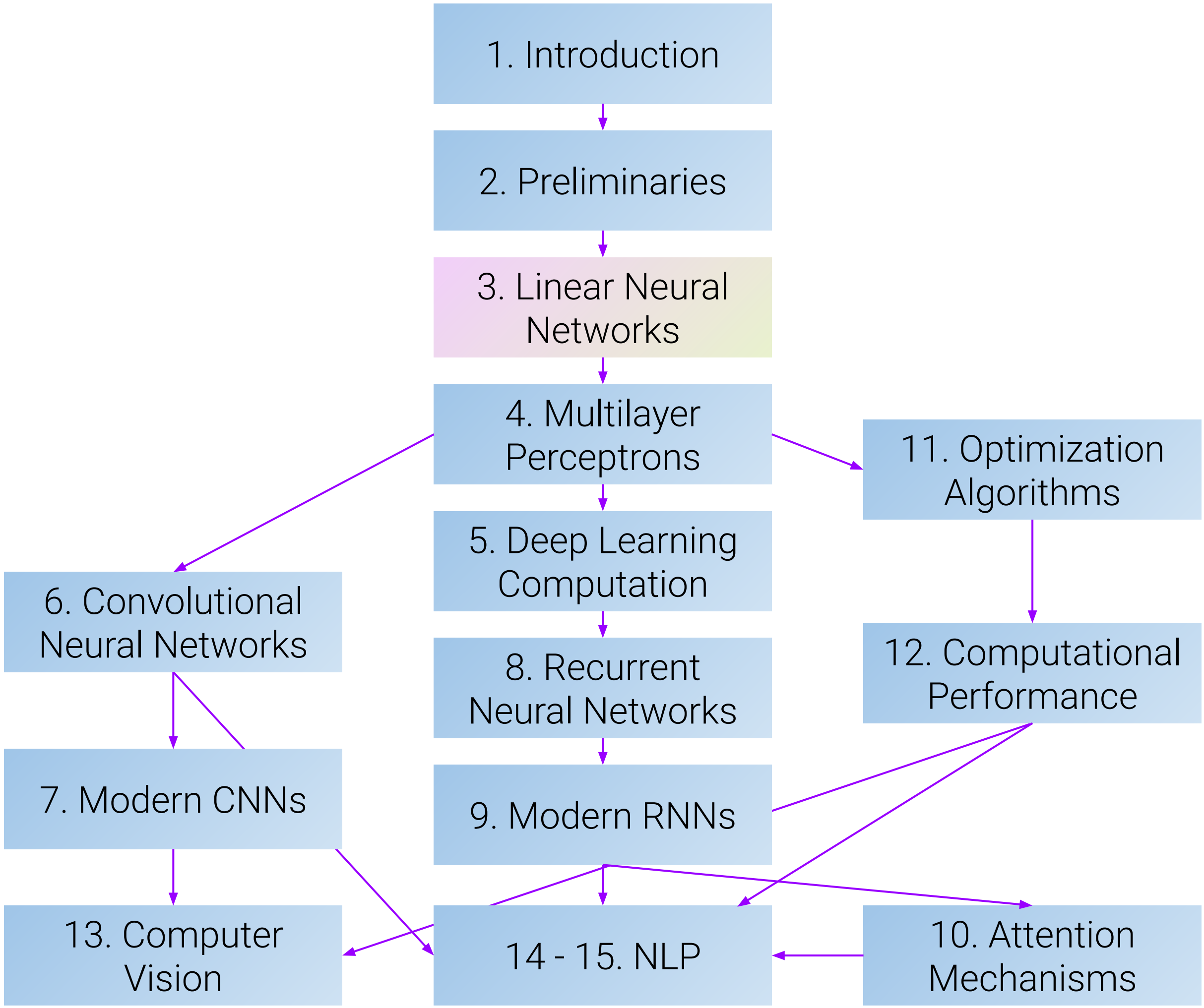
Linear Neural Networks

Session #3

A study group by dair.ai

Chapter 3



1. Introduction
2. Preliminaries
3. Linear Neural Networks
4. Multilayer Perceptrons
5. Deep Learning Computation
6. Convolutional Neural Networks
7. Modern CNNs
8. Recurrent Neural Networks
9. Modern RNNs
10. Attention Mechanisms
11. Optimization Algorithms
12. Computational Performance
13. Computer Vision
14 - 15. NLP

Agenda

- Linear Regression
  - Implementation from scratch
  - Concise implementation
- Softmax regression
  - The image classification dataset
  - Implementation from scratch
  - Concise implementation

Linear Regression

- Regression is used for modeling relationship between one or more ***independent*** variables and a ***dependent*** variable
- The aim is to train a model that is able to predict a numerical value
  - Predicting prices (homes, stocks, etc.)
  - Predicting length of stay (patients in hospital)
  - Demand forecasting (sales)
- Linear regression differs from ***classification***
  - Classification aims to predict from set of categories (cat vs. dogs, positive vs. negative)

Basic Elements of Linear Regression

- Assumption:
  - *Linear relationship* between independent variables **x** and the dependent variable ***y***
  - ***y*** can be expressed as a *weighted sum* of the elements in **x,** given noise on the observations
  - Noise well behaved (follow Gaussian distribution)
- Example:
  - We would like to estimate price of houses based on ***area*** and ***age***
  - We want to develop a predictive model for predicting house prices
- We need:
  - A ***training dataset*** or ***training set***
  - Rows referred to as *examples*, *data points*, *data instances*, *sample*
  - Dependent variable is called the *label* or *target*
  - Independent variables are called the *features* or *covariates*

area	**age**	**price**
15	10	15000
25	15	25000

Notations

superscript

- **n** denotes the # of examples
- Index the data examples $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}]^\top$
- Corresponding labels $y^{(i)}$

Linear Model

Based on the ***linearity assumption*** we say that the target is a weighted sum of the features and translation (bias)

$$\mathrm{price} = w_{\mathrm{area}} \cdot \mathrm{area} + w_{\mathrm{age}} \cdot \mathrm{age} + b$$

Weights - Influence of features on the predictions

Bias - Take on this value when features take value of 0

Affine transformation of input features

Linear Model

- **Goal:** choose weights and bias such that on average, predictions of the model fit the true prices observed in the data
- Linear models rely on the ***affine transformation*** specified by the chosen ***weights*** and ***bias***
- So, generally speaking we have $\hat{y} = w_1 x_1 + \ldots + w_d x_d + b$
- Compact form using vectors: $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$

Linear Model

We refer to features of the entire dataset as:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

Features of dataset

Goal is to find **w** and **b** such that for a new example (from same distribution) and it's label (in expectation) the model makes prediction that produces the lowest error

Linear Models

- We have a formulation based on the linearity assumption
- However,
    - we need a ***quality measure*** (measure goodness/badness of the model)
    - and a ***procedure*** to update and improve the model quality

We need a loss function and an optimizer

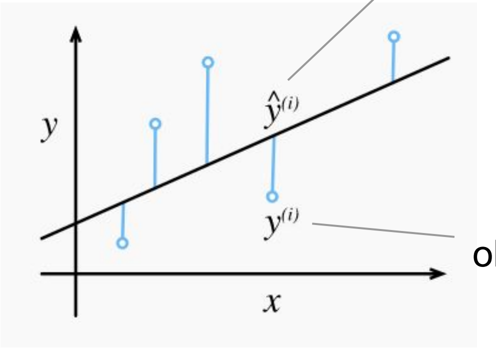
Loss function

- A function that quantifies the difference between real and predicted value of the target
- The smaller the value of loss the better
- A popular loss function: ***squared error***
- Empirical error is a function of the parameters

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

Term cancels when we take the derivative of the loss

estimation



observation

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\overbrace{\mathbf{w}^\top \mathbf{x}^{(i)} + b}^{\hat{y}^{(i)}} - y^{(i)}\right)^2$$

Number of examples

When training the model, we seek parameters that minimize the total loss across all training examples

Note the superscript suggests an operation applied to a single example

Average loss on the entire dataset

Loss function

When training the model we are looking for weights and bias (parameters) that ***minimize the total loss on all training examples***

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \; L(\mathbf{w}, b)$$

Optimization

- We seek to iteratively reduce the error of the model and improve its quality:
  - Updating the parameters in the direction that incrementally lowers the loss function
- We use the ***gradient descent algorithm*** to achieve it
- We can directly take the derivative of the average loss on the entire dataset
  - This requires pass over the entire dataset before making a single update
- A better solution is called ***minibatch stochastic gradient descent***
  - Sampling random minibatch of examples and
  - Take the derivative of the average loss on the minibatch with regard to the model parameters and then compute the update

Minibatch Stochastic Gradient Descent

Term multiplied to the gradient (learning rate)

Partial derivative of the average loss of the minibatch

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w}, b)} l^{(i)}(\mathbf{w}, b)$$

Subtract the result from the current parameters

Minibatch size

Optimization Algorithm

- Initialize model parameters (typically random)
- Sample random minibatches
- Update parameters in the direction of the negative gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right),$$

$$b \leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right).$$

Prediction using Linear Regression Model

- We adjust hyperparameters (e.g., learning rate) assessed on validation set
- We aim to find parameters that achieve low loss on ***unseen data***
    - Also referred to ***generalization*** (discussed later on)
- Given those learned parameters it is now possible to estimate targets given features of a new instance
- We use the squares loss below to quantify goodness/badness of the model
    - Using maximum likelihood estimate principles we get ***negative log likelihood***

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2$$

Normal Distribution and Squared loss

Linear regression with the square loss can be motivated by assuming that the observations arise from noisy distributions. Hence,

$$y = \mathbf{w}^\top \mathbf{x} + b + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right)$$

$$P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p(y^{(i)} | \mathbf{x}^{(i)})$$

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2$$

Linear regression as single-layer neural network

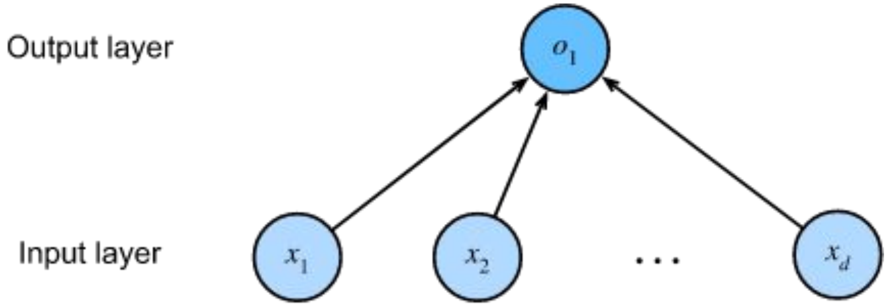
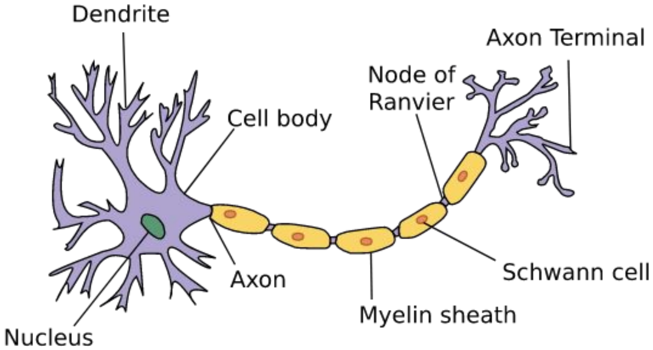


Figure source: d2l.ai

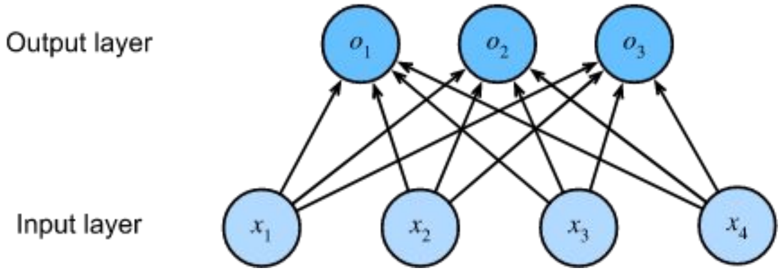


Classification

- Classification aims to predict from set of categories (e.g., cat vs. dogs, positive vs. negative)
- Hard assignment of examples to categories (classes)
- Soft assignments; assess probabilities (discussed later on)
- We want a model that estimates the ***conditional probabilities*** with all possible classes
    - Model with multiple outputs (one per class)
- Goal: optimize our parameters to produce probabilities that maximize the likelihood of the observed data
    - One main approach is softmax regression

Softmax Regression

- Supports classification (binary and multiclass)
- Applies softmax function produces outputs as probabilities
- Take logits (output of the linear model) and transform to output probabilities that meet a desired criteria
    - Nonnegativity
    - Differentiable
    - Probabilities sum up to 1



Softmax regression is a single-layer neural network.
Figure source: d2l.ai

Cross-entropy loss

- Used to measure quality of predicted probabilities
- Common loss function used in classification problems
- Computes the expected value of the loss for a distribution over labels
- Concretely, cross-entropy objective
  - Maximizes the likelihood of the observed data
  - Measures difference between two probability distributions
  - Minimize the surprisal require to communicate the labels (refer to information theory)

$$l(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{j=1}^{q} y_j \log \hat{y}_j$$

Assignment

- Implement backward function
- Calculate the derivatives with respect to the parameters to obtain gradients from scratch and do the update them manually
- Individual assignment
- Solution provided next week

Demo