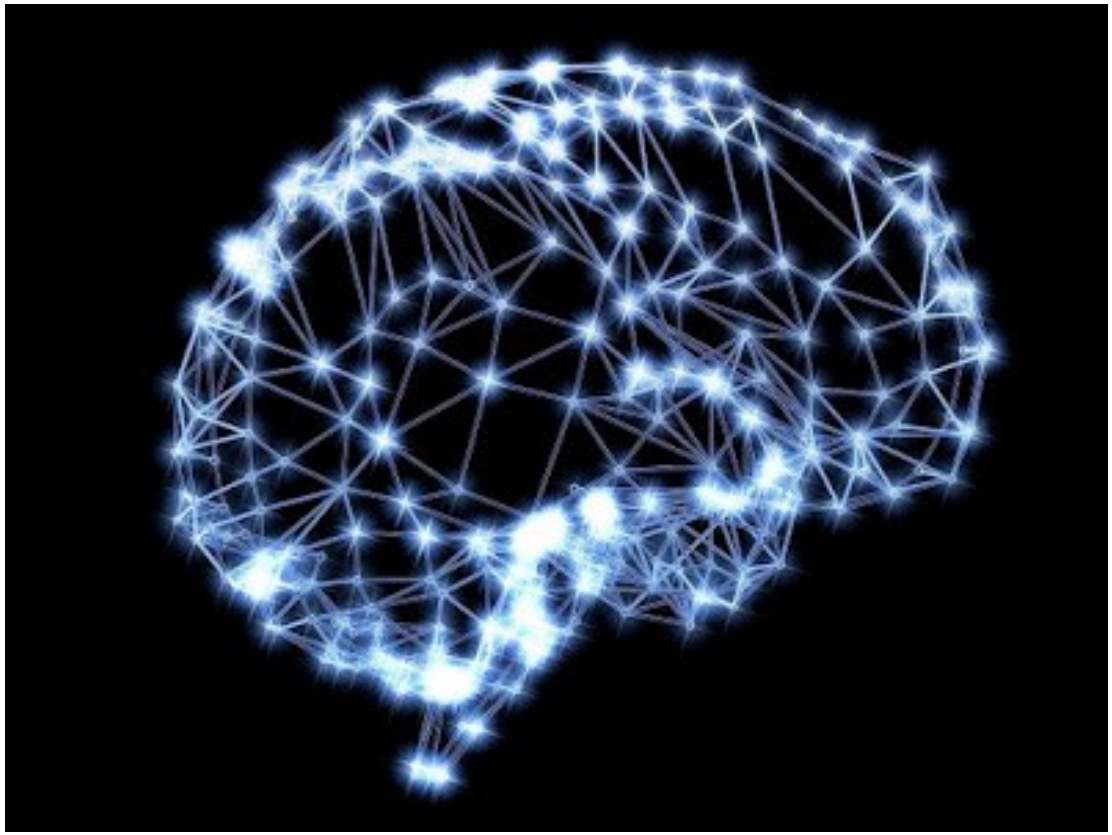
Neural Networks and Deep Learning

Shaobo Li

University of Kansas

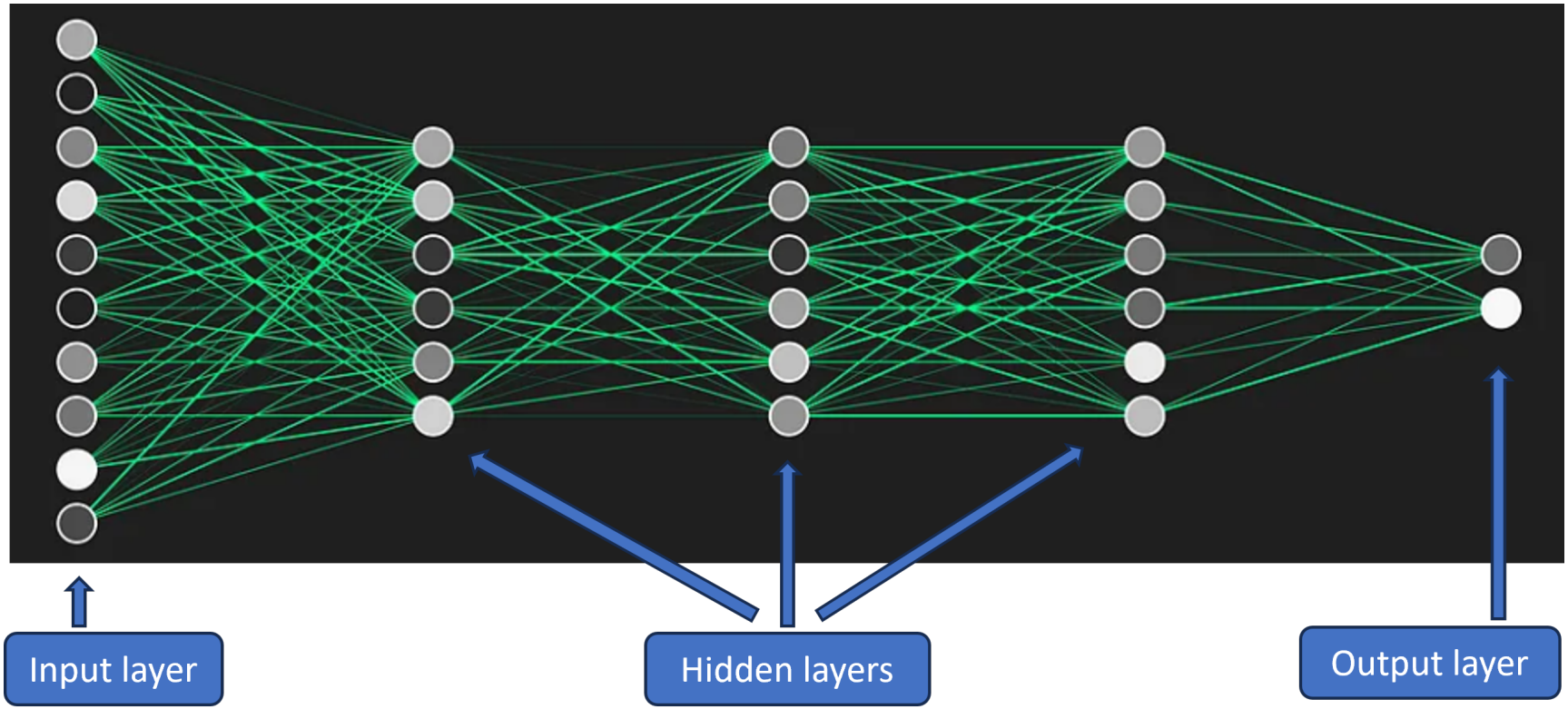
Human brain



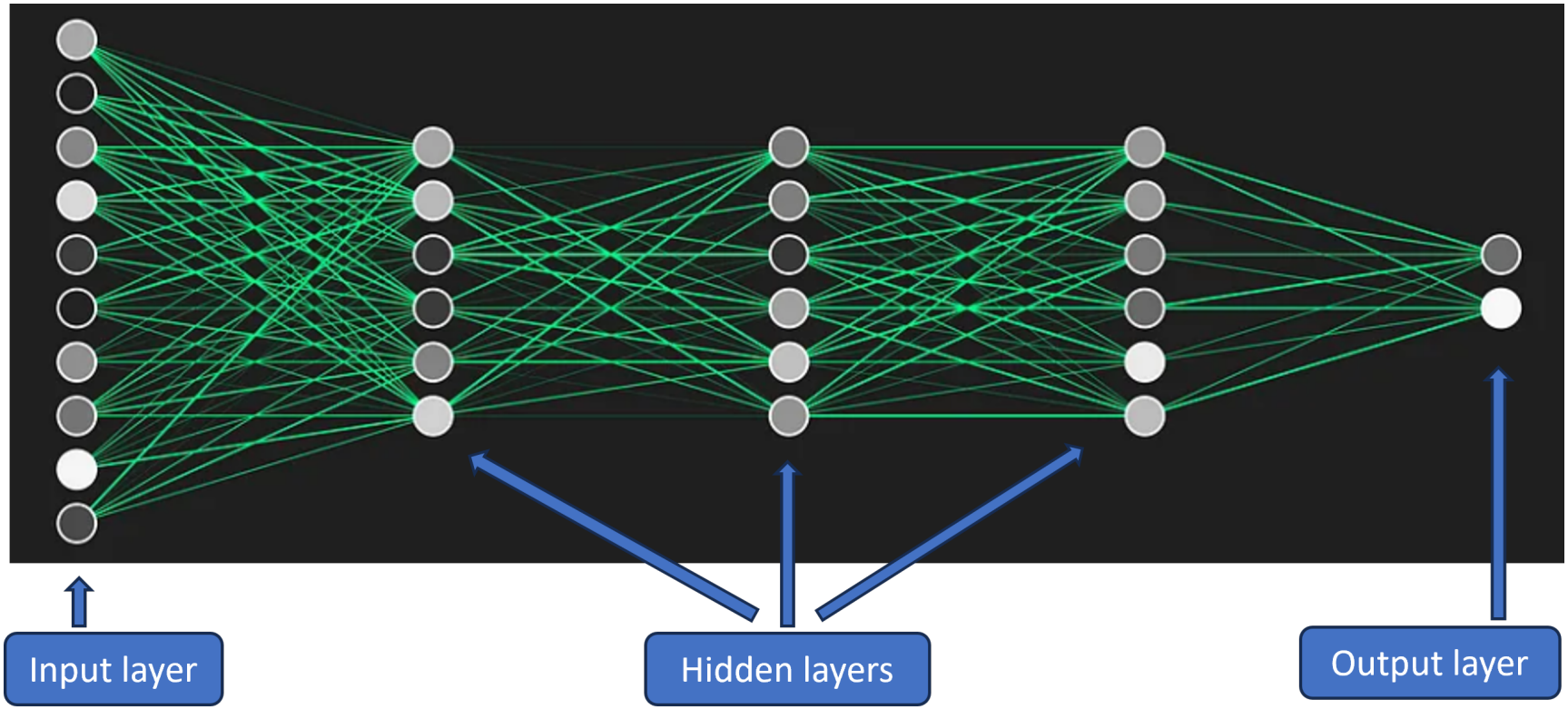
Human brain

- Receive inputs
- Processing electric and chemical signals
- Neuron switching time: ~ 0.001 second
- Number of neurons: $\sim 10^{11}$
- Connections per neuron: $\sim 10^4 - 10^5$
- Scene recognition time: ~ 0.1 second
- Much parallel computation

Artifical Neural Network



Artifical Neural Network



- Each edge represents a weight
- Each node represents a univariate function

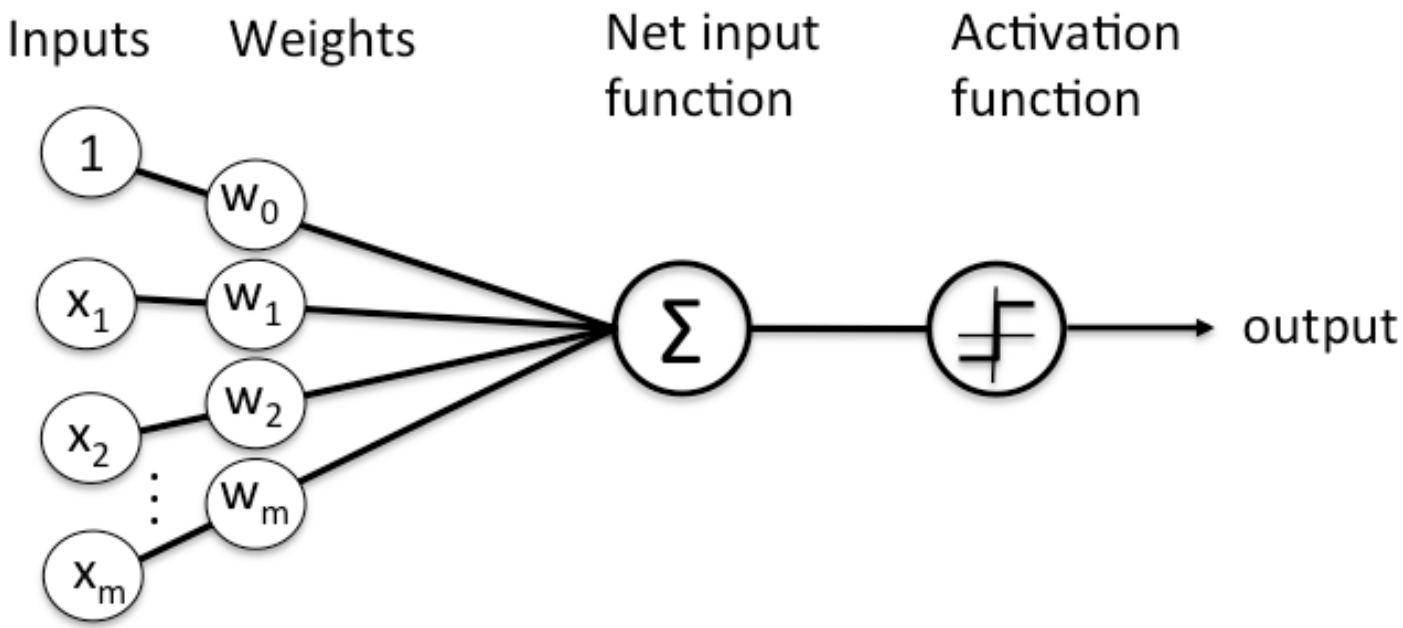
Image source: here

How the system is working

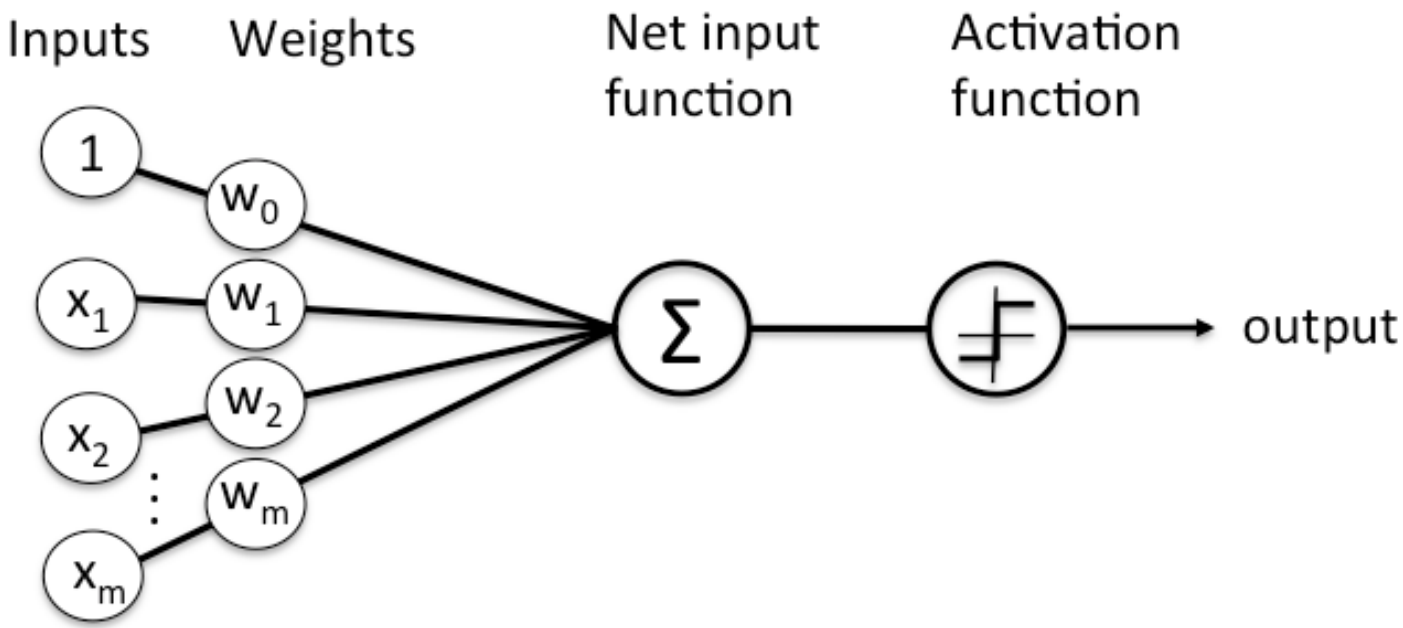
1. Input layer takes in the data $X_1, X_2, \ldots, X_p$
2. Form linear combinations: $w_{0,k} + w_{1,k}X_1 + w_{2,k}X_2 + \ldots + w_{p,k}X_p$
   - $k = 1, ..., K_1$, where K_1 is the number of neurons in the first hidden layer
   - w_0 acts as an intercept, which is to correct the bias
3. Pass to first hidden layer: to each neurons
   - Each neuron is an activation function $\sigma(w_0 + \sum_{j=1}^{p} w_j X_j)$
   - It is a nonlinear transformation.
4. A new set weights are applied to form linear combinations and pass to the second hidden layer, and so on.....
5. Output layer: prediction
   - For regression: one node with an identity function
   - For classification: multiple nodes with the softmax function

You should watch this great video

A simple neural network – Perceptron

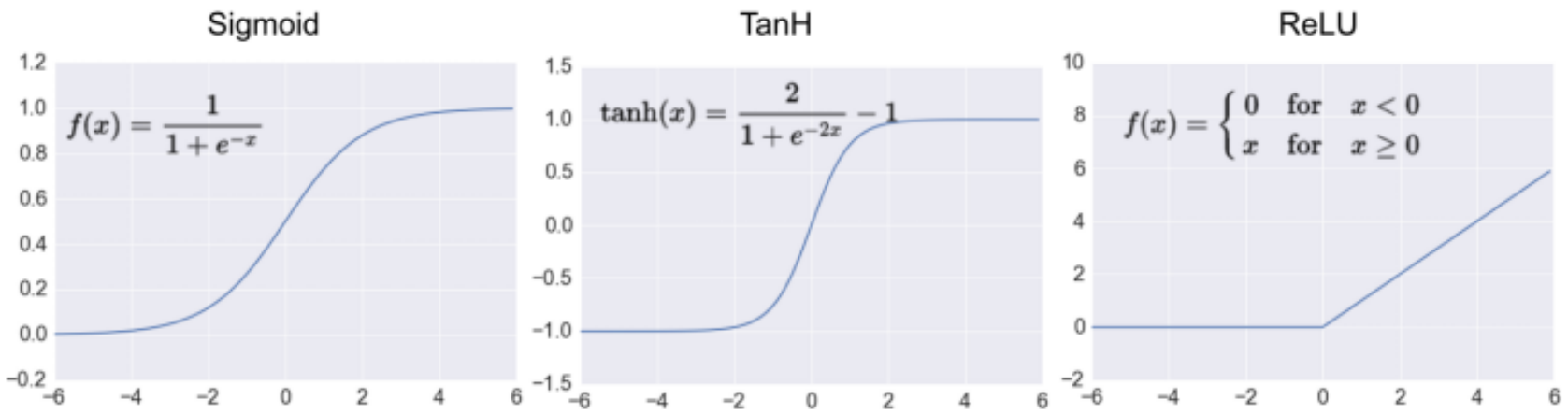


A simple neural network – Perceptron

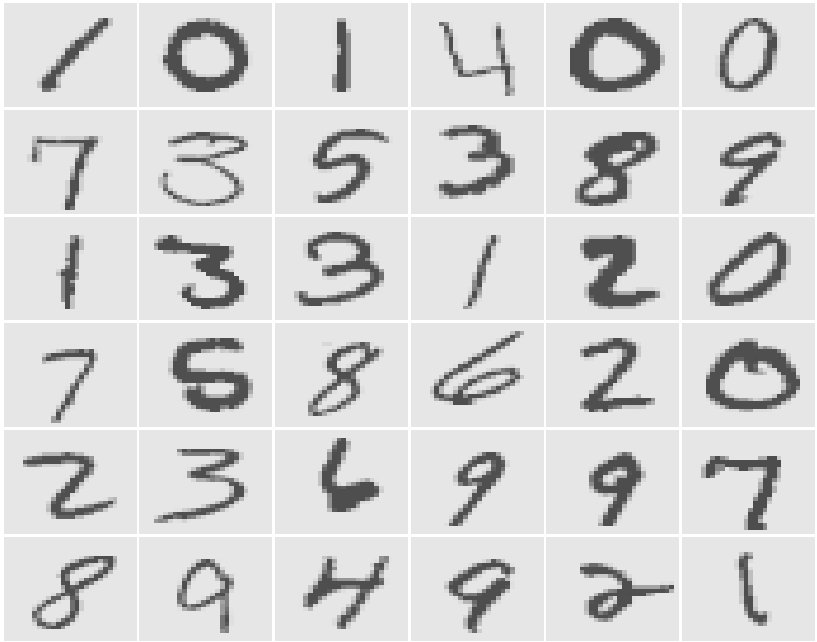


- It is equivalent to linear regression if
  - the response variable is continuous, and
  - the activation function is the identity function
- It is equivalent to logistic regression if
  - the response variable is binary, and
  - the activation function is the sigmoid function

Other popular activation funtions

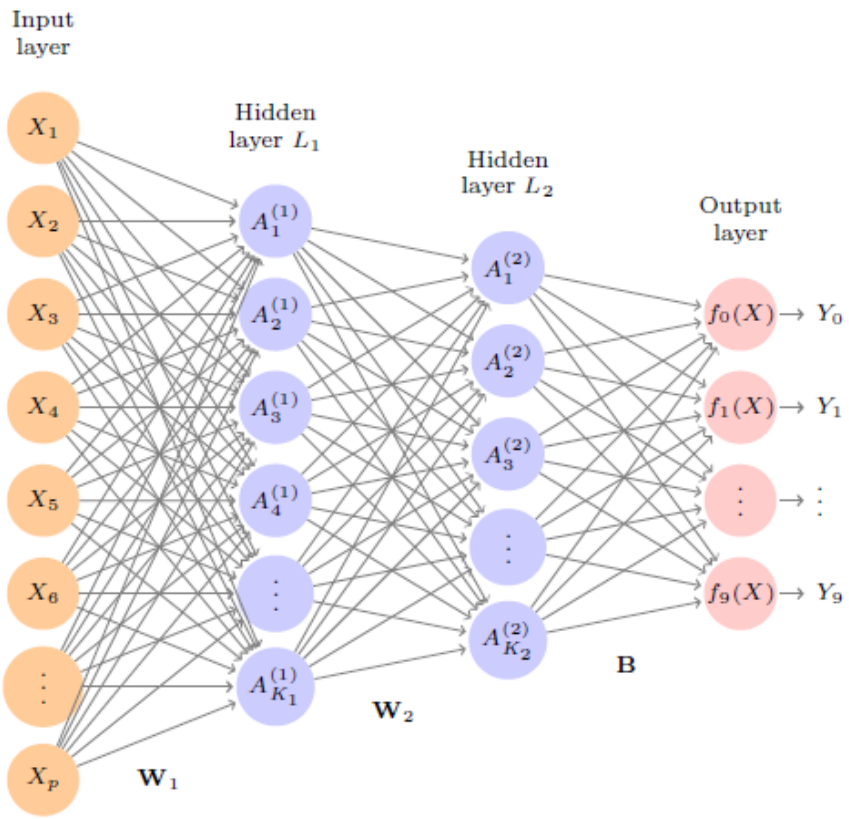


Example — MNIST data

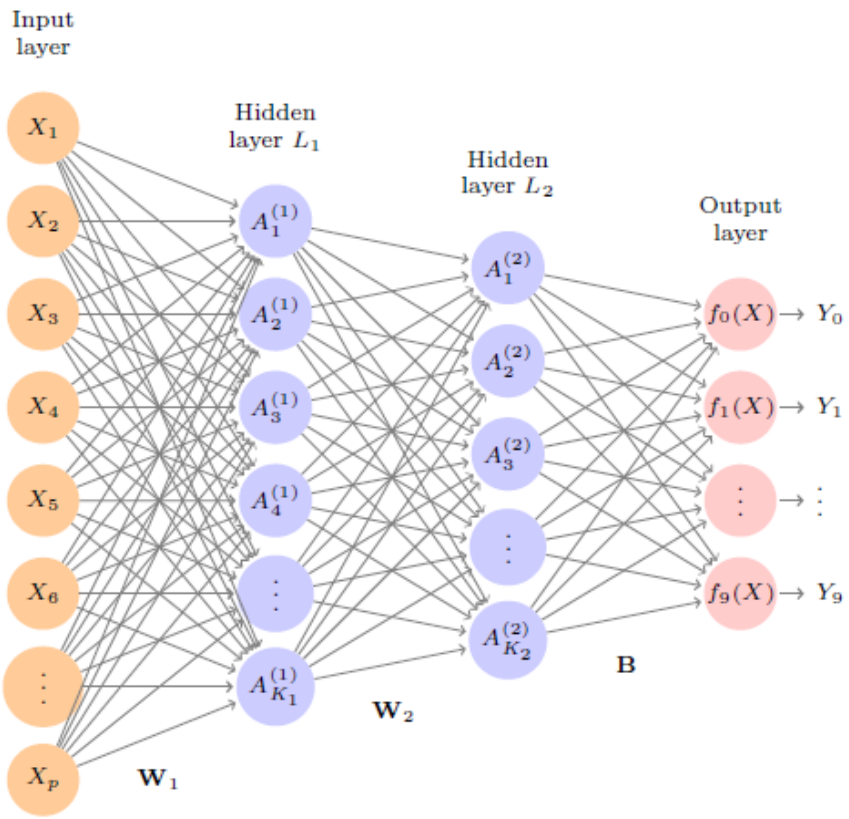


- Each grayscale image has 28×28 pixels
- Each pixel has a value ranges 0-244, representing the darkness
- Goal: predict/classify an image to a specific number (label)
- 60,000 training images and 10,000 test images

A neural network for MNIST data



A neural network for MNIST data



- Input layer: $p = 784$ units (pixels)
- Two hidden layers: $K_1 = 256$ and $K_2 = 128$
- Output layer: 10 units, representing 10 labels
- How many parameters in total?

Train a neural network

- A huge number of parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, ...\}$
- A single objective function, $R(\boldsymbol{\theta})$ e.g.,
  - squared error (for regression)
  - cross-entropy (for classification)

$$R(\boldsymbol{\theta}) = -\sum_{i=1}\sum_{j=1} y_{ij} \log(f_j(x_i, \boldsymbol{\theta})),$$

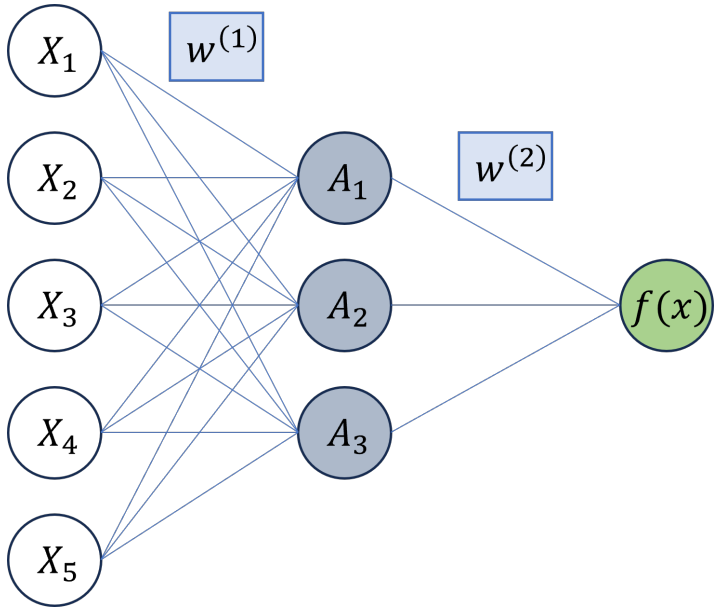
  where i is the index for observations and j is for labels.
- $R(\boldsymbol{\theta})$ is generally nonconvex
- Gradient descent can find local optimal
- Requires derivatives
- Parameters are nested from layer to layer (how complicated!!!)

Chain rule

- This property of derivative saves our life!
- $y = f(x)$; $z = g(y)$; $s = h(z)$
- In one equation, $s(x) = h(g(f(x)))$
- Derivative with respect to x: $\frac{\partial s}{\partial x} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$
- An example:
  $y = x + 2$; $z = e^y$; $s = z^2 - z$
  Then, $s(x) = e^{2(x+2)} - e^{x+2}$
  $s'(x) =?$
  Using chain rule, did you get the same result?

A fundamental method — Backpropagation

- We illustrate with a single layer neural network:



- Let the loss function be squared loss

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \text{where } \hat{y}_i = f(x_i; \hat{\theta})$$

- Two sets of parameters $\theta = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$.

A fundamental method — Backpropagation (cont'd)

- From the hidden layer to the output layer:[1]

$$\hat{y} = g(\hat{w}_0^{(2)} + \hat{w}_1^{(2)} A_1 + \hat{w}_2^{(2)} A_2 + \hat{w}_3^{(2)} A_2)$$

  where $g()$ is the identity function (we keep it for generality)

- From the input layer to the hidden layer:

$$A_1 = \sigma(\hat{w}_{01}^{(1)} + \hat{w}_{11}^{(1)} X_1 + \cdots + \hat{w}_{51}^{(1)} X_5)$$
$$A_2 = \sigma(\hat{w}_{02}^{(1)} + \hat{w}_{12}^{(1)} X_1 + \cdots + \hat{w}_{52}^{(1)} X_5)$$
$$A_3 = \sigma(\hat{w}_{03}^{(1)} + \hat{w}_{13}^{(1)} X_1 + \cdots + \hat{w}_{53}^{(1)} X_5)$$

- Using matrix expression

$$\mathbf{A} = \sigma(\mathbf{w}_0^{(1)} + \mathbf{w}^{(1)} \mathbf{X})$$

[1] We drop subscript i for simplicity

A fundamental method — Backpropagation (cont'd)

- Recall the loss function[2]

$$R(\boldsymbol{\theta}) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}\left[y - g(z)\right]^2$$

where $z = w_0^{(2)} + w_1^{(2)} A_1 + w_2^{(2)} A_2 + w_3^{(2)} A_3$.

- We first estimate $\mathbf{w}^{(2)} = (w_0^{(2)}, w_1^{(2)}, w_2^{(2)}, w_3^{(2)})$

$$\frac{\partial R(\mathbf{w}^{(2)})}{\partial \mathbf{w}^{(2)}} = \frac{\partial R}{\partial g} \times \frac{\partial g}{\partial z} \times \frac{\partial z}{\partial \mathbf{w}^{(2)}} = -(y - g)g'(z)\mathbf{A}$$

where $\mathbf{A} = (1, A_1, A_2, A_3)$

- Then use gradient descent to find optimal $\mathbf{w}^{(2)}$

[2] we drop $\frac{1}{n}\sum_{i=1}^{n}$ for simplicity

A fundamental method — Backpropagation (cont'd)

- Now update the weights at the hidden layer given $\hat{\mathbf{w}}^{(2)}$
- Below is an example for estimating $\mathbf{w}_1^{(1)} = (w_{01}^{(1)}, w_{11}^{(1)}, ..., w_{51}^{(1)})$

$$\frac{\partial R(\mathbf{w}^{(1)})}{\partial \mathbf{w}^{(1)}} = \frac{\partial R}{\partial g} \times \frac{\partial g}{\partial z} \times \frac{\partial z}{\partial A_1} \times \frac{\partial A_1}{\partial z_1} \times \frac{\partial z_1}{\partial \mathbf{w}_1^{(1)}}$$
$$= -(y - g)g'(z)w_1^{(2)}\sigma'(z_1)\mathbf{X}$$

  where $\mathbf{X} = (1, X_1, ..., X_5)$ and $z_1 = \mathbf{X}^T\mathbf{w}_1^{(1)}$.
- Using gradient descent to update the parameters.
- Exercise:
  - Rewrite all these using matrix form for all observations.
  - Generalize it to multiple hidden layers.
  - Implement the algorithm in R.

Regularization and Stochastic Gradient Descent

- To speed up the optimization
- Stochastic gradient descent (SGD):
  - A general learning algorithm based on gradient descent
  - Efficient when n is large
  - Idea: use "minibatch" instead of all n observations to compute the gradient
- Regularization:
  - To avoid overfitting
  - Popular methods include ridge and Lasso penalties

Convolutional Neural Network (CNN)

- A deep neural network for image classification
- Convolution layers
  - a large number of convolution filters, each is a matrix
  - each determines if a particular local feature is in the image
  - convolution: element-wise matrix multiplication and summation
  - a visualization example
- Pooling layer
  - reduce the size of "convolved features"
  - a visualization example

An illustration – stripe detector

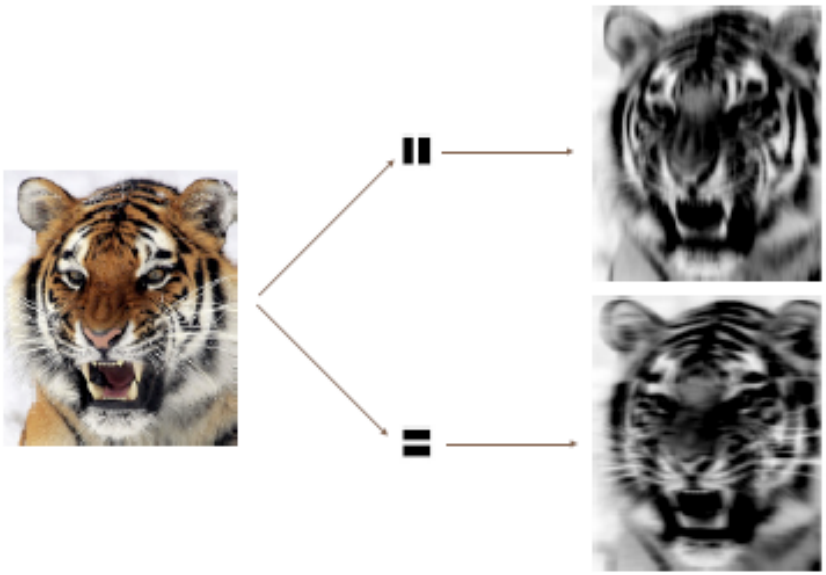


FIGURE 10.7. *Convolution filters find local features in an image, such as edges and small shapes. We begin with the image of the tiger shown on the left, and apply the two small convolution filters in the middle. The convolved images highlight areas in the original image where details similar to the filters are found. Specifically, the top convolved image highlights the tiger's vertical stripes, whereas the bottom convolved image highlights the tiger's horizontal stripes. We can think of the original image as the input layer in a convolutional neural network, and the convolved images as the units in the first hidden layer.*

Image source: ISLR (page 414)

Architecture of a CNN

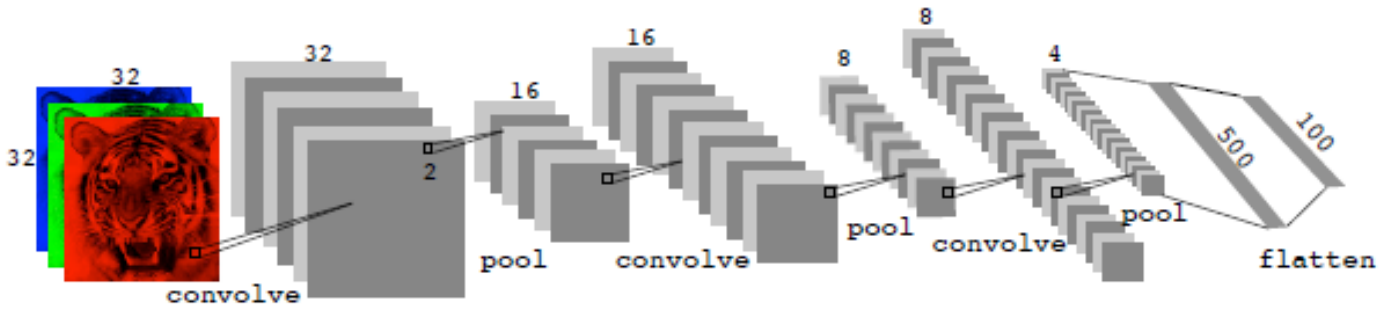


FIGURE 10.8. *Architecture of a deep CNN for the* CIFAR100 *classification task. Convolution layers are interspersed with* 2×2 *max-pool layers, which reduce the size by a factor of 2 in both dimensions.*

Image source: ISLR (page 416)

Other Deep Neural Networks

- Recurrent neural networks
- Generative neural networks
- A good article everyone should read.