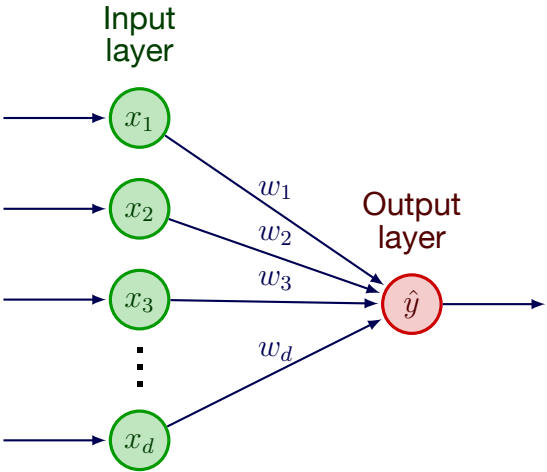
Nonconvex optimization

Source: Julian McAuley, Personalized Machine Learning (2022).

Neural Networks

Recommender Systems

Linear Regression (revisited)

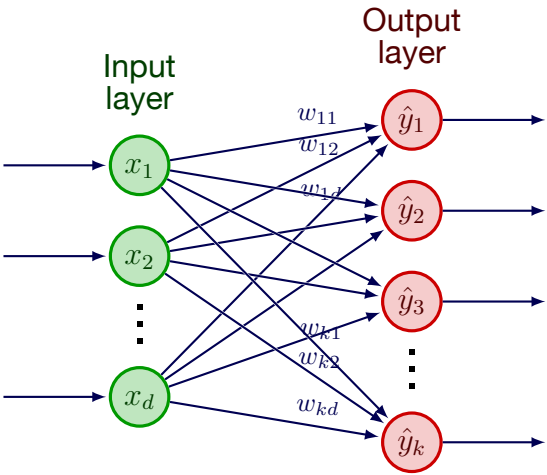


Input: $x = (x_1, x_2, \ldots, x_d)$

Prediction:

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

Multivariate Regression



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

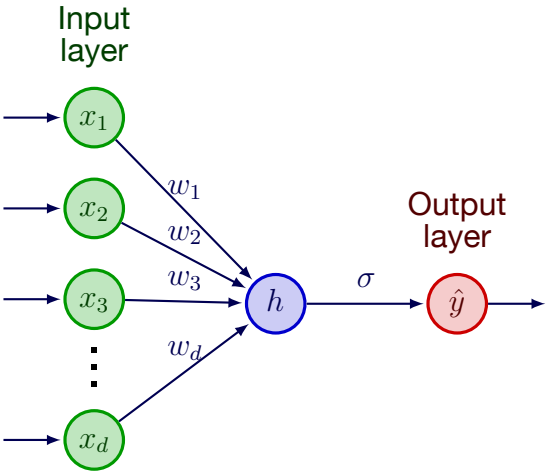
Weight matrix:

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

Prediction:

$$\hat{y} = Wx$$

Logistic Regression



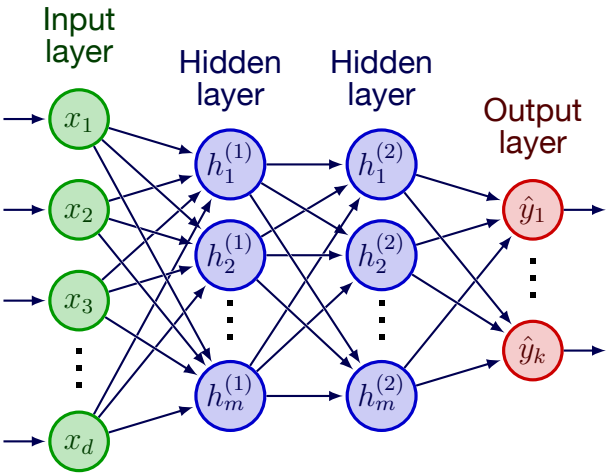
Input: $x_1, x_2, \ldots, x_d$

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

Prediction:

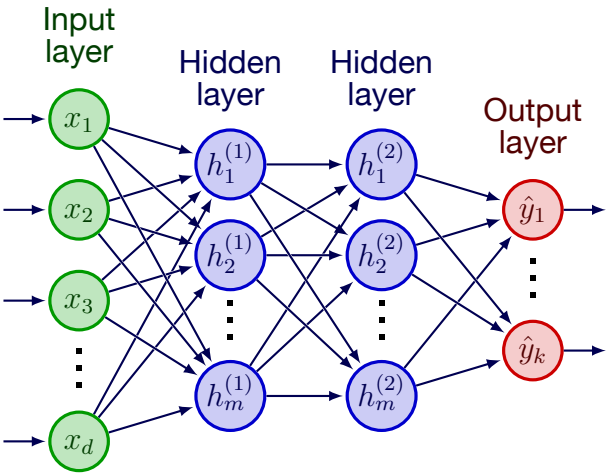
$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_d x_d)$$

Neural Networks



Layers: $l = 1, 2, 3, 4$

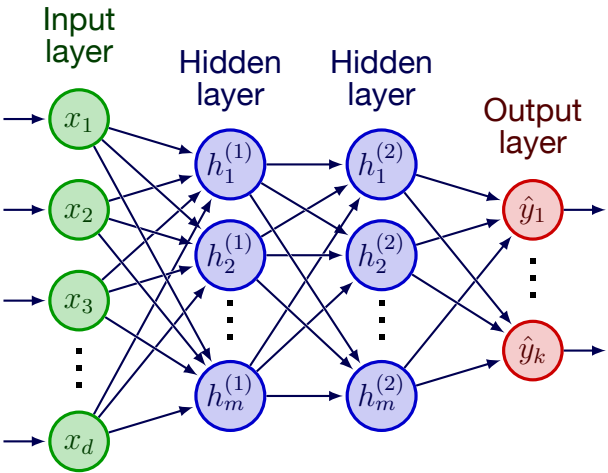
Neural Networks



Layers: $l = 1, 2, 3, 4$

$\sigma_l =$ **activation function**

Neural Networks

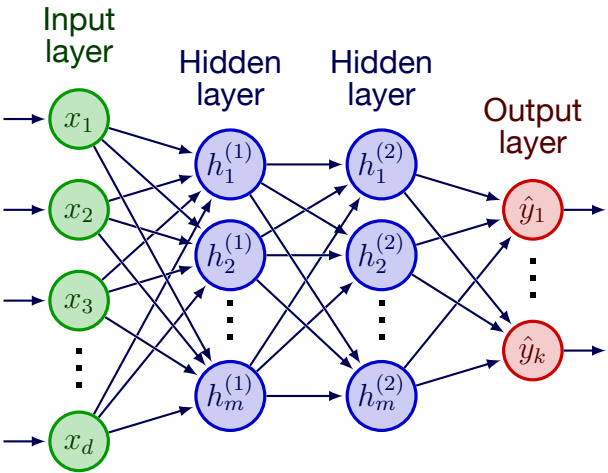


Layers: $l = 1, 2, 3, 4$

$\sigma_l =$ **activation function**

$w_{ij}^{(l)}$: weight from $h_j^{(l)} \rightarrow h_i^{(l+1)}$

Neural Networks



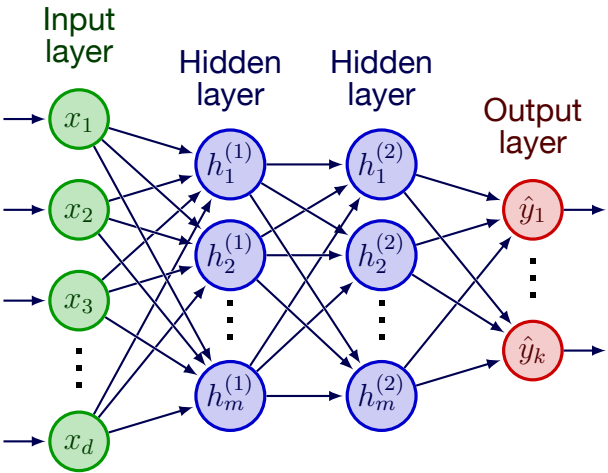
Layers: $l = 1, 2, 3, 4$

$\sigma_l =$ **activation function**

$w_{ij}^{(l)}$: weight from $h_j^{(l)} \rightarrow h_i^{(l+1)}$

$W_l = \left(w_{ij}^{(l)}\right)$

Neural Networks



Layers: $l = 1, 2, 3, 4$

$\sigma_l =$ **activation function**

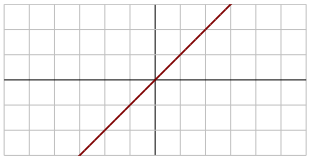
$w_{ij}^{(l)}$: weight from $h_j^{(l)} \to h_i^{(l+1)}$

$$W_l = \left(w_{ij}^{(l)}\right)$$

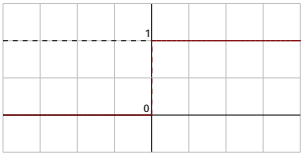
Prediction:

$$h^{(1)} = \sigma_1\left(W_1 x\right)$$
$$h^{(2)} = \sigma_2\left(W_2 h^{(1)}\right)$$
$$\hat{y} = \sigma_3\left(W_3 h^{(2)}\right)$$

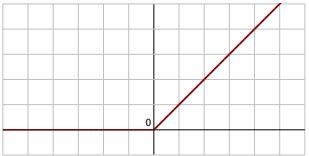
Activation functions



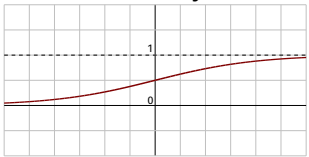
identity



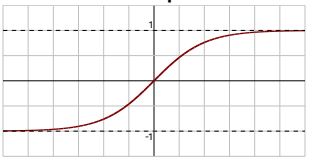
step



ReLU/ramp

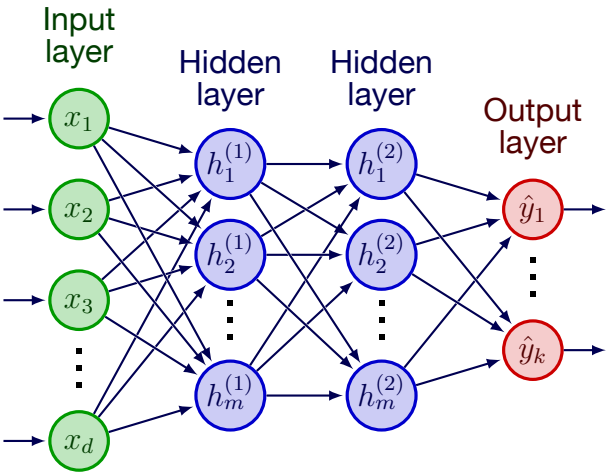


logistic/sigmoid



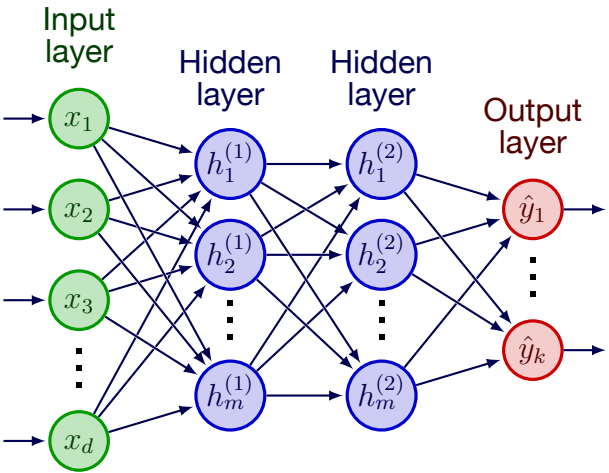
tanh

Optimizing neural networks



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

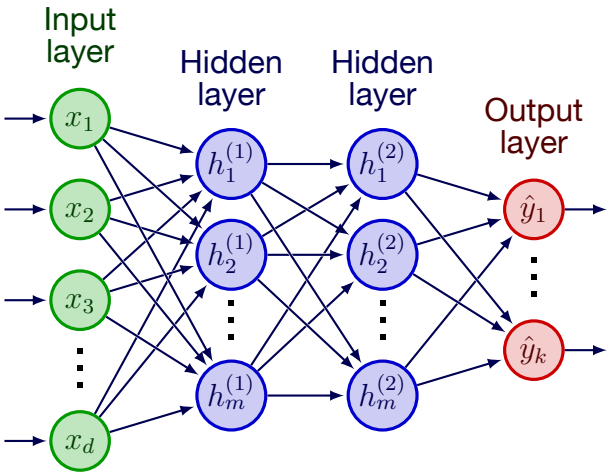
Optimizing neural networks



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

Parameters: $w = (w_{11}^{(1)}, \ldots, w_{kd}^{(l)})$

Optimizing neural networks

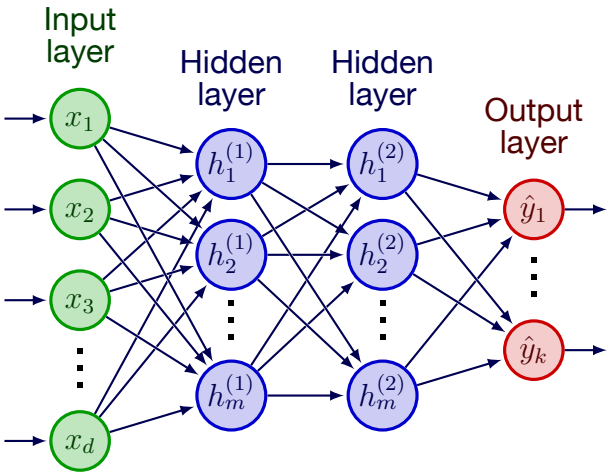


Data: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$

Parameters: $w = (w_{11}^{(1)}, \ldots, w_{kd}^{(l)})$

Prediction: $\hat{y}^{(i)} = \mathrm{NN}(x^{(i)}; w)$

Optimizing neural networks



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

Parameters: $w = (w_{11}^{(1)}, \ldots, w_{kd}^{(l)})$

Prediction: $\hat{y}^{(i)} = \mathrm{NN}(x^{(i)}; w)$

Goal: minimize the **loss function**

Find w that minimizes $\sum_{i=1}^{n} \ell(\hat{y}^{(i)}, y^{(i)})$

Examples of loss functions

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

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

Examples of loss functions

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

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

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

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

Examples of loss functions

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

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

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

$$\ell(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$ (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 $\sum_i y_i = 1$

$$\ell(\hat{y}, y) = -\sum_{i=1}^{k} y_i \log \hat{y}_i$$ (categorical cross-entropy)

Gradient descent

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

Goal: Minimize loss $L(w)$ where $w = (w_1, \ldots, w_K)$ is the parameters

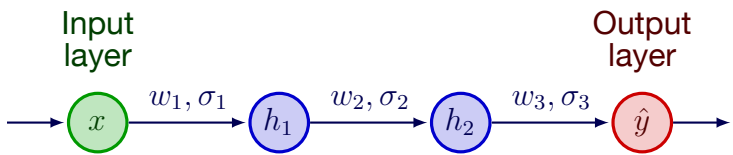
Specify the **learning rate** $\eta > 0$.

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

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

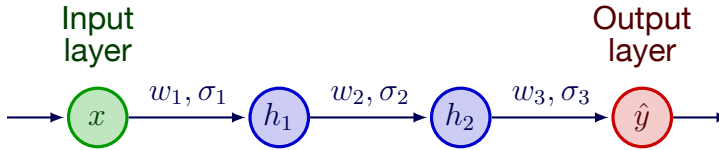
Computing gradient

Let's consider a simple network with $w_1, w_2, w_3 \in \mathbb{R}$



Step 1: With w_1, w_2, w_3 from the previous iteration, compute forward from x to $\hat{y}$, and store values of x, h_1, h_2 and $\hat{y}$ along the way

Computing gradient

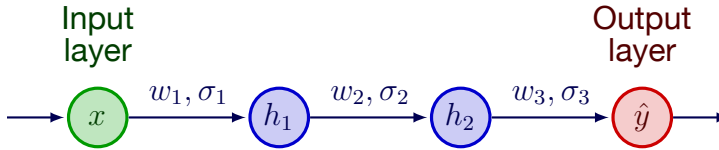


Step 2: compute the gradient **backward**

Using chain rule,

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_3(w_3 h_2)} \cdot \frac{\partial \sigma_3(w_3 h_2)}{\partial w_3 h_2} \cdot \frac{\partial w_3 h_2}{\partial w_3}$$

Computing gradient



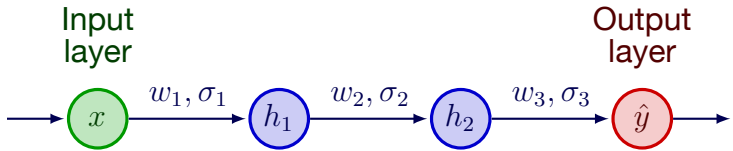
Step 2: compute the gradient **backward**

Using chain rule,

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_3(w_3 h_2)} \cdot \frac{\partial \sigma_3(w_3 h_2)}{\partial w_3 h_2} \cdot \frac{\partial w_3 h_2}{\partial w_3}$$

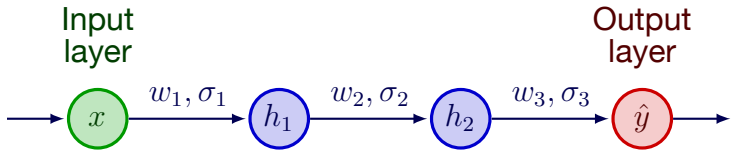
$$= \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \cdot \sigma_3'(w_3 h_2) \cdot h_2$$

Computing gradient



$$\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_3(w_3 h_2)} \cdot \frac{\partial \sigma_3(w_3 h_2)}{\partial w_3 h_2} \cdot \frac{\partial w_3 h_2}{\partial h_2}$$

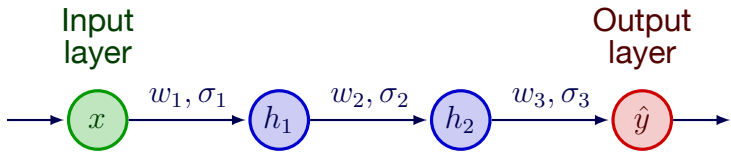
Computing gradient



$$\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_3(w_3 h_2)} \cdot \frac{\partial \sigma_3(w_3 h_2)}{\partial w_3 h_2} \cdot \frac{\partial w_3 h_2}{\partial h_2}$$

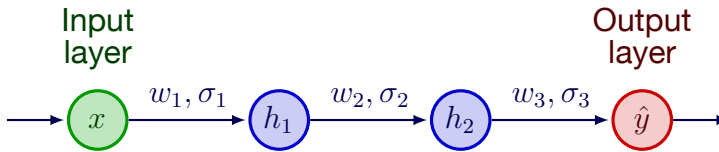
$$= \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \cdot \sigma_3'(w_3 h_2) \cdot w_3$$

Computing gradient



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial h_2} \cdot \frac{\partial h_2}{\partial \sigma_2(w_2 h_1)} \cdot \frac{\partial \sigma_2(w_2 h_1)}{\partial w_2 h_1} \cdot \frac{\partial w_2 h_1}{\partial w_2}$$

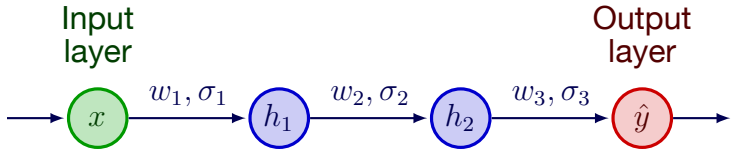
Computing gradient



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial h_2} \cdot \frac{\partial h_2}{\partial \sigma_2(w_2 h_1)} \cdot \frac{\partial \sigma_2(w_2 h_1)}{\partial w_2 h_1} \cdot \frac{\partial w_2 h_1}{\partial w_2}$$

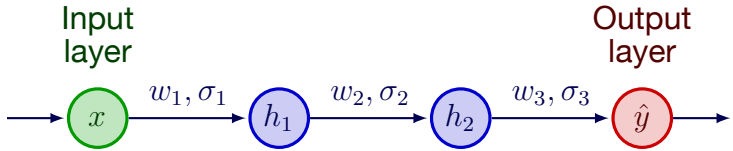
$$= \frac{\partial L}{\partial h_2} \cdot \sigma_2'(w_2 h_1) \cdot h_1$$

Computing gradient



$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2} \cdot \sigma_2'(w_2 h_1) \cdot w_2$$

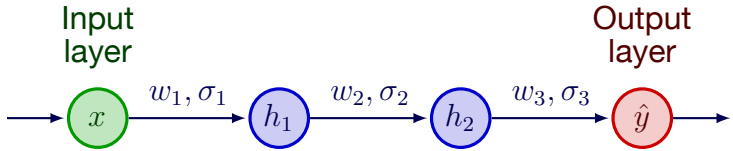
Computing gradient



$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2} \cdot \sigma_2'(w_2 h_1) \cdot w_2$$
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_1} \cdot \sigma_1'(w_1 x) \cdot h_1$$

From this, we have computed $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}$

Computing gradient



$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2} \cdot \sigma_2'(w_2 h_1) \cdot w_2$$
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_1} \cdot \sigma_1'(w_1 x) \cdot h_1$$

From this, we have computed $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}$

This way of computing the gradient backward is called **backpropagation**

Summary

- To optimize the parameters of neural networks, we use **gradient descent**

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

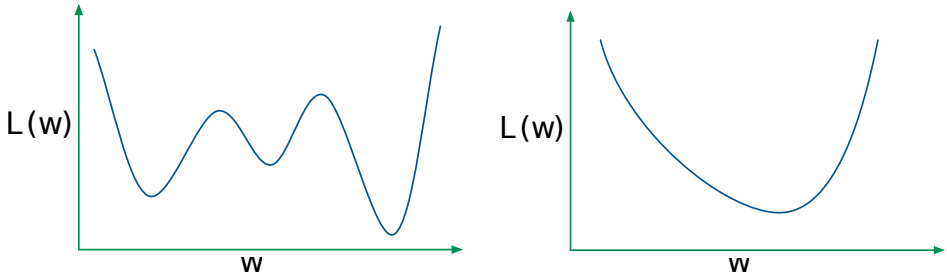
Summary

- 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

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



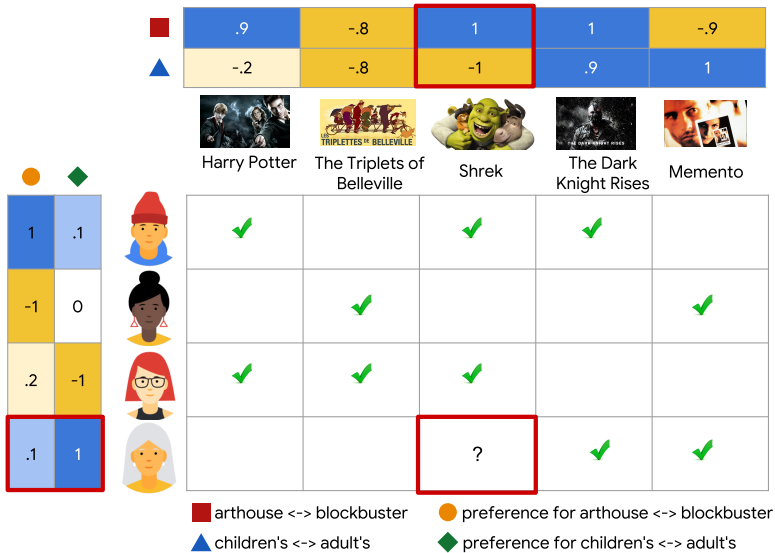
How to escape from local (but not global) minima?

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

Neural Networks

Recommender Systems

Recommender systems



Matrix Factorization

- We assume that there is some underlying low-dimensional structure among the interactions between users and items

Matrix Factorization

- We assume that there is some underlying low-dimensional structure among the interactions between users and items

- In simple terms, we assume that users' opinions, or the properties of the items they consume, can be efficiently summarized
  - Do you tend to like action movies (and is this an action movie)?
  - Do you tend to enjoy movies with a high budget, certain actors, or a long runtime?

Matrix Factorization

- We assume that there is some underlying low-dimensional structure among the interactions between users and items
- In simple terms, we assume that users' opinions, or the properties of the items they consume, can be efficiently summarized
  - Do you tend to like action movies (and is this an action movie)?
  - Do you tend to enjoy movies with a high budget, certain actors, or a long runtime?
- Purchases, clicks, or ratings can be explained by such factors, the goal of model-based recommendation is to discover them

Example

Consider the interaction between users and items below

$$R = \begin{bmatrix} 1 & \cdot & 1 & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot \end{bmatrix} \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix}$$

$$\begin{matrix} i_1 & i_2 & i_3 & i_4 & i_5 \end{matrix}$$

Example

Consider the interaction between users and items below

$$R = \begin{bmatrix} 1 & \cdot & 1 & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot \end{bmatrix} \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix}$$
$$\begin{matrix} i_1 & i_2 & i_3 & i_4 & i_5 \end{matrix}$$

We can approximate $R_{u,i}$ by $\gamma_u \cdot \gamma_i$ where

$$\begin{aligned} \gamma_{u_1} &= (1,0) & \gamma_{i_1} &= (1,0) \\ \gamma_{u_2} &= (1,0) & \gamma_{i_2} &= (1,0) \\ \gamma_{u_3} &= (1,0) & \gamma_{i_3} &= (1,0) \\ \gamma_{u_4} &= (0,1) & \gamma_{i_4} &= (0,1) \\ \gamma_{u_5} &= (0,1) & \gamma_{i_5} &= (0,1) \end{aligned}$$

Example

We have factorized R into matrices of **users** and **items**

$$R = \begin{matrix} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix} \begin{matrix} \begin{bmatrix} 1 & \cdot & 1 & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot \end{bmatrix} \\ i_1 \quad i_2 \quad i_3 \quad i_4 \quad i_5 \end{matrix} \approx \underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}_{\gamma_U} \cdot \underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}}_{\gamma_I^T}$$

For example,

- Each column in γ_U might correspond to male and female users
- Each column in γ_U might correspond to men's and women's clothing

Example

We have factorized R into matrices of **users** and **items**

$$R = \begin{bmatrix} 1 & \cdot & 1 & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot \end{bmatrix} \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix} \approx \underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}_{\gamma_U} \cdot \underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}}_{\gamma_I^T}$$

$$\begin{matrix} i_1 & i_2 & i_3 & i_4 & i_5 \end{matrix}$$

For example,

- Each column in γ_U might correspond to male and female users
- Each column in γ_U might correspond to men's and women's clothing

This model predicts that user u_2 will give item i_4 a rating of $\gamma_{u_2} \cdot \gamma_{i_4} = 0$

Latent Factor Models

The goal of Matrix Factorization is to describe a (partially observed) matrix in terms of lower-dimensional factors

$$\underbrace{\begin{bmatrix} & R & \end{bmatrix}}_{|U|\times|I|} = \underbrace{\begin{bmatrix} & \gamma_U & \end{bmatrix}}_{|U|\times K} \times \underbrace{\begin{bmatrix} & \gamma_I^T & \end{bmatrix}}_{K\times|I|}$$

- $|U|$ is the number of users
- $|I|$ is the number of items
- K is the number of hidden factors

Latent Factor Models

The goal of Matrix Factorization is to describe a (partially observed) matrix in terms of lower-dimensional factors

$$\underbrace{\begin{bmatrix} & R & \end{bmatrix}}_{|U|\times|I|} = \underbrace{\begin{bmatrix} & \gamma_U & \end{bmatrix}}_{|U|\times K} \times \underbrace{\begin{bmatrix} & \gamma_I^T & \end{bmatrix}}_{K\times|I|}$$

- $|U|$ is the number of users
- $|I|$ is the number of items
- K is the number of hidden factors

The model of hidden factors based on matrix factorization is called **latent factor models (LFM)**

LFM objective

Let $T = \{(u_1, i_1), \ldots, (u_n, i_n)\}$ be **observed** user-item interactions
Let $R_{u,i}$ be the rating that user u gave to item i

LFM objective

Let $T = \{(u_1, i_1), \ldots, (u_n, i_n)\}$ be **observed** user-item interactions
Let $R_{u,i}$ be the rating that user u gave to item i

The variables are matrices $\gamma_U \in \mathbb{R}^{|U| \times K}$ and $\gamma_I \in \mathbb{R}^{|I| \times K}$

LFM objective

Let $T = \{(u_1, i_1), \ldots, (u_n, i_n)\}$ be **observed** user-item interactions
Let $R_{u,i}$ be the rating that user u gave to item i

The variables are matrices $\gamma_U \in \mathbb{R}^{|U| \times K}$ and $\gamma_I \in \mathbb{R}^{|I| \times K}$

The usual choice of objective is the mean-squared error

$$\min_{\gamma_U, \gamma_I} \frac{1}{n} \sum_{(u,i) \in T} (\gamma_u \cdot \gamma_i - R_{u,i})^2$$

LFM objective

Let $T = \{(u_1, i_1), \ldots, (u_n, i_n)\}$ be **observed** user-item interactions
Let $R_{u,i}$ be the rating that user u gave to item i

The variables are matrices $\gamma_U \in \mathbb{R}^{|U| \times K}$ and $\gamma_I \in \mathbb{R}^{|I| \times K}$

The usual choice of objective is the mean-squared error

$$\min_{\gamma_U, \gamma_I} \frac{1}{n} \sum_{(u,i) \in T} (\gamma_u \cdot \gamma_i - R_{u,i})^2$$

We want our model to make accurate predictions on unseen data, so we will **regularize** the variables

Regularized LFM objective

We regularize γ_u and γ_i for all users u and items i

$$\min_{\gamma_U, \gamma_I} \frac{1}{n} \sum_{(u,i) \in T} (\gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda_1 \sum_{u \in U} \|\gamma_u\|^2 + \lambda_2 \sum_{i \in I} \|\gamma_i\|^2$$

Regularized LFM objective

We regularize γ_u and γ_i for all users u and items i

$$\min_{\gamma_U, \gamma_I} \frac{1}{n} \sum_{(u,i) \in T} (\gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda_1 \sum_{u \in U} \|\gamma_u\|^2 + \lambda_2 \sum_{i \in I} \|\gamma_i\|^2$$

- However, this regularizer will encourage γ_u's and γ_i's to be close to zero

Regularized LFM objective

We regularize γ_u and γ_i for all users u and items i

$$\min_{\gamma_U, \gamma_I} \frac{1}{n} \sum_{(u,i) \in T} (\gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda_1 \sum_{u \in U} \|\gamma_u\|^2 + \lambda_2 \sum_{i \in I} \|\gamma_i\|^2$$

- However, this regularizer will encourage γ_u's and γ_i's to be close to zero
- As such, the predictions $\gamma_u \cdot \gamma_i$ will also be pushed toward zero

Regularized LFM objective

We regularize γ_u and γ_i for all users u and items i

$$\min_{\gamma_U, \gamma_I} \frac{1}{n} \sum_{(u,i) \in T} (\gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda_1 \sum_{u \in U} \|\gamma_u\|^2 + \lambda_2 \sum_{i \in I} \|\gamma_i\|^2$$

- However, this regularizer will encourage γ_u's and γ_i's to be close to zero
- As such, the predictions $\gamma_u \cdot \gamma_i$ will also be pushed toward zero
- Consequently, the system will systematically underpredict ratings

Modifying LFM objective

A natural way to increase the prediction is by **adding an intercept term** α

$$\hat{R}_{u,i} = \alpha + \gamma_u \cdot \gamma_i$$

Modifying LFM objective

A natural way to increase the prediction is by **adding an intercept term** α

$$\hat{R}_{u,i} = \alpha + \gamma_u \cdot \gamma_i$$

- However, we still have the same problem
- The regularizer will push γ_u's and γ_i's to be close to zero

Modifying LFM objective

A natural way to increase the prediction is by **adding an intercept term** α

$$\hat{R}_{u,i} = \alpha + \gamma_u \cdot \gamma_i$$

- However, we still have the same problem
- The regularizer will push γ_u's and γ_i's to be close to zero
- The predictions $\alpha + \gamma_u \cdot \gamma_i$ will be pushed toward α
- Not everyone is going to have the same rating level of α!

Modifying LFM objective

We add intercept terms for each user and item

$$\hat{R}_{u,i} : \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

- β_u encodes how much user u's ratings are higher or lower than α
- β_i encodes how much item i tends to receive higher or lower ratings than α

Modifying LFM objective

We add intercept terms for each user and item

$$\hat{R}_{u,i} : \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

- β_u encodes how much user u's ratings are higher or lower than α
- β_i encodes how much item i tends to receive higher or lower ratings than α

In practice, β_u and β_i are also regularized, but not α

Regularized LFM objective

Our goal now is to minimize the following nonconvex objective function

$$f(\alpha,\beta,\gamma)=\frac{1}{n}\sum_{(u,i)\in T}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2+\lambda_1\sum_{u\in U}(\beta_u^2+\|\gamma_u\|^2)+\lambda_2\sum_{i\in I}(\beta_i^2+\|\gamma_i\|^2)$$

We will try to minimize this function with gradient descent

Partial derivatives of $f(\alpha, \beta, \gamma)$

$$f(\alpha,\beta,\gamma) = \frac{1}{n}\sum_{(u,i)\in T}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2+\lambda_1\sum_{u\in U}(\beta_u^2+\|\gamma_u\|^2)+\lambda_2\sum_{i\in I}(\beta_i^2+\|\gamma_i\|^2)$$

The partial derivatives of f are ($I_u = \{i : (u,i) \in T\}$)

$$\frac{\partial f}{\partial \alpha} = \frac{1}{n}\sum_{(u,i)\in T} 2(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})$$

$$\frac{\partial f}{\partial \beta_u} = \frac{1}{n}\sum_{i\in I_u} 2(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i}) + 2\lambda_1\beta_u$$

$$\frac{\partial f}{\partial \gamma_{uk}} = \frac{1}{n}\sum_{i\in I_u} 2\gamma_{ik}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i}) + 2\lambda_1\gamma_{uk}$$

The partial derivatives of β_i and γ_{ik} can be computed similarly

Gradient descent for LFM

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

1. Start with a random $\alpha^0, \beta_u^0, \gamma_{uk}^0, \beta_i^0, \gamma_{ik}^0$.
2. At $t > 0$, for all u, i and k with $\theta^t = (\alpha^t, \beta_u^t, \beta_i^t, \gamma_{uk}^t, \gamma_{ik}^t)$,

$$\alpha^{t+1} \leftarrow \alpha^t - \frac{\partial f}{\partial \alpha}(\theta^t)$$

$$\beta_u^{t+1} \leftarrow \beta_u^t - \frac{\partial f}{\partial \beta_u}(\theta^t)$$

$$\gamma_{uk}^{t+1} \leftarrow \gamma_{uk}^t - \frac{\partial f}{\partial \gamma_{uk}}(\theta^t)$$

Similar update formula for β_i^{t+1} and γ_{ik}^{t+1}

Alternating Least Squares

Alternatively, we can minimize at step t by solving the first-order conditions

$$\frac{\partial f}{\partial \alpha} = \frac{1}{n} \sum_{(u,i) \in T} 2(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) = 0$$

$$\frac{\partial f}{\partial \beta_u} = \frac{1}{n} \sum_{i \in I_u} 2(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) + 2\lambda_1 \beta_u = 0$$

$$\frac{\partial f}{\partial \gamma_{uk}} = \frac{1}{n} \sum_{i \in I_u} 2\gamma_{ik}(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) + 2\lambda_1 \gamma_{uk} = 0$$

Alternating Least Squares

Solving the following equation for α

$$\frac{\partial f}{\partial \alpha} = \frac{1}{n} \sum_{(u,i) \in T} 2(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) = 0$$

we obtain

$$\alpha = \frac{1}{n} \sum_{(u,i) \in T} (R_{u,i} - \beta_u - \beta_i - \gamma_u \cdot \gamma_i)$$

Alternating Least Squares

Solving the following equation for β_u

$$\frac{\partial f}{\partial \beta_u} = \frac{1}{n} \sum_{i \in I_u} 2(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) + 2\lambda_1 \beta_u = 0$$

we obtain

$$\beta_u = \frac{\sum_{i \in I_u} (R_{u,i} - \alpha - \beta_i - \gamma_u \cdot \gamma_i)}{|I_u|/n + \lambda_1}$$

Alternating Least Squares

Solving the following equation for β_u

$$\frac{\partial f}{\partial \gamma_{uk}} = \frac{1}{n} \sum_{i \in I_u} 2\gamma_{ik}(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i}) + 2\lambda_1 \gamma_{uk} = 0$$

we obtain

$$\gamma_{uk} = \frac{\sum_{i \in I_u}(R_{u,i} - \alpha - \beta_i - \gamma_u \cdot \gamma_i + \gamma_{uk}\gamma_{ik})}{\sum_{i \in I_u} \gamma_{ik}^2 / n + \lambda_1}$$

The solution for β_i and γ_{ik} can be computed similarly

Alternating Least Squares for LFM

1. Start with a random $\alpha^0, \beta_u^0, \gamma_{uk}^0, \beta_i^0, \gamma_{ik}^0$.
2. At $t > 0$, for all u, i and k,

$$\alpha^{t+1} \leftarrow \frac{1}{n} \sum_{(u,i) \in T} (R_{u,i} - \beta_u^t - \beta_i^t - \gamma_u^t \cdot \gamma_i^t)$$

$$\beta_u^{t+1} \leftarrow \frac{\sum_{i \in I_u} (R_{u,i} - \alpha^t - \beta_i^t - \gamma_u^t \cdot \gamma_i^t)}{|I_u|/n + \lambda_1}$$

$$\gamma_{uk}^{t+1} \leftarrow \frac{\sum_{i \in I_u} (R_{u,i} - \alpha^t - \beta_i^t - \gamma_u^t \cdot \gamma_i^t + \gamma_{uk}^t \gamma_{ik}^t)}{\sum_{i \in I_u} (\gamma_{ik}^t)^2/n + \lambda_1}$$

Similar update formula for β_i^{t+1} and γ_{ik}^{t+1}

Summary

- We can extract features of users and items by factorize the rating matrix

Summary

- We can extract features of users and items by factorize the rating matrix
- However, the objective function of matrix factorization is non-convex

Summary

- We can extract features of users and items by factorize the rating matrix
- However, the objective function of matrix factorization is non-convex
- There are several methods that can be used to minimize the objective, such as gradient descent and alternating least squares
- These methods are sensitive to initialization, so we might have to optimize several times with random initializations