Notes for Final

6/22/2024

SVM

* Support Vector Machines are best suited for classification, small to medium data. – text p 175
* Soft Margin Classification less sensitive to outliers than hard. P. 176
* Linear SVM requires data that is linearly separable.
* Can use polynomial features on linear SVM in scikit for nonlinear SVM. P.179
* SVM regression is possible p 184
* K Means is non neural net approach to classifying unlabeled data.

Decision Trees

* Can handle both regression and Classification p. 195
* Split nodes ask more questions, leaf nodes are answer
* Gini impurity = node is pure (gini=0) if all training instances it applies to belong to the same class p. 197
* CART algorithm only has 2 children per split node
* Decision Trees are white box algorithms. Random forests and neural nets are black box p. 199
* CART is greedy; searches for optimum split at the top level.
* Complexity of trees is O(log(m)/log(2)) since they are mostly balanced.
* Entropy is another impurity measure, entropy = 0 when all belong to one class. P. 201
* Regression trees try to minimize MSE instead of impurity.

In machine learning, MSE stands for Mean Squared Error. It is a common loss function used for regression tasks, which measures the average of the squares of the errors or deviations—that is, the difference between the actual values and the predicted values produced by a model. The formula for MSE is:

[ \text{MSE} = \frac{1}{n} \sum\_{i=1}^{n} (y\_i - \hat{y}\_i)^2 ]

Where:

* (n) is the number of data points,
* (y\_i) is the actual value for the (i)th data point, and
* (\hat{y}\_i) is the model's predicted value for the (i)th data point.

MSE is always non-negative, and a value of 0 indicates perfect predictions. Lower values of MSE indicate better model performance, with fewer errors. Since MSE squares the errors before averaging them, it gives a relatively high weight to large errors. This means models are heavily penalized for making large prediction errors, making MSE a useful metric when large errors are particularly undesirable.

* Data shape very important for trees p. 206
* Main negative is high variance.

Neural Nets

* Logical neurons diagram p. 303. Binary inputs and one binary output.
* Perceptron is TLU, and inputs and outputs are numbers p 304. Perceptron is a layer of TLU’s all connected to every input, this is a dense layer (fully connected layer).
* How to train perceptron: for every output neuron that produced a wrong prediction, it reinforces the weight that would have led to right prediction. P. 306
* Backpropagation is combo of reverse-mode autodiff and gradient descent p. 310
* Handles mini batches and fully trains data multiple times, epochs.
* Mini batch passes forward through all layers, next the algo measures the output error using loss function.
* Then it calculates how much each connection to the output layer contributed to error using chain rules. The algo then moves back down each layer calculating the same by propagating the error gradient backwards.
* Finally, algo performs gradient descent step to tweak connections p 311
* Good activation functions: sigmoid, Relu, others p. 312
* P. 316 modern MLP diagram
* Keras sequential model is single stack of layers connected sequentially p. 319
* Loss functions improve gradient descent, not autodiff p. 324
* Fine tune models by changing number of hidden layers, number of neurons per hidden layer, learning rate & batch size, optimizer and activation functions p. 349
* Activation function transforms input into output, the loss function calculates how well the neuron performs.
* **Activation Function**: This function is applied to the output of a neuron or layer of neurons, transforming the input signal into an output signal. It decides whether a neuron should be activated or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions introduce non-linearity into the network, enabling it to learn complex patterns. Common examples include ReLU (Rectified Linear Unit), Sigmoid, and Tanh functions.
* **Loss Function**: Also known as the cost function, it measures how well the neural network performs by comparing the predicted outputs of the network to the actual target values. The loss function quantifies the difference between the two, guiding the optimization process during training. The goal of training is to minimize this loss, improving the model's accuracy. Examples of loss functions include Mean Squared Error (for regression tasks) and Cross-Entropy Loss (for classification tasks).

Clustering Algorithms

* K Means and DB SCAN p. 260
* K means you have to specify number of clusters, K.
* Centroids are center of clusters p. 267
* Silhouette score helps find best k p. 271
* DBSCAN is p.279

Training Deep Neural Networks

* Vanishing gradient – lower layer’s weights remain unchanged as gradient is already too small p. 358
* The problem was sigmoid function. And Glorot and He initialization were proposed. Keras uses Glorot by default.
* ReLU does not saturate for positive values p. 361
* It does lead to dead neurons that stop training.
* Batch normalization normalizes input at each layer p. 367. Only works in training, not testing.
* There is runtime penalty.
* Gradient clipping can help when BN is not possible p. 372.
* Faster optimizer than help speed up network p. 379
* Adam optimizer p. 384
* Learning rate is very important p. 388
* Too high = training will diverge, too low = will take very long time.
* Learning schedules vary the rate per cycle.
* Regularization can avoid overfitting p. 392.
* Dropout is popular technique, every training step, some neurons are dropped out, which causes network to shift.
* P. 400 NN practical guide.
* The softmax function in a neural network is used as an activation function in the output layer for multi-class classification problems. It converts the raw output scores (often referred to as logits) from the network into probabilities by taking the exponential of each output and then normalizing these values by dividing by the sum of all the exponentials. This ensures that the output values are in the range (0, 1) and sum up to 1, making them interpretable as probabilities. Each value thus represents the probability that the input belongs to one of the classes. The softmax function is defined as follows for a vector (z) of raw class scores from the final layer of a neural network, where (z\_i) is the score for class (i), and (K) is the total number of classes: This function is particularly useful for classification problems where each input is to be assigned to one and only one class, and the classes are mutually exclusive.
* SGD vs Adam:

Stochastic Gradient Descent (SGD) and Adam are both optimization algorithms used for minimizing the loss function during the training of machine learning models, especially in neural networks. They have key differences in how they approach optimization:

**Stochastic Gradient Descent (SGD):**

* **Basic Concept:** SGD updates the model's parameters by computing the gradient of the loss function with respect to the parameters for a single sample or a mini-batch (a small subset of the dataset). It then adjusts the parameters in the opposite direction of the gradient to minimize the loss.
* **Learning Rate:** It uses a fixed learning rate, although variations of SGD introduce learning rate schedules or adaptive learning rates.
* **Convergence:** The path to convergence can be noisy due to the stochastic nature of the algorithm, potentially leading to longer training times. However, this can also help the model escape local minima.
* **Simplicity:** It is simpler and has less computational overhead compared to more sophisticated optimizers like Adam.

**Adam (Adaptive Moment Estimation):**

* **Basic Concept:** Adam combines ideas from two other extensions of SGD, AdaGrad and RMSProp. Like RMSProp, Adam maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g., natural language and computer vision problems). Adam also keeps an exponentially decaying average of past gradients, similar to momentum, which helps to accelerate convergence.
* **Learning Rate:** Adam computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like RMSProp, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.
* **Convergence:** Adam generally converges faster than SGD because it uses the concept of momentum by combining the advantages of AdaGrad and RMSProp. It is well-suited for problems with large datasets or many parameters.
* **Complexity:** Adam is more complex and requires more computation for each iteration. It also has more hyperparameters to tune (e.g., the decay rates of the moving averages).

**Comparison Summary:**

* **Performance:** Adam often converges faster than SGD due to its adaptive learning rates and momentum. However, SGD can sometimes outperform Adam in terms of final model accuracy and generalization, especially with a well-tuned learning rate schedule.
* **Usage:** Adam is preferred for its ease of use and generally good performance across a wide range of problems, especially those with large datasets or high-dimensional spaces. SGD, due to its simplicity and effectiveness, remains a strong choice, particularly when simplicity and memory footprint are considerations.
* **Tuning:** Adam has more hyperparameters (e.g., learning rate, beta1, beta2, epsilon) which can be a double-edged sword; it provides more control but also requires more tuning. SGD, with its fewer parameters, can be easier to tune in practice.

CNN

* Convolution led to success of deep networks. Used in image recognition. Visual cortex p. 480.
* P. 481 neurons in the convolutional layer are only connected to certain pixels in input layer, and then the next layer is only focused on that section from the first layer.
* Filters = kernels. P. 484
* Padding makes sure output feature maps are all same size p. 488

In a Convolutional Neural Network (CNN), a kernel, also known as a filter, is a small matrix used to apply operations like edge detection, blurring, sharpening, etc., to the input data. Kernels are fundamental to the convolution operation that characterizes CNNs. They slide over the input data (e.g., an image) to produce a feature map, highlighting features like edges, colors, or textures.

**Key Points about Kernels in CNNs:**

* **Size:** Kernels are typically small (e.g., 3x3, 5x5) compared to the input data size. The choice of size affects the level of detail captured in the feature map.
* **Depth:** For input data with multiple channels (e.g., RGB images), the kernel has a depth that matches the input depth, allowing it to process all input channels simultaneously.
* **Convolution Operation:** As the kernel slides (or convolves) across the input data, it performs element-wise multiplication with the part of the input it covers, and the results are summed up to produce a single value in the feature map. This process is repeated across the entire input.
* **Learning:** The values in the kernel are learnable parameters. During training, the network learns the optimal values for these kernels to extract meaningful features from the input data, aiding in tasks like classification or object detection.
* **Multiple Kernels:** A CNN layer can have multiple kernels, each detecting different features. The output feature maps from these kernels can be stacked together to form the complete output of the layer.
* Pooling layers subsample (shrink) the input to reduce computational load. P. 491.
* Typical arch is convolutional layer – relu – convolutional – relu – etc -pooling layer – more convolution – pooling – etc. p. 495

GAN

* Autoencoders work on unlabeled data p. 635. They find patterns that helps me produce output very similar to input.
* Autoencoders look like multilayer perceptrons.
* P. 639 simple autoencoder example.
* Dimensionality reduction helps pre-train unlabeled data.
* You can build a convolutional autoencoder, good for images p. 648
* GAN = generator vs. discriminator p. 660
* Phase 1 = train discriminator pahse 2 = generator
* Gaussian noise used to train p. 662
* Mode collapse: generator gets good at one specific things, and only gets good at that. 664
* DCGAN = deep convolutional GAN 665
* Progressive growing GAN starts small and scales up closer to output p. 668
* Diffusion models start complicated and deconstruct. P. 674

Encoders

**Autoencoders:** In an autoencoder architecture, the encoder is the first half of the network. It compresses the input into a latent-space representation. It consists of convolutional layers (and often pooling layers) that progressively reduce the spatial dimensions of the input. The encoder is followed by a decoder, which attempts to reconstruct the input from the latent representation. Autoencoders are used for tasks like dimensionality reduction, denoising, and feature learning.

**Segmentation:** In segmentation tasks, especially with architectures like U-Net, the encoder part of the network captures the context of the input image. It works by reducing the spatial dimensions while increasing the depth (number of channels), capturing high-level features. This encoded representation is then used by the decoder part of the network to generate a pixel-wise segmentation map.

**Generative Models:** Some generative models, like Variational Autoencoders (VAEs), use an encoder to map input data to a distribution in latent space. The decoder then samples from this distribution to generate new data points. The encoder in such models is typically a CNN, leveraging its ability to efficiently process and compress image data.

**Encoder-Decoder Architectures for Translation and Segmentation:** Many tasks, such as image-to-image translation (e.g., style transfer) and semantic segmentation, use encoder-decoder architectures. The encoder compresses the input into a latent representation, capturing the necessary information, and the decoder uses this representation to generate the output.

In summary, encoders in CNNs are crucial for transforming input data into a form that captures the essential information in a more compact, latent space, facilitating various advanced tasks beyond simple image classification.

Genetic Algorithms

* Select random sample
* Select best results from sample and crossbreed
* Add in mutation
* Run algo again
* Important that crossover produce valid answers.
* Avoid re-evaluations, only evaluate children and mutants.

Bayes’ Rule

* p(C|X)=p(C)p(X|C)/p(X)
* Probability of C given X.
* Naïve Bayes assumes independence of factors
* A black text with a white background

  Description automatically generated with medium confidence

Attention

* P. 604. Important to translation
* Transformer p. 609
* Attention positive is explainability 626
* In machine learning, encoders, decoders, autoencoders, and transformers are concepts and architectures that serve different purposes but are interconnected in how they process and represent data.
* **Encoders and Decoders:**
* **Encoders** are components that transform input data into a different, often more compact representation. In the context of neural networks, an encoder might take an image and compress it into a lower-dimensional vector that captures its essential features.
* **Decoders** perform the inverse operation. They take the encoded data and reconstruct the original input or transform it into a new output format. For example, a decoder might take a compressed image representation and reconstruct the original image or generate a textual description of the image.
* **Autoencoders:**
* **Autoencoders** are neural networks that use encoder-decoder architectures for unsupervised learning. They aim to learn a compressed, efficient representation of the input data. An autoencoder first encodes the input into a latent-space representation (using an encoder) and then decodes this representation back into the original input format (using a decoder). The goal is often dimensionality reduction or feature learning. Autoencoders are trained by minimizing the difference between the input and its reconstruction, encouraging the network to capture the most important features in the latent space.
* **Transformers:**
* **Transformers** are a type of neural network architecture that has become prominent for handling sequential data, particularly in natural language processing (NLP). Unlike traditional sequence models like RNNs and LSTMs, transformers use self-attention mechanisms to weigh the importance of different parts of the input data. Transformers consist of encoder and decoder blocks, but they operate differently from the encoders and decoders in autoencoders:
* In **transformer encoders**, the input data (e.g., a sentence) is processed as a whole, with self-attention mechanisms allowing each part of the input to interact with every other part, capturing complex relationships.
* **Transformer decoders** are used in tasks that generate output sequences (e.g., translation). They process the encoded input in combination with previous outputs to generate the next item in the sequence.
* **Relationship:**
* The concepts of encoders and decoders form the basis of both autoencoders and transformers, focusing on transforming data into different representations and back. However, their applications and mechanisms differ.
* **Autoencoders** are primarily used for unsupervised learning tasks like dimensionality reduction and feature learning, focusing on reconstructing the original input as closely as possible.
* **Transformers**, leveraging encoder and decoder blocks, excel in handling sequential data, enabling state-of-the-art performance in tasks like translation, text generation, and more, through their ability to capture long-range dependencies and parallel processing capabilities.