L3 - Learning algorithm for linear neuron

- 2017/11/03 10:40
- Learning the weights of a linear neuron -- 与 linear regression 一样
  - differ with perceptron
    - essence
      - perceptron: move the estimated weights closer to the actual weights
        - can't work for multi-layer NN
      - linear neuron: move the output closer to the real target
  - How
    - Linear neurons (also called linear filters)
      - hypethesis: a weighted sum of its inputs
        - $y = w'x$
      - The error: the squared difference between the desired output and the actual output. Sum squared Error

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$

    - The iterative approach:
      - Start with random guesses for the prices and then adjust them to get a better fit to the observed prices of whole meals.
        - The "delta-rule" for learning is

$$\Delta w_i = \varepsilon \, x_i (t - y)$$

          - 针对一个point，SGD
        - The batch delta rule:

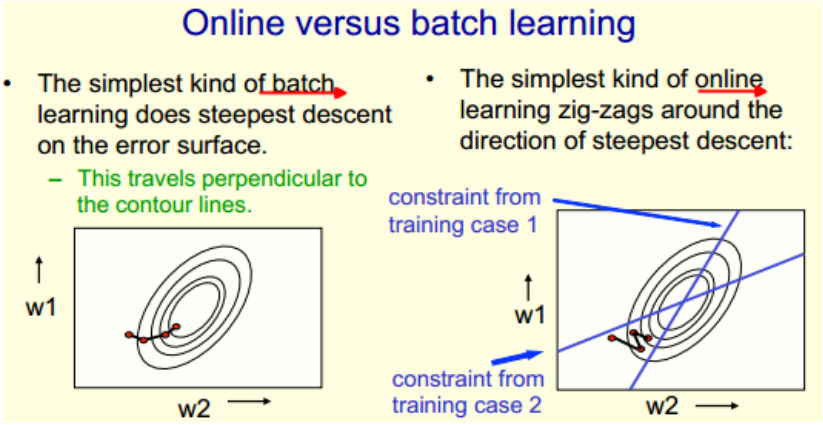
$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon \, x_i^n (t^n - y^n)$$

- 针对所有data，BGD
  - Instead of showing the weights get closer to a good set of weights, show that the actual output values get closer the target values
    - This can be true even for non-convex problems where averaging two good sets of weights may give a bad set of weights.
- Comments
  - this can't guantee the perfect answer, but when the learning rate is small enough, we can get very close to the best answer.
  - convergence speed
    - learning rate control the speed
      - small: slow but closer to the optimum
      - big: faster but less accurate, and may diverge if too big
      - So we have to choose a learning rate. This is annoying.
    - if two input dimensions are highly correlated, it can be very slow

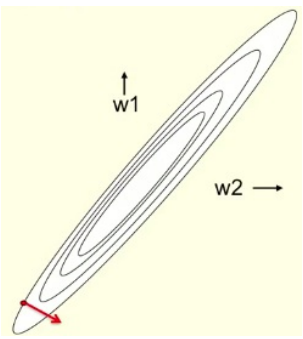
- The error surface for a linear neuron
  - meaning
    - to get a nice geometrical understanding of what's happening when we learn the weights of a linear neuron.
  - J/E over wi

- for quadratic error, it's a quadratic bowl
- For multi-layer, non-linear nets the error surface is much more complicated.
    - As long as the **weights aren't too big, the error surface will still be smooth**, but it may have many local minimum.

- Online versus batch learning
    - GD: doing steepest descent on the error surface
    - batch learning - Batch GD
        - take it at right angles to those elliptical contour lines
    - Online learning - Stochastic GD
        - To get the training case correct, we must lie on one of those blue lines.
        - the delta rule will move us perpendicularly towards that line
        - if we alternate between the two training cases, we'll zigzag backwards and forwards, moving towards the solution point which is where those two lines intersect.



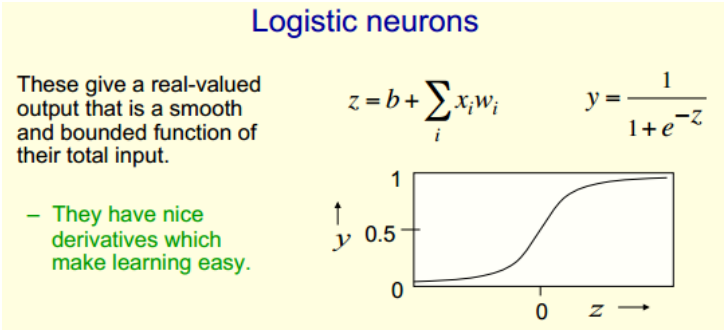
- why learning can be slow
    - cause
        - when the ellipse is very elongated, which happens when the lines that correspond to training cases is almost parallel



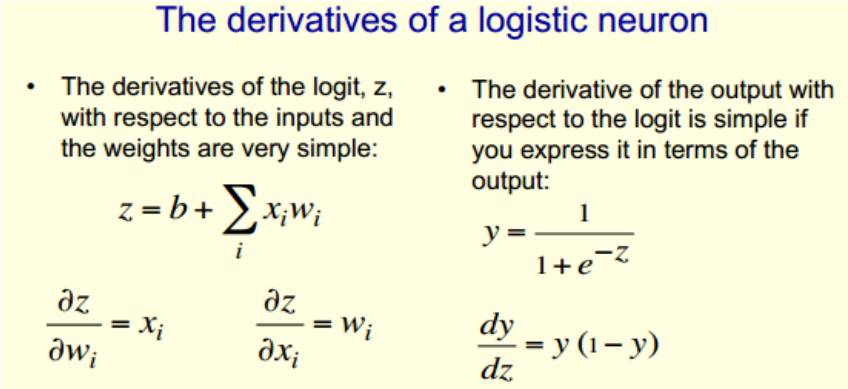
    - how

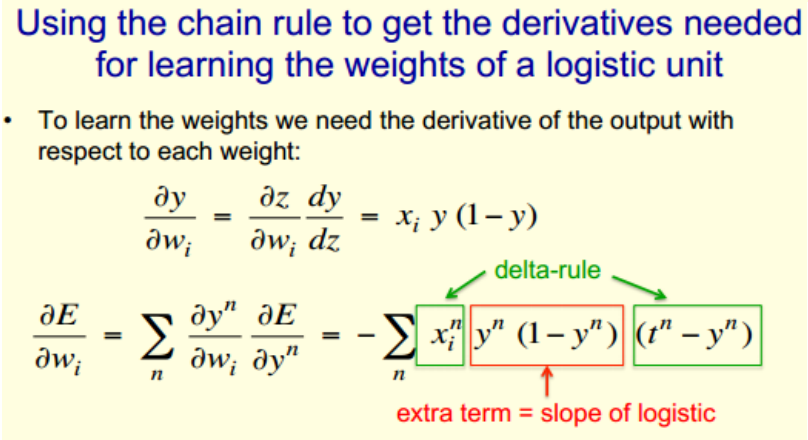
- If you look at the red arrow in the picture, the gradient is big in the direction in which we don't want to move very far, and it's small in the direction in which we want to move a long way.

- Learning the weights of a logistic output neuron
  - two things required
    - logistic neuron
    - derivative of a logistic neuron
  - logistic neuron



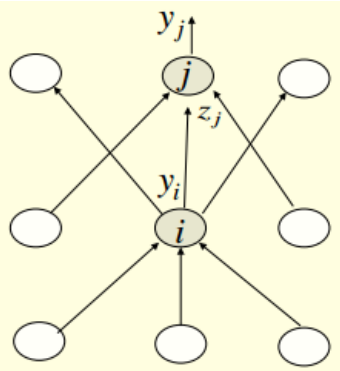
    - output ranges (0, 1), also can be treated as probability of the output based on the input
  - derivative of a logistic neuron





- 这个CF还是原来的Sum squared error, 所以相比于ML中学的多了这个 slope of logistic

- The backpropagation algorithm - invented in 1980s
  - meaning
    - in order to learn the hidden unit, because
      - Networks without hidden units are very limited in the input-output mappings they can model
      - We need to **automate the loop of designing features** for a particular task and seeing how well they work without requiring insights into the task or repeated trial and error
  - intuitive solution: perturbation of the weights. But it's very inefficient.
  - The idea behind backpropagation
    - We don't know what the hidden units ought to do, but **we can compute how fast the error changes as we change a hidden activity.**
      - use error derivatives w.r.t. hidden activities to train the hidden units
    - We can compute error derivatives for all the hidden units efficiently at the same time
  - Derivatives
    - 三个公式分别对应
    - First convert the discrepancy between each output and its target value into an error derivative. - dE/dzj 计算Error对每个最终输入z的偏导
    - • Then compute error derivatives in each hidden layer from error derivatives in the layer above. - dE/dyi = 上层的偏导 * 对应的 w
    - • Then use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights.



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

- How to use the derivatives computed by the backpropagation algorithm
  - To get a fully specified learning procedure, we still have to take care of other issues about how to use these error derivatives, such as
    - optimization issues: how to use error derivatives (lecture 6)
    - generalization issues (lecture 7)
  - optimization issues in using weight derivatives
    - **how often update the weight**
      - - Online: after each training case.
      - - Full batch: after a full sweep through the training data.
      - - Mini-batch: after a small sample of training cases.
        - more suitable for complex NN or with large training sets
    - **how much to update the weight**
      - - Use a fixed learning rate?
      - - Adapt the global learning rate?
        - increase when E decrease steady; decrease when E increase
      - - Adapt the learning rate on each connection separately?
      - - Don’t use steepest descent?
        - the e.g. before where the steepest direction is othogonal to the minima

- generalization issues
  - Overfitting: The downside of using powerful models
  - cause
    - The training data contains information about the regularities in the mapping from input to output. But it also contains two types of noise.
      - - The target values may be unreliable, wrong labels (usually only a minor worry).
      - - There is sampling error.
    - When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
  - how to define a good model
    - A model is convincing when it fits a lot of data surprisingly well.
    - It is not surprising that a complicated model can fit a small amount of data well.
- to prevent overfitting for e.g.
  - **fit small degree polynomials**
  - **constrain the coefficients of the polynomial to be small values**
  - **get more data**

- A large number of different methods have been developed.
  - - Weight-decay
    - keep the weights small or even zero
  - - Weight-sharing
    - keep many weights have small value
  - - Early stopping
    - test while training, stop training when test error getting worse
  - - Model averaging
    - train many different NN and average the weights
  - - Bayesian fitting of neural nets
    - fancy type of Model averaging
  - - Dropout

    - try and make your model more robust by randomly emitting hidden units when you're training it.
- - Generative pre-training