Neural Networks

Monday, October 25, 2021 8:23 PM

2. **What is Deep Learning**

- Deep learning = ML using large/deep NN
- ML = technique to create new funcs using example behavior rather than explicit instructions
- Learning from examples
  - easier to solve rather than using explicit instructions
  - examples = data, expected answer = labelled data
- ML algorithms allow to create approx functions using set of example data
- Why NN?
  - we know how to train them efficiently
  - backpropagation quickly and efficiently finds high quality approximate function
- Why Deep NN?
  - effective when we have very large data training set
  - For problems where input is unstructured, problems with complex relationships but clear goals
- **Terminology**
  - Data Science: process of using data analysis to build understanding
  - ML: Process of using example data to create approximate functions to be applied to new data
  - NN: ML using an interconnected network of trainable artificial neurons that map some input X to output Y
  - DL: ML using multi layered neural networks, trained w large data sets
  - Supervised learning: ML when example data provides both expected input and output, supervised by identifying and correcting mistakes
  - Labelled Data: example data with expected output, used in supervised learning
  - Unsupervised learning: ML when only expected input is provided, ML learns relationships between inputs themselves
  - Unlabeled data: example data without expected outputs, used in unsupervised learning
  - Reinforcement learning: ML using high-level goals, trial and error during training

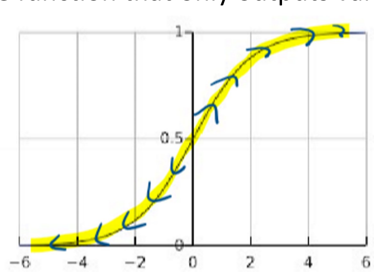
3. **ML with Logistic Regression**

- **Logistic Regression**
  - assumes that we can make prediction (hypothesis) based on a linear combination of the outputs

    $$z = w_0 x_0 + w_1 x_1 + \ldots + w_n x_n + b$$

  - **Binary Classification**
    - can use 0 and 1 to represent each group
    - **Sigmoid Function:** impose function that only outputs values bw 0 and 1

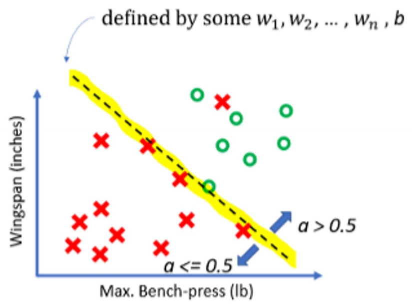
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- This sigmoid "forces" large values to be '1', and small values to be '0'

- Working logistic regression equation:

  $$a = \sigma(w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b)$$



- Finding parameters w1,w2,... b
  - Gradient Descent

- **Cost Function J**
  - Compare combinations of w_n and b to know which works best
  - create cost func J, measure of fitness
  - if *J(w1',w2',...b) < J(w1,w2...b)* then w1',w2'...b = better set of parameters
  - Choose a cost function using accuracy
    - *Accuracy = (right answers/total answers)*
    - want a cost function that gives guidance to next guess
    - derivative of func = rate of change, where func is going
    - **Gradient Descent**, adjusting parameters with understanding of where you are going

      if $J(w_1, w_2, \dots, w_n, b)$ is the overall cost, then

      $\frac{\partial J(w_1, w_2, \dots, w_n, b)}{\partial w_1}$ is the rate of change of the cost w.r.t w_1
  - can improve parameters by incrementally adjusting them based on derivative

    $$w = w - \propto \frac{\partial J(w,b)}{\partial w}$$

    $$b = b - \propto \frac{\partial J(w,b)}{\partial b}$$

    Where, $\propto$, is size of the adjustment,
    normally called the **learning rate**.
  - **Log Function**

    $$\frac{d}{da} log(a) = \frac{1}{a}$$

    - Cost for data point (Loss, L) using log func

      When y = 1: $L(a,y) = -\log(a)$
      When y = 0: $L(a,y) = -\log(1-a)$

      $$L(a,y) = -(y\log(a) + (1-y)\log(1-a))$$
    - Calculate partial derivatives

      $$L(a,y) = -(y \log a + (1-y)\log(1-a)) \Rightarrow \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} = \frac{a-y}{a(1-a)}$$

      $$a = \sigma(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{\partial a}{\partial z} = \sigma(z)(1-\sigma(z)) = a(1-a)$$

      $$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b \Rightarrow \frac{\partial z}{\partial w_1} = x_1, \frac{\partial z}{\partial w_2} = x_2, \dots, \frac{\partial z}{\partial b} = 1$$

      $$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1} = x_1(a-y)$$

      $$\frac{\partial L}{\partial w_n} = x_n(a-y) \qquad \frac{\partial L}{\partial b} = (a-y)$$
    - across each of the m data points for cost function J:

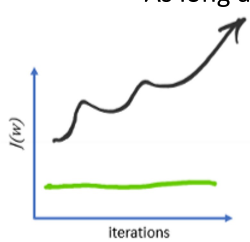
$$J = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{i}\log(a^{(i)}) + \sum_{i=1}^{m}(1 - y^{(i)})\log(1 - a^{(i)})\right)$$

$$\frac{\partial J}{\partial w_n} = -\frac{1}{m}\sum_{i=1}^{m} x_n^{(i)}(a^{(i)} - y^{i}) \qquad \frac{\partial J}{\partial b} = -\frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{i})$$

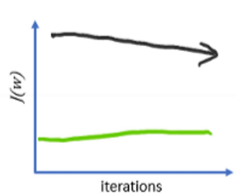
- Steps:
  1. Assume: $a = \sigma(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b)$
  2. Initialize w_{1-n}, b to random values (or zero)
  3. Repeatedly apply: $w = w - \propto \frac{\partial J(w,b)}{\partial w}$ $\quad b = b - \propto \frac{\partial J(w,b)}{\partial b}$
  4. Stop when J < target error

4. Implementation and Evaluation of Logistic Regression

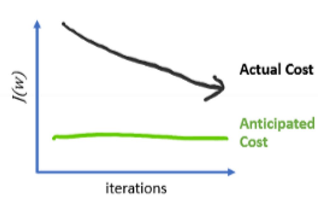
- Inaccuracy in ML
  - AI does not match nature of data (not linearly separable)
  - algorithm did not find best parameters
  - example data is not representative of new data
    - Not enough data to represent funciton
    - data is noisy
    - underlying behavior not deterministic
- Failure to find best parameters
  - accuracy impacted by ability to get to good parameters
  - **hyperparameters**
    - in logistic regression, 2 tuneable components: learning rate and number of iterations
  - Learning rate is too large
    - final parameters are worst than random
  - Learning rate is too small
    - final parameters are better than random but not optimal
  - Number of iterations is too small
    - Final parameters are better than random, but not optimal
  - Number of iteration is too large
    - As long as learning rate small enough, only costs CPU cycles



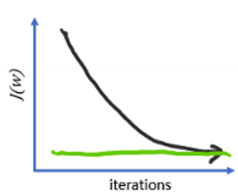
Learning rate is too **high.**



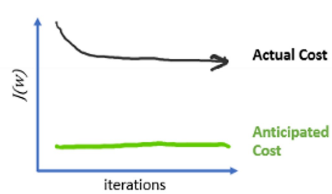
Learning rate is too **low.**



Number of iterations to **low.**



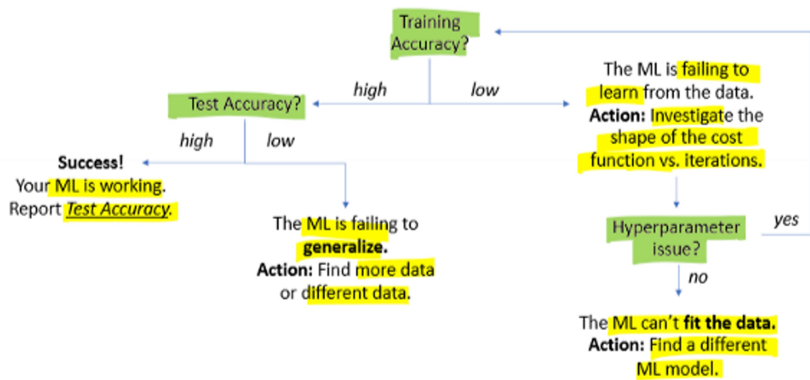
Good behavior.



Learning unsuccessful.
(Data/model fit issue)

Issue	Symptom	Action
AI Model Doesn't Fit Data.	Training accuracy is low and hyperparameter tuning doesn't help.	Consider a different AI model (NN, for instance)
We are not finding the best parameters.	Unexpected shape of cost/iterations graph.	Tune the hyperparameters.
Example data does not represent the new data. (Lack of data, noisy data, non-deterministic data)	High training accuracy but test accuracy is low.	Try to find more data, better data or different data.

- **Building a Test Data set**
  - help understand ML
  - take some data, put off to the side, lavelled data has the right answer



- **Vectorization**
  - structure learning code correctly

inputs

$$A = \sigma(w^T X + b)$$ → bias

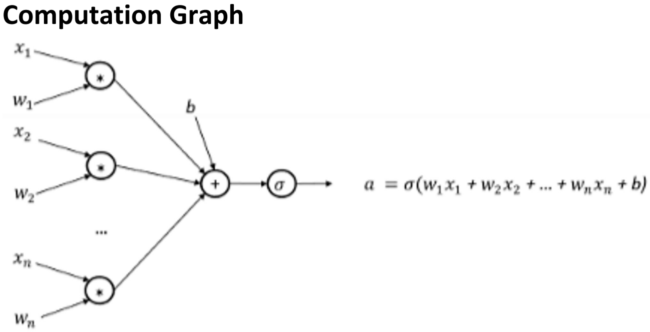
vector

- Where: Answer For all inputs

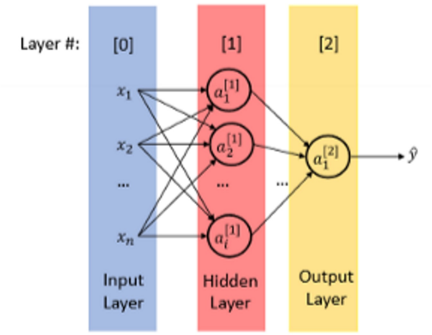
X is a (n, m) input matrix (*i.e all* of the input features for each example)
n is the number of features in the input data
m is the number of examples in the training data
w is a (n,1) parameter matrix
b is the bias
A is a $(1,m)$ vector which represents the hypothesis for each of the m samples

5. **Neural Network**

- Logistic Regression assumes a linear relationship
- NNs can learn very complex non-linear relationships rather than having to specify them
- **Computation Graph**



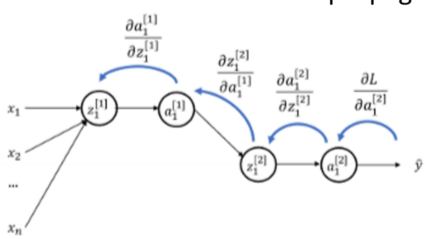
  - **Activation Function**
    - non-linear activation function is the key to allowing combinations of logistic regression units to produce complex functions
    - without non-linear activation functions, all combs of logistic regression would continue to be linear



- can learn parameters of NN using gradient descent for Logistic regression, using cost function

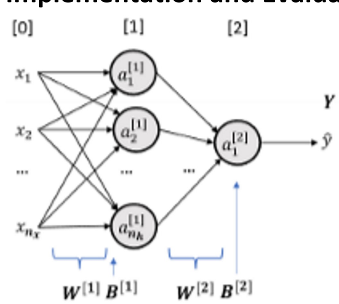
$$J(\hat{y}, y) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^i \log(\hat{y}^{(i)}) + \sum_{i=1}^{m}(1 - y^{(i)})\log(1 - \hat{y}^{(i)})\right)$$

  - Calculate y^ using computation graph
  - determine loss
  - update each parameter (using partial derivative of cost)
  - repeat until J < target
- Partial derivatives and Backpropagation



  - Backpropagatin the error and attributing it to each node

6. **Implementation and Evaluation of NN**



- If we consider a single example, i:

$$z^{[1](i)} = W^{[1]}x^{(i)} + B^{[1]}$$
$$a^{[1](i)} = g(z^{[1](i)}), \text{ where } g(z) = \tanh(z)$$

- And:

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + B^{[1]}$$
$$\hat{y}^{(i)} = a^{[2](i)} = \sigma(z^{[2](i)})$$

- Backpropagation - Vectorization

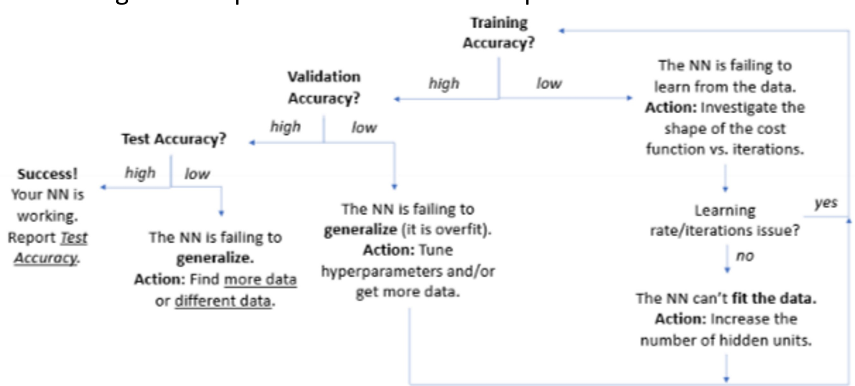
	Element-wise	Vectorized - single sample, i	Vectorize for all m samples
(1)	$dz_1^{[2]} = (\hat{y} - y)$	$dz^{[2](i)} = (\hat{y}^{(i)} - y^{(i)})$	$dZ^{[2]} = (\hat{Y} - Y)$ (1,m) (1,m) (1,m)
(2)	$dw_{1,1}^{[2]} = dz_1^{[2]}a_1^{[1]}$	$dW^{[2](i)} = dz^{[2](i)}a^{[1](i)T}$	$dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]T}$ (1,n_h) (1,m)(m,n_h)
(3)	$db_1^{[2]} = dz_1^{[2]}$	$dB^{[2](i)} = dz^{[2](i)}$	$dB^{[2]} = \frac{1}{m}\sum_{rows} dZ^{[2]}$ (1,1) (1,m)
(4)	$dz_1^{[1]} = w_1^{[2]}dz_1^{[2]}g'(z_1^{[1]})$	$dz^{[1](i)} = W^{[2](i)T}dz^{[2](i)} * g'(z^{[1(i)]})$	$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g'(Z^{[1]})$ (n_h,m) (n_h,1)(1,m)
(5)	$dw_{1,1}^{[1]} = dz_1^{[1]}x_1$	$dW^{[1](i)} = dz^{[1](i)}x^{(i)T}$	$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$ (n_h,n_x) (n_h,m) (m,n_x)
(6)	$db_1^{[1]} = dz_1^{[1]}$	$dB^{[1](i)} = dz^{[1](i)}$	$dB^{[1]} = \frac{1}{m}\sum_{rows} dZ^{[1]}$ (n_h,1) (n_h,m)

Brad Quinton, Scott

- Initializing parameters in NN
  - in LR, initialize parameters to any starting values, usually 0
  - Using 0 or uniform non-zero value does not work with NN?
- **Overfit in NN**

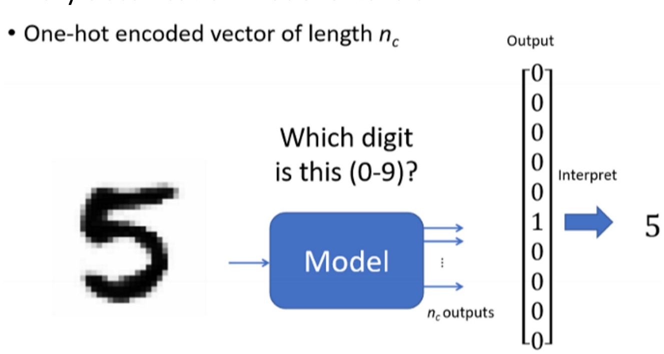
- it is possible that the NN finds an approx func that fits training data but not new data
- need to tune hyperparameters for optimal performance
- because of overfit in NN, split 3 data sets

- need 3rd data set to measure expected performance
- don't want to simply fit hyperparameters to validation set
- Validation set gives data to tune hyperparameters
- Test data gives independent ref to measure performance of AI



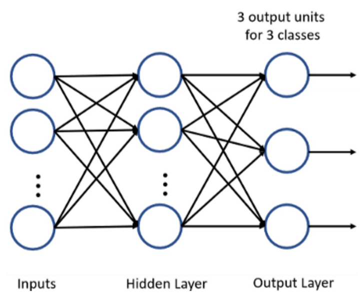
7. Multiclass Classification

- Binary Classification
  - 2 choices
- Multiclass classification
  - n_c number of possible classes, ex. MNIST dataset
  - input has 1 label
- Multilabel classification
  - input has 1+ label
  - cannot use softmax
  - user separate classifiers or sigmoids on outputs
  - labels cannot be one hot encoded vectors
- One hot encoding
  - Binary classification model extension

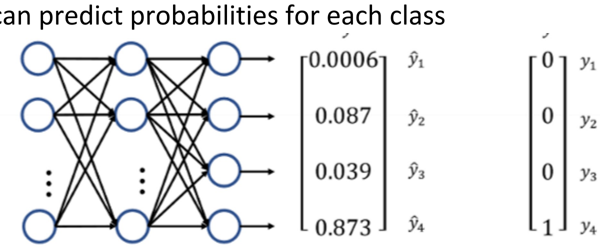


- Extending binary classifier for n_c classes
  - **Multiple Binary Classifiers**
    - One vs All
      - build multiple binary classifiers, one binary classifier/class each classifier predicts whether input is in class or not
    - One vs One

            - ***n_c(n_c -1)/2*** binary classifiers, all possible combinations of 2 classes
            - each classifier only receives data about pair of classes discriminating between
            - majority voting scheme to select predicted class
            - scales poorly with class numbers
            - performs same as OvA
            - DNN more efficient
    - **Single NN with multiple outputs**
        - one NN, change output layer to have 1 node per class
        - each output acts as binary classifier for that class (0 or 1)



        - can possibly use sigmoid as activation on output layer? interpret as probability
- **Softmax Activation Function**
    - normalizes outputs st output nodes produce value bw 0-1 and sum to 1
    - can predict probabilities for each class



$$L(\hat{y}, y) = -\sum_{j=1}^{n_c} y_j log(\hat{y}_j) = ???$$

    - produces vector length n_c

$$g_i(Z) = \frac{e^{z_i}}{\sum_{j=1}^{n_c} e^{z_j}}$$

    - Categorical Cross Entropy Loss (Softmax Loss) is a generalization of Binary Cross Entropy Loss

$$L(\hat{y}, y) = -\sum_{j=1}^{n_c} y_j log(\hat{y}_j)$$

    - generalization of sigmoid
    - Cost function

$$J(W, B) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

    - Backpropagation

8. **Deep Neural Networks**

- bias shifts the function, else only lines passing through origin
- 3 layer NN?
- Fully connected FC or dense layers: each input connects to each node
    - each FC layer can have diff number of units
    - diff number of weights&biases?
    - ex. 2 hidden layers:

$$Y(X) = \sigma\left(W^{[3]}\tanh\left(W^{[2]}\tanh\left(W^{[1]}X + B^{[1]}\right) + B^{[2]}\right) + B^{[3]}\right)$$

- Feature engineering
    - with LR, can manually transform features to encode non-linearity
    - data is now linearly separable
- Works best in problems with unstructured data as input, or complex relationships but clear goals
- **Backpropagation through softmax and categorical cross entropy loss**