MSDS 458

Artificial Intelligence & Deep Learning

Module 1 - AI: Past, Present, and Future

Deep Learning with Python

Chapter 1 - What is Deep Learning

1. Artificial Intelligence, Machine Learning, and Deep Learning

1.1 Artificial Intelligence

1.1.1 Artificial Intelligence

- John McCarthy, Dartmouth Mathematics Professor organized a summer workshop in 1956 that marked start of AI
- Symbolic AI: explicit, symbolic representation of knowledge written by humans (expert systems)

1.1.2 Machine Learning

- Machine Learning: an automatic search process for transformations that produce useful representations of data, guided by a feedback signal.

1.1.3 Learning Rules and Representations of Data

- Hypothesis Space: the set of possible transformations that a machine learning model iterates through while learning

1.1.4 The 'Deep' in Deep Learning

- Deep Learning: subset of machine learning that employs layered representations of data in increasing meaningfulness. Deep learning uses neural networks, the depth of which is equal to the number of layers.

1.1.5 How Deep Learning Works

- Weights: scalar parameters assigned to each node in each layer.
- Loss Function: describes the accuracy of the output
- Learning: process through which the weights are optimized to reduce the loss function, uses backpropogation
- Training Loop: the individual iteration of learning

1.2 Before Deep Learning: A Brief History of Machine Learning

1.2.1 Probabilistic Modeling

- Naive Bayes: classification algorithm based on Bayes' Rule. Assumes input data are all independent, a naive assumption
- Logistic Regression: classification algorithm, the 'hello, world' of machine learning

1.2.2 Early Neural Networks

- First successful, practical application of NNs was made by Yann LeCun of Bell Labs, who combined early ideas of CNNs with the backpropogation algorithm (gradient descent optimization method to train chains of parameteric operations) to classify digits, called LeNet

1.2.3 Kernel Methods (set of classification methodologies)

- Support Vector Machine: Maps data to new high-dimensional representation, then a classifying boundary is learned in a process called maximizing the margin. Explicitly calculating the coordinates of points in the new space is unnecessary. Only the distance between points matters, and this can be calculated using kernel functions, which calculate distances between points under transformation. Requires feature engineering.

1.2.5 CNNs

- Convolutional Neural Networks are used for computer vision, image classification. Replaced SVMs and random forests for most of their applications.

1.2.6 What Makes Deep Learning Different

- Deep learning automates feature engineering
- Layers increase in the complexity of their representations of data
- Learning process optimizes layers jointly

Chapter 2 - The Mathematical Foundations of Neural Networks

2.2 - Data Representations for Neural Networks

- Tensor: containers for numerical data, allow for an arbitrary number of dimensions called ranks, and axes
- Tensor Ranks:
  - Scalars: Rank 0 tensors, a single number (ndim = 0)
  - Vectors: Rank 1 tensors
  - Matrices: Rank 2 tensors, two axes
  - Rank 3 and Higher Tensors
- Tensor Attributes:
  - Rank: number of axes
  - Shape: how many dimensions are on each axis
  - Data type
- Axis 0 is refered to as the samples axis
- In the context of batches, the Axis 0 is called the batch axis
- In the context of vector data, data is stored in a matrix (rank 2 vector), the first axis is the samples axis and the second is called the features axis.

2.3 - Tensor Operations

- Element-wise Operations: apply operations to pairs of corresponding elements within two tensors
- Broadcasting: allows for element-wise operations on tensors with different dimensions. To do so, the values of the smaller tensor are duplicated such that the shapes of the two tensors match.
- Tensor Product: can be applied to tensors with arbitrarily large dimensions.
  - Vector product: $[1,2,3] \cdot [1,2,3]$

    $(1 \cdot 1) + (2 \cdot 2) + (3 \cdot 3)$

    $1 + 4 + 9$

    14
  - Vector Matrix product: $\begin{bmatrix} 1,2,3 \\ 4,5,6 \end{bmatrix} \cdot [1,2,3]$

    $(1 \cdot 1) + (2 \cdot 2) + (3 \cdot 3),\ (4 \cdot 1) + (5 \cdot 2) + (6 \cdot 3)$

    $1 + 4 + 9,\ 4 + 10 + 18$

    $14, 32$

    The second dimension of the matrix must be the same as the first of the vector.

    Result has same shape as vector.
  - Matrix product: Given matrices x and y of respective shapes (a,b) and (b,c), creates a new matrix of shape (a,c) where each coefficient is the vector product of matrix x rows and matrix y columns.

    $\begin{bmatrix} 1,2,3 \\ 4,5,6 \end{bmatrix} \cdot \begin{bmatrix} 1,6 \\ 2,4 \\ 5,2 \end{bmatrix}$

    $\begin{bmatrix} (1 \cdot 1) + (2 \cdot 3) + (3 \cdot 5),\ (1 \cdot 2) + (2 \cdot 4) + (3 \cdot 6) \\ (4 \cdot 1) + (5 \cdot 3) + (5 \cdot 5),\ (4 \cdot 2) + (5 \cdot 4) + (6 \cdot 6) \end{bmatrix}$

    $\begin{bmatrix} 1 + 6 + 15,\ 2 + 8 + 18 \\ 4 + 15 + 25,\ 8 + 20 + 36 \end{bmatrix}$

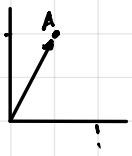
    $\begin{bmatrix} 22, 28 \\ 44, 64 \end{bmatrix}$

  - Once a tensor of ndim > 1 is involved, tensor dot product is no longer symmetric, meaning $dot(x,y) \neq dot(y,x)$
- Tensor Reshaping: special case - transposition

2.3 - Tensor Operations

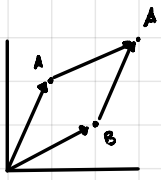
- **Geometric Interpretation of Tensor Operations**

Vectors can be represented by points in multidimensional space.
The vector [0.5, 1] can be represented as:



Adding the tensor [1, 0.5]:

Vector Addition:



Tensor addition is thus equivalent to a translation (moving an object without distorting it.)

Rotation: clockwise rotation of a 2-D object by angle theta can be accomplished via a dot product of a 2x2 matrix and a vector

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling: vertical and horizontal scaling of a 2-D object can be accomplished via a dot product of a 2x2 diagonal matrix and a vector

$$\begin{bmatrix} 1 & 0 \\ 0 & -0.5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Linear Transformation: a dot product of an arbitrary matrix is a linear transformation (rotation and scaling are, therefore, linear transformations)

Affine Transformation: combination of a linear transformation and a translation. Thus, matrix · matrix + vector

2.4 - Gradient-Based Optimization

output = f(w · x + b)

where: f = activation function
w = weight vector
x = input vector
b = bias

- The weights and biases of nodes in a neural network are set randomly at first, a step called random initialization
- The weights and biases are optimized to minimize the loss function (feedback signal) through each training loop via gradient-based optimization
- The mathematical operations applied to input data within neural networks are linear and continuous, thus differentiable
- The function that describes the relationship between the models' weights and biases and the model's loss is differentiable, thus the weights and biases can be optimized along their gradients in synchrony
- Gradient: the derivative of a tensor operation / function, representing the curvature of the multi-dimensional surface described by the function.

Given an input vector x, matrix of weights w, and y_true,
y_pred = dot(x, w) and loss = diff(y_pred, y_true).
f(w) is the function that maps w to loss given fixed x and y_true.
f'(w) is a matrix of shape w that describes the direction and magnitude of change in loss associated with incrementally adjusting w_ij values. This tensor is said to be the gradient of the loss value with respect to w around a value of w. Each value within f'(w) represents the partial derivative of loss with respect to w_ij.

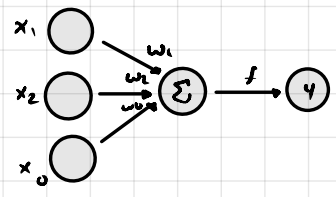
- Learning Rate: the magnitude with which weights are adjusted down the gradient at each training loop. If too small, the learning can get stuck at a local minimum. If too big, the learning can jump around too much and not reach a minimum.
- Stochastic Gradient Descent: performing gradient descent on sets of randomly selected batches of data at each training loop.
  True SGD uses a single data point for each loop.
  Mini-batch SGD uses batches of data points for each loop.
- Batch Gradient Descent uses all datapoints for all loops (most expensive)
- SGD variants perform optimizations according to different methods
- Momentum is applied in some SGD optimizers to address local minima by adjusting weights according to both gradient at the current training loop and previous weight adjustments
- Backpropagation Algorithm: calculates the contribution of each parameter to loss by connecting their tensor functions via the chain rule.

Module 2 - Neural Networks : The Simplest Possible Networks

Solving the XOR Problem With Neural Networks

Perceptron - Binary linear classification algorithm that separates positive from negative numbers

Structure:



Computation:
- Calculate the dot product of the input x and weight w vectors
- Add the bias
- Apply the activation function to the calculated sum

$$y = f(w \cdot x + b)$$

Activation Functions: sigmoid activation functions output a value between 0 and 1, indicating the classification probabilities
θ = threshold for classification

$$\text{Class}_1: w \cdot x + b \geq 0$$
$$\text{Class}_2: w \cdot x + b < 0$$

Gradient: Δw = node input value · (true - predicted) (correct classification results in $\Delta w = 0$)

Learning Algorithm: $w_{new} = w_{old} + \text{learning rate} \cdot \Delta w$

Example:

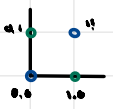
	w	input	y pred	y true	y true - y pred	Δw
w_0	0.5	0.5	0.25	1	0.75	0.375
w_1	0.875	0.5	0.4375	1	0.5625	0.28125
w_2	1.09375	0.5	0.546	1		

The 2D XOR Problem

An XOR function takes two binary integers as inputs and returns 1 if they are different and 0 if they are the same

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Geometrically:

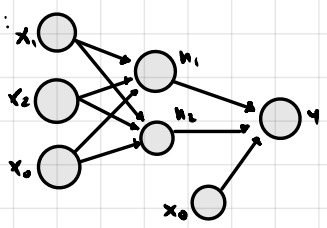


The perceptron described in the structure diagram above would not be able to solve this problem because the classes are not linearly separable, meaning it is not possible for a linear plane to discriminate the classes correctly. Thus, non-linearity is required.

Multi-Layer Perceptron

The hidden layer in an MLP allows for the network to incorporate non-linearity.

Structure:



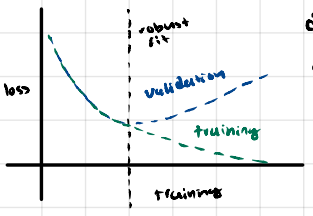
Module 4 - Deep Neural Networks, Architecture and Design

Deep Learning with Python

Chapter 5 - Fundamentals of Machine Learning

5.1 - Generalization and Overfitting

In all machine learning tasks, a balance must be achieved between generalization and optimization. Generalization refers to a model's predictive power on new data, while optimization refers to a model's accuracy on training data. Overfitting (also over-optimization) reduces generalizability. The optimal balance is when validation and training accuracy diverge. Prior to this point, training and validation loss are positively correlated and the data are underfit. After this point, the training and validation loss are negatively correlated, the data are overfit. Patterns in the data learned beyond this point are specific to the training data and not generalizable.



- Noise in data causes spurious correlations that can be picked up and learned, this is a common cause of overfitting.
- The Manifold Hypothesis states that all natural data lies on low-dimensional manifold within high-dimensional space in which it is encoded. This implies that machine learning algorithms fit simple, low-dimensional, highly structured subspaces within their input space. Additionally, it is possible to interpolate between samples in this space.
- 'Machine learning models are high dimensional, smooth, continuous, differentiable curves, that are then fit to training data to identify the low-dimensional, generalized representations of the population.
- Training data is of paramount importance! Ideally, training data is dense within the input space. (especially true around decision boundaries)
- A model's ability to generalize is more a function of the natural structure of the input data than properties of the model
- Regularization: modulates the amount of information the model can store, combatting overfitting. This smoothes the curve, allowing for regular patterns to be modeled

5.2 - Evaluating Machine Learning Models

- Data should always be split into three subsets: train, validation, test
The necessity for validation and test sets to be separate arises from:
- Information Leakage: information about the validation set leaks into a machine learning algorithm after successive training loops.
- To achieve valid testing metrics, algorithms can't be learned on test data, test data must be entirely new. Thus, both validation and test sets are necessary.
- Simple Holdout Validation: split data into training and test set, no validation set.
- K-Fold Validation: partition training data by K folds, then for every partition K_i, train on $K - K_i$ and validate with K_i. Final score is $\bar{K}$. Useful when model shows sensitivity to variations in train:test split
- Iterated K-Fold Validation with Shuffling: apply K-fold validation P time, randomizing data each time. Final score is $\sum \bar{K}_i$ for $K \in P$. Expensive process due to training P models.

5.3 - Improving Model Fit

- To achieve optimal fitting, a model must first be able to overfit the data. Overfitting implies that the model has achieved some level of generality, after which it starts to overfit. If this point is not achieved, i.e. training metrics don't decrease and then eventually increase, generality has not been achieved. This indicates a structural problem in the approach, in the data, model architecture, or both. Increasing a model's capacity for information by adding nodes, layers, or changing node types, layer types, etc... can achieve generality assuming data and tasks.

5.4 - Improving Generalization

- Feature Engineering: applying transformations to the training data that simplify the machine learning task. 'Make the manifold smoother, simpler, better organized.' Allows problems to be solved more simply and with less data.
- Early Stopping: method to stop training after model has generalized and not overfit.
- Regularization: training techniques to combat overfitting by constraining complexity of curvature, making manifold smooth, more regular, generic.
  - Simple method is to reduce model complexity, either in layers or nodes
- Weight Regularization: applies a factor to model coefficients that reduces overfitting, applies 'cost' to weights
  - L1 - cost is proportional to the absolute value of the weight
  - L2 - cost is proportional to the square of the the weight

* Regularization makes models more resistant to overfitting, this will be exhibited by a validation curve with lower slope in the section after the inflection point, less assuming model generalizes and regularization works.

- Dropout: regularization technique in which nodes in the input, hidden, and output layers are dropped at random according to a probability coefficient.
  - In dense (fully connected) networks, multiple nodes can extract the same features, called co-adaptation. This results in amplification of the feature. To avoid this, dropout can be used to decrease level of connectedness.

5. Summary

- Machine learning algorithms must successfully fit the 'latent manifold' in the data (which is the low-dimensional representation of the population within the input space) in order to achieve generality (i.e. be able to interpolate new data between sample points correctly)

Chapter 6 - The Universal Machine Learning Workflow

6.1 - Define The Task

Concept Drift - Decline in the accuracy of production models over time due to changes in the properties of input data

Target Leaking - the presence of features in training data that contain information about the targets

- For balanced classification problems, use accuracy and area under ROC (receiver operating characteristic) curve
- For imbalanced classification problems, use precision, recall, and weighted formulations of accuracy and area under ROC curve.
- Loss functions must be both computable and differentiable to be compatible with backpropagation
- Choosing activation and loss functions:

Problem	Activation	Loss
Binary classification	Sigmoid	Binary crossentropy
Multiclass, single label	Softmax	Categorical
Multiclass, multilabel	Sigmoid	Binary

Weight Pruning - reducing parameters by rank significance

Module 5 - Convolutional Neural Networks

Deep Learning with Python

Chapter 8: Deep Learning for Computer Vision

8.1 - Introduction to Convnets

- Feature: a learned, significant pattern in the data, contained within kernels
- The Convolutional Operation: convolving kernels across features of input data, producing a single value in the output feature map.
- Convolutional Layer: layer in which the convolutional operation is applied to input data
- Pooling Layer: layer in which the output feature map of a convolutional layer undergoes dimensionality reduction
  - Pooling window: size of window in feature map where values are drawn from
  - Max pooling takes the max value
  - Average pooling takes the average value
  - L2 uses the square
  * Improves generality, reduces overfitting
- Normalization Layer
- Dropout Layer
- Flattening Layer (conv & pooling → dense)

~

- Learned features (patterns) are translation invariant
- Features are also learned within spatial hierarchies

Chapter 9: Advanced Deep Learning for Computer Vision

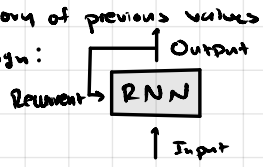
Module 6 - Recurrent Neural Networks

Deep Learning with Python

Chapter 10 : Deep Learning for Timeseries

10.3 - Recurrent Neural Networks

- Recurrent neural networks build on feedforward neural networks by adding a hidden state to each node that maintains information on the data that has previously passed through the node. This allows for memory of previous values in sequential data.
- Design:



- Computation: where a simple feed-forward neural network uses $y = f(w \cdot x + b)$

  where : f = activation function
  w = weights
  x = input
  b = bias

  RNNs use: $y = f(w \cdot x + u \cdot h + b)$

  where : f = activation function
  w = weights
  x = input
  u = hidden state weights
  h = hidden state
  b = bias

- Vanishing Gradient Problem: gradients can become vanishingly small in deep neural networks and for distant observations in a sequence. This makes training slow and/or impossible. As a result, RNNs have a hard time capturing long-term sequential patterns.

10.3A - Long Short-Term Memory (LSTM)

- LSTM NNs attempt to solve the vanishing gradient problem by introducing the notion of 'carry', which represents the carrying-forward of information from for-previous elements in a sequence (this is the information that is lost in simple RNNs)
- Carry at any point is calculated by three transformations, each of which take the form of a simple recurrent unit. Thus, LSTMs take the form $y = f(w \cdot x + u \cdot h + v \cdot c + b)$

  where : w = weight
  x = input
  h = hidden state
  u = hidden state weights
  c = carry
  v = carry weights
  b = bias

  Using i, f, and k to represent the transformations used to calculate carry, $C_{t+1} = i_t \cdot k_t + c_t \cdot f_t$

  where : $i_t = f(w_i \cdot x + v_i \cdot h + b_i)$ $\quad i$ = input
  $f_t = f(w_f \cdot x + v_f \cdot h + b_f)$ $\quad f$ = forget
  $k_t = f(w_k \cdot x + v_k \cdot h + b_k)$ $\quad k$ = cell state update

- What each of these three transformations do is defined by their weights

* Note: Chollet's description of LSTM models differs from the mainstream. A more mainstream description would be:

10.4 - Advanced RNNs

10.4.1 - Recurrent Dropout

- Dropout can be used in RNN, but the optimal way to use it is with a constant dropout mask. This was discovered by Yarin Gal as part of his '16 dissertation

10.4.3 - Bidirectional RNNs

- Bidirectional recurrent layers double traditional recurrent layers, each using an opposite sequence of the input data
- This allows RNNs to learn alternative representations of the data, which are then merged to create a fuller representation
- Whether or not bidirectionality is useful is dependent on the problem

10.5 - Gated Recurrent Units (GRUs)

Module 7 : Natural Language Processing with RNNs, LTSMs, 1D-CNNs

Deep Learning with Python

Chapter 11 : Deep Learning for Text

Module 8: Deep Generative Learning: Autoencoders and GANs

Deep Learning with Python

Chapter 12: Generative Deep Learning

Module 9: Reinforcement Learning

Deep Learning with Python