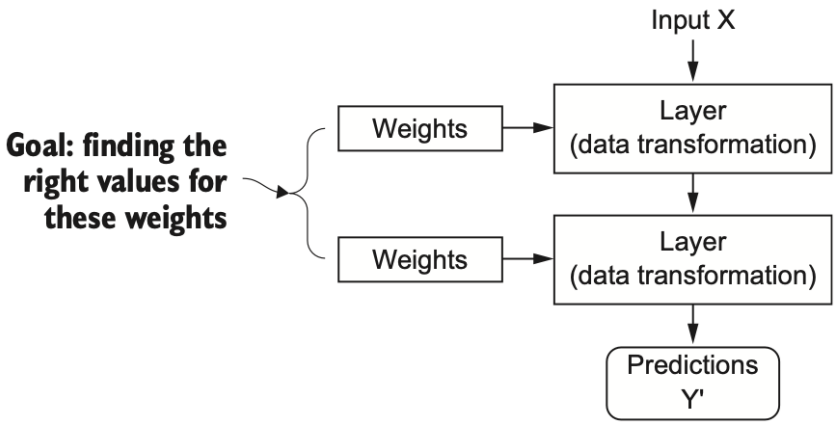


Figure 1.7 A neural network is parameterized by its weights.

The specification of what a layer does to its input data is stored in the layer's **weights**. The transformation implemented by a layer is **parameterized** by its weights. **Learning** means finding a set of values for the weights of all layers in a network, such that the network will correctly map example inputs to their associated targets.

```
const output1 = layer1(x)
const output2 = layer2(output1)
const y = layer3(output2)
```

How Deep Learning Works - Loss Function

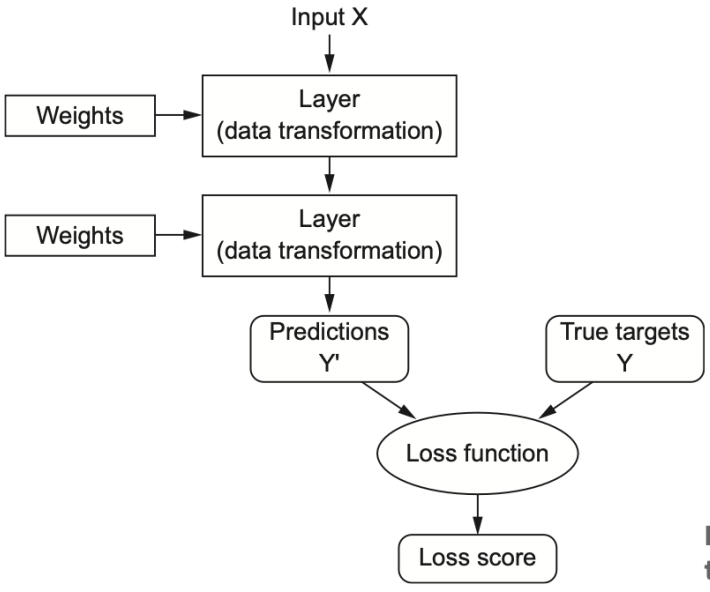


Figure 1.8 A loss function measures the quality of the network's output.

Loss function (also called the **objective function**) is used to measure the distance between the output and the target, giving a distance score to capture how well the network has done on this specific example.

```
const y = layer3(layer2(layer1(x)))
const loss = lossFunction(y, target)
```

How Deep Learning Works - Back Propagation

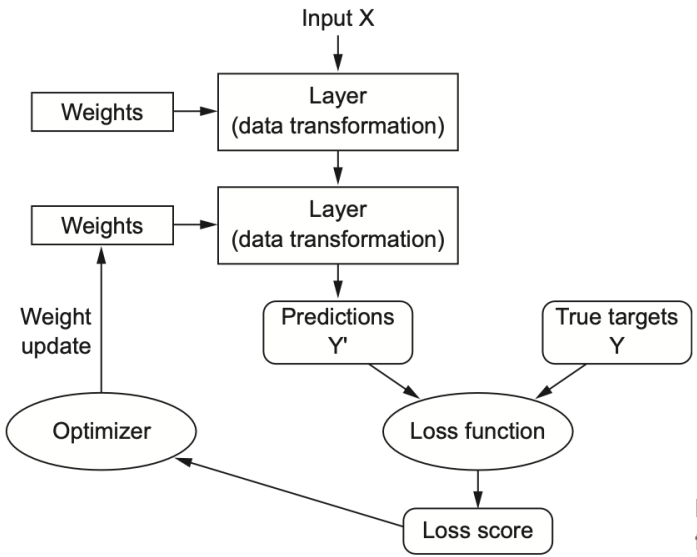


Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

The fundamental trick in deep learning is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the current example. This adjustment is the job of the **optimizer**, which implements what's called the **Backpropatation** algorithm.

```
1   optimizer.update([layer1, layer2, layer3], loss)
```

How Deep Learning Works - Layers

```
class Layer {
  constructor(activate: Function) {
    this.W = new Tensor()
    this.B = new Tensor()
    this.activate = activate
  }

  // Y = activation(W·X + B)
  forward(X: Tensor) {
    return this.activate(this.W.mul(X).add(this.B))
  }
}

const ReLU = (x) => x > 0 ? x : 0
const layer = new Layer(ReLU)
const Y = layer.forward(X)
```

A **layer** is a data-processing module that takes as input one or more tensors and that outputs one or more tensors. The layer's **weights**, one or several tensors learned with SGD, which together contain the network's **knowledge**.

How Deep Learning Works - Models

A deep-learning model is a directed, acyclic graph of layers. The topology of a network defines a **hypothesis space**. By choosing a network topology, you constrain your **space of possibilities** to a specific series of tensor operations, mapping input data to output data. What you'll then be searching for is a good set of values for the weight tensors involved in these tensor operations.

Picking the right network architecture is more an art than a science.

```
class Model {
  layers: Layer[]

  constructor () {
    this.layers = []
  }

  add(layer: Layer) {
    this.layers.push(layer)
  }

  // Y = layer3(layer2(layer1(X)))
  // model.layers = [layer1, layer2, layer3]
  predict(X: Tensor) {
    let result = X
    this.layers.forEach(layer => {
      result = layer.forward(result)
    })
    return result
  }
}

const model = new Model()
const layer1 = new Layer(ReLU)
model.add(layer1)
const Y = model.predict(X)
```

How Deep Learning Works - Loss Function

Cross Entropy Error

Loss function (objective function), the quantity that will be minimized during training. Cross-entropy error is often used in multi-category classification problems.

$$E = -\sum_k t_k \log y_k$$

```
const LossFunction = (t, y) => -_.sum(_.map((t, i) => t * Math.log(y[i])))
const Y = model.predict(X)
const loss = LossFunction(Y, target)
```

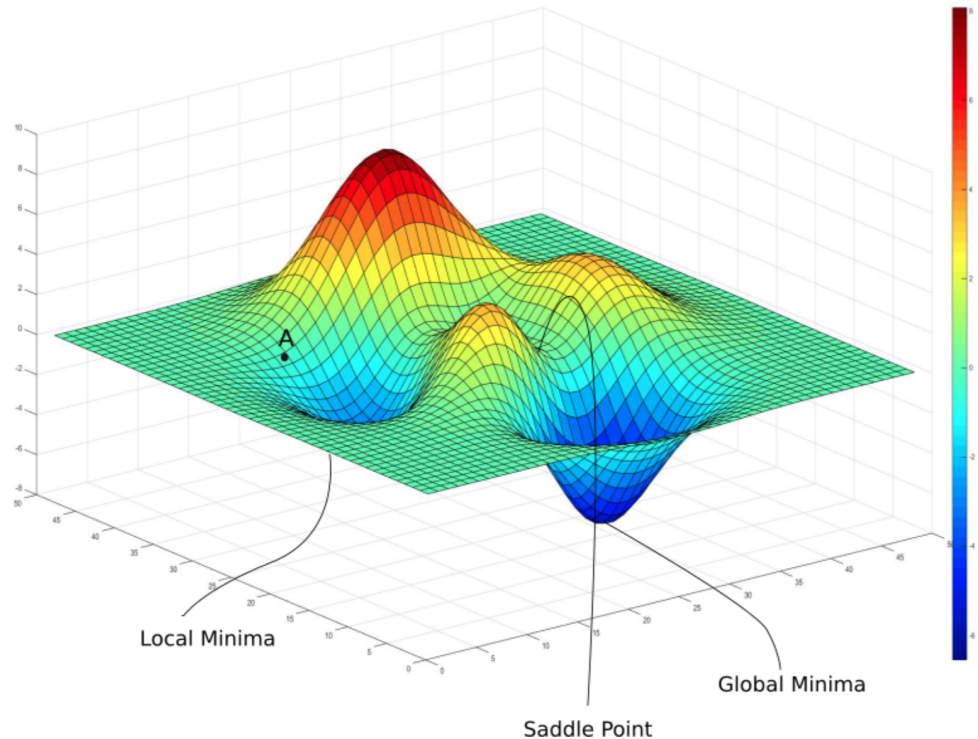
Mean Squared Error

Often used in continuous prediction problems, such as linear regression.

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2$$

How Deep Learning Works - Optimizer

```
1   const loss = LossFunction(Y, target)
```



How Deep Learning Works - Optimizer

SGD (Stochastic Gradient Desend)

Optimizer determines how the network will be updated based on the loss function. SGD updates the weights with the following manner.

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

```
class SGD {
  constructor(learningRate) {
    this.lr = learningRate
  }

  update(W: Tensor, gradients: Tensor) {
    return W.sub(gradients.mul(this.lr))
  }
}

const loss = LossFunction(model.predict(X), target)
const optimizer = new SGD(0.001)
optimizer.update(model.W, loss)
```

Training

```
const train = (data) => {
  const model = new Model()
  const LossFunction = CrossEntropyError()
  const optimizer = new SGD()

  _.each(data, ([X, target]) => {
    const Y = model.predict(X)
    const loss = LossFunction(Y, target)
    optimizer.update(model.W, Y - loss)
  })

  return model
}

const model = train()
model.predict(newX)
```

Training is a process of modifying the parameters of a model to minimize the total loss. After training, we have a model to predict new data.

Evaluation of Models

Evaluating a model always boils down to splitting the available data into three sets: training, validation, and test.

- Train on training data
- Evaluate the model on validation data
- Test the model **one final time** on the test data

The reason of separating three sets instead of two is that developing a model always involves tuning its configuration. Tuning hyperparameters is also a form of learning. As a result, tuning the configuration of the model based on its performance on the validation set can quickly result in overfitting to the validation set. Central to this phenomenon is the notion of **information leaks**.

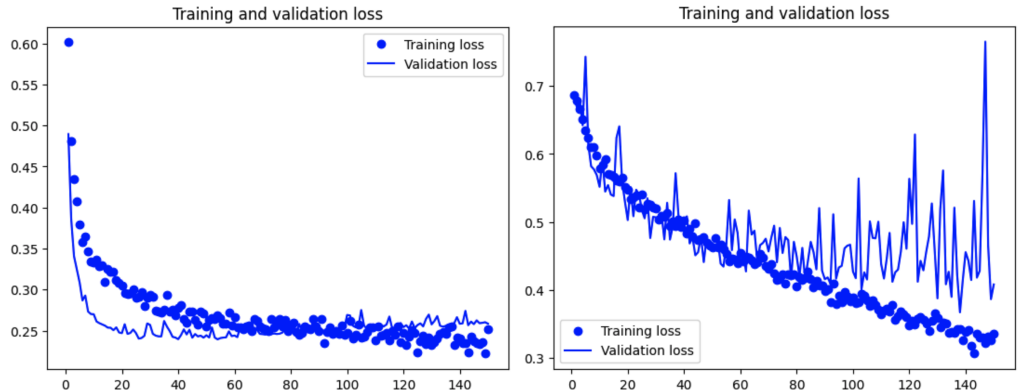
There are three common ways to split data into three sets when little data is available.

1. Simple hold-out validation
2. K-fold validation
3. Iterate K-fold validation with shuffling

Training - Evaluation of Models

```
const evaluate = (model, data) => {
  let correct = 0
  let loss = 0

  _.each(data, ([X, target]) => {
    const Y = model.predict(X)

    loss += (Y - target)
    if (Y === target) {
      correct += 1
    }
  })

  const accuracy = correct / data.length
  return { accuracy, loss }
}

const rawData = getData()
const [trainData, validationData] = split(rawData, 0.8)
const model = train(trainData)
const accuracy = evaluate(model, validationData)
```

Training - Overfitting and Underfitting

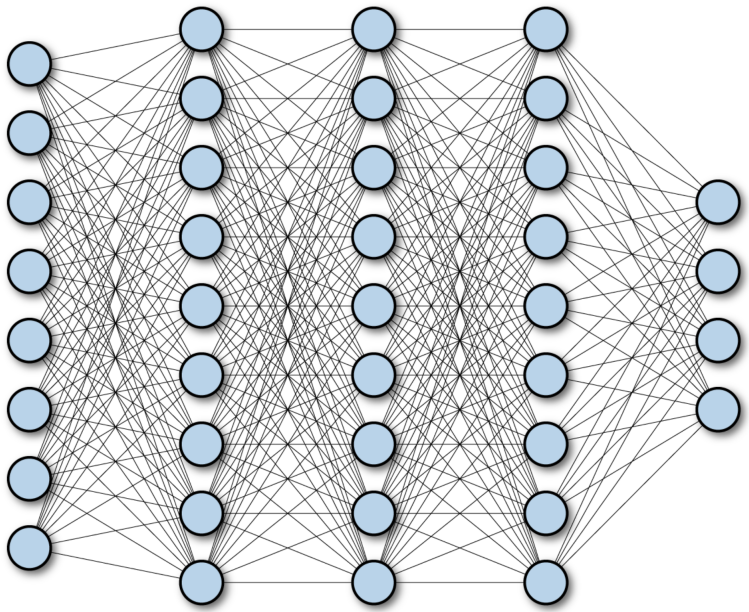


The fundamental issue in machine learning is the tension between optimization and generalization. **Optimization** refers to the process of adjusting a model to get the best performance possible on the training data. Whereas **generalization** refers to how well the trained model performs on data it has never seen before.

To prevent a model from learning misleading or irrelevant patterns found in the training data, the best solution is to **get more training data**.

The next-best solution is to modulate the quantity of information that the model is allowed to store or to add constraints on what information it's allowed to store. The processing of fighting overfitting this way is called **regularization**.

Deep Learning Networks - FCN



```
Y = layer3(layer2(layer1(X)))
```

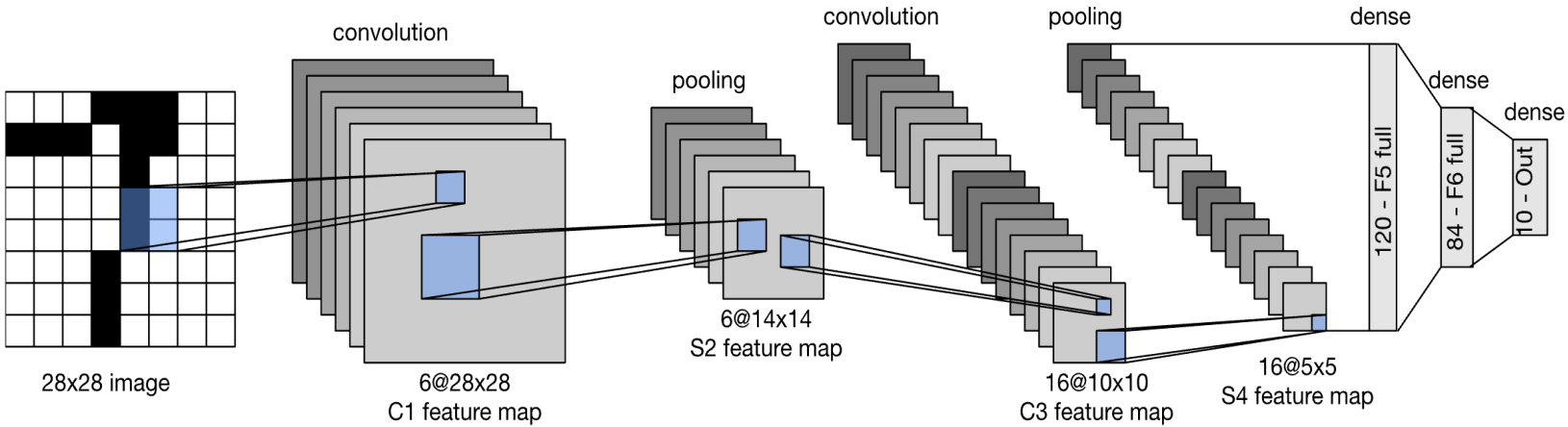
FCN (Full Connected Network): When all the inputs and outputs are connect with the neighboring layers, then the network is called FCN.

$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{X} + \mathbf{b}$$

A simple example.

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}_{\mathbf{W}} \times \underbrace{\begin{bmatrix} 5 \\ 6 \end{bmatrix}}_{\mathbf{X}} + \underbrace{\begin{bmatrix} 7 \\ 8 \end{bmatrix}}_{\mathbf{b}} = \begin{bmatrix} 1 \times 5 + 2 \times 6 + 7 \\ 3 \times 5 + 4 \times 6 + 8 \end{bmatrix} = \underbrace{\begin{bmatrix} 24 \\ 47 \end{bmatrix}}_{\mathbf{Y}}$$

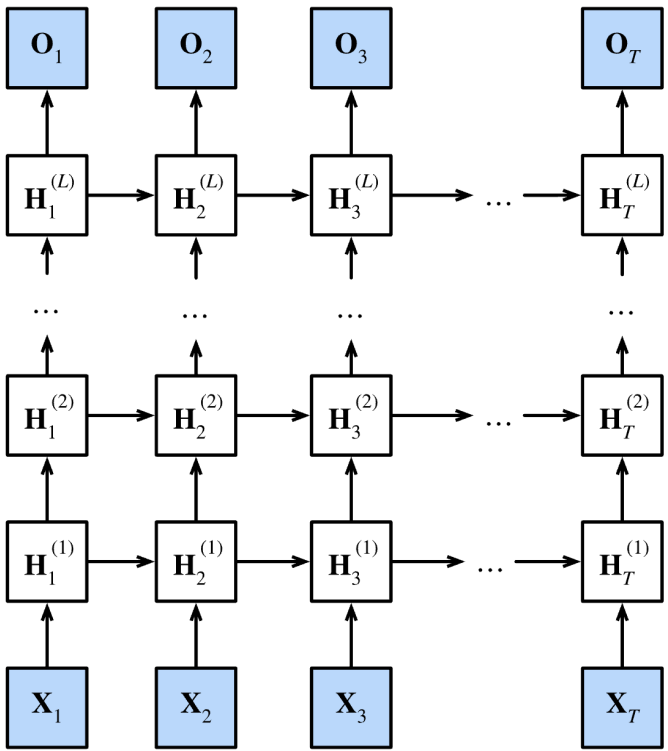
Deep Learning Networks - CNN



CNN (Convolutional Neural Network): When `layer` performs convolutional operations, then the network is called CNN.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)(d)\tau$$

Deep Learning Networks - RNN



RNN (Recurrent Neural Network): When `layer` has a memory unit to remember the previous output, then the network is called RNN.

$$
\begin{aligned}
f &= \sigma(x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)}) \\
g &= \tanh(x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)}) \\
i &= \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)}) \\
o &= \sigma(x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)}) \\
c_t &= f \odot c_{t-1} + i \odot g \\
h_t &= o \odot \tanh(c_t)
\end{aligned}
$$

where f is the forget gate, g is the gain parameter, i is the input gate, o is the output gate, c_t is the carrier at time t, h_t is the hidden state at time t.

Q & A