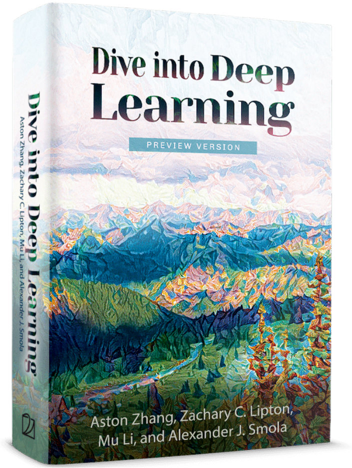
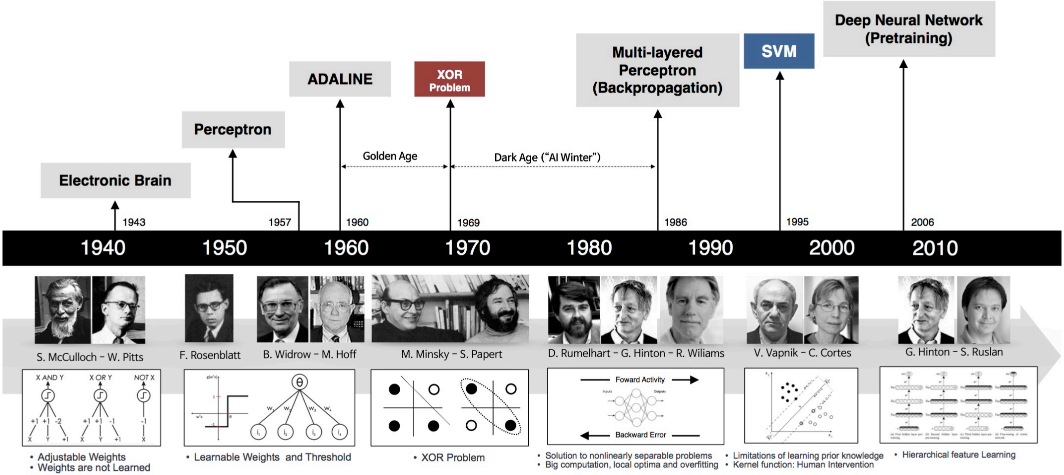
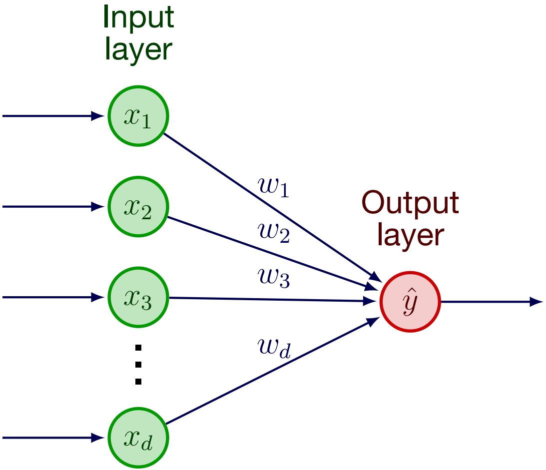
Introduction to deep learning


Dive into Deep Learning
PREVIEW VERSION
Aston Zhang, Zachary C. Lipton,
Mu Li, and Alexander J. Smola
Dive into Deep Learning
Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola

https://d2l.ai


Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Linear Regression (revisited)

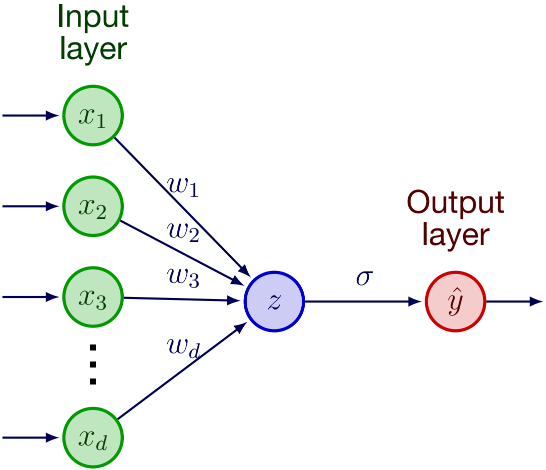


Input: $x = (x_1, x_2, \ldots, x_d)$

Prediction:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + b$$

Logistic Regression



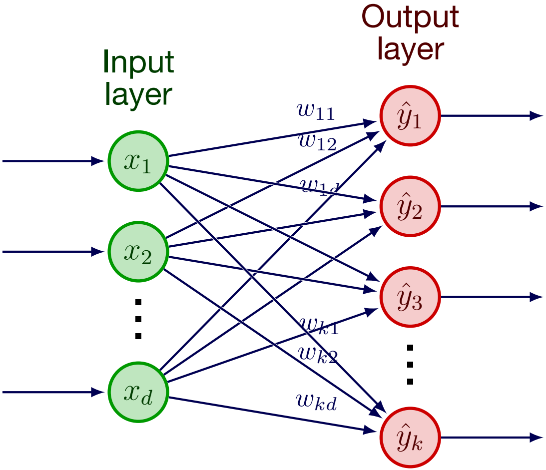
Input: $x_1, x_2, \ldots, x_d$

Sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$

Prediction:

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + b)$$

Multivariate Regression



Input: $x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$ **Output:** $\hat{y} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_k \end{pmatrix}$

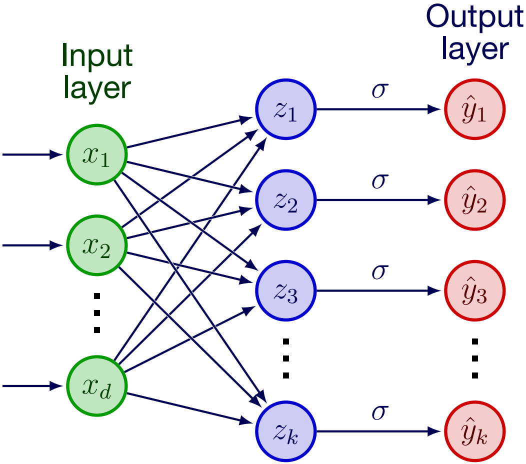
Weight matrix:

$$W = \begin{pmatrix} w_{11} & \dots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{k1} & \dots & w_{kd} \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix}$$

Prediction:

$$\hat{y} = Wx + b$$

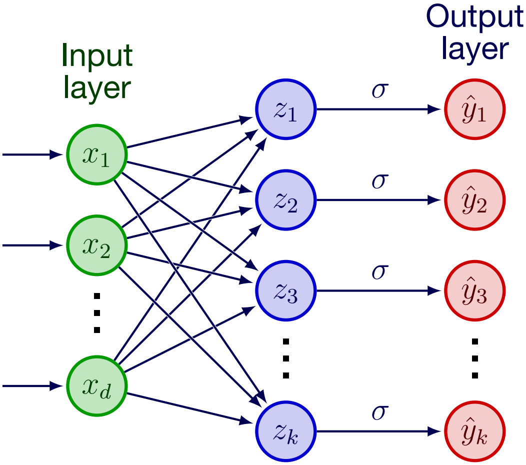
Multiclass Classification



Input: $x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$ **Output:** $\hat{y} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_k \end{pmatrix}$

$$\hat{y}_1 + \hat{y}_2 + \ldots + \hat{y}_k = 1$$

Multiclass Classification



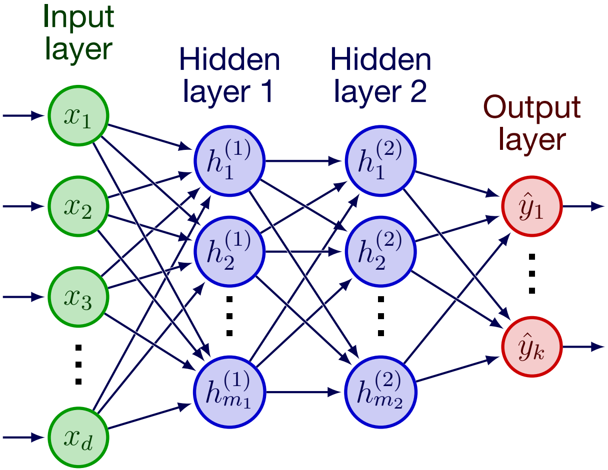
Input: $x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$ **Output:** $\hat{y} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_k \end{pmatrix}$

$$\hat{y}_1 + \hat{y}_2 + \ldots + \hat{y}_k = 1$$

Softmax function:

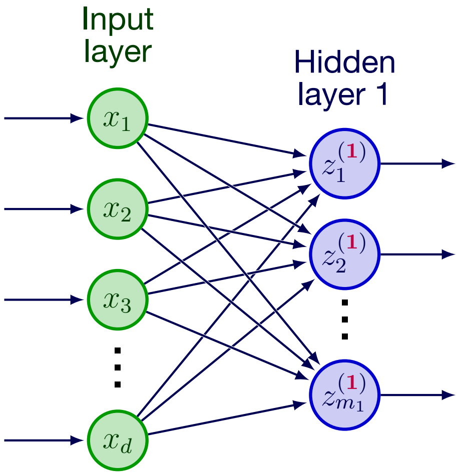
$$\hat{y}_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \ldots + e^{z_k}}$$

Neural Networks



Not shown: **activation function** $\sigma_1, \sigma_2, \sigma_3$ after each (non-input) layer

Prediction steps (1)

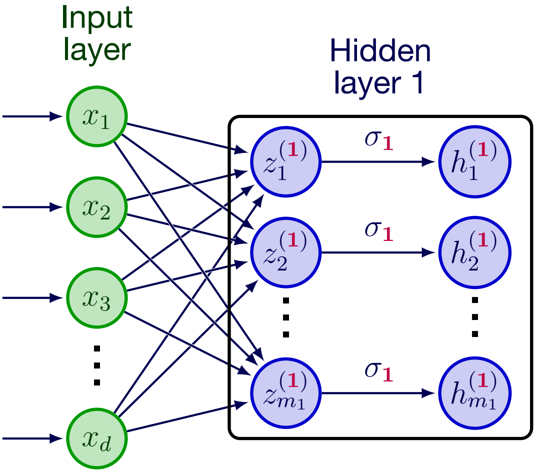


Hidden Layer 1:

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & \dots & w_{1d}^{(1)} \\ \vdots & \ddots & \vdots \\ w_{k1}^{(1)} & \dots & w_{m_1 d}^{(1)} \end{pmatrix} \qquad b^{(1)} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_{m_1}^{(1)} \end{pmatrix}$$

$$z^{(1)} = W^{(1)} x + b^{(1)}$$

Prediction steps (2)



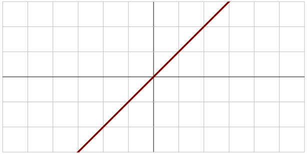
Hidden Layer 1:

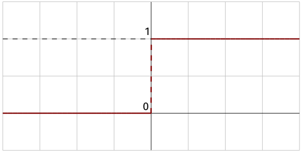
σ_1: **activation function**

$$h^{(1)} = \begin{pmatrix} h_1^{(1)} \\ \vdots \\ h_{m_1}^{(1)} \end{pmatrix} \qquad z^{(1)} = \begin{pmatrix} z_1^{(1)} \\ \vdots \\ z_{m_1}^{(1)} \end{pmatrix}$$

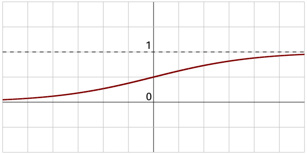
$$h^{(1)} = \sigma_1 \left(z^{(1)} \right)$$

Examples of activation functions

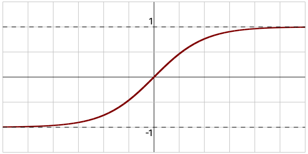

identity



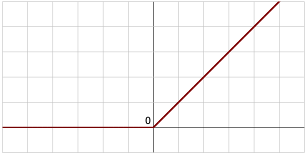
step



logistic/sigmoid

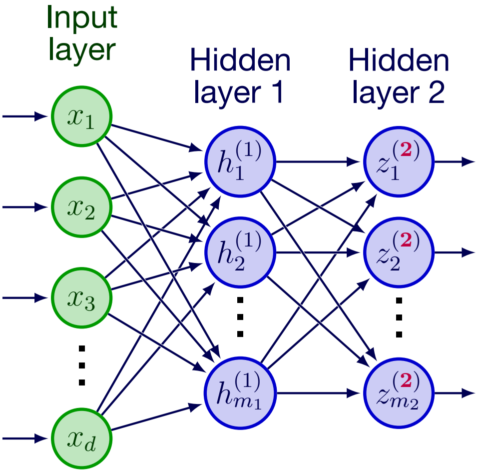


tanh



ReLU/ramp

Prediction steps (3)

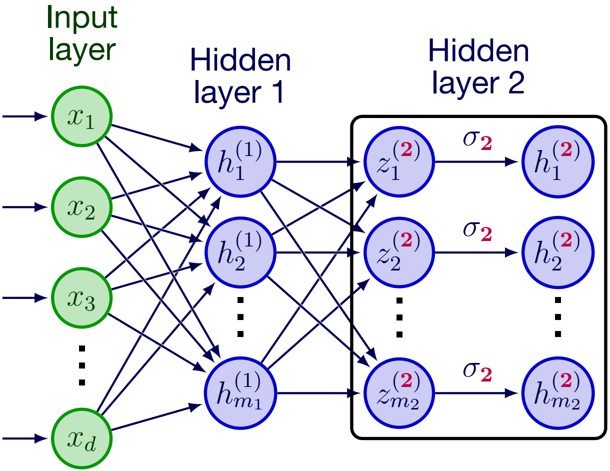


Hidden Layer 2:

$$W^{(2)} = \begin{pmatrix} w_{11}^{(2)} & \cdots & w_{1m_1}^{(2)} \\ \vdots & \ddots & \vdots \\ w_{m_2 1}^{(2)} & \cdots & w_{m_2 m_1}^{(2)} \end{pmatrix} \quad b^{(2)} = \begin{pmatrix} b_1^{(2)} \\ \vdots \\ b_{m_2}^{(2)} \end{pmatrix}$$

$$z^{(2)} = W^{(2)} h^{(1)} + b^{(2)}$$

Prediction steps (4)



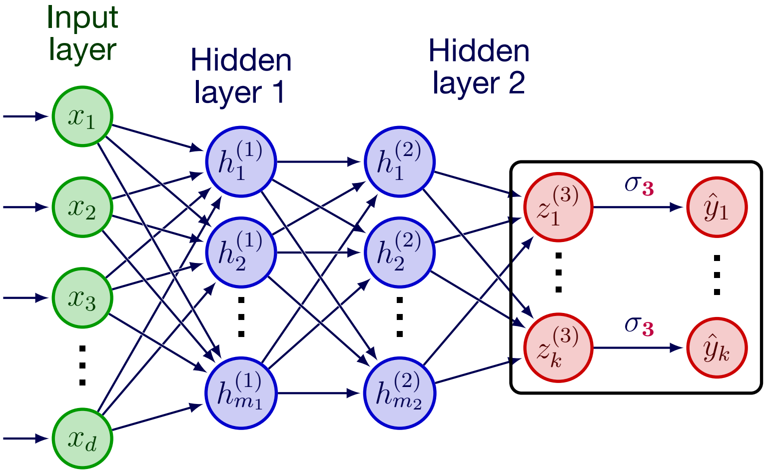
Hidden Layer 2:

σ_2: activation function

$$h^{(2)} = \begin{pmatrix} h_1^{(2)} \\ \vdots \\ h_{m_2}^{(2)} \end{pmatrix} \qquad z^{(2)} = \begin{pmatrix} z_2^{(2)} \\ \vdots \\ z_{m_2}^{(2)} \end{pmatrix}$$

$$h^{(2)} = \sigma_2\left(z^{(2)}\right)$$

Prediction steps (5)



Output Layer:

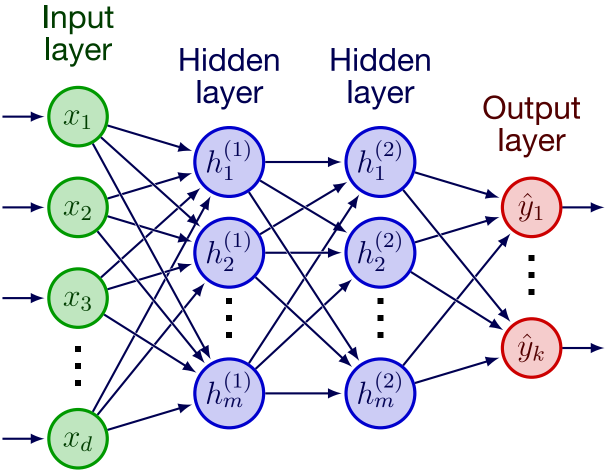
$$W^{(\mathbf{3})} = \begin{pmatrix} w_{11}^{(\mathbf{3})} & \dots & w_{1m_2}^{(\mathbf{3})} \\ \vdots & \ddots & \vdots \\ w_{k1}^{(\mathbf{3})} & \dots & w_{km_2}^{(\mathbf{3})} \end{pmatrix}$$

$$b^{(\mathbf{3})} = \begin{pmatrix} b_1^{(\mathbf{3})} \\ \vdots \\ b_k^{(\mathbf{3})} \end{pmatrix}$$

$$z^{(\mathbf{3})} = W^{(\mathbf{3})} h^{(2)} + b^{(\mathbf{3})}$$

$$y = \sigma_{\mathbf{3}} \left(z^{(\mathbf{3})} \right)$$

Optimizing neural networks



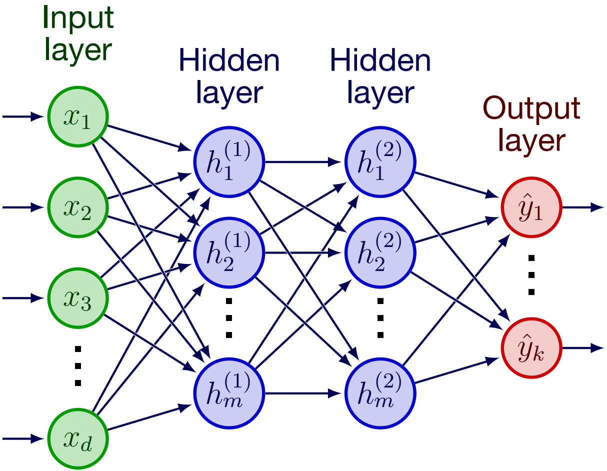
Data: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$

Parameters: $w = (w_1, \ldots, w_K)$

Prediction:

$$\hat{y}^{(i)} = \mathrm{NeuralNet}(x, w)_i$$

Optimizing neural networks



Data: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$

Parameters: $w = (w_1, \ldots, w_K)$

Prediction:

$$\hat{y}^{(i)} = \text{NeuralNet}(x, w)_i$$

Specify a **loss function**

$$\underbrace{L}_{\text{a function of } w_1 \ldots, w_K} = \sum_{i=1}^{n} \ell(y^{(i)}, \hat{y}^{(i)})$$

Goal: Minimize L over $w_1, \ldots, w_K$

Examples of loss functions

- Regression $(y, \hat{y} \in \mathbb{R}^k)$

$$\ell(y, \hat{y}) = \|y - \hat{y}\|_2^2$$ (mean-squared error)

Examples of loss functions

- Regression $(y, \hat{y} \in \mathbb{R}^k)$

$$\ell(y, \hat{y}) = \|y - \hat{y}\|_2^2 \qquad \text{(mean-squared error)}$$

- Binary Classification $(y \in \{0, 1\}, \hat{y} \in (0, 1))$

$$\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \qquad \text{(binary cross-entropy)}$$

Examples of loss functions

- Regression $(y, \hat{y} \in \mathbb{R}^k)$

$$\ell(y, \hat{y}) = \|y - \hat{y}\|_2^2$$ (mean-squared error)

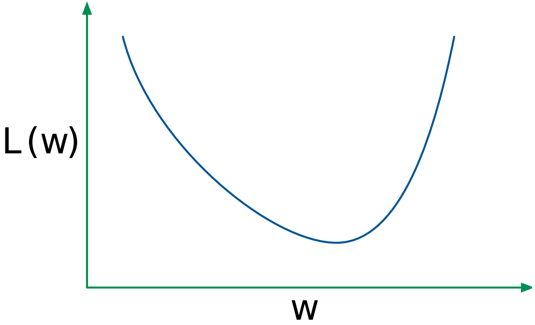
- Binary Classification $(y \in \{0, 1\}, \hat{y} \in (0, 1))$

$$\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$ (binary cross-entropy)

- Multiclass Classification
    - $y = (0, \ldots, 0, 1, 0, \ldots, 0)$; 1 at i-th position if (x, y) is in class i
    - $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_k)$; $y_i \in (0, 1)$ and $\sum_i y_i = 1$

$$\ell(y, \hat{y}) = -y_1 \log \hat{y}_1 - y_2 \log \hat{y}_2 - \ldots - y_k \log \hat{y}_k$$ (categorical cross-entropy)

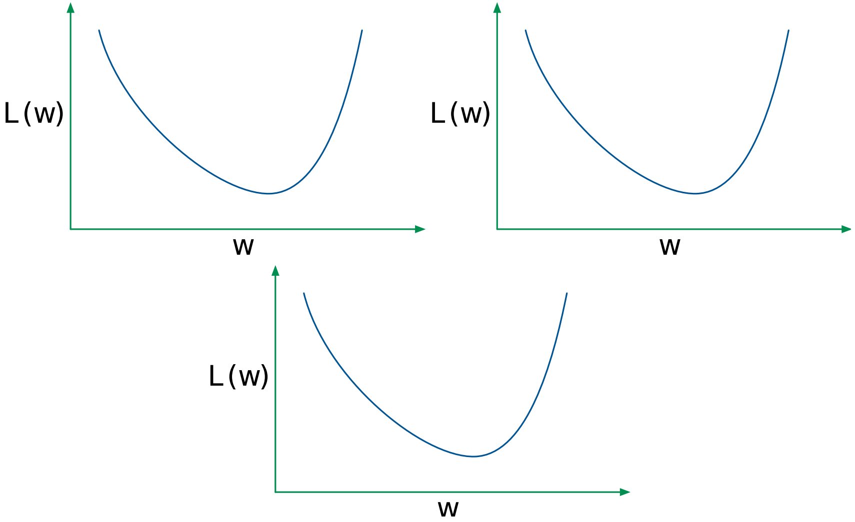
Optimization method: Gradient Descent



$$w^{t+1} \leftarrow w^t - \eta \frac{\partial L}{\partial w}(w^t)$$

$\eta =$ **Learning rate**

Learning rate affects the optimization



Neural network optimization

Let $L(w_1, \ldots, w_K) = \sum_{i=1}^{n} \ell(\hat{y}^{(i)}, y^{(i)})$

Goal: Minimize loss $L(w_1, \ldots, w_K)$ over $w_1, \ldots, w_K$

Specify the **learning rate** $\eta > 0$.

1. Start with random $w_1^0, \ldots, w_K^0$.
2. At $t > 0$, for $i = 1, \ldots, K$,

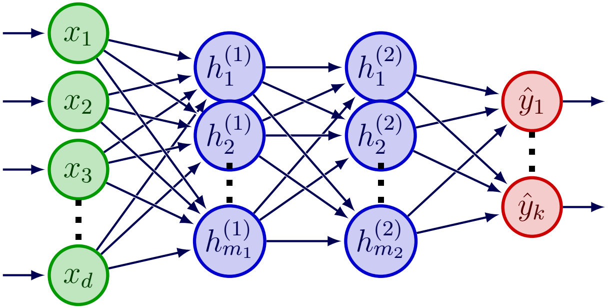
$$w_i^{t+1} \leftarrow w_i^t - \eta \frac{\partial L}{\partial w_i}(w^t).$$

Typically, $K > 100,000$. Can we efficiently compute $\frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_K}$?

Backpropagation (1)

Rumelhart, Geoffrey E. Hinton & Ronald J. Williams (1986)

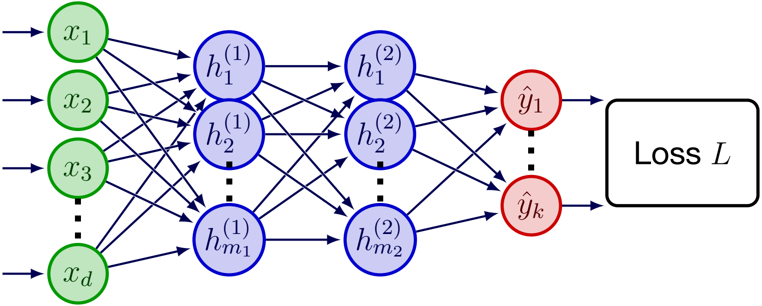
- At step t, the current parameters are $w_1^t, \dots, w_K^t$
- With these parameters, pass the input data (x) from **left** to **right**, and store the output at each node



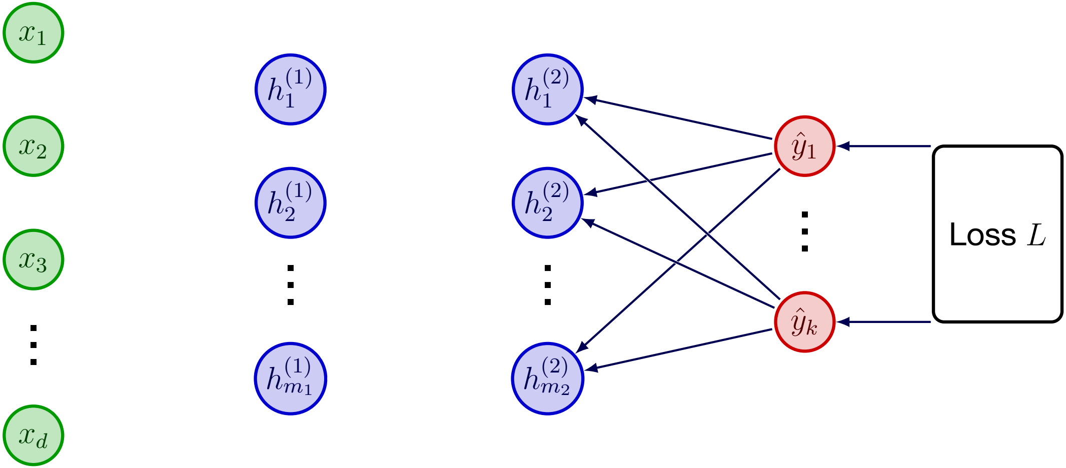
Backpropagation (2)

Rumelhart, Geoffrey E. Hinton & Ronald J. Williams (1986)

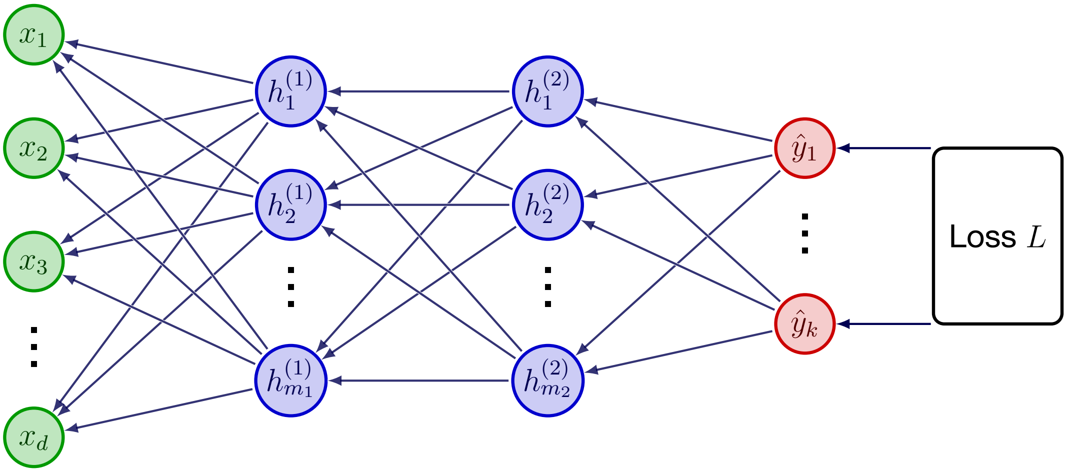
- Compute the Loss L
- With the power of Chain Rule, we can compute the partial derivative from **right** to **left**



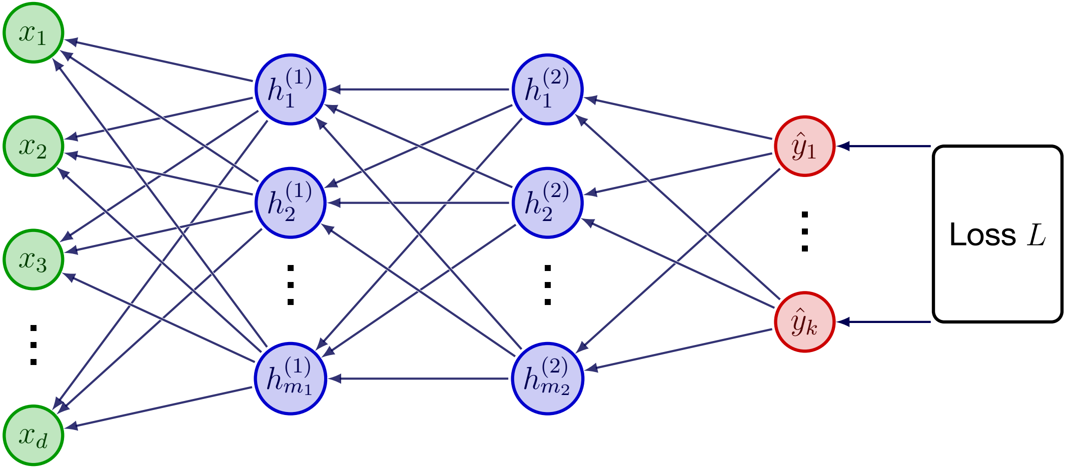
Backpropagation (3)



Backpropagation (3)



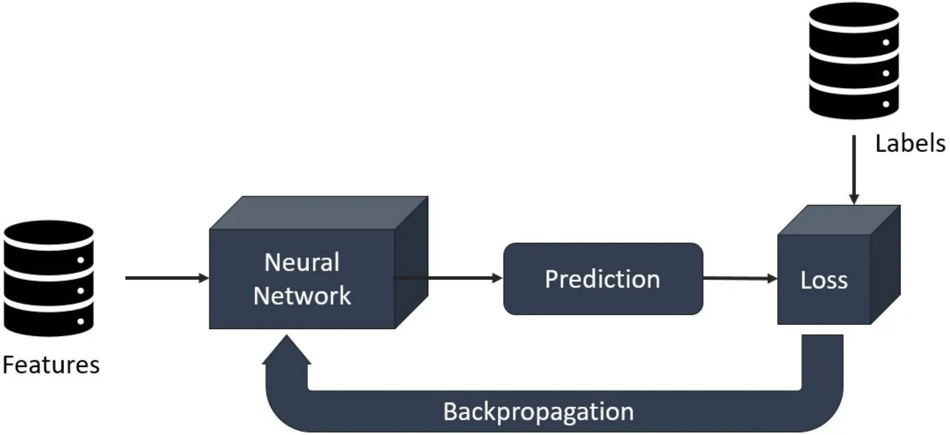
Backpropagation (3)



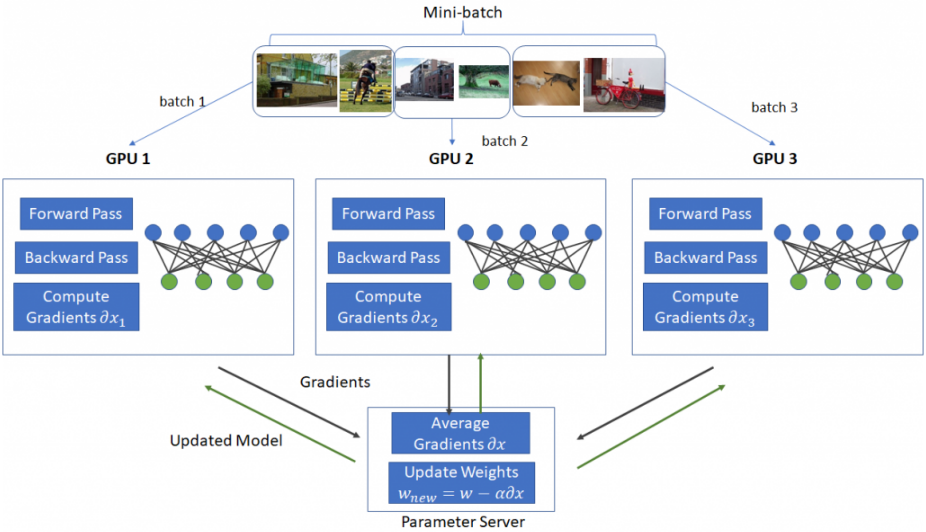
3Blue1Brown’s animation of backpropagation

https://www.youtube.com/watch?v=Ilg3gGewQ5U

Training cycle



Training with GPUs



Set these values before training an NN

- Number of layers
- Number of nodes in each layer
- Activation functions at each layer

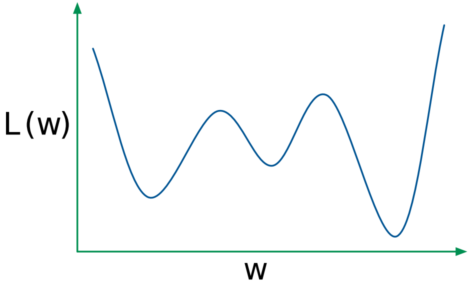
Set these values before training an NN

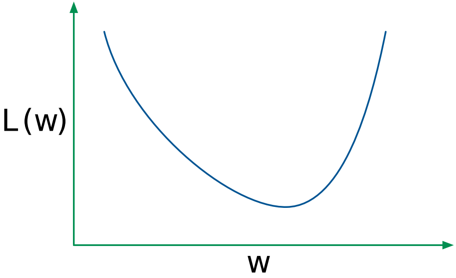
- **Number of layers**
- **Number of nodes in each layer**
- **Activation functions at each layer**
- **Batch size**: How many instances in each minibatch?
- **Epoch**: How many cycles over the whole dataset?
- **Learning rate**: Should not be too large or too small

Summary

- To optimize the parameters of neural networks, we use **gradient descent**
- In each iteration, we first compute and store the values of each node forward
- Then, use the stored values to compute the gradient backward
- Finally, use the gradient wrt parameters to update the parameters with $w_i^{t+1} \leftarrow w_i^t - \eta \frac{\partial L}{\partial w_i}$

Challenge: Nonconvex Optimization





How to escape from local (but not global) minima?

- Multiple random initializations
- Start with high learning rate, then decrease it later
- Use modern optimization technique such as **Adam** and **RMSProp**