DATASCI 347 Machine Learning

Lecture 17: Neural Networks

Ruoxuan Xiong

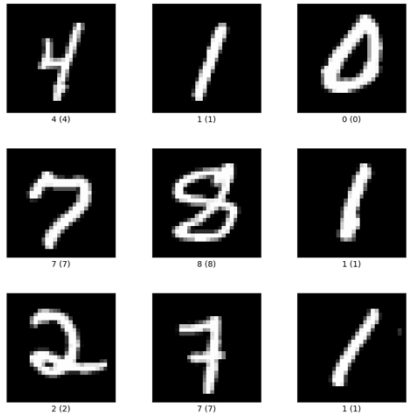
Suggested reading: ISL Chapter 10

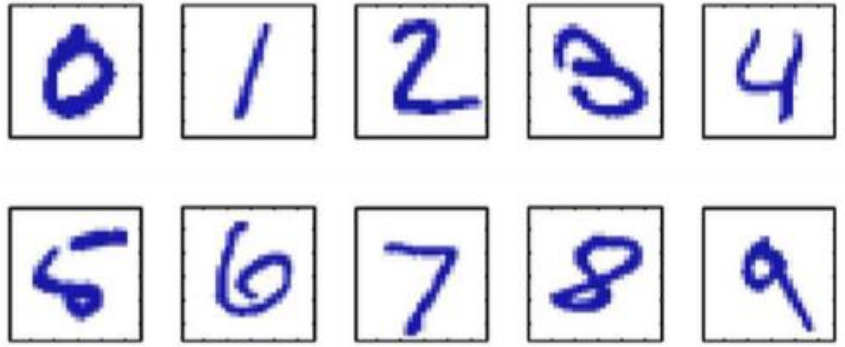
Announcements

- Extension of HW3 deadline to Friday 4/3
- Office hour this week: Tuesday 3-4 PM

Recognizing handwritten digits

- Input is a collection of handwritten digits from 0 to 9 in black and white





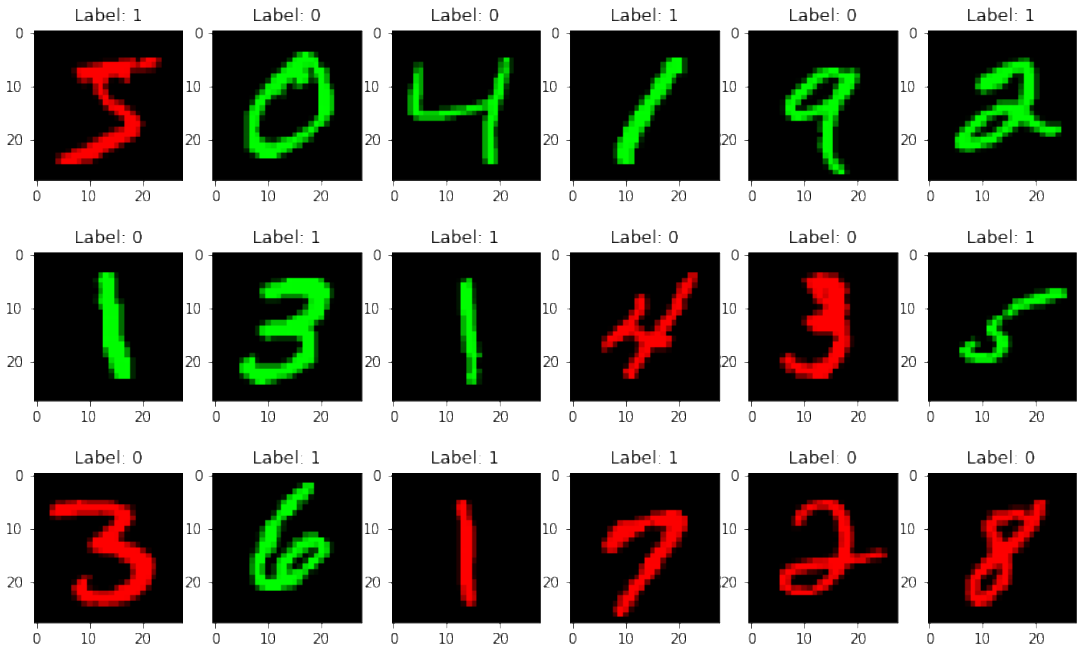
- **MNIST data set**
  - 50,000 handwritten digits for training
  - 5,000 for validation
  - 5,000 for testing

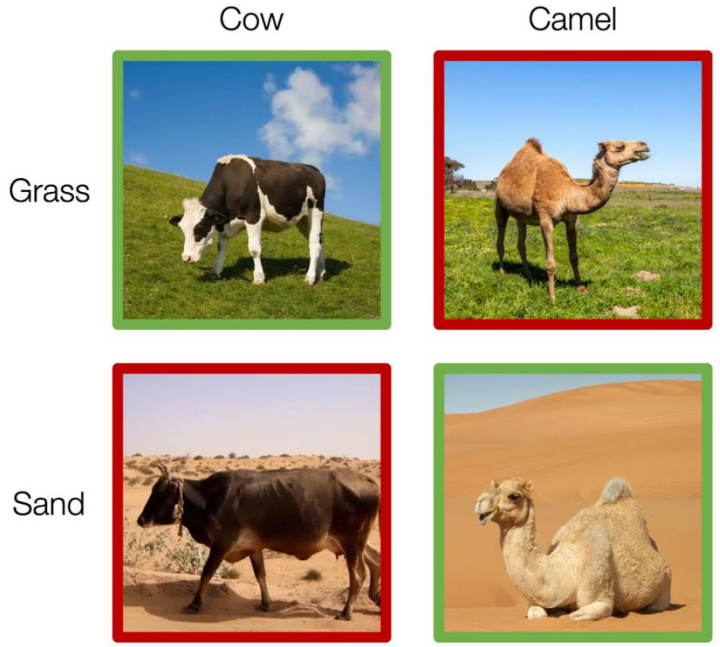
Link to dataset http://yann.lecun.com/exdb/mnist/

EMORY

Colored digits

- Colored MNIST: Colored digits in a black ground
  - Input is represented from 3 times 28 times 28 pixels



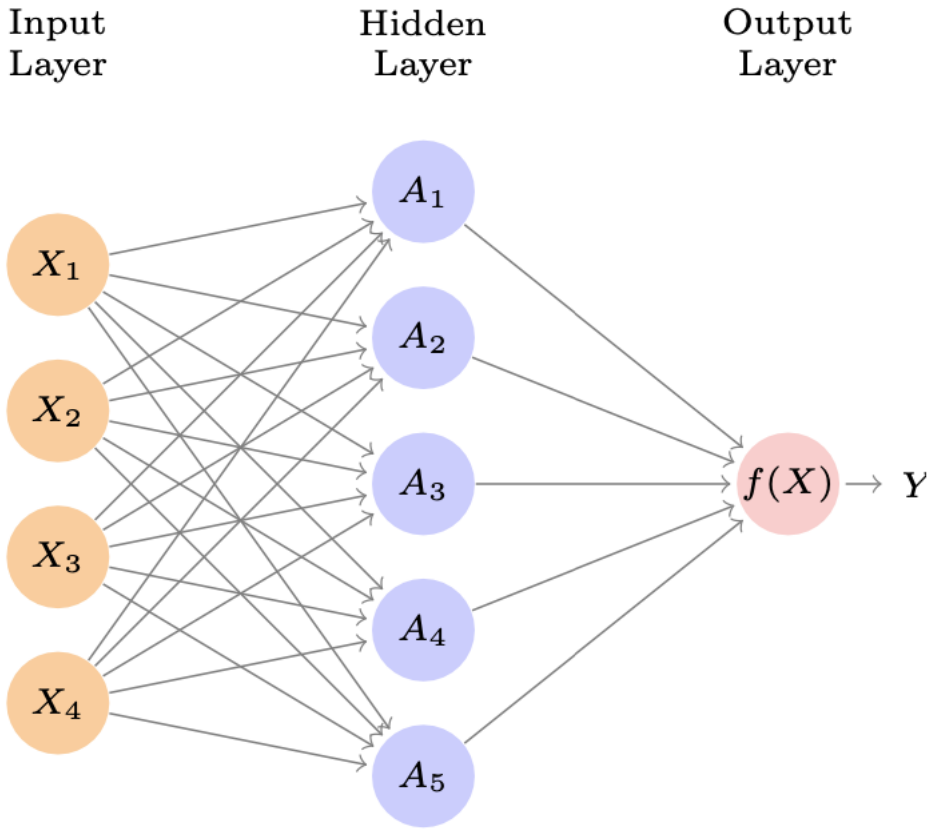


  - A naive model may simply predict the digit based on its color---a problem known as **spurious correlation**
  - Link: https://github.com/facebookresearch/InvariantRiskMinimization/blob/main/code/colored_mnist/main.py

EMORY

Single-layer neural network

- A neural network takes an input vector $X = (X_1, X_2, \cdots, X_p)$ and builds a nonlinear function $f(X)$ to predict Y



- This simple approach works well on MNIST (over 90% test accuracy)

Single-layer neural network

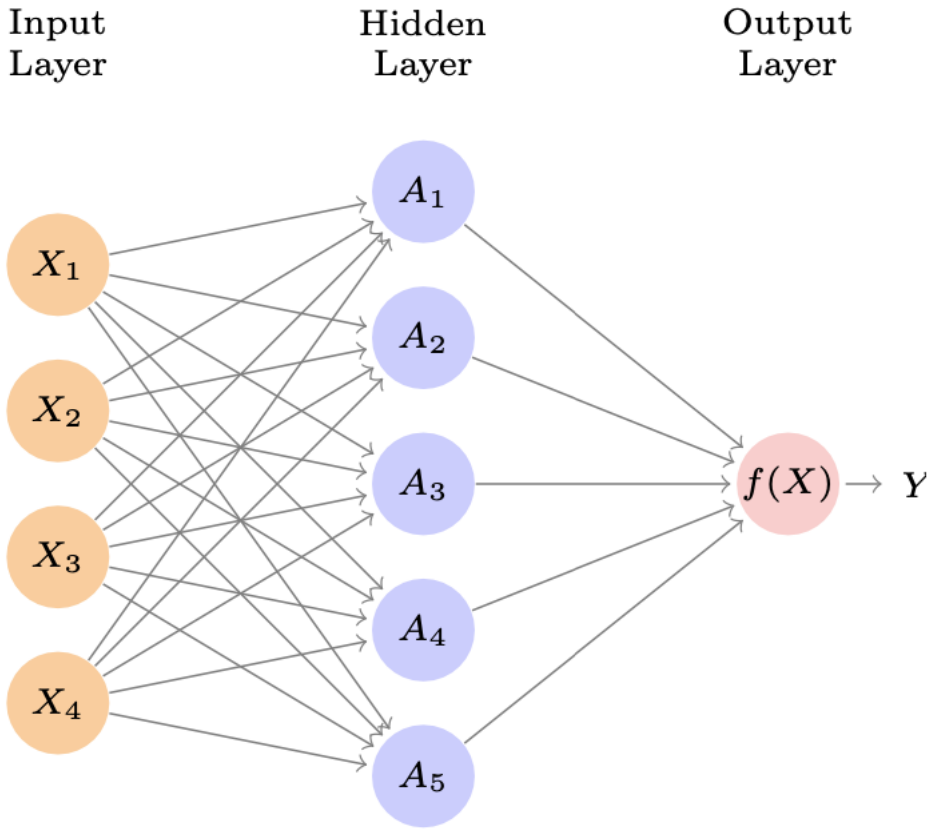
- The neural network model has the form

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

Here $h_k(X)$ is a neuron / hidden unit (analogous to neurons in the brain)

$$A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

g is a nonlinear activation function



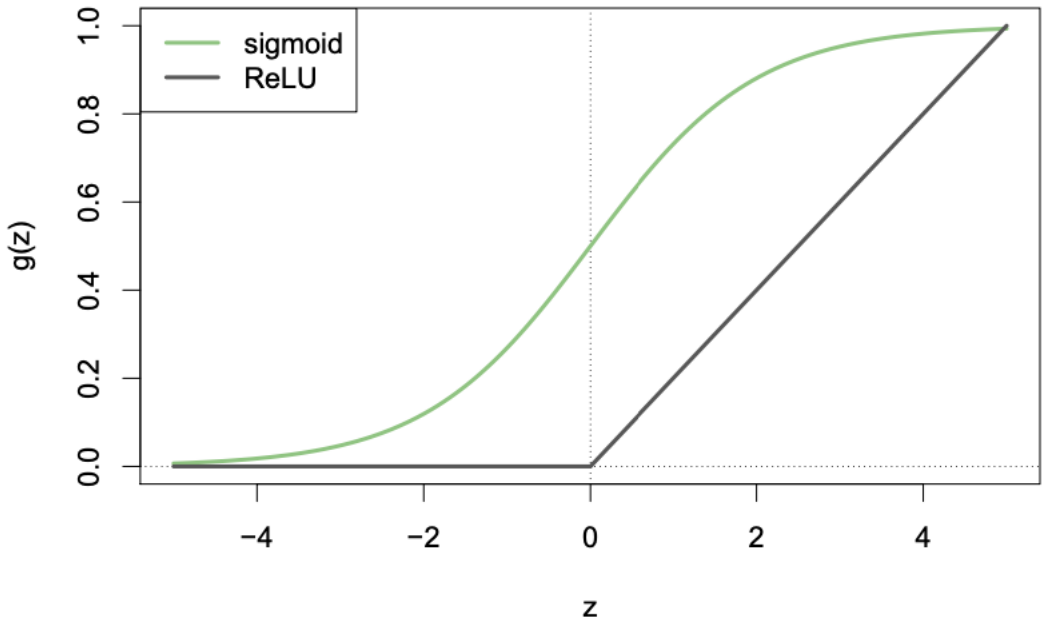
Activation function

- In the early instances of neural networks, the *sigmoid activation function* is preferred

$$g(z) = \frac{1}{1+e^{-z}}$$

- In modern neural networks, *ReLU (rectified linear unit)* is the preferred choice

$$g(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$$



A toy example

- Two features $X = (X_1, X_2)$ and two hidden units $h_1(X)$ and $h_2(X)$ with $g(z) = z^2$

- We specify parameters as (9 parameters in total)

$$\beta_0 = 0 \quad \beta_1 = \frac{1}{4} \quad \beta_2 = -\frac{1}{4}$$
$$w_{10} = 0 \quad w_{11} = 1 \quad w_{12} = 1$$
$$w_{20} = 0 \quad w_{21} = 1 \quad w_{22} = -1$$

$$h_1(X) = (w_{10} + w_{11}X_1 + w_{12}X_2)^2 = (0 + X_1 + X_2)^2$$

$$h_2(X) = (w_{20} + w_{21}X_1 + w_{22}X_2)^2 = (0 + X_1 - X_2)^2$$

$$f(X) = \beta_0 + \beta_1 h_1(X) + \beta_2 h_2(X) = \frac{1}{4}(0 + X_1 + X_2)^2 - \frac{1}{4}(0 + X_1 - X_2)^2 = X_1X_2$$

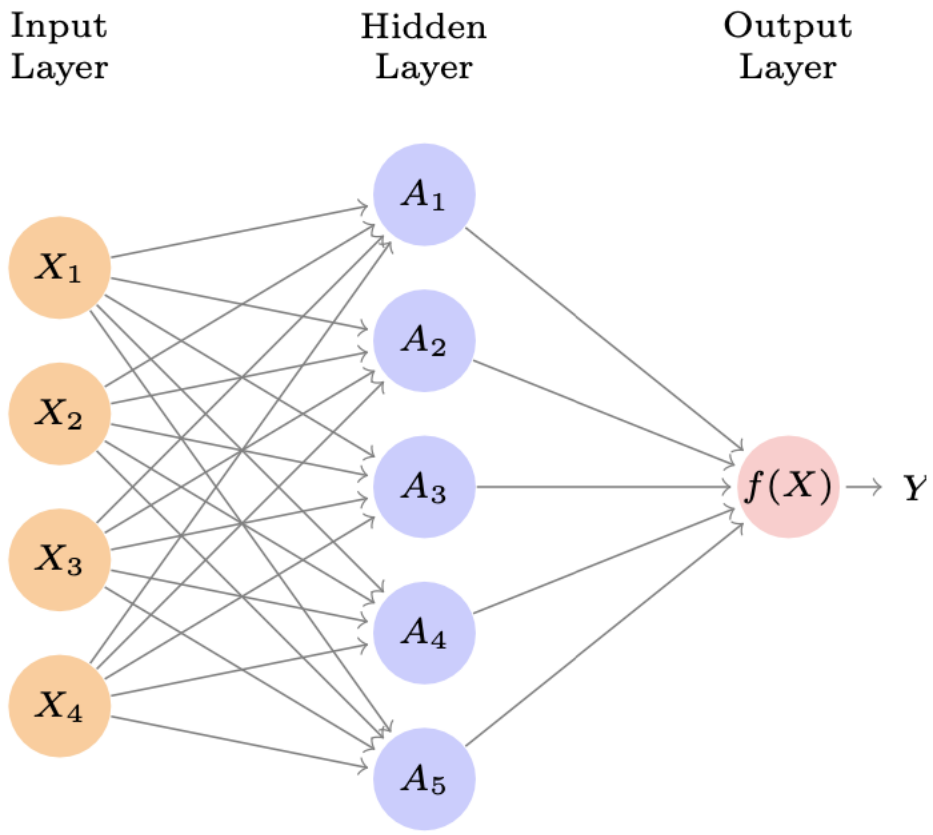
EMORY

Another example

- Three features $X = (X_1, X_2, X_3)$ and one hidden unit $h(X)$ with ReLU activation $g(z) = z$ if $z \geq 0$ and 0 otherwise
- There is a brand-new restaurant that had just opened in Decatur
  - X_1 = "Is the dinner over \$30 per person?"; $w_1 = -30$
  - X_2 = "Is the parking fee over \$10?"; $w_2 = -10$
  - X_3 = "Is the wait time over half an hour?"; $w_3 = -10$
  - Intercept $w_0 = 40$ (can be interpreted as budget)
- $y = h(X) = g(w_0 + w_1X_1 + w_2X_2 + w_3X_3)$ determines whether to go or not
  - If $X_1 = 1, X_2 = 1, X_3 = 0$, then $y = 1$
  - If $X_1 = 0, X_2 = 1, X_3 = 1$, then $y = 1$
  - If $X_1 = 1, X_2 = 1, X_3 = 1$, then $y = 0$

Design choices

- Width: Number of neurons in the hidden layer
- In the following example, width is five

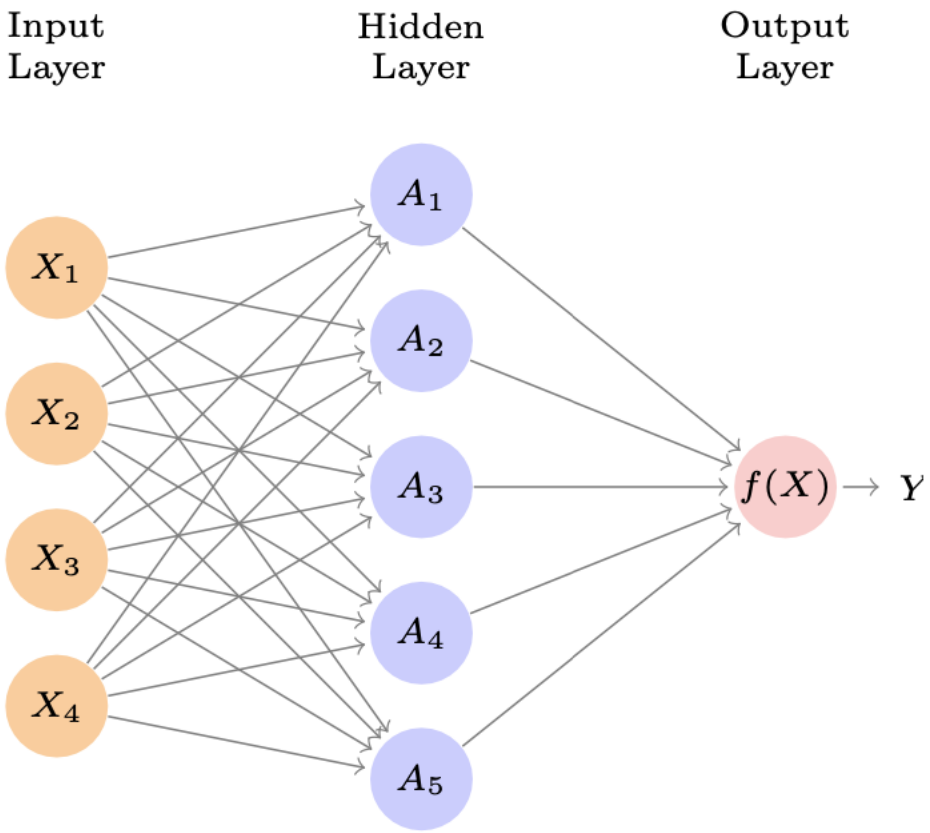


Design choices

- Width also determines the number of parameters in the network
- Quiz: how many parameters in total?

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

$$A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

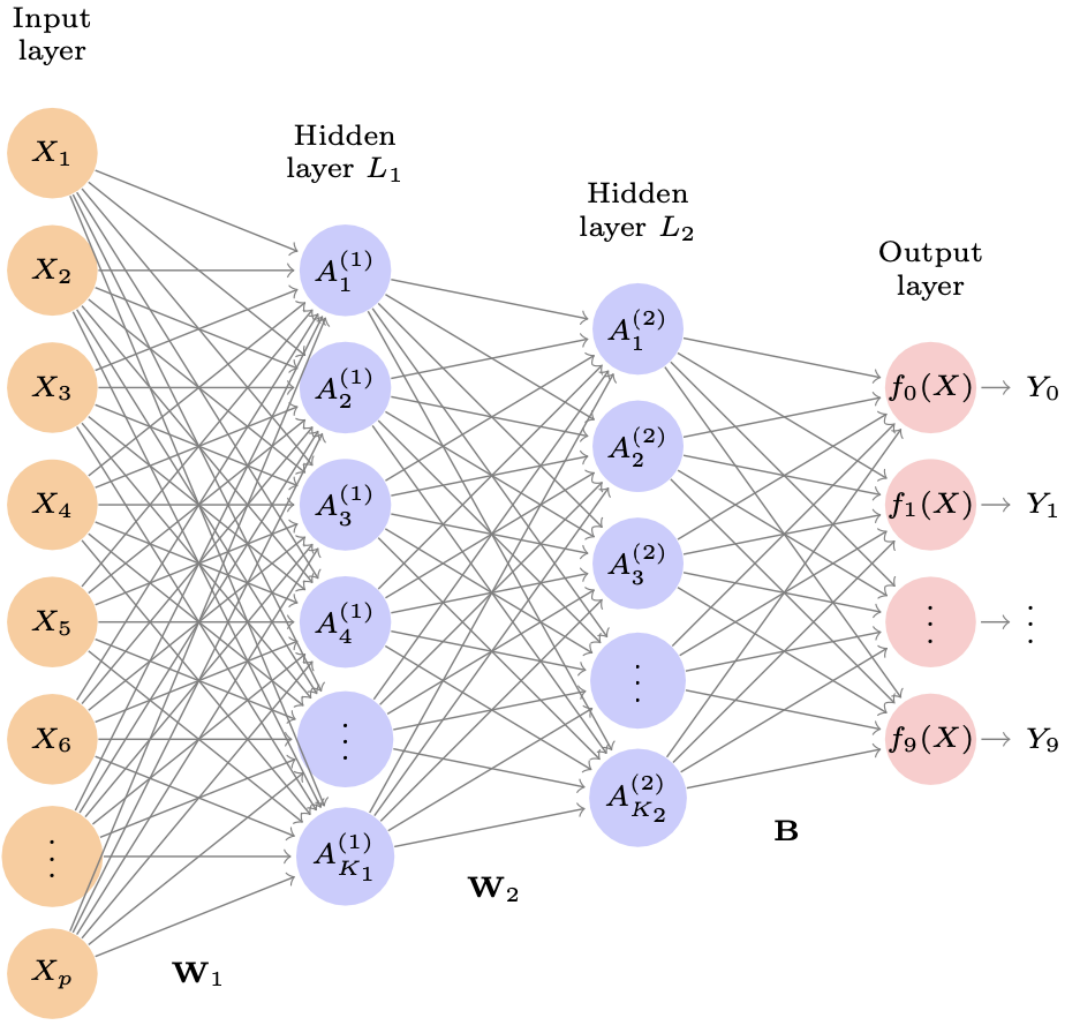


Multilayer neural networks

- Modern neural networks typically have more than one hidden layer and often many units per layer

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

Test error rate on the MNIST data



Parameters in modern neural networks

- Number of parameters is often very large for modern neural networks
- Large parameter space comes with large model capacity

Deep networks

ConvNet	# Params
AlexNet	60M
VGG19	140M
ResNet-50	25M

$\gg$

IMAGENET

1.5M

- Number of parameters can be much higher than the number of labeled examples

Loss function to fit a neural network

- Fitting a neural network requires estimating the unknown parameters (β_k and w_{kj})

- For a regression problem, typically *squared-error* is used

$$\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2$$

Loss function to fit a neural network

- For a classification problem, typically *cross-entropy (negative multinomial log-likelihood)* is used

  $-\sum_{i=1}^{n}\sum_{m} y_{im}\log(f_m(x_i))$

  $Y = (Y_0, Y_1, \cdots, Y_M)$ with *a one* in the position corresponding to label and zeros elsewhere ($M+1$ classes)

  $f_m(X) = \Pr(Y = m \mid X)$ the *probability* in class m

- Example: for MNIST, the label space is $\{0,1,2,\ldots,9\}$. The probability $f_m(X)$ for all m may look like [0.01, **0.9**, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01, 0.01]

PyTorch

CrossEntropyLoss = Negative Log Likelihood applied to SoftMax

- L is the label space
- y_n is the label of x_n
- $x_{n,c}$ is the softmax output probability of x_n for label c

```
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=- 100,
      reduce=None, reduction='mean', label_smoothing=0.0) [SOURCE]
```

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html