Review
---

1. Perceptron(Linear separator)
- Hypothesis set: sign(W^Tx)
- Learning algorithm: PLA
    - idea: start with some weights and try to improve it
    - algorithm:
    ```
    for i in max_iter:
        for (x, y) in in_samples:
            if sign(W^Tx) != y:
                all_mispred_samples.append((x, y))
        for (x, y') in all_mispred_samples:
            W_(i-1) += y * x
    ```
    - pocket algorithm: use the model with the smallest in-sample error
- Theoretical basis: If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.

2. Linear Regression
- y = W^Tx, E = (y-W^Tx)^2
- Has analytic solution:
    - Normal equation: take the derivative of E , we get X^TXw = X^Ty
    - The linear regression algorithm gets the smallest possible Ein in one step

3. Logistic Regression(for classification task)
- predict the probability with: sigmoid(W^Tx)
- sigmoid: 1 / (1 + e^(-s))
- Cross entropy loss(for binary case): E = ln(1 + e(-y * W^T * x))
    - y takes {+1, -1}, y is the real label
- Cross entropy is an alternative representation of maximum likelihood estimation,

the loss function used in logReg
    is actually derived from MLE

4. Softmax Regression(for multi-class classification)
- formula:
- relationship with logistic regression:
    - equivalence between softmax and sigmoid
    - equivalence between binary cross entropy and multi-class cross entropy:
- Cross entropy loss(for multi-class case)

5. Gradient Descent
- Batch-GD: update for the entire batch
- SGD: update for each sample
- Mini-Batch GD: update for each mini-batch (accumulate loss -> compute gradient -> update)

6. Neural Network
- Backpropagation Algorithm
```
Compute_X(sample):
    s, x = new_array(), new_array()
    s[0] = sample
    for l in (1, L):
        x[l] = s[l-1] * W[l]
        s[l] = actv(x[l])
    return x
```
```
Compute_sensitivity(x):
    E' = Loss'(S[L])
    sen[L] = E' * actv'(s[L])
    for l in (L-1, 1):
        sen[l] = sen[l+1] * W[l+1]^T * actv'(x[l])
    return sen
```
```
Backprop():
```



```
    for s in all_samples:
        x = Compute_X(s)             // compute output of each layer
        sen = Compute_sensitivity(s) // compute sensitivity of each layer
        E += Loss(s)
        for l in L:                  // for each layer
            G[l] = x[l-1] * sen[l-1]   // compute gradient for layer l
            G[l] += G[l] + 1/N * G[l]  // accumulate gradient
    for l in L:
        W[l] -= lr * G[l]            // update weight
```
- Generalization
    - L1 regularization
    - L2 regularization (weight decay)
    - Early stopping
    - Dropout
    - Data augmentation
- Better GD
    - variable learning rate
    - Steepest Descent (Line Search): binary search to decide the lr that minimize E in one step
    - ... and others

7. CNN
- Domain knowledge
    - translation invariance
    - locality
- Basics (hk is the size of the kernel)
    - Conv
        - H(i+1) = H(i) + hk -1
        - Backprop(DeConv): 1. full-padding 2. conv with inverted filters
    - Padding
        - "same" padding (or half-padding): Ph

= (hk - 1)/2 on each side
        - "valid" padding: not use any padding ...
            - without padding, pixels on boarders are under-represented
        - "full" padding: Ph = hk - 1 on each side, increase by hw-1, useful when doing reverse convolution
    - Strides (Sk is the stride)
        - H(i+1) = (H(i) - hk) / Sk + 1
    - Bias
        - one for each channel, added on each pixel
    - Pooling
        - diff with conv: apply on each feature map, does not change the number of feature maps
        - with stride of 1, H(i+1) = H(i) + hk -1
        - provide nonlinearity and translation invariance
        - global average pooling before FC to reduce computation load
    - Skip connection
        - provide unimpeded gradient flow
        - provide multiple level of abstractions and let the network itself to decide which level to use; the network
          becomes a "bag" of different models, similar to ensemble
        - the above point can be seen from an optimization perspective, in which says that deeper networks have more
          complicated loss surface and require much more time and more sophisticated optimization techniques to converge.
          The skip connections ease this difficulty by allowing the network to converge at "less-representative" minima

7. CNN - training
- preprocessing
    - zero-centered
    - normalization
- weight initialization
    - small random number (gaussian with zero mean and 1e-2 standard deviation)
        - for deeper networks, activation outputs become zero
        - weight updating becomes super slow, sometimes completely stop, G[l-1] = X^T * G[l-1]
    - Xavier initialization
        - small random / sqrt(fan_in)
- batch normalization
    - covariate shift: the change of the distribution of input data
    - almost eliminate gradient vanishing, alleviate internal covariate shift, regularization , network converge faster, can use larger learning rate
    - at test time, should use empirical parameters obtained at training stage
    - for fc layer, we do per-dimension batch norm; for conv, we do per-channel batch norm
- optimization
    - problems with sgd:
        - stuck in local minima
        - slow at saddle point
    - momentum: v[t] = alpha * v[t-1] + G; w[t+1] = w[t] - lr * v[t]
        - jump over local minima
        - speed up at saddle point
    - second-order optimization
        - learning rate determined by hessian matrix, point to minima so converge faster
        - computationally expensive
- model ensembles
    - train multiple independent model
    - at test time, take the average of their results

File - /Users/tiandi03/go/src/github.com/tiandi111/Notes/tamu/636DL/mid_term.md