Gradient Descent

Problem Statement

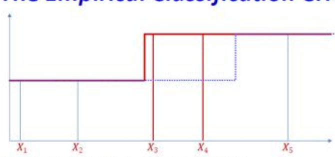
- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_N, \boldsymbol{d}_N)$
- Minimize the following function

$$Loss(W) = \frac{1}{N}\sum_i div(f(X_i; W), d_i)$$

w.r.t W

- This is problem of function minimization
  - An instance of optimization

The *Empirical Classification* error

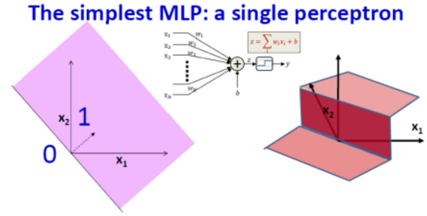


- The obvious error metric in a classifier is binary
  - The classifier is either right (error=0) or wrong (error=1)
    - Either $f(X; \boldsymbol{W}) = d$, or $f(X; \boldsymbol{W}) \neq d$

$$EmpiricalError(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} 1(f(X_i; \boldsymbol{W}) \neq d_i)$$

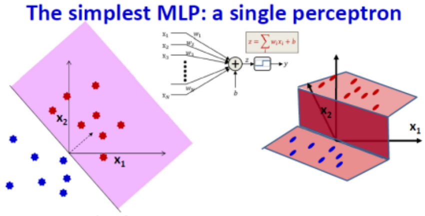
- **Learning the classifier:** Minimizing the count of misclassifications

The simplest MLP: a single perceptron



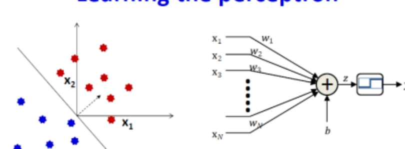
- Learn this function
  - **A step function across a hyperplane**

The simplest MLP: a single perceptron



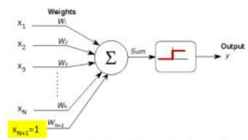
- Learn this function
  - **A step function across a hyperplane**
  - **Given only samples from it**

Learning the perceptron



- Given a number of input output pairs, learn the weights and bias
  - $y = \begin{cases} 1 & if \ \sum_{i=1}^{N} w_i X_i + b \geq 0 \\ 0 & otherwise \end{cases}$ Boundary: $\sum_{i=1}^{N} w_i X_i + b = 0$
  - Learn $W = [w_1 .. w_N]^T$ and b, given several (X, y) pairs

Restating the perceptron



- Restating the perceptron equation by adding another dimension to X

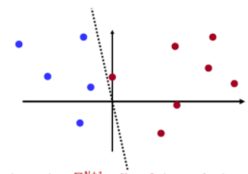
$$y = \begin{cases} 1 \ if \ \sum_{i=1}^{N+1} w_i X_i \geq 0 \\ 0 \ otherwise \end{cases}$$

where $X_{N+1} = 1$

- Note that the boundary $\sum_{i=1}^{N+1} w_i X_i = 0$ is now a hyperplane through origin

38

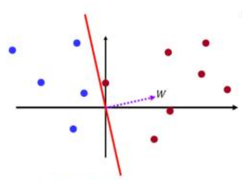
The Perceptron Problem



- Find the hyperplane $\sum_{i=1}^{N+1} w_i X_i = 0$ that perfectly separates the two groups of points

39

The Perceptron Problem

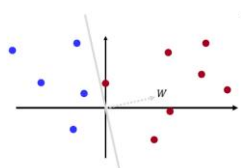


When the inner product of two vectors is zero, what would be the angle between two vectors?

- Find the hyperplane $\sum_{i=1}^{N+1} w_i X_i = 0$ that perfectly separates the two groups of points
  - Let vector $W = [w_1, w_2, \ldots, w_{N+1}]^T$ and vector $X = [x_1, x_2, \ldots, x_N, 1]^T$
  - $\sum_{i=1}^{N+1} w_i X_i = W^T X$ is an inner product
  - $W^T X = 0$ is the hyperplane comprising all Xs orthogonal to vector W
    - Learning the perceptron = finding the weight vector W for the separating hyperplane
    - W points in the direction of the positive class

40

The Perceptron Problem



- Learning the perceptron: Find the weights vector W such that the plane described by $W^T X = 0$ perfectly separates the classes
  - $W^T X$ is positive for all red dots and negative for all blue ones

41

Perceptron Algorithm: Summary

- Cycle through the training instances
- Only update W on misclassified instances
- If instance misclassified:
  - If instance is positive class (positive misclassified as negative)

    $$W = W + X_i$$

  - If instance is negative class (negative misclassified as positive)

    $$W = W - X_i$$

43

Perceptron Learning Algorithm

- Given N training instances $(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)$
  - $y_i = +1$ or -1

Using a +1/-1 representation for classes to simplify notation

- Initialize W
- Cycle through the training instances:
- do
  - For $i = 1..N_{train}$

    $$O(X_i) = sign(W^T X_i)$$

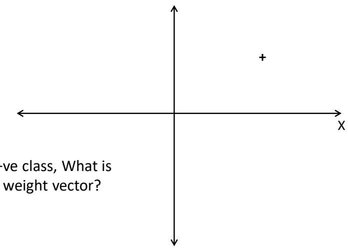
    - If $O(X_i) \neq y_i$

      $$W = W + y_i X_i$$

- until no more classification errors

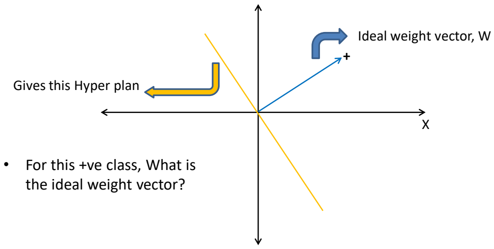
44

Ideal Weight vector



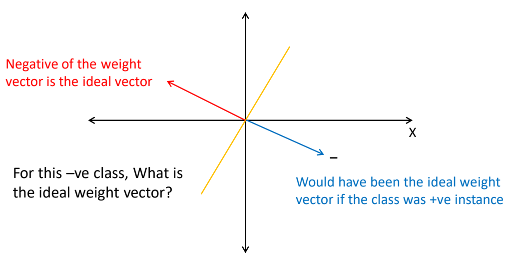
- For this +ve class, What is the ideal weight vector?

Ideal Weight vector



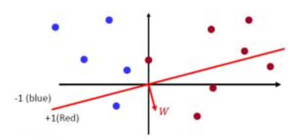
- For this +ve class, What is the ideal weight vector?

Ideal Weight vector



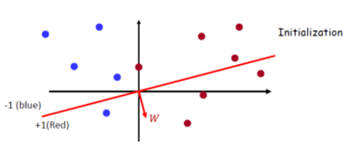
- For this –ve class, What is the ideal weight vector?

A Simple Method: The Perceptron Algorithm

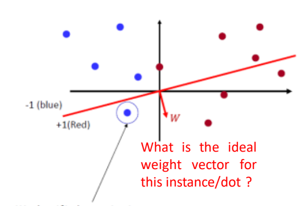


- **Initialize:** Randomly initialize the hyperplane
  - I.e. randomly initialize the normal vector W
- **Classification rule** $sign(W^T X)$
  - Vectors on the same side of the hyperplane as W will be assigned +1 class, and those on the other side will be assigned -1
- The random initial plane will make mistakes

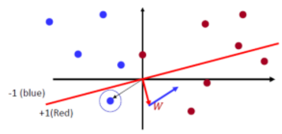
Perceptron Algorithm



Perceptron Algorithm

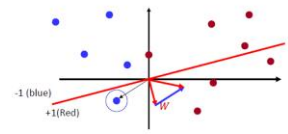


Perceptron Algorithm



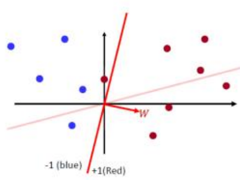
Misclassified *negative* instance, *subtract* it from W

Perceptron Algorithm



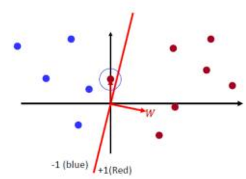
The new weight

Perceptron Algorithm



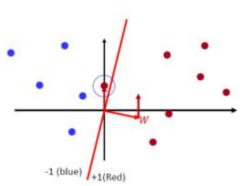
The new weight (and boundary)

Perceptron Algorithm



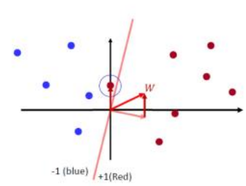
Misclassified *positive* instance

Perceptron Algorithm



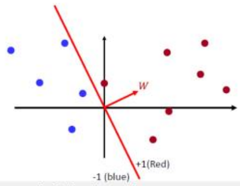
Misclassified *positive* instance, *add* it to W

Perceptron Algorithm



The new weight vector

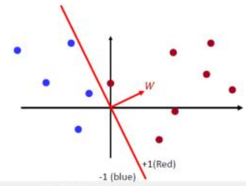
Perceptron Algorithm



The new decision boundary
Perfect classification, no more updates, we are done

54

Perceptron Algorithm



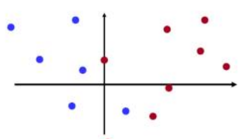
The new decision boundary
Perfect classification, no more updates, we are done

If the classes are linearly separable, guaranteed to converge in a finite number of steps

5

The Perceptron Solution: when classes are not linearly separable

Key: Red 1, Blue = -1

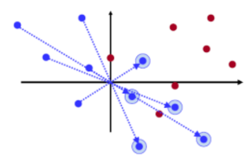


- When classes are not linearly separable, not possible to find a separating hyperplane
  - No "support" plane for reflected data
  - Some points will always lie on the other side
- Model does not support perfect classification of this data
- **Perceptron algorithm will never converge**

58

A simpler solution

Key: Red 1, Blue = -1

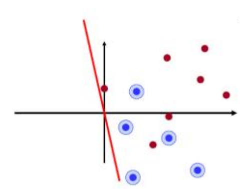


- *Reflect* all the negative instances across the origin
  - Negate every component of vector X
- If we use class $y \in \{+1, -1\}$ notation for the labels (instead of $y \in \{0,1\}$), we can simply write the "reflected" values as $X' = yX$
  - Will retain the features X for the positive class, but reflect/negate them for the negative class

59

The Perceptron Solution

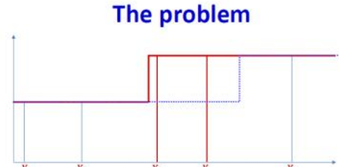
Key: Red 1, Blue = -1



- Learning the perceptron: Find a plane such that all the modified (X') features lie on one side of the plane
  - Such a plane can always be found if the classes are linearly separable

60

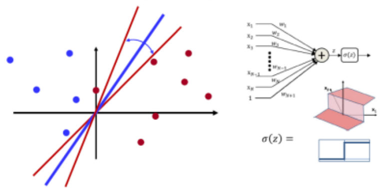
The problem



- Our binary error metric is not useful
  - To improve the classifier we must move the blue dotted line left
  - But if we move it only slightly, moving it either right or left results in no change in error

94

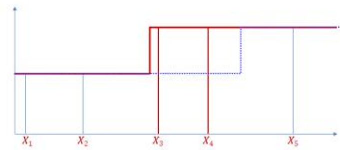
Why this problem?



- The perceptron is a flat function with zero derivative everywhere, except at 0 where it is non-differentiable
  - You can vary the weights a *lot* without changing the error
  - There is no indication of which direction to change the weights to reduce error

95

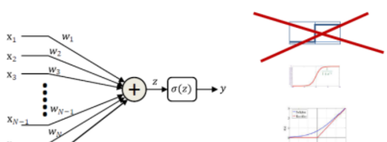
The solution



- Change our way of computing the mismatch such that modifying the classifier slightly lets us know if we are going the right way or not
  - This requires changing both, our activation functions, and the manner in which we evaluate the mismatch between the classifier output and the target output
  - Our mismatch function will now not actually count errors, but a *proxy* for it

98

Solution: Differentiable activation

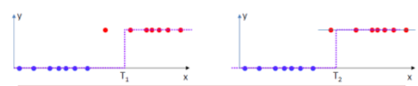


Activation functions $\sigma(z)$

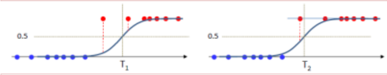
- Let's make the neuron differentiable, *with non-zero derivatives over much of the input space*
  - Small changes in weight can result in non-negligible changes in output
  - This enables us to estimate the parameters using gradient descent techniques..

99

Differentiable Mismatch function



- Threshold activation: shifting the threshold from T_1 to T_2 does not change classification error
  - Does not indicate if moving the threshold left was good or not

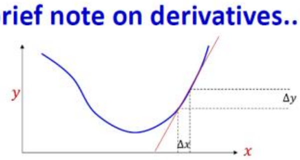


- Smooth, continuously varying activation: Classification based on whether the output is greater than 0.5 or less
  - Quantify *how much* the output differs from the desired target value (0 or 1)
  - Moving the function left or right changes this quantity, even if the classification error itself doesn't change

100

A brief note on derivatives..

derivative



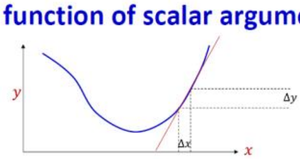
- A derivative of a function at any point tells us how much a minute increment to the *argument* of the function will increment the *value* of the function
  - For any $y = f(x)$, expressed as a multiplier α to a tiny increment Δx to obtain the increments Δy to the output

$$\Delta y = \alpha \Delta x$$

  - Based on the fact that at a fine enough resolution, any smooth, continuous function is locally linear at any point

5

Scalar function of scalar argument



- When x and y are scalar

$$y = f(x)$$

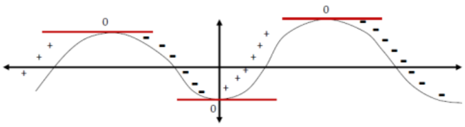
  - Derivative:

$$\Delta y = \alpha \Delta x$$

  - Often represented (using somewhat inaccurate notation) as $\frac{dy}{dx}$
  - Or alternately (and more reasonably) as $f'(x)$

6

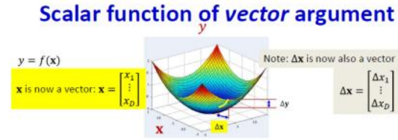
Scalar function of scalar argument



- Derivative $f'(x)$ is the *rate of change* of the function at x
  - How fast it increases with increasing x
  - The magnitude of f'(x) gives you the steepness of the curve at x
    - Larger |f'(x)| → the function is increasing or decreasing more rapidly
- It will be positive where a small increase in x results in an *increase* of f(x)
  - Regions of positive slope
- It will be negative where a small increase in x results in a *decrease* of f(x)
  - Regions of negative slope
- It will be 0 where the function is locally flat (neither increasing nor decreasing)

7

Multivariate scalar function: Scalar function of *vector* argument



$$\Delta y = \alpha \Delta \mathbf{x}$$

- Giving us that α is a row vector: $\alpha = [\alpha_1 \quad \cdots \quad \alpha_D]$

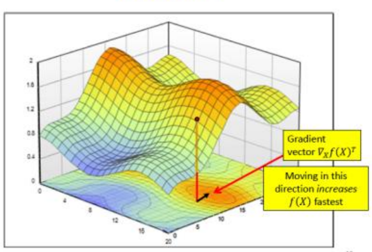
$$\Delta y = \alpha_1 \Delta x_1 + \alpha_2 \Delta x_2 + \cdots + \alpha_D \Delta x_D$$

- The *partial* derivative α_i gives us how y increments when *only* x_i is incremented
- Often represented as $\frac{\partial y}{\partial x_i}$

$$\Delta y = \frac{\partial y}{\partial x_1} \Delta x_1 + \frac{\partial y}{\partial x_2} \Delta x_2 + \cdots + \frac{\partial y}{\partial x_D} \Delta x_D$$

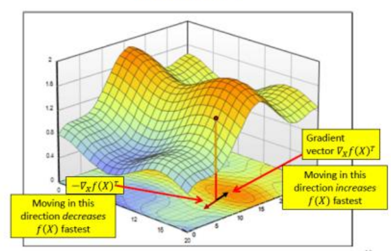
8

Gradient



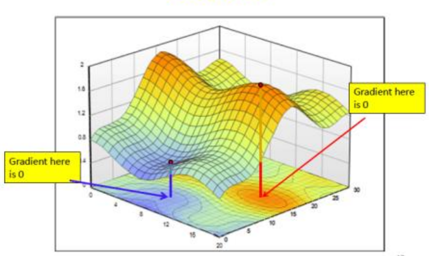
15

Gradient



16

Gradient



17

Gradient

- At any location X, there may be many directions in which we can step, such that f(X) increases
- The direction of the gradient is the direction in which the function increases fastest

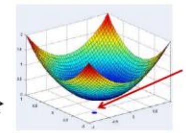
The Hessian

- The Hessian of a function $f(x_1, x_2, \dots, x_n)$ is given by the second derivative

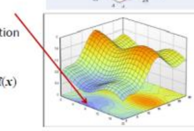
$$\nabla_x^2 f(x_1,\dots,x_n) := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdot & \cdot & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

19

The problem of optimization

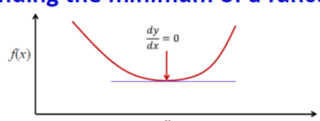


- General problem of optimization: Given a function $\mathbf{f}(x)$ of some variable x …
- Find the value of x where $\mathbf{f}(x)$ is minimum



22

Finding the minimum of a function



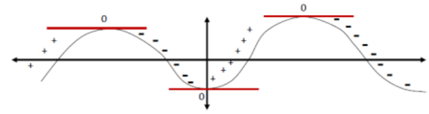
- Find the value x at which $f'(x) = 0$
  - Solve

$$\frac{df(x)}{dx} = 0$$

- The solution is a "turning point"
  - Derivatives go from positive to negative or vice versa at this point
- But is it a minimum?

23

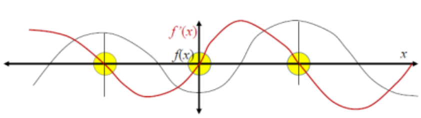
Turning Points



- Both *maxima* and *minima* have zero derivative
- Both are turning points

26

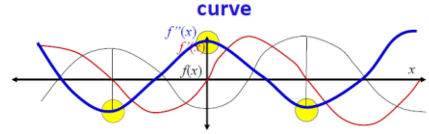
Derivatives of a curve



- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative

27

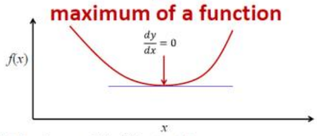
Derivative of the derivative of the curve



- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative
- The *second derivative* $f''(x)$ is –ve at maxima and +ve at minima!

28

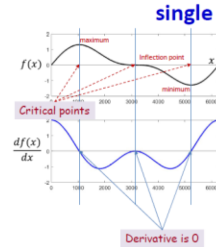
Solution: Finding the minimum or maximum of a function



- Find the value x at which $f'(x) = 0$: Solve
$$\frac{df(x)}{dx} = 0$$
- The solution x_{soln} is a ***turning point***
- Check the double derivative at x_{soln} : compute
$$f''(x_{soln}) = \frac{df'(x_{soln})}{dx}$$
- If $f''(x_{soln})$ is positive x_{soln} is a minimum, otherwise it is a maximum

29

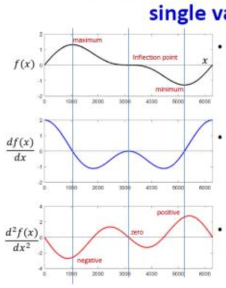
A note on derivatives of functions of single variable



- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points
- The *second* derivative is
  - Positive (or 0) at minima
  - Negative (or 0) at maxima
  - Zero at inflection points

30

A note on derivatives of functions of single variable



- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points
- The *second* derivative is
  - ≥ 0 at minima
  - ≤ 0 at maxima
  - Zero at inflection points
- It's a little more complicated for functions of multiple variables..

31

Unconstrained Minimization of function (Multivariate)

1. Solve for the X where the derivative (or gradient) equals to zero
$$\nabla_X f(X) = 0$$
2. Compute the Hessian Matrix $\nabla_X^2 f(X)$ at the candidate solution and verify that
   - Hessian is positive definite (eigenvalues positive) -> to identify local minima
   - Hessian is negative definite (eigenvalues negative) -> to identify local maxima

34

Unconstrained Minimization of function (Example)

- Minimize
$$f(x_1, x_2, x_3) = (x_1)^2 + x_1(1 - x_2) + (x_2)^2 - x_2x_3 + (x_3)^2 + x_3$$
- Gradient
$$\nabla_X f^T = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}$$

35

Unconstrained Minimization of function (Example)

- Set the gradient to null
$$\nabla_X f = 0 \Rightarrow \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$
- Solving the 3 equations system with 3 unknowns
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

36

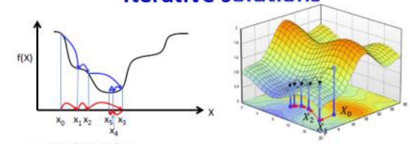
Unconstrained Minimization of function (Example)

- Compute the Hessian matrix $\nabla_X^2 f = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$
- Evaluate the eigenvalues of the Hessian matrix

  $\lambda_1 = 3.414,\ \lambda_2 = 0.586,\ \lambda_3 = 2$
- All the eigenvalues are positives => the Hessian matrix is positive definite
- The point $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$ is a minimum

37

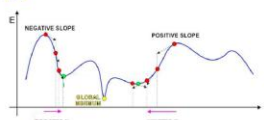
Iterative solutions



- Iterative solutions
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of f(X)
  - Stop when f(X) no longer decreases
- Problems:
  - Which direction to step in
  - How big must the steps be

39

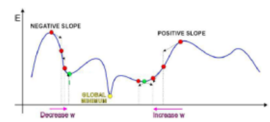
The Approach of Gradient Descent



- Iterative solution:
  - Start at some point
  - Find direction in which to shift this point to decrease error
    - This can be found from the derivative of the function
      - A negative derivative → moving right decreases error
      - A positive derivative → moving left decreases error
  - Shift point in this direction

40

The Approach of Gradient Descent



- Iterative solution: Trivial algorithm
  - Initialize x^0
  - While $f'(x^k) \neq 0$

    $x^{k+1} = x^k - \eta^k f'(x^k)$
- η^k is the "step size"

43

So far…

- Minimum of a function f(x) is when : f'(x) = 0 and the second derivative f''(x) is positive
- At any location X, there may be many directions in which we can step, such that f(X) increases
- The direction of the gradient is the direction in which the function increases fastest

Gradient descent/ascent (multivariate)

- The gradient descent/ascent method to find the minimum or maximum of a function f iteratively
  - To find a *maximum* move *in the direction of the gradient*

    $$x^{k+1} = x^k + \eta^k \nabla_x f(x^k)^T$$
  - To find a *minimum* move *exactly opposite the direction of the gradient*

    $$x^{k+1} = x^k - \eta^k \nabla_x f(x^k)^T$$
- Many solutions to choosing step size η^k

47

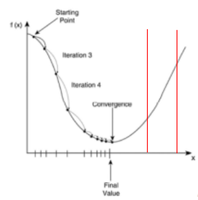
Gradient descent convergence criteria

- The gradient descent algorithm converges when one of the following criteria is satisfied

$$\boxed{\left|f(x^{k+1}) - f(x^k)\right| < \varepsilon_1}$$

- Or

$$\left\|\nabla_x f(x^k)\right\| < \varepsilon_2$$



48

Overall Gradient Descent Algorithm

- Initialize:
  - x^0
  - $k = 0$
- do
  - $x^{k+1} = x^k - \eta^k \nabla_x f(x^k)^T$
  - $k = k + 1$
- while $\left|f(x^{k+1}) - f(x^k)\right| > \varepsilon$

49

Preliminaries

Problem Statement

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- Minimize the following function

$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

w.r.t W

- This is problem of function minimization
  - An instance of optimization

54

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- Minimize the following function

$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

w.r.t W

57

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

58

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

What is f() and what are its parameters W?

Problem Setup: Things to define

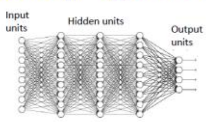
- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

What is the divergence div()?

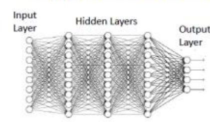
What is f() and what are its parameters W?

What is f()? Typical network



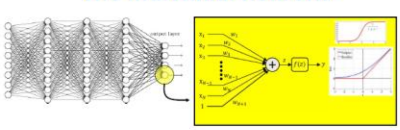
- Multi-layer perceptron
- A *directed* network with a set of inputs and outputs
  – No loops

Typical network



- We assume a "layered" network for simplicity
  – Each "layer" of neurons only gets inputs from the earlier layer(s) and outputs signals only to later layer(s)
  – We will refer to the inputs as the ***input layer***
    - No neurons here – the "layer" simply refers to inputs
  – We refer to the outputs as the ***output layer***
  – Intermediate layers are ***"hidden" layers***

The individual neurons



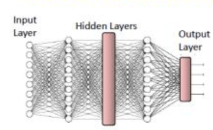
- Individual neurons operate on a set of inputs and produce a single output
  – **Standard setup:** A continuous activation function applied to an affine function of the inputs

$$y = f\left(\sum_i w_i x_i + b\right)$$

  – More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

Vector Activations

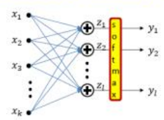


- We can also have neurons that have *multiple coupled* outputs

$$[y_1, y_2, \ldots, y_l] = f(x_1, x_2, \ldots, x_k; W)$$

  – Function $f()$ operates on set of inputs to produce set of outputs
  – Modifying a single parameter in W will affect *all* outputs

Vector activation example: Softmax



- Example: Softmax *vector* activation

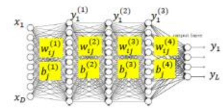
$$z_i = \sum_j w_{ji} x_j + b_i$$

Parameters are weights w_{ji} and bias b_i

$$y = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

68

Notation



- The input layer is the 0^{th} layer
- We will represent the output of the i-th perceptron of the k^{th} layer as $y_i^{(k)}$
  - **Input to network:** $y_i^{(0)} = x_i$
  - **Output of network:** $y_i = y_i^{(N)}$
- We will represent the weight of the connection between the i-th unit of the k-1th layer and the jth unit of the k-th layer as $w_{ij}^{(k)}$
  - The bias to the jth unit of the k-th layer is $b_j^{(k)}$

71

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- Minimize the following function

$$Loss(W) = \frac{1}{T} \sum_i div(f(X_i; W), d_i)$$

What is f() and what are its parameters W?

72

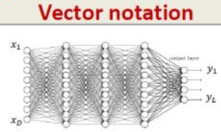
Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

$$Loss(W) = \frac{1}{T} \sum_i div(f(X_i; W), d_i)$$

73

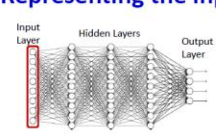
Input, target output, and actual output: Vector notation



- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$
- $X_n = [x_{n1}, x_{n2}, \ldots, x_{nD}]^\top$ is the nth input vector
- $d_n = [d_{n1}, d_{n2}, \ldots, d_{nL}]^\top$ is the nth desired output
- $Y_n = [y_{n1}, y_{n2}, \ldots, y_{nL}]^\top$ is the nth vector of *actual* outputs of the network
  - Function of input X_n and network parameters
- We will sometimes drop the first subscript when referring to a *specific* instance

74

Representing the input



- Vectors of numbers
  - (or may even be just a scalar, if input layer is of size 1)
  - E.g. vector of pixel values
  - E.g. vector of speech features
  - E.g. real-valued vector representing text
  - Other real valued vectors

75

Multi-class output: One-hot representations

- Consider a network that must distinguish if an input is a cat, a dog, a camel, a hat, or a flower
- We can represent this set as the following vector, with the classes arranged in a chosen order:

  [cat dog camel hat flower]^T
- For inputs of each of the five classes the desired output is:

  cat: $[1\,0\,0\,0\,0]^T$

  dog: $[0\,1\,0\,0\,0]^T$

  camel: $[0\,0\,1\,0\,0]^T$

  hat: $[0\,0\,0\,1\,0]^T$

  flower: $[0\,0\,0\,0\,1]^T$
- For an input of any class, we will have a five-dimensional vector output with four zeros and a single 1 at the position of that class
- This is a *one hot vector*

80

Problem Setup: Things to define

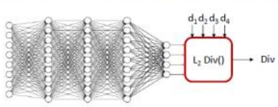
- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- Minimize the following function

$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

What is the divergence div()?

Note: For Loss(W) to be differentiable w.r.t W, div() must be differentiable

Examples of divergence functions



- For real-valued output vectors, the (scaled) L_2 divergence is popular

$$Div(Y, d) = \frac{1}{2}\|Y - d\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$$

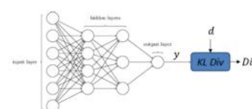
  - Squared Euclidean distance between true and desired output
  - Note: this is differentiable

$$\frac{dDiv(Y, d)}{dy_i} = (y_i - d_i)$$

$$\nabla_Y Div(Y, d) = [y_1 - d_1, y_2 - d_2, \ldots]$$

88

For binary classifier



- For binary classifier with scalar output, $Y \in (0,1)$, d is 0/1, the Kullback Leibler (KL) divergence between the probability distribution $[Y, 1-Y]$ and the ideal output probability $[d, 1-d]$ is popular

$$Div(Y, d) = -d\log Y - (1-d)\log(1-Y)$$

  - Minimum when $d = Y$
- Derivative

$$\frac{dDiv(Y, d)}{dY} = \begin{cases} -\frac{1}{Y} & if\ d = 1 \\ \frac{1}{1-Y} & if\ d = 0 \end{cases}$$

89

Summary

- Neural nets are universal approximators
- Neural networks are trained to approximate functions by adjusting their parameters to minimize the average divergence between their actual output and the desired output at a set of "training instances"
  - Input-output samples from the function to be learned
  - The average divergence is the "Loss" to be minimized
- To train them, several terms must be defined
  - The network itself
  - The manner in which inputs are represented as numbers
  - The manner in which outputs are represented as numbers
    - As numeric vectors for real predictions
    - As one-hot vectors for classification functions
  - The divergence function that computes the error between actual and desired outputs
    - L2 divergence for real-valued predictions
    - KL divergence for classifiers