Introduction to Machine Learning

Lecture 9

Instructor:
Dr. Tom Arodz

Recap: the story so far…

- Ideal way of making predictions:
  - Perfect feature
  - Accurate, efficient simulation
  - Fully-known probability distribution $p(y|x)$ from which data comes from
    - or $p(x|y)$, we can use Bayes theorem to get $p(y|x)$ from it
- Traditional machine learning:
  - heuristics / ad-hoc approaches (e.g. perceptron in HW1)
  - maximum likelihood (or Bayesian learning)
    - pick a distribution shape for $p(x|y)$, based on domain knowledge
    - fit best distribution $p(x|y)$ to each class in training data
    - get $p(y|x)$ from these distributions using Bayes theorem
- “Modern” machine learning:
  - produce numbers that we can treat as $p(y|x)$ from some function `f(x)` – not any specific distribution
  - Tweak the parameters `θ` of function `f(x;θ)` to make predictions better

Towards ML Approach #3

- "Modern" machine learning: probability distrib. governing the problem not known, let's model it using a class of function
  - Predicting: use trained parameters θ_{opt}
    - p(class|x,S)=`some_f(`θ_{opt}`)(x).to_class_probs()`
  - Training:
    - Search for single value of parameters θ that minimizes:

  θ_{opt} = arg min$_\theta$ `error_metric(`
  `S, some_f(θ)(x in S).to_class_probs()`
  `)`

- Compare with maximum likelihood (MLE) :
  - Predicting: use trained parameters θ_{opt}
    - p(class|x,S)=`mult_normal(`θ_{class}`).pdf(x)*P[class]`
  - Training:
    - θ_{class}=arg max$_\theta$ `mult_normal(θ).pdf(`S_{class}`)`

Recap: ML Approach #3

- "Modern" machine learning:

```
minimize_θ: error_metric(S,
        some_f(θ)(x in S).to_class_probs())
```

- Filling the gaps:
  - `some_f(θ)(x)` or `some_f(θ;x)`
    - Some function that takes two groups of inputs
      - Raw input features x
      - Trainable parameters θ
        - the parameters help us pick a specific function from a family of functions
  - `to_class_probs()`
    - `some_f(θ;x)` may not return probabilities, how to convert to: ≥0, sum=1
  - `error_metric`
    - Classification error? Do we need something more complex?
  - `minimize_θ`
    - how? For Gaussians in MLE, we had analytical solution, that's unlikely for `some_f`

9.1. Iterative minimization

Building ML Approach #3

- Modern machine learning:

```
minimizeθ: error_metric(
  S, some_f(θ)(x in S).to_class_probs())
```

- Minimization:

  - minimize$_\theta$
    - how?

Building ML Approach #3

- Modern machine learning:

```
minimizeθ: error_metric(
   S, some_f(θ)(x in S).to_class_probs())
```

- Let's simplify the notation:

  - minimize$_\theta$: $\boldsymbol{\Sigma}_{(x,y)\in S}$ error_metric(
    ```
                 y,
                 some_f(θ,x).to_class_probs()
             )
    ```

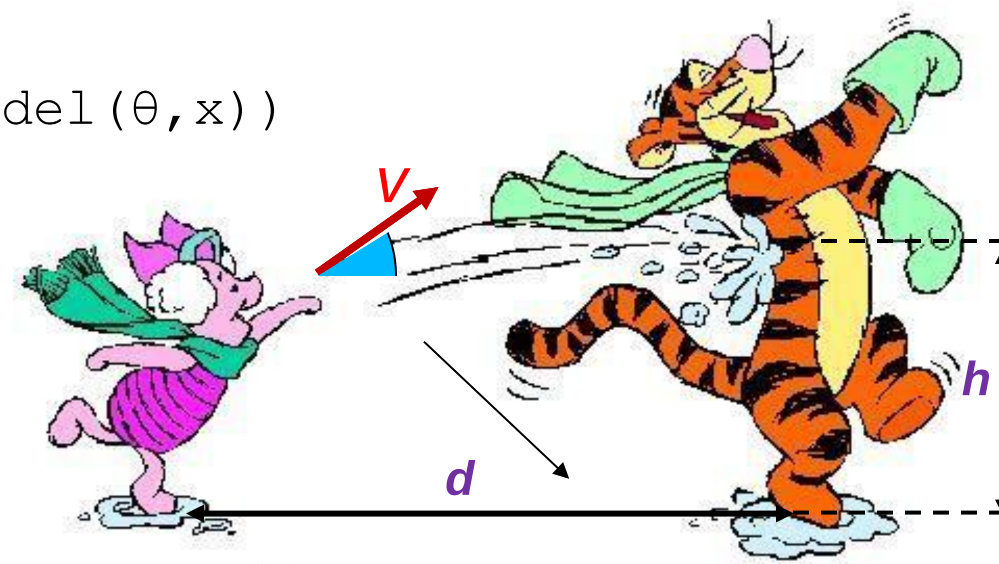
  - minimize$_\theta$: $\boldsymbol{\Sigma}_{(x,y)\in S}$ L(y,model(θ,x))

Building ML Approach #3

- Modern machine learning:
  - `minimize`$_\theta$`:`
    `Σ`$_{(x,y)\in S}$ `error_metric(`
    `y,`
    `some_f(θ,x).to_class_probs()`
    `)`
- Let's assume that `error_metric`, `some_f`, and `to_class_probs` are smooth functions: they have derivatives
(as opposed to e.g. |x| - abs. value of x does not have derivative a x=0)
- That means: `L` and `model` are smooth functions, too (have derivatives)
  - `minimize`$_\theta$`:` `Σ`$_{(x,y)\in S}$ `L(y,model(θ,x))`
    - Note that **Σ** (sum) is also a smooth function
- Problems to solve: find minimum of a differentiable function
  - `minimize`$_\theta$`: function(θ)`

Building ML Approach #3

- Training:
  - `minimize`$_\theta$`: function(θ)`
    - More specifically:
  - `minimize`$_\theta$`:` $\mathbf{\Sigma}_{(x,y)\in S}$ `L(y,model(θ,x))`

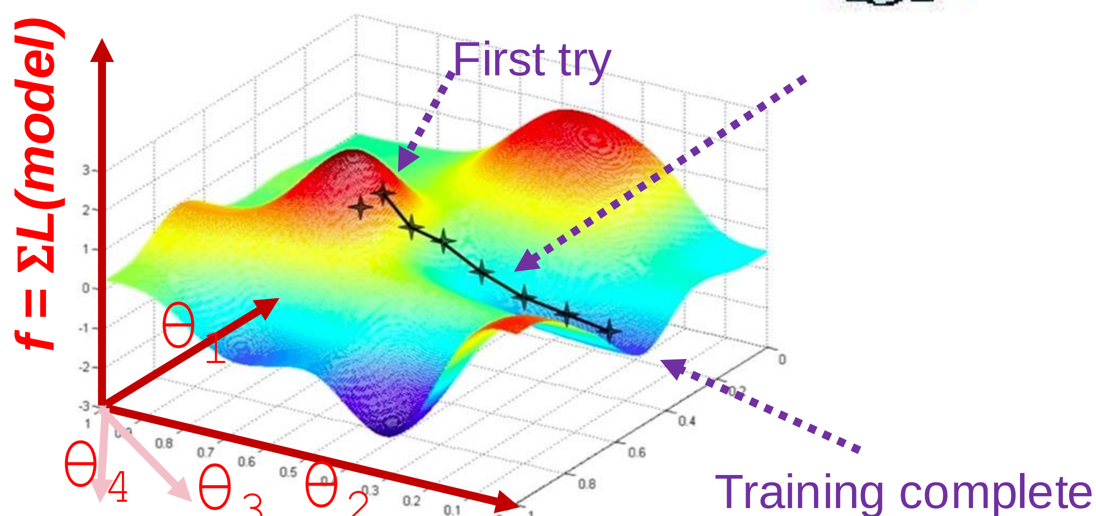


- Like Perceptron / HW1

  but

  we assumed differentiability of **f**

  it will help us figure out in which direction we should tweak θ



Optimization

- Training:
  - $\text{minimize}_\theta\text{: function}(\theta)$

- We will change notation to just minimize:
  - $\text{minimize}_x\text{: } f(x)$

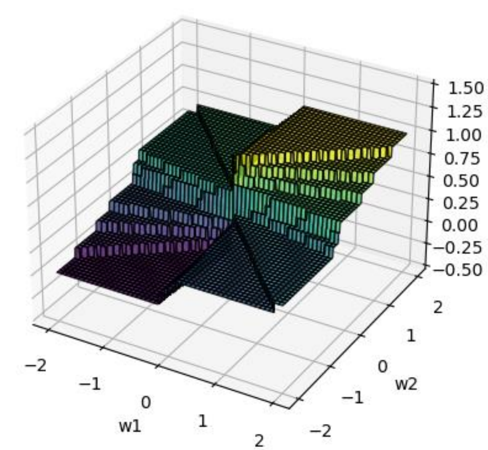
- For a moment:
  - x will not mean "feature values" of our training set
  - instead, it will mean the parameters we want to find to minimize the value of f()
  - the training data will be "folded in" (hidden inside) the shape of the function f()
    - that means: f(x) is very complex, its shape depends on training set, and difficult to view/analyze "holistically"

Optimization

- Task
  - `minimize`$_x$`: f(x)`
- Possible ways to find minimum of a function:
  - Analytical: solve for "derivative of f = 0"
    - E.g. $\min_x x^2-4x$ is at $d(x^2-4x)/dx = 2x-4=0$, i.e. at $x=2$
      - We used this approach in Maximum Likelihood Estimation for Gaussians
    - Cons:
      - in ML, f is highly complex, finding the equation to solve and solving it will be computationally hard
      - in ML, f has many points with df/dx=0: maxima, saddle points, many minima

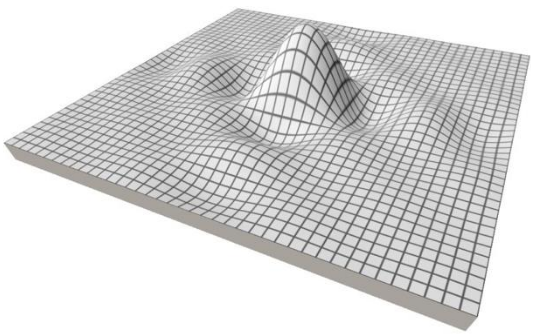
Optimization

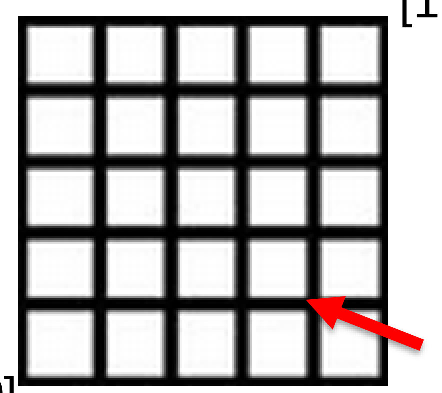
- Task
  - `minimize`$_x$`: f(x)`
- Possible ways to find minimum of a function:
  - Analytical won't work, ultimately because:
    - f(x) is very complex, its shape depends on training set, and difficult to view/analyze "holistically"
  - What is "cheap" is calculating value of the function f(x) for specific x. So, one alternative to "analytical" is:
  - Probing: like constructing the plots in HW1 with meshgrid
    - try f(x) at a large number of possible x
    - pick best, i.e., lowest value of f(x))



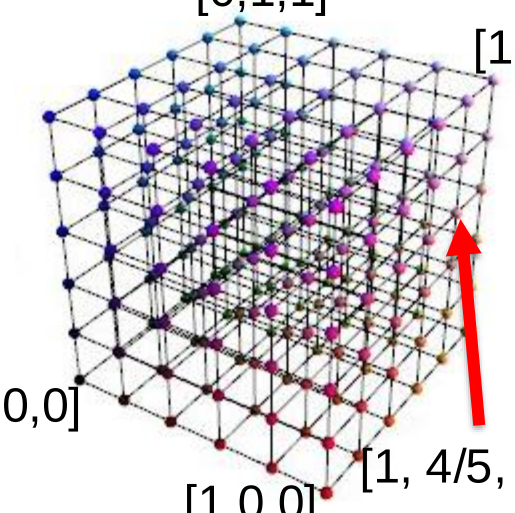
Optimization

- Task
  - `minimize`$_x$`: f(x)`
- Possible ways to find minimum of a function:
  - Probing: like plots in HW1
    - try f(x) at a large number of possible x
    - pick best, i.e., lowest value of f(x))





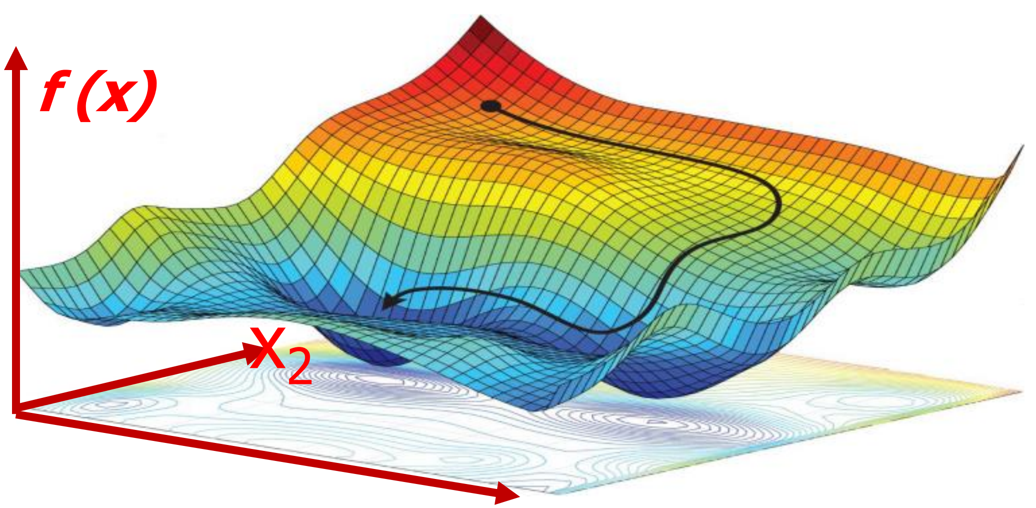
- p=5,
- If n=2, we need 6^2=36 points in the grid
- If n=3, we need 6^3=216 points in the grid
- each cell has side length of 1/5



- Won't work, to many points to try!

Optimization

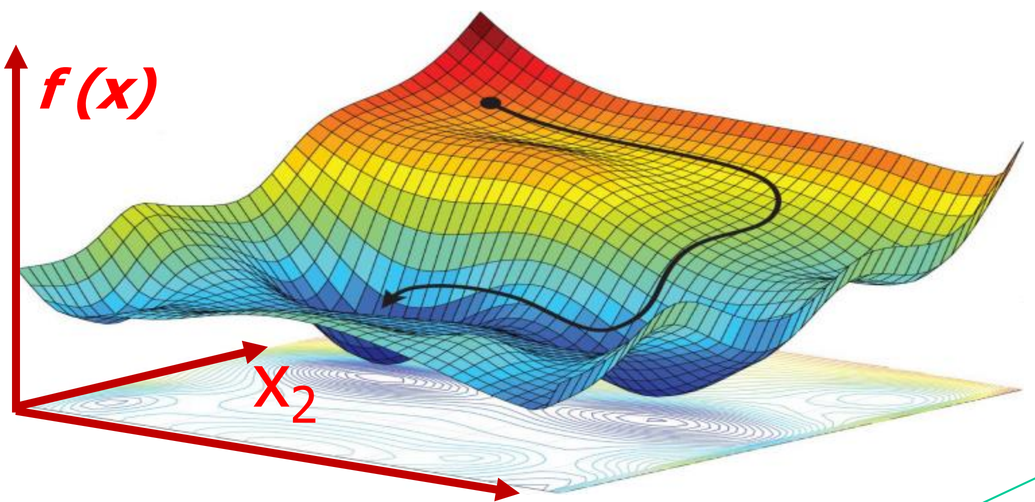
- Task
  - `minimize`$_x$`: f(x)`
- Possible ways to find minimum of a function:
  - Iterative minimization
    - Like training in HW1



    - Relies on "local" rules to find how to update in position (x here, w in HW1)

Optimization

- Task
  - $\text{minimize}_x\text{: } f(x)$
- Possible ways to find minimum of a function:
  - Iterative minimization



- Relies on "local" rules to find how to update in position
  - Heuristic rules like in HW1
  - Rules based on value of f(x) for neighboring x
  - **Rules based on derivatives (gradient) of f at x**

ML overwhelmingly uses this approach: "gradient descent"

Gradient descent

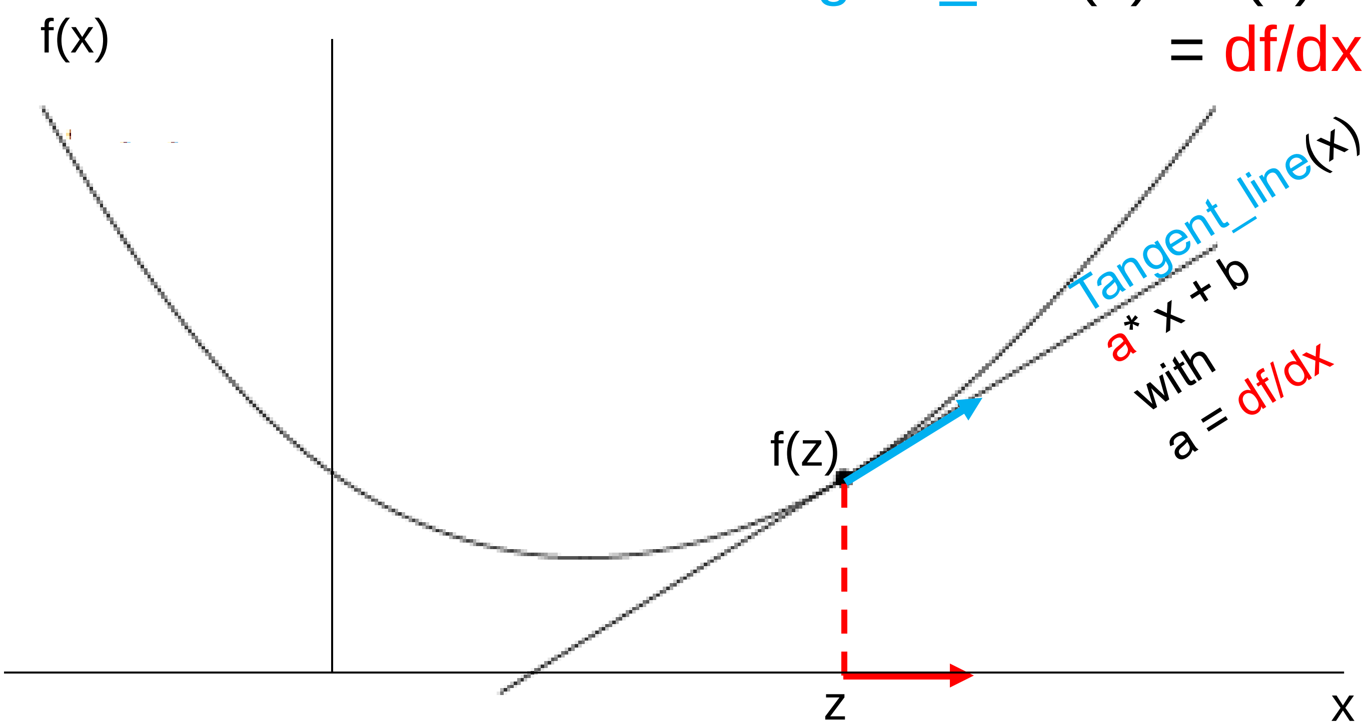
For 1D, $\nabla f = df/dx = f'$

First derivative or gradient: denoted ∇f or f'

- gradient of a n-dimensional function $f: R^n \rightarrow R$ is a generalization of derivative of a single-dimensional function $f: R \rightarrow R$

Tangent_line(x) = f(z) + df/dx * (x-z)

= df/dx * x + [f(z)-df/dx *z]



Gradient descent

gradient of a n-dimensional function $f:R^n \rightarrow R$ is:

an n-dimensional **vector** of partial derivatives,
pointing (for a given z) in direction of steepest slope of f at that z

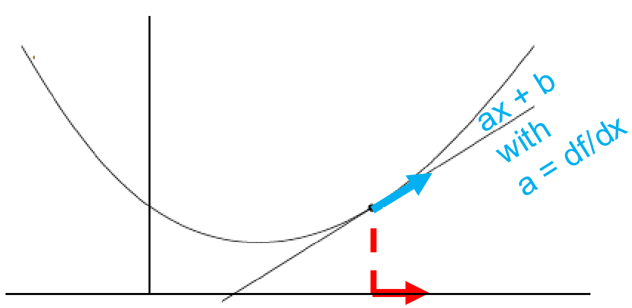
Pointing always in the direction in which the function grows (locally)

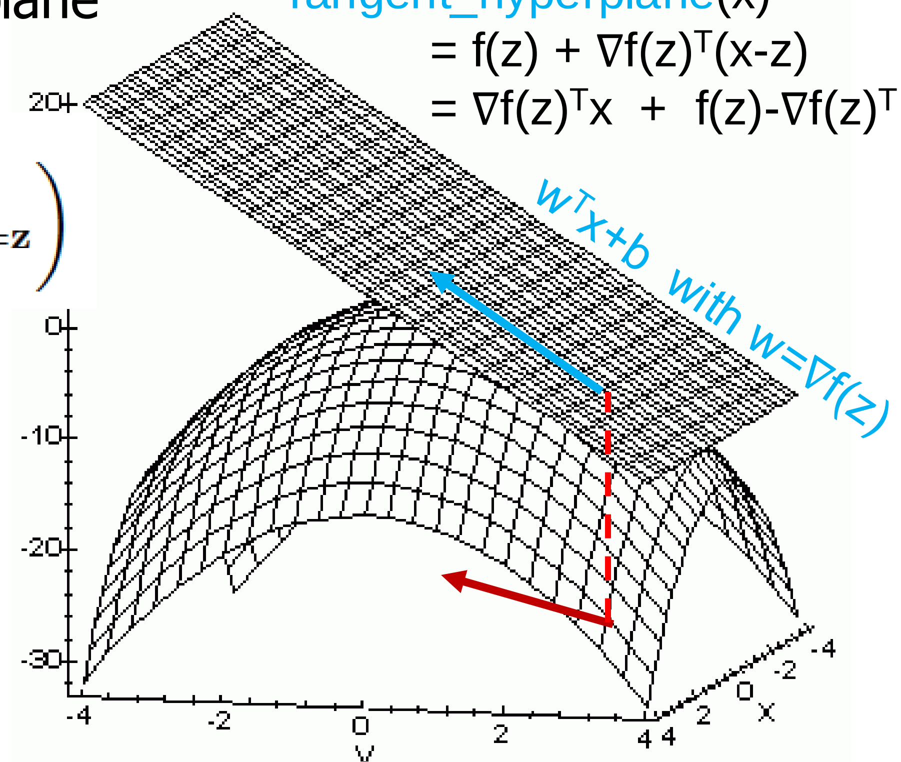
positive value in a vector = f grows (vector points) towards $+\infty$; negative value = f grows towards $-\infty$, vector points towards $-\infty$

Defines a line or plane or hyperplane tangent to f

Tangent_hyperplane(x)
$= f(z) + \nabla f(z)^T(x-z)$
$= \nabla f(z)^T x + f(z) - \nabla f(z)^T z$

$$\nabla f(\mathbf{z}) = \left(\frac{\partial}{\partial x_1} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}}, \ldots, \frac{\partial}{\partial x_n} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}} \right)$$





Gradient descent

Gradient descent (with learning rate *c*):

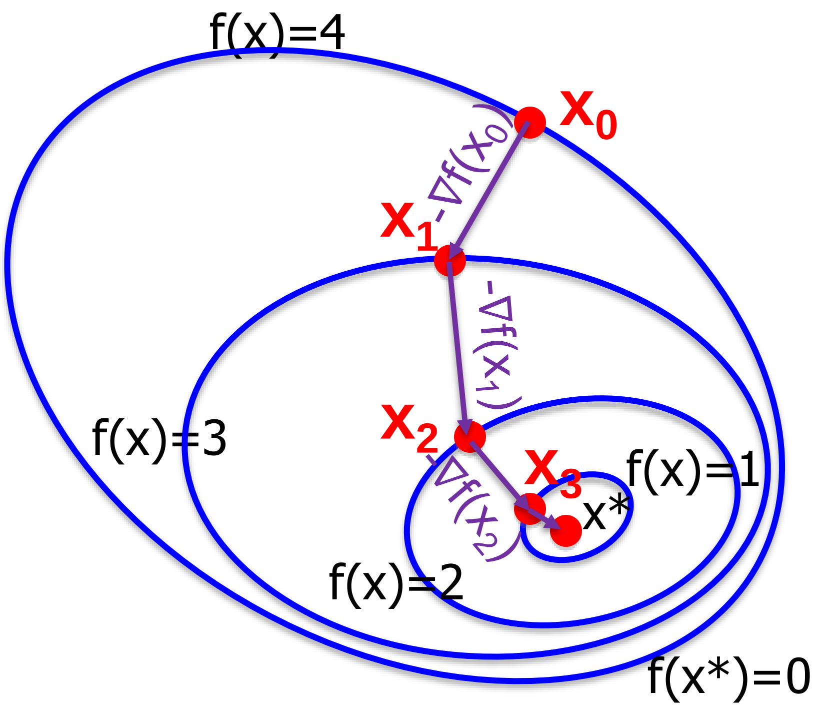
We start from x_0

We calculate $x_1 = x_0 - c\, \nabla f(x_0)$

We calculate $x_2 = x_1 - c\, \nabla f(x_1)$

$\mathbf{x_{n+1} = x_n - c\, \nabla f(x_n)}$

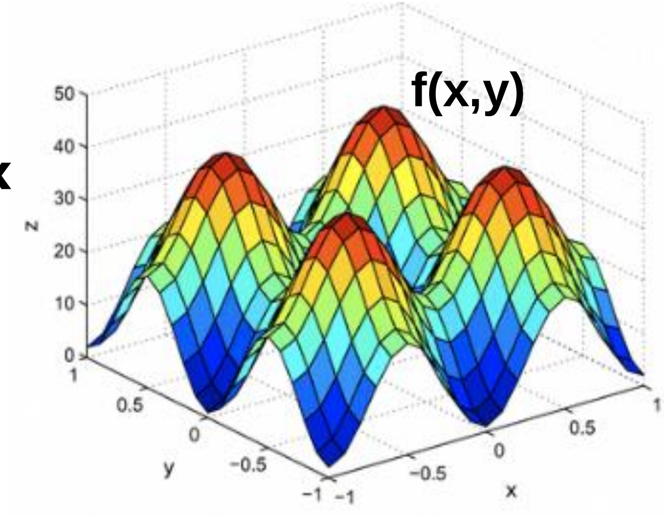
If we choose L large enough
grad.desc. goes down in each step,
converging towards
local minimum of function f



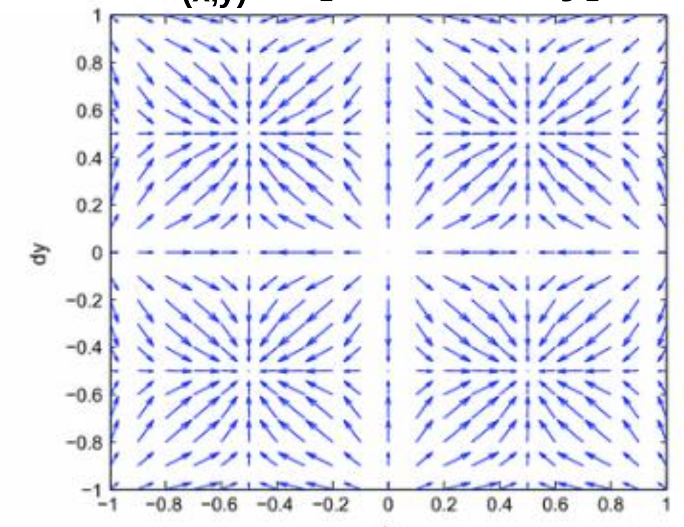
Recap: Gradient descent

Gradient of f(x) at z:

vector of partial derivatives of **f** w.r.t. **x** at the current point **z** pointing towards the direction of steepest increase of function locally at point z



$\nabla_{(x,y)} f = [\partial f / \partial x, \partial f / \partial y]$



Gradient descent (with learning rate *c*):

We start from x_0

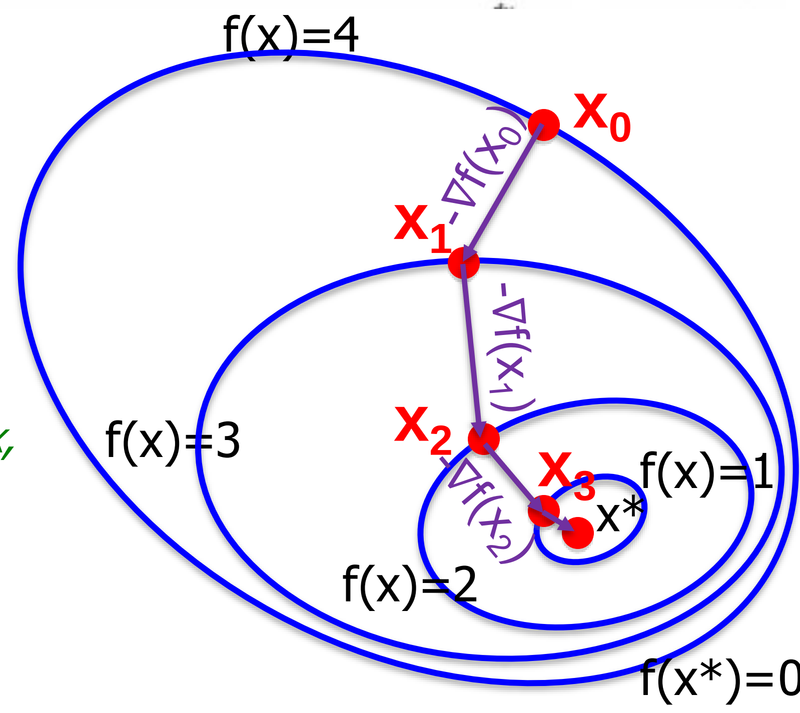
We calculate $x_1 = x_0 - c\, \nabla f(x_0)$

We calculate $x_2 = x_1 - c\, \nabla f(x_1)$

$\mathbf{x_{n+1} = x_n - c\, \nabla f(x_n)}$

Note: we don't need full formula of ∇f as function of x, just the code to get its value for specific x_0 x_1 …

In ML, we apply grad. desc. to find minimum of a function of parameters θ, (e.g. θ=(w,b)), instead of function of x



Recap

- Modern machine learning:

```
minimize_θ: error_metric(
   S, some_f(θ)(x in S).to_class_probs())
```

- Filling the gaps:
  - `some_f(θ)(x)` or `some_f(θ;x)`
    - Some function (a "model") that takes two groups of inputs
      - Raw input features x
      - Trainable parameters θ
  - `to_class_probs()`
    - `some_f(θ;x)` may not return probabilities, how to convert to: ≥0, sum=1
  - `error_metric`
    - Classification error? Do we need something more complex?
  - `minimize_θ`
    - **How?**
      Gradient descent, over parameters θ of the model `some_f(θ;x)`
    - $\boldsymbol{\theta_{n+1} = \theta_n - c\, \nabla_\theta\, \text{error\_metric}(S, \text{model}(\theta_n, S))}$

9.2. Correct/incorrect as the *error metric*

Building ML Approach #3

`minimize`$_\theta$`: error_metric(S, model(θ, S))`

- Assume "model" return raw predictions from some_f, or predictions converted to class probabilities

- Filling the gaps:
  - `error_metric`
    - Classification error (error rate or error count) is the most natural metric of classifier performance
      - error_rate(training_set S) = error_count(training_set S) / |S|
      - error_count(training_set S) = $\mathbf{\Sigma}_{(x,y)\in S}$ is_incorrect(y, predicted_y(x))
        - is_incorrect returns 1 (incorrect) or 0 (correct)

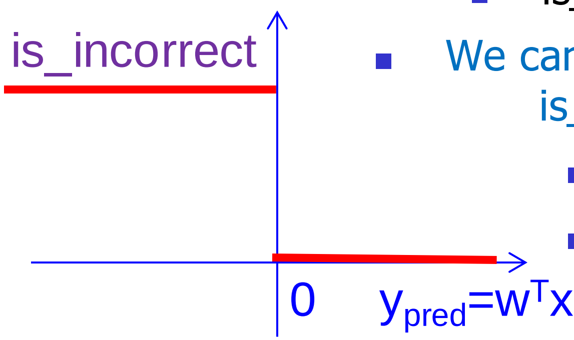
    - With gradient descent as the minimization method, the update of the parameters θ would be (c=learning rate):
      - `new θ = old θ - c` ∇_θerror_count(`S, model(θ, S)`)

        or, equivalently
      - `new θ = old θ - c` $\mathbf{\Sigma}_{(x,y)\in S}$ ∇_θis_incorrect (`y,model(θ,x)`)

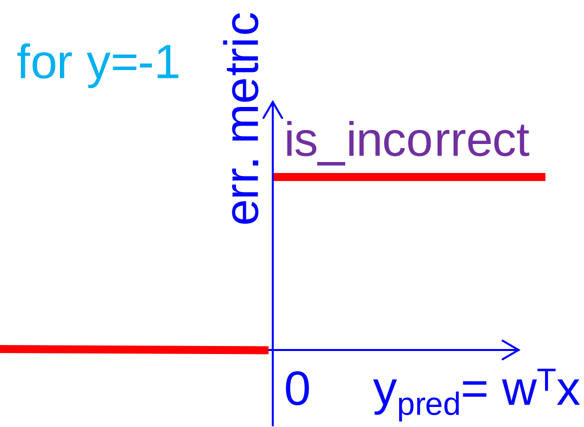
Choosing the error metric

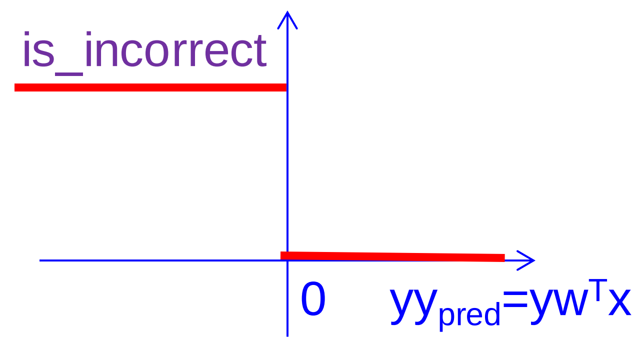
- error_metric
  - Classification error (error rate or error count) is the most natural metric of classifier performance
    - error_rate(training_set S) = error_count(training_set S) / |S|
    - error_count(training_set S) = $\Sigma_{(x,y) \in S}$ is_incorrect(y, predicted_y(x))
      - is_incorrect returns 1 (incorrect) or 0 (correct)
    - We can also use:
      is_incorrect(y, predicted_y(x)) = is_negative(y * predicted_y(x))
      - is_negative(u) returns 1 for u<=0, and returns 0 otherwise
      - can be codes as sign(1+sign(-1 * y * predicted_y(x)))

for y=+1



for y=-1





when using the two-part term
$y_k f(w;x_k)= y_{true}*y_{pred}$
there's just one common plot

Choosing the error metric

```
minimize_θ: error_metric(
  S, some_f(θ)(x in S).to_class_probs())
```

- Filling the gaps:
  - `minimize_θ`
    - how? Gradient descent!
  - `error_metric`
    - Classification error is the most natural metric of classifier performance

How to use it
with gradient descent?

∇error_count = ?

∇is_incorrect = ?

∇_wis_negative(yw^Tx) = ?

Choosing the error metric

```
minimizeθ: error_metric(
  S, some_f(θ)(x in S).to_class_probs())
```

- Filling the gaps:
  - `minimize`$_\theta$
    - how? Gradient descent!
  - `error_metric`
    - Classification error is the most natural metric of classifier performance

∇error_count = 0

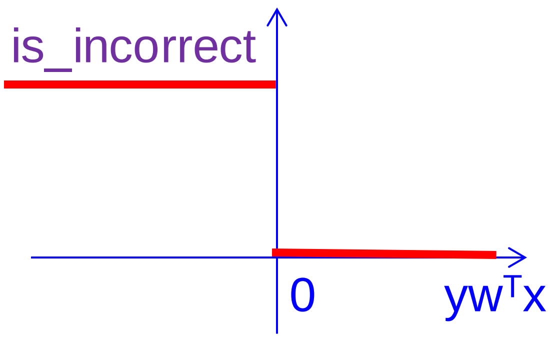
∇is_incorrect = 0

∇_wis_negative(yw^Tx) = 0

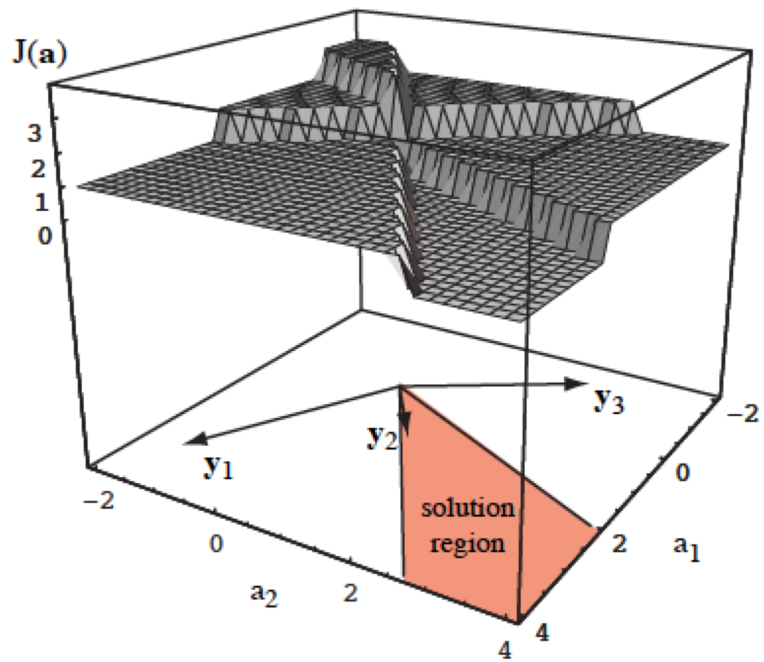
Flat everywhere (∇=0) except when is_incorrect changes from 1 to 0, error_count drops by 1 (sudden drops: derivative not defined)

Choosing the error metric

- minimize$_\theta$
  - how? Gradient descent!
- error_metric
  - Classification error is the most natural metric of classifier performance



`0 if yh(x)>0,`
`1 otherwise`



∇error_count

∇is_incorrect

Non-differentiable

Not continuous

Flat!

- We don't know in which direction to move to get a (even slightly) better solution!

Choosing the error metric

```
minimizeθ: error_metric(
  S, some_f(θ)(x in S).to_class_probs())
```

- Filling the gaps:
  - `error_metric`
    - Classification error is the most natural metric of classifier performance
    - But will not work with gradient-based (or other local) optimization approaches
    - We will need a different error metric
      - One that has no flat regions!

9.3. Mean squared error

Towards ML Approach #3

`minimize`$_\theta$`: error_metric(S, model(`θ`, S))`

- Filling the gaps:
  - `minimize`$_\theta$
    - how? Gradient descent!
  - `error metric`
    - **Mean square error** is a well-known error metric:
      - MSE(training_set S) = $\Sigma_{(x,y)\in S}$ (y – predicted_y(x))^2 / |S|
        - We assume that true y is +1 or -1,
    - With gradient descent as the minimization method, the update of the parameters θ would be (c=learning rate):
      - `new` θ `= old` θ `– c` ∇_θ `MSE(S, model(`θ`, S))`

        or, equivalently
      - `new` θ `= old` θ `– c/|S|` $\Sigma_{(x,y)\in S}$ ∇_θ`(y – model(`θ`,x))`^{2}

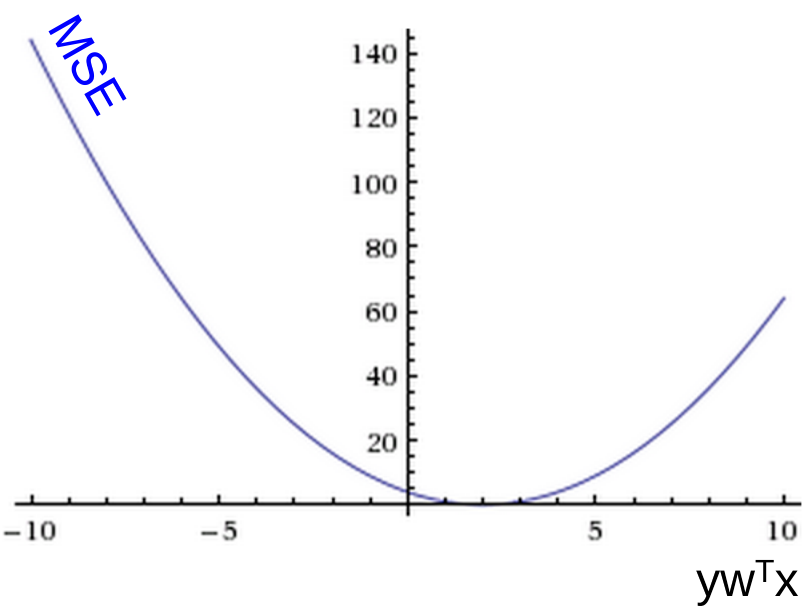
Is MSE a good choice for classification?

- MSE for classification:

multiply by $y^2=1$

$$loss(w,x,y) = (y-w^Tx)^2 = (1-yw^Tx)^2$$

- MSE is:
  - Differentiable (gradient exists everywhere)
  - Not flat (we will get non-zero gradient)
  - Nonnegative



Is MSE a good choice for classification?

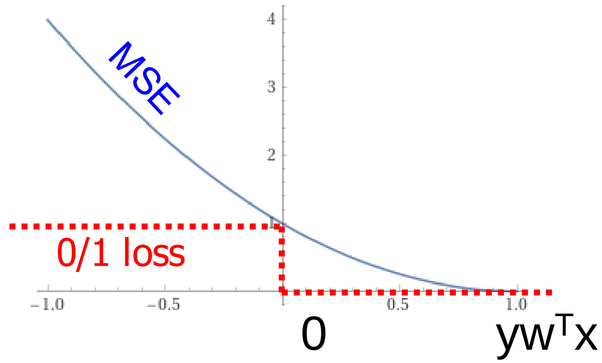
- MSE for classification:

multiply by $y^2=1$

$$loss(w,x,y) = (y-w^Tx)^2 = (1-yw^Tx)^2$$

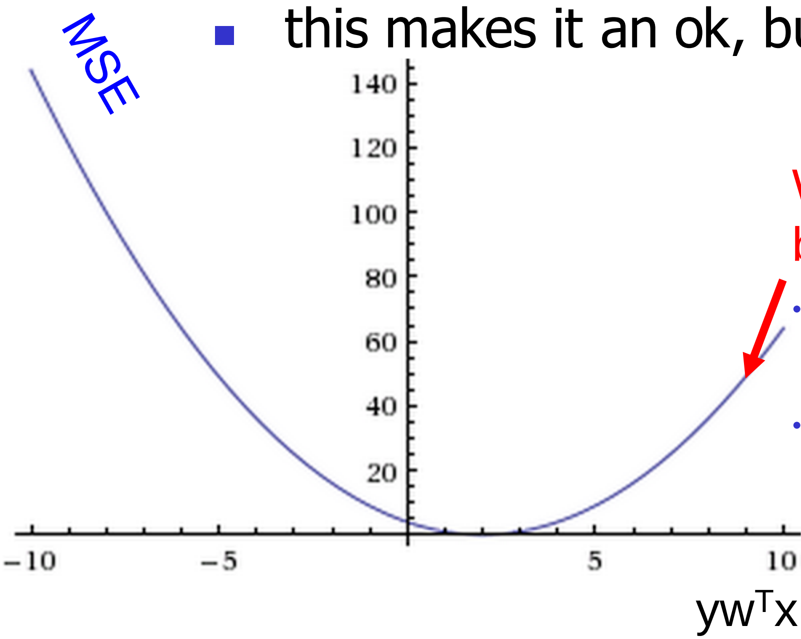
- MSE:
  - Has some relationship to 0/1 loss (is-incorrect)
    - If MSE =0 then 0/1_loss = 0
    - Upper-bounds *0/1 loss*:

MSE (w,x,y) >= 0/1_ loss(w,x,y)



Problem with MSE for classification

- MSE is:
  - Not flat
  - Upper-bounds 0/1 loss
  - Non-monotonic:
    - decreases, then increases, as yw^Tx increases
      - E.g. for class +1, prediction +4 (correct)
        has same MSE as prediction -2 (incorrect)
    - this makes it an ok, but not the best choice for classification



We get large loss (penalty) not only for incorrect, but for correct predictions too!

- If data is Gaussian, very few points should be that far from decision boundary.
- But what if data is not Gaussian? BAD!
  - non-monotonic loss is suspicious for classification (but ok for regression)

Updated schedule

- Test 1 will be on Wednesday, 10/9
  - Test will be done via Canvas, online (from home)

- Study problems will be posted in Canvas on Friday, 9/27
- We will discuss the study problems during our Monday, 10/7 class

- HW2 is due next Tuesday, 10/1, at 5pm