Summary

sagisk

October 30, 2021

1 Chapter 1: An Introduction To Neural Networks

1.1 The Perceptron:

Input: $X = \{x_1, ..., x_d\}$.

Labels: $y \in \{-1, 1\}$

Weights: $W = \{w_1, ..., w_d\}$.

Prediction: $\hat{y} = \text{sign}\{WX\} = \text{sign}\{\sum_j w_j x_j\}$

Error: $E(X) = (y - \hat{y}) \in \{-2, 0, 2\}$

Objective: minimize $L = \sum_{(X,y)\in D}(y - \hat{y})^2 = \sum_{(X,y)\in D}(y - \text{sign}(WX))^2$

For binary classification we can write:

$L_i^{0/1} = \frac{1}{2}(y_i - \text{sign}(WX_i))^2 = \frac{1}{2}y_i^2 + \frac{1}{2}\text{sign}(WX_i)^2 - y_i\text{sign}(WX_i) = 1 - y_i\text{sign}(WX_i)$ However, since $\text{sign}(WX_i)$ part is dominating in our loss function and we need to make the loss function differentiable we can write:

$L_i = \max\{-y_i(WX_i), 0\}$.

Gradient: Uses a smooth approximation $\nabla_{\text{smooth}} L = \sum_{(X,y)\in D}(y - \hat{y})X$

Weight update: $W = W + aE(X)X = W + a(y - \hat{y})X$

For binary classification:

$W = W - a\nabla_W L_i$

Advantages:

- Works well when the data is linearly separable.

Similarities to other algorithms:

- SVM (Support Vector Machines): In SVM we have $L_i^{SVM} = \max\{1 - y_i(WX_i), 0\}$. In SVM updates on W have the same form as for the Perceptron i.e. $W = W + a\sum_{(X,y)\in S^+} yX$. where S^+ is the set of misclassified training points for which $y(WX_i) < 1$. The main difference between SVM and Perceptron is that Perceptron updates only misclassified points v.s. SVM also updates the points that are correctly classified butt are near the decision boundary.

1.2 Activation functions:

- Pre-activation value: $a_h = WX$
- Post activation value: $h = \Phi(a_h)$

Common functions:

- Linear:
  1. $\Phi(v) = v$.
  2. Fields: Regression.
  3. Derivative: 1.
- Sign:
  1. $\Phi(v) = \text{sign}(v)$.

2. Derivative: If $v \neq 0$ then 0 else non-differentiable.

- Sigmoid:
  1. $\Phi(v) = \frac{1}{1+e^{-v}}$.
  2. Fields: Binary classification to $(0,1)$.
  3. Derivative: $\Phi(v)(1-\Phi(v))$.
- Tanh:
  1. $\Phi(v) = \frac{2v-1}{2v+1}$.
  2. Fields: Binary classification to $[-1,1]$.
  3. Derivative: $1-\Phi(v)^2$
- ReLU (Rectified Linear Unit):
  1. $\Phi(v) = \max\{v, 0\}$.
  2. Fields: Replace Sigmoid.
  3. Derivative: For non-negative 1 else 0.
- Hard tanh:
  1. $\Phi(v) = \max\{\min[v,1], -1\}$. Replace tanh.
  2. Derivative: For arguments in $[-1,1]$ - 1 else 0.

If the multilayer network uses only the identity activation function in all its layers reduces to a single-layer network performing linear regression. Proof: $h_1 = \Phi(W_1 x) = W_1 x, ..., h_{p+1} = \Phi(W_{p+1}h_p) = W_{p+1}h_p, o = \Phi(W_{k+1}h_k) = W_{k+1}h_k = W_{k+1}W_k...W_1 x$. Hence, such a network can be represented as a single-layer network.

1.3 Loss functions:

- Regression:
  1. Squared loss: $(y-\hat{y})^2$.
  2. Hinge loss for $y \in \{-1,1\}$: $\max\{0, 1-y\hat{y}\}$
- Binary targets: $\log(1+e^{-y\hat{y}})$
- Categorical targets: $-\log(\hat{y_r})$

1.4 Multilayer Networks:

- Feed-Forward:
  - Input-to-hidden layer: $h_1 = \Phi(W_1^T x)$
  - Hidden-to-hidden layer: $h_{p+1} = \Phi(W_{p+1}^T h_p)$
  - Hidden-to-output layer: $o = \Phi(W_{k+1}^T h_k)$
- Backpropogation: The main goal of this phase is to learn the gradient. Let the weigth of the connection from hidden unit h_r to h_{r+1} be $w(h_r, h_{r+1})$. Then,

$$\frac{dL}{dw(h_{r-1},h_r)} = \frac{dL}{do}\Big[\sum_{h_r,h_{r+1},\dots,h_k,o} \frac{do}{dh_k}\prod_{i=1}^{k-1}\frac{dh_{i+1}}{dh_i}\Big]\frac{dh_r}{dw(h_{r-1},h_r)}$$

Where firstly the $\frac{dL}{do}$ is computed. Then the computation goes from later layers to the earlier layers (backwards). So, that while computing the gradient for the earlier layers we already knew the consecutive values for the later values to use them in the computations.

The basic idea of deep learning is that repeated composition of functions can often reduce the requirements on the number of base functions (computational units) by a factor that is exponentially related to the number of layers in the network. Or in other word the deeper the network the less units (compared to narrower networks) it will require to solve the task. Moreover, the early layers learn more detailed patterns, wheres the later layers learn higher-level patterns.

1.5 Practical Issues in Training:

1. Overfitting:
   - When happens:
     (a) More parameters then data.
   - Solutions:
     (a) Increase the number of training points.
     (b) Regularization: By adding penalty to the loss function, sharing parameters between nodes (depending on the context like c.n.n.), early stopping (stop training when the error on validation set starts to increase), ensemble methods (bagging: train multiple networks on samples of the training data and average the outputs, dropout: randomly drop nodes from layers with all their corresponding edges).
2. Vanishing/Exploding Gradient:
   - When happens:
     (a) Every time (especially in deep networks): To explain suppose we have several layers with one node in each. Then by backpropgation the local derivative of the loss function is given as the product of the derivatives of the activation functions at node times the incoming weight.
   - Solution:
     (a) Some possible solutions regularization and pretraining (see below).
3. Local/Spurious Optima:
   - When happens:
     (a) Common with higher dimensionalities. Spurious optima are local optima point in the training phase that are not present in the test phase.
   - Solutions:
     (a) Pretraining: Greedy, layerwise fashion train the network every layer one at a time. From outer layers to the inner layers. Each layer provides learn the initialization points for the next layer.

References