Machine Learning

Multiclass classification and Learning as Optimization

Dan Goldwasser

dgoldwas@purdue.edu

Multi-Categorical Output Tasks

- So far, our discussion was limited to ***binary predictions***
  - Well, ***almost*** (?)
- What happens if our decision is not over binary labels?
  - *Many interesting classification problems are not!*
  - **Credit card**: Approved, Deny, Further investigation needed
  - **Document classification**: sports, finance, politics
  - **OCR**: *0,1,2,3..9,A,..,Z*

How can we approach these problems?

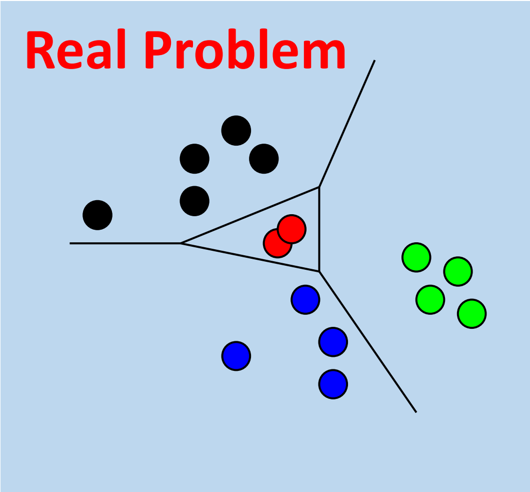
- *We will look into different reductions to binary classification problems!*

Hint: What is the computer science solution to: *"I can solve problem A, but now I have problem B, so…"*

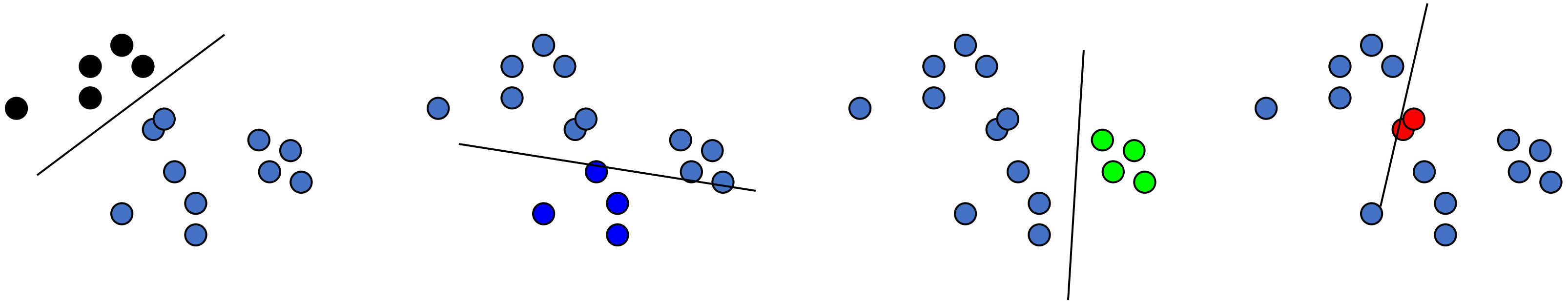
Learning via One-Versus-All

- Find $v_r, v_b, v_g, v_y \in \mathbf{R}^n$ such that
  - $v_r x > 0$ iff y = red ⊗
  - $v_b x > 0$ iff y = blue √
  - $v_g x > 0$ iff y = green √
  - $v_y x > 0$ iff y = yellow √

$\mathbf{H} = \mathbf{R}^{kn}$



- *Classification*: $f(x) = \text{argmax}_i \, (v_i x)$

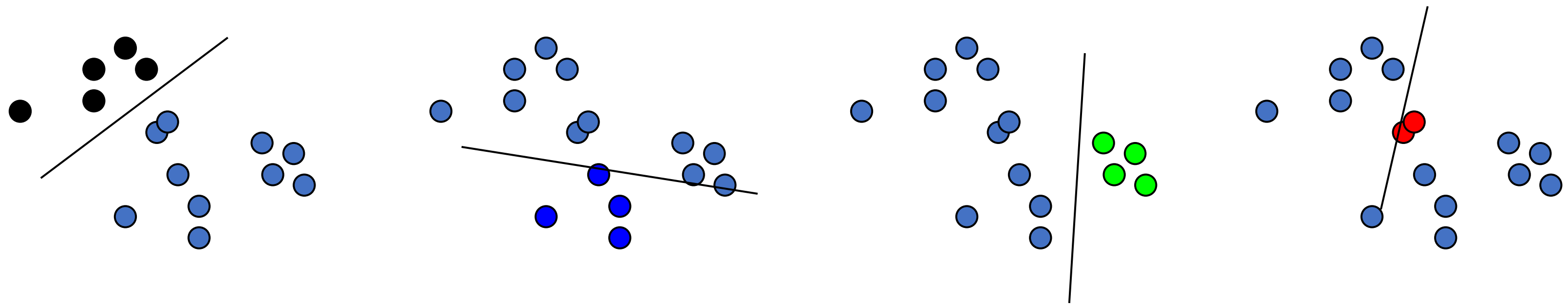


Problems with Decompositions

- **Learning optimizes over *local* metrics**
  - Poor *global* performance
  - What is the metric?
    - We don't care about *local* classifiers performance
  - ***Poor decomposition*** $\Rightarrow$ ***poor performance***
  - Especially true for Error Correcting Output Codes
- **Difficulty:** *how to ensure that the resulting problems are separable.*
- Can we ensure separability and learn the real objective?
- Real objective: final performance, rather than local metric

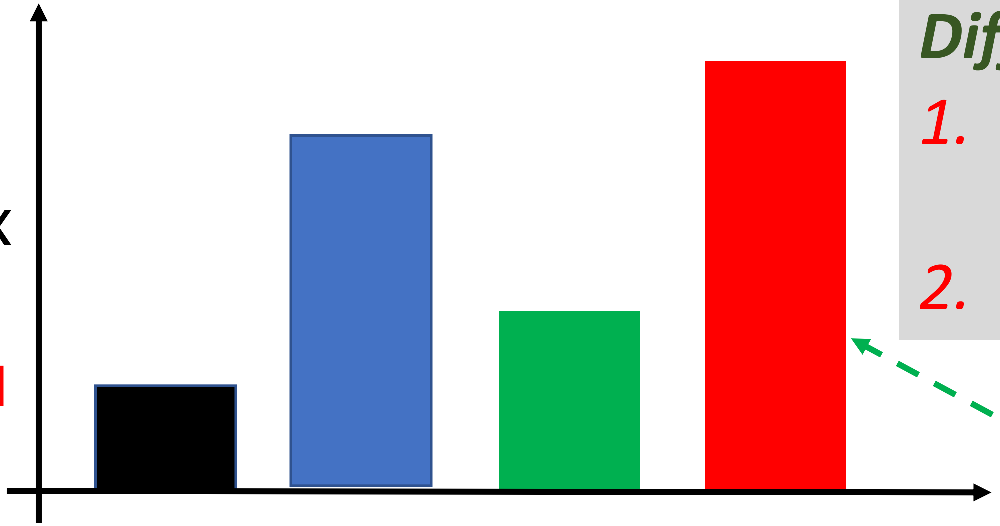
Intuition

- One vs. All



- *We can think about the problem in a different way*

Score =
$F_L(x) = W_L x$



Each W_L is defined over different features

Difference compared to 1vs.All

1. *Define the learning problem for **each label** over different feature space.*
2. *"Global" training objective*

Highest score = predicted label

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into $w = (w_1, w_2, \ldots w_k) \in R^{nk}$
- **Key Construction**: (**Kesler Construction**)
- Represent each example (x,y), as an nk-dimensional vector, x_y with x embedded in the y-th part of it (y=1,2,…k) and the other coordinates are 0

E.g., $\mathbf{x}_y = (\mathbf{0},x,\mathbf{0},\mathbf{0}) \in \mathbf{R}^{kn}$ *(here k=4, y=2)*

Example: *predict sentiment of a product review*

Features *– words (unigram)*
Labels*: Good, Bad, Neutral.*

The feature corresponding to "sick" when the label is "good"

$$\underbrace{\text{(["sick",good], …,}}_{\text{Good label features}}\ \underbrace{\text{["sick",neutral], …,}}_{\text{Neutral label features}}\ \underbrace{\text{["sick",bad],..)}}_{\text{bad label features}}$$

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into $w = (w_1, w_2, \ldots w_k) \in R^{nk}$
- **Key Construction:** (**Kesler Construction**)
- Represent each example (x,y), as an nk-dimensional vector, x_y with x embedded in the y-th part of it ($y=1,2,\ldots k$) and the other coordinates are 0

E.g., $\mathbf{x}_y = (\mathbf{0},x,\mathbf{0},\mathbf{0}) \in \mathbf{R}^{kn}$ (*here k=4, y=2*)

- Now we can understand the decision
  - Predict y iff $\forall y' \in \{1,\ldots k\}, y!=y$ $(w_y^T - w_{y'}^T)\ x > 0$

Conclusion: The set $(x_{yy'}, +) \equiv (x_y - x_{y'}, +)$ is ***linearly separable*** from the set $(x_{yy'}, -)$ using the linear separator $w \in R^{kn}$,

Learning via Kesler Construction

- A **perceptron update** rule applied in the nk-dimensional space due to a mistake in $w^T x_{ij} > 0$
- Implies the following update:

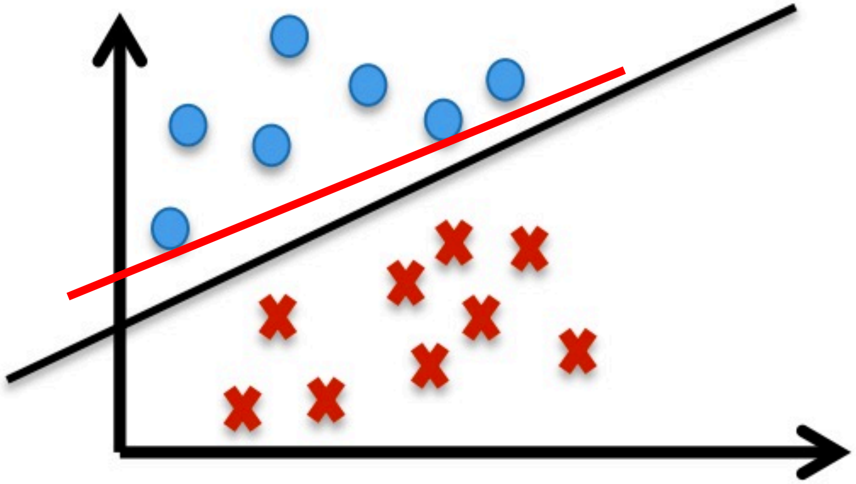
- Given example (x,i) (example $x \in R^n$, labeled i)
  - $\forall j = 1,\ldots,k,\ i \neq j$
  - If $(w_i^T - w_j^T)\ x < 0$ (mistaken prediction)
    - $w_i \leftarrow w_i + x$ (*promotion*)
    - $w_j \leftarrow w_j - x$ (*demotion*)

Conservative version: only demote the the weights corresponding to the label that go the highest score **(incorrectly)**

For any given example, you can potentially make K-1 promotion steps for w_i and K-1 demotion steps, for the *different* w_j

Learning using the Perceptron Algorithm

- Perceptron guarantee: *find a linear separator (if the data is separable)*



- There could be many models consistent with the training data
  - ***How did the perceptron algorithm deal with this problem?***
- Trading some training error for better generalization

Reminder: Loss functions

- To formalize performance let's define a *loss function:*

$$loss(y, \hat{y})$$

  - Where $\hat{y}$ is the gold label

- *The loss function measures the error on a single instance*
  - Specific definition depends on the learning task

Regression

$$loss(y, \hat{y}) = (y - \hat{y})^2$$

Binary classification

$$loss(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & otherwise \end{cases}$$

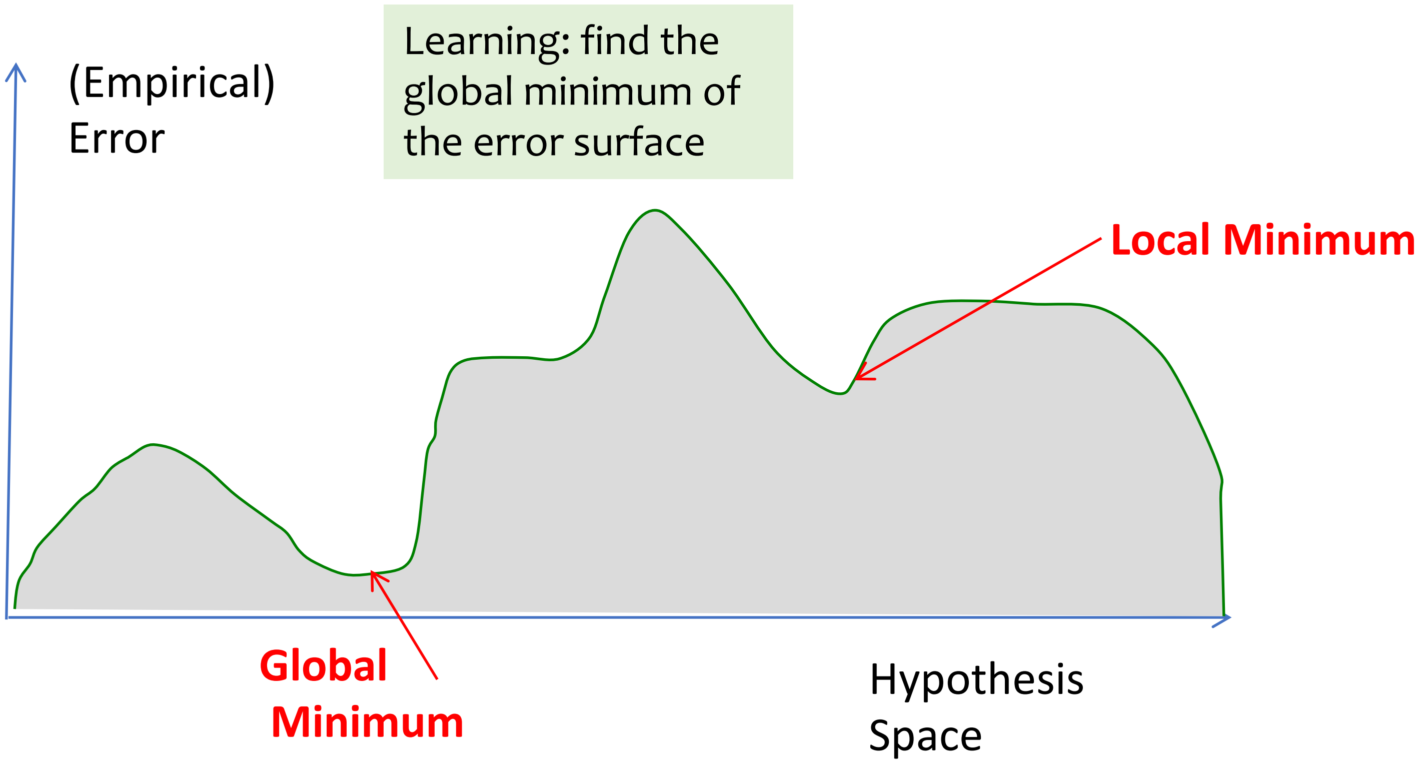
Empirical Risk Minimization

- Learning: Given a training dataset D, return the classifier f(x) that minimizes the empirical risk
- *Given this definition we can view learning as a **minimization problem***
  - The objective function to minimize (empirical risk) is defined with respect to a specific loss function
  - Our minimization procedure (aka **learning**) will be influenced by the choice of loss function
    - **Some are easier to minimize than other!**

$$R_D(f) = \frac{1}{D} \sum_{D} loss(y_i, f(x_i))$$

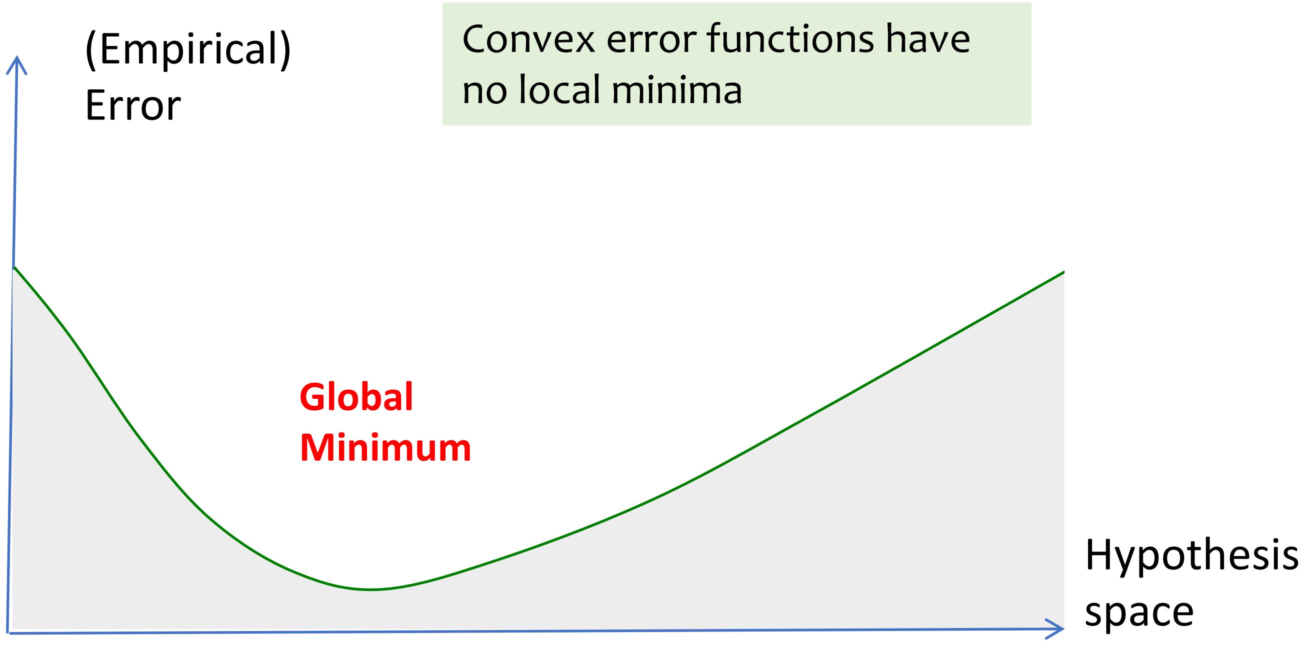
Error Surface

- **Linear classifiers**: hypothesis space parameterized by w
- *Error/Loss/Risk* are all functions of w



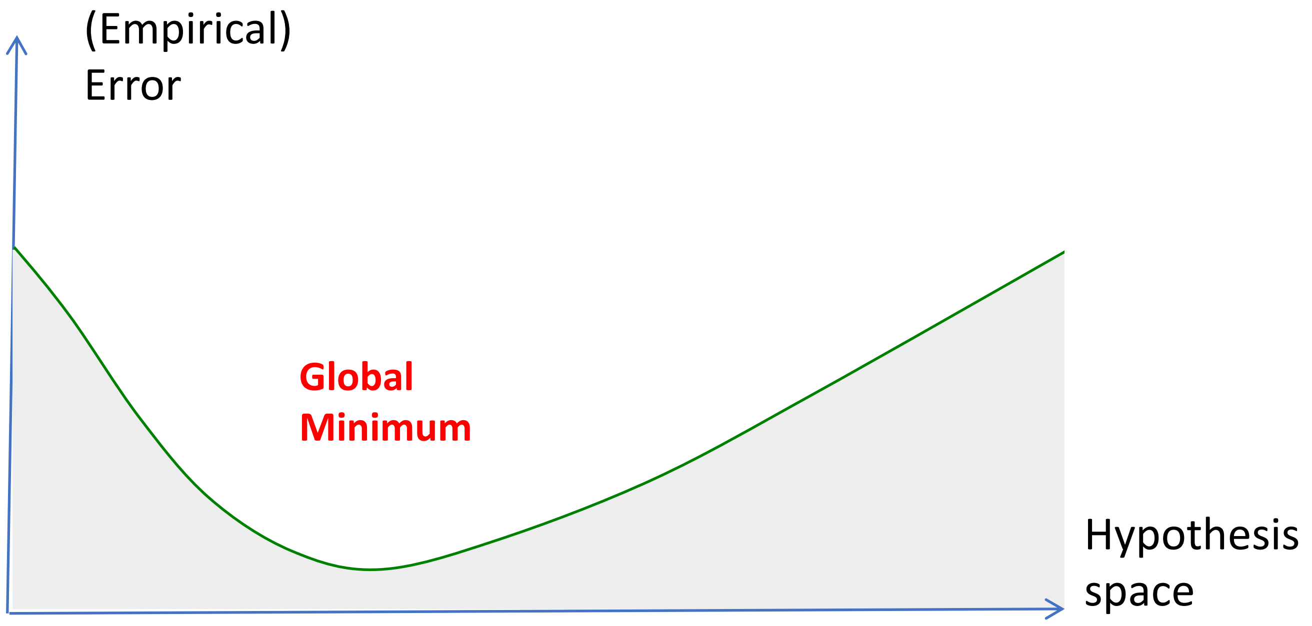
Convex Error Surfaces

- **Convex functions** have a single minimum point
  - Local minimum = global minimum
  - *Easier to optimize*



Loss minimization

- **Let's consider the square loss.**
  - Convex loss function, error surface has a global minimum (~any local minimum is also global).



Gradient Descent for Squared Loss

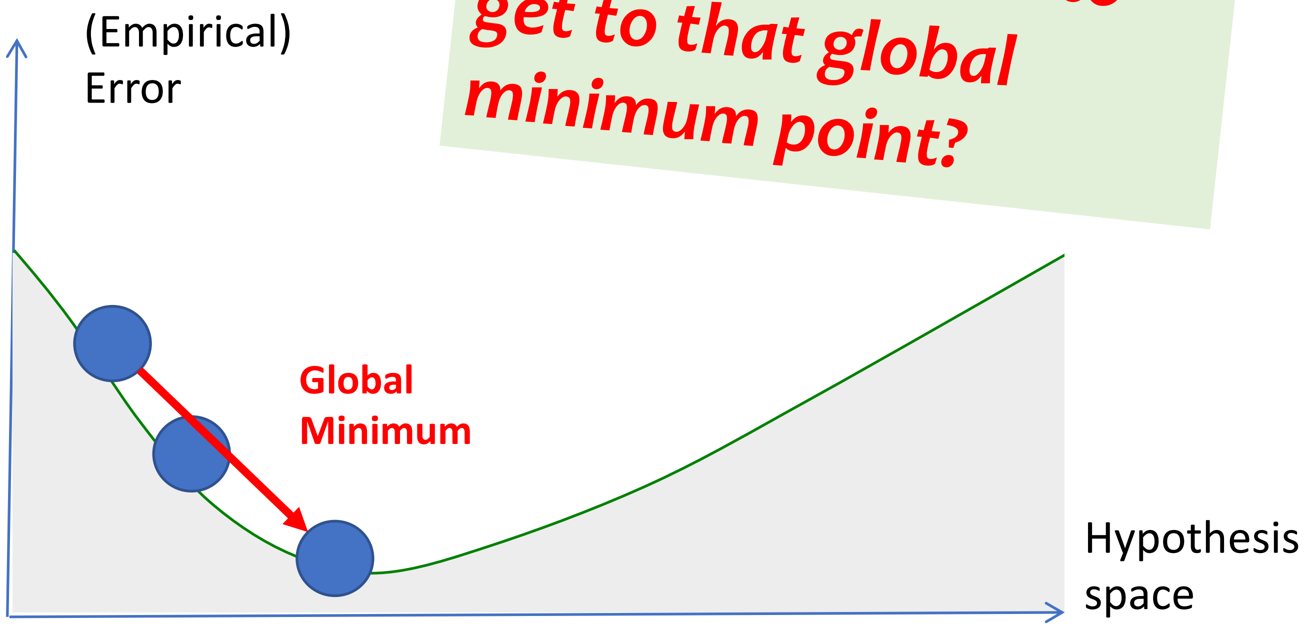
$$Err(w) = \frac{1}{2}\sum_{d\in D}(y_d - f(x_d))^2$$

Initialize $\mathbf{w^0}$ randomly
for i = 0…T:
    $\Delta\mathbf{w} = (0, \ldots, 0)$
    for every training item d = 1…D:
        $f(\mathbf{x_d}) = \mathbf{w^i}\cdot\mathbf{x_d}$
        for every component of $\mathbf{w}$ j = 0…N:
            $\Delta w_j \mathrel{+}= \alpha(y_d - f(\mathbf{x_d}))\cdot x_{dj}$
    $\mathbf{w^{i+1}} = \mathbf{w^i} + \Delta\mathbf{w}$
    return $\mathbf{w^{i+1}}$ when it has converged

Loss minimization

- **Let's consider the square loss.**
  - Convex loss function, error surface has a global minimum (~any local minimum is also global).

Do we really want to get to that global minimum point?



Expected vs. Empirical Risk

- The risk (aka generalization error) of a classifier is its ***expected loss*** (the loss averaged over all possible datasets)

$$R(f) = \int L(y, f(x))P(x, y)dx, y$$

$$\epsilon \triangleq \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[\ell(y, f(\boldsymbol{x}))] = \sum_{(\boldsymbol{x},y)} \mathcal{D}(\boldsymbol{x}, y)\ell(y, f(\boldsymbol{x}))$$

$$R_D(f) = \frac{1}{D}\sum_{D} loss(y_i, f(x_i))$$

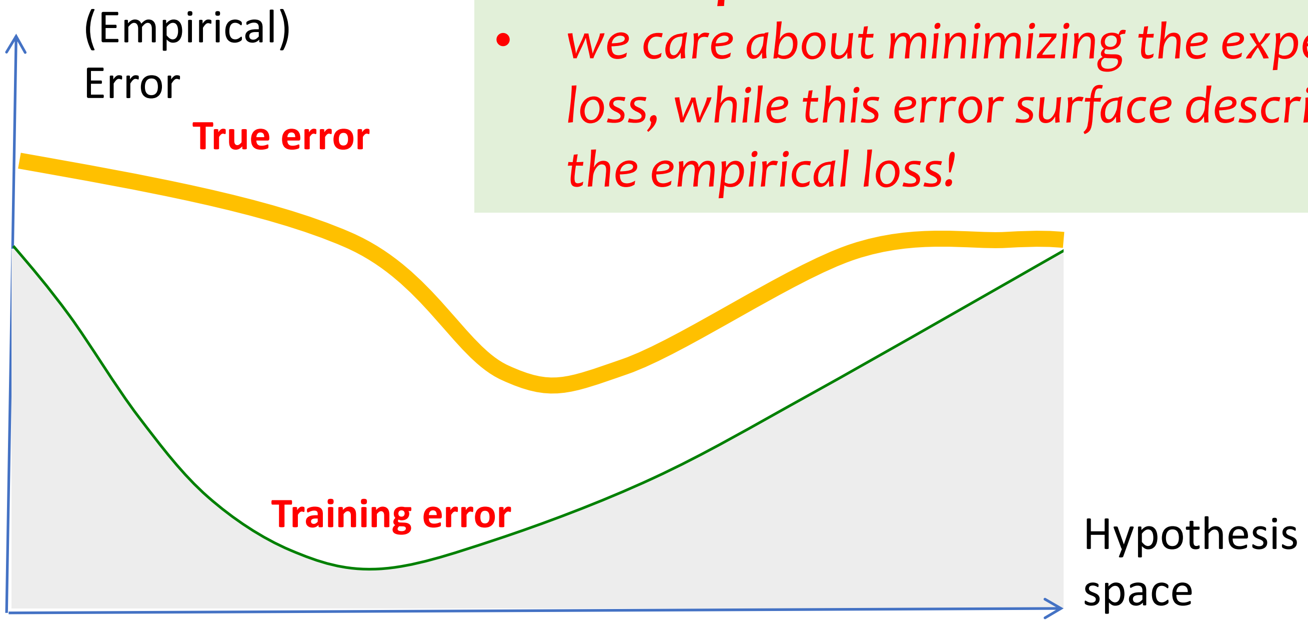
The empirical risk of a classifier on a dataset D is its average loss on the items in d

Loss minimization

- **Let's consider the square loss.**
  - Convex loss function, error surface has a global minimum (~any local minimum is also global).

Do we really want to get to that global minimum point?

- *we care about minimizing the expected loss, while this error surface describes the empirical loss!*



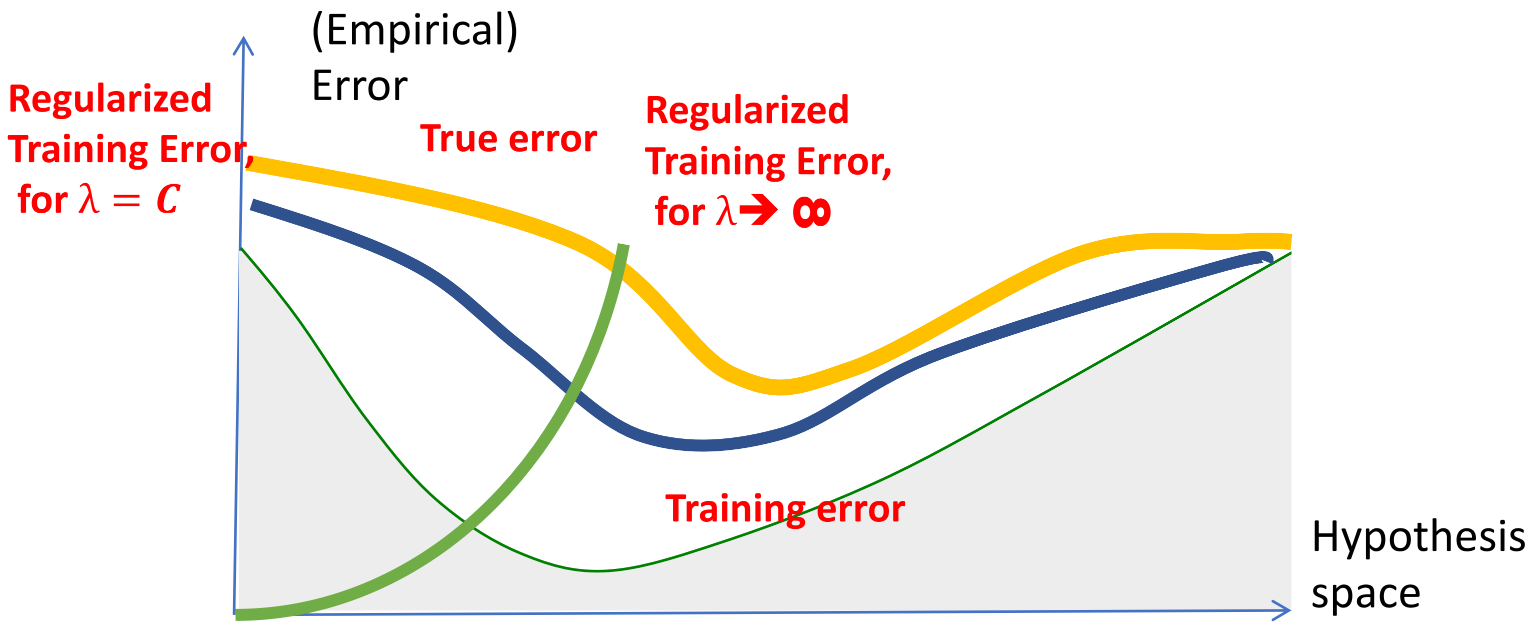
Regularization

- ***A form of inductive bias – we prefer simpler functions!***
- A very popular choice of regularization term is to minimize the norm of the weight vector
  - For convenience: ½ squared norm

$$\min_{\mathrm{w}} \quad \sum_n loss(y_n, \mathrm{w}_n) + \frac{\lambda}{2}||\mathrm{w}||^2$$

Loss minimization

- **Let's consider the square loss.**
  - Convex loss function, error surface has a global minimum (~any local minimum is also global).

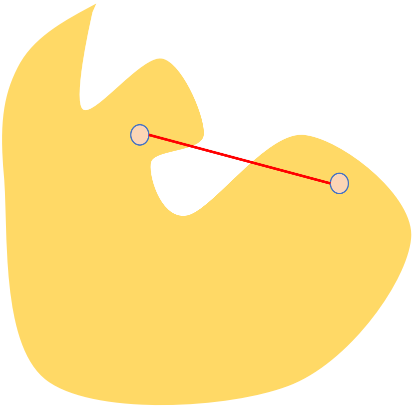


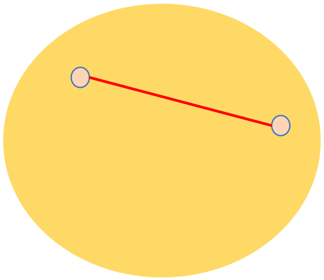
Convex Sets

- A **set** S is **convex** if

$$\forall x, y \in S, \alpha x + (1-\alpha) y \in S \qquad (0 \leq \alpha \leq 1)$$

(the line segment joining x and y is contained in S)





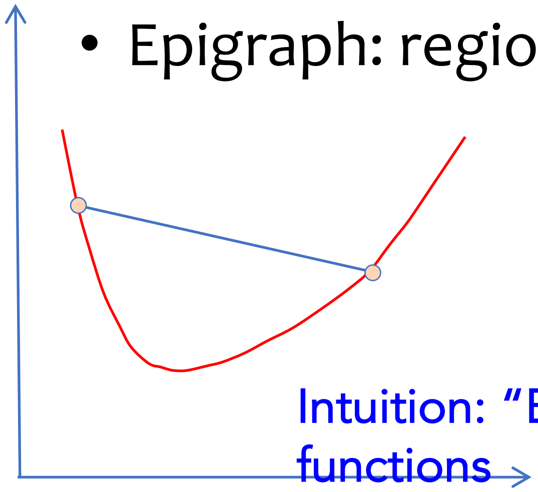
Convex Function

- A function f defined on a convex set is convex if

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \qquad (0 < \alpha < 1)$$

f is convex if the chord joining any two points is always above the graph.

- A function f is convex if its epigraph is a convex set
  - Epigraph: region above the graph of the function f



Intuition: "Bowl shaped" functions

If f is convex $\rightarrow$ - f is concave

Checking Convexity

- According to definition: chord lies above function

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y) \qquad (0 < \alpha < 1)$$

- If f is differentiable: f lies above all tangent lines

$$f(y) \geq f(x) + f'(x)(y-x) \qquad f(y) \geq f(x) + \nabla f(x)(y-x)$$

- For twice differentiable functions: 2nd derivative is non-negative

$$f''(x) \geq 0 \qquad \nabla^2 f(x) \succeq 0$$

- If the above functions are strict inequalities, f is strictly convex

Example: Checking Convexity

$$f(x) = x\log x$$

$$f'(x) = \log x + 1$$

$$f''(x) = \frac{1}{x}$$

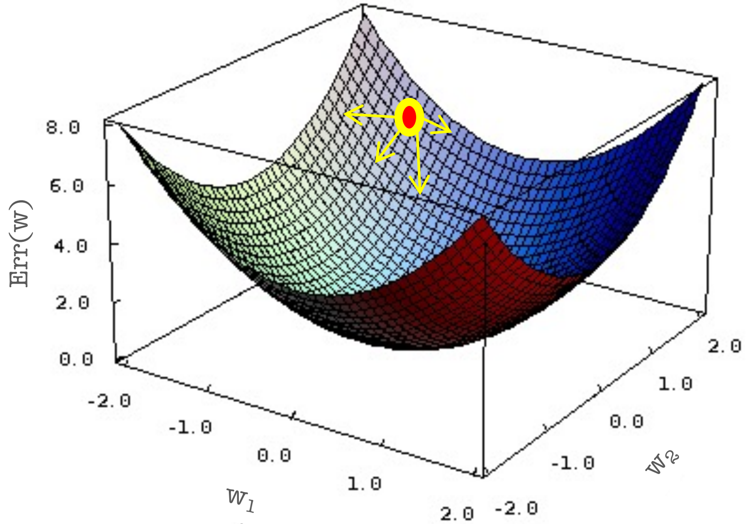
- f''(x) > 0 for all x>0 ➔ f(x) is (strictly) convex!
  - *Note that the domain of log(x) is x>0*

Error Surface for Squared Loss

We add the ½ for convenience

$$Err(\mathrm{w}) = \frac{1}{2}\sum_{d\in D}(\hat{y}_d - y_d)^2$$

$$y = \mathrm{w}^T x$$



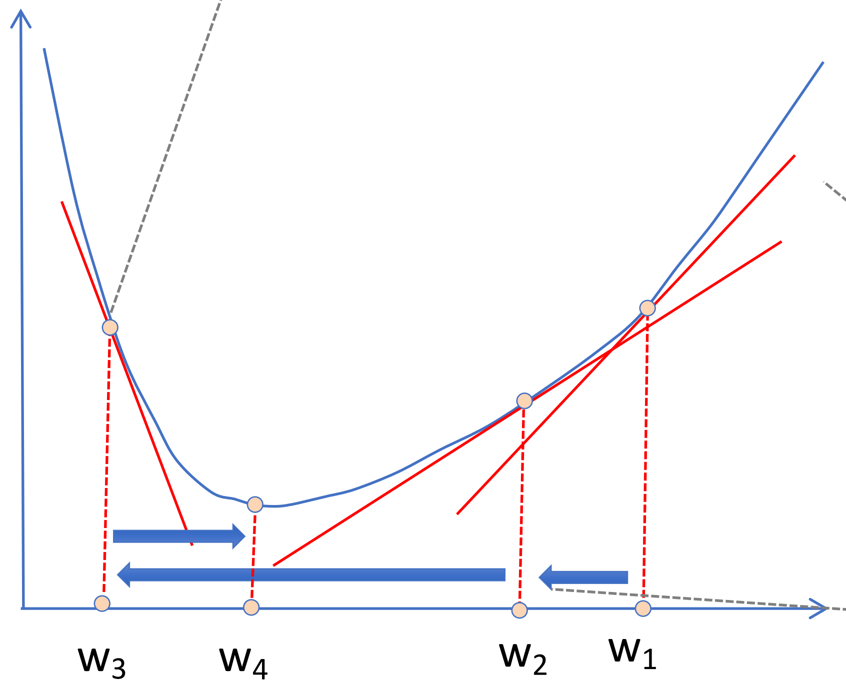
Since $\hat{y}$ is a constant (for a given dataset), the Error function is a *quadratic function* of W (paraboloid)

➔ Squared Loss function is convex!

How can we find the global minimum?

Gradient Descent Intuition

(3) We also need to determine the step size (aka learning rate).
What happens if we overshoot?



(1) The derivative of the function at w_1 is the slope of the tangent line
→ Positive slope (increasing)
Which direction should we move to decrease the value of Err(w) ?

(2) The gradient determines the direction of steepest increase of Err(w) *(go in the opposite direction)*

What is the gradient of Error(w) at this point?

The Gradient of Error(w)

The gradient is a generalization of the derivative

$$\nabla Err(\mathrm{w}) = \left(\frac{\partial Err(w)}{\partial w_0}, \frac{\partial Err(w)}{\partial w_1}, \ldots, \frac{\partial Err(w)}{\partial w_n} \right)$$

The gradient is a vector of partial derivatives.

It Indicates the *direction of steepest increase* in Err(w), for each one of w's coordinates

Computing $\nabla Err(w^i)$ for Squared Loss

$$Err(w) = \frac{1}{2}\sum_{d \in D}(y_d - f(x_d))^2$$

$$\frac{\partial \mathrm{Err}(\mathbf{w})}{\partial \mathrm{w_i}} = \frac{\partial}{\partial \mathrm{w_i}} \frac{1}{2} \sum_{\mathrm{d} \in \mathrm{D}} (\mathrm{y_d} - \mathrm{f}(\mathbf{x}_\mathrm{d}))^2$$

$$= \frac{1}{2} \frac{\partial}{\partial \mathrm{w_i}} \sum_{\mathrm{d} \in \mathrm{D}} (\mathrm{y_d} - \mathrm{f}(\mathbf{x}_\mathrm{d}))^2$$

$$= \frac{1}{2} \sum_{\mathrm{d} \in \mathrm{D}} 2(\mathrm{y_d} - \mathrm{f}(\mathbf{x}_\mathrm{d})) \frac{\partial}{\partial \mathrm{w_i}} (\mathrm{y_d} - \mathbf{w} \cdot \mathbf{x}_\mathrm{d})$$

$$= -\sum_{d \in D} (y_d - f(\mathbf{x}_d)) x_{di}$$

Batch Update Rule for Each w_i

- **Implementing gradient descent**: *as you go through the training data, accumulate the change in each w_i of W*

$$\Delta w_i = \alpha \sum_{d=1}^{D} (y_d - \mathbf{w}^i \cdot \mathbf{x}_d) x_{di}$$

Gradient Descent for Squared Loss

Initialize $\mathbf{w^0}$ randomly
for i = 0…T:
    $\Delta\mathbf{w} = (0, …., 0)$
    for every training item d = 1…D:
        $f(\mathbf{x_d}) = \mathbf{w^i} \cdot \mathbf{x_d}$
        for every component of $\mathbf{w}$ $j = 0…N$:
            $\Delta w_j \mathrel{+}= \alpha(y_d - f(\mathbf{x_d})) \cdot x_{dj}$
    $\mathbf{w^{i+1}} = \mathbf{w^i} + \Delta\mathbf{w}$
    return $\mathbf{w^{i+1}}$ when it has converged

Batch vs. Online Learning

- The Gradient Descent algorithm makes updates after going over the entire data set
  - Data set can be huge
  - Streaming mode (we cannot assume we saw all the data)
  - Online learning allows “adapting” to changes in the target function
- Stochastic Gradient Descent
  - Similar to GD, updates after each example
  - Can we make the same convergence assumptions as in GD?
- Variations: update after a subset of examples

Stochastic Gradient Descent

Initialize $\mathbf{w^0}$ randomly
for m = 0…M:

$$f(\mathbf{x_m}) = \mathbf{w^i} \cdot \mathbf{x_m}$$

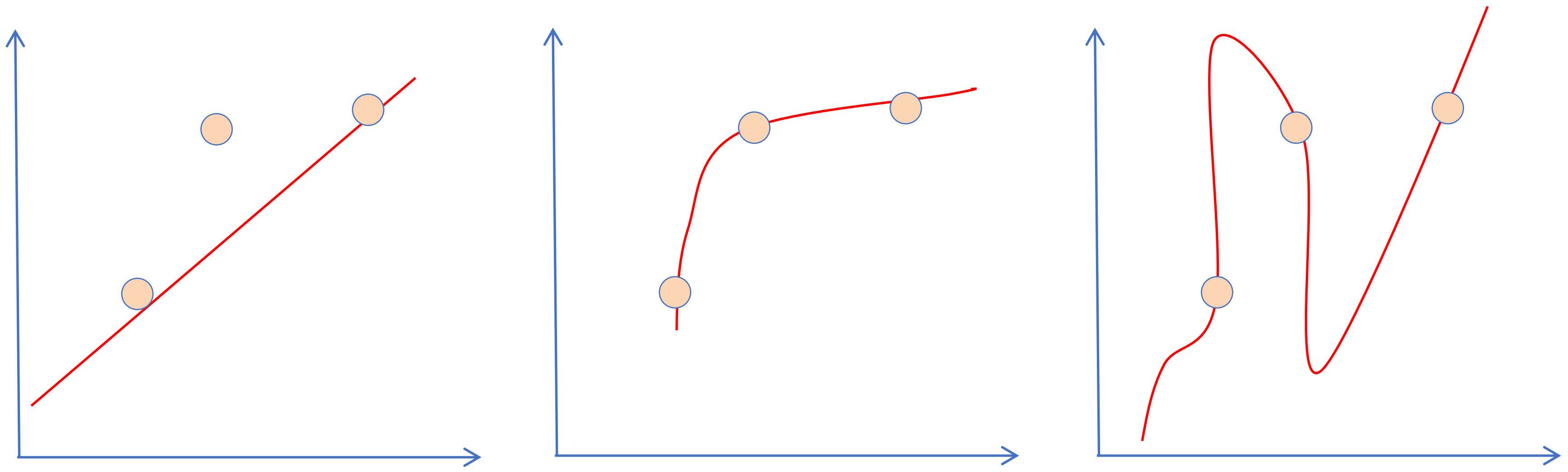
$$\Delta w_j = \alpha(y_d - f(\mathbf{x_m})) \cdot x_{mj}$$

$$\mathbf{w^{i+1}} = \mathbf{w^i} + \Delta\mathbf{w}$$

return $\mathbf{w^{i+1}}$ when it has converged

Polynomial Regression

- GD: general optimization algorithm
  - Works for classification (different loss function)
  - Incorporate polynomial features to fit a complex model
  - **Danger – overfitting!**

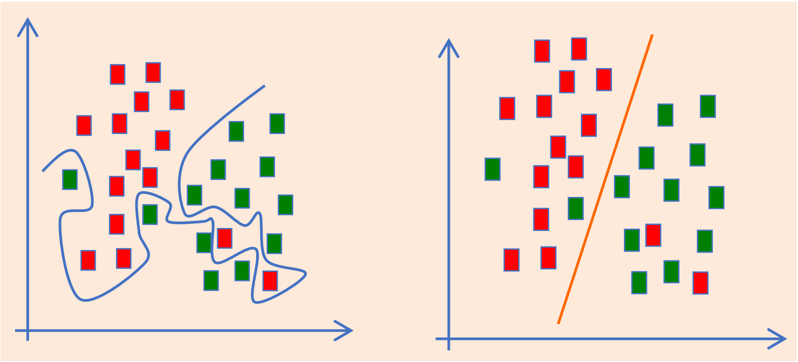


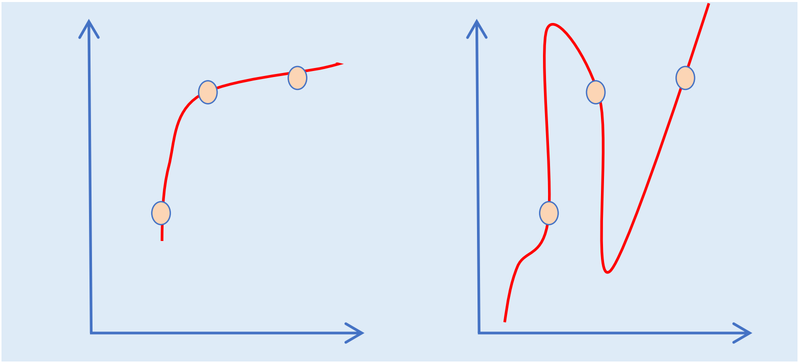
Regularization

- Both for regression and classification, for a given error we prefer a *simpler model*
  - *Keep W small: ε changes in the input cause ε*w in the output*
- *Some times we are even willing to trade a higher error rate for a simpler model (why?)*
- *Add a regularization term:*
  - *This is a form of* ***inductive bias***

$$\min_{\mathrm{w}} = \sum_n loss(y_n, \mathrm{w}_n) + \lambda R(\mathrm{w})$$

How different values affect learning?





Regularization

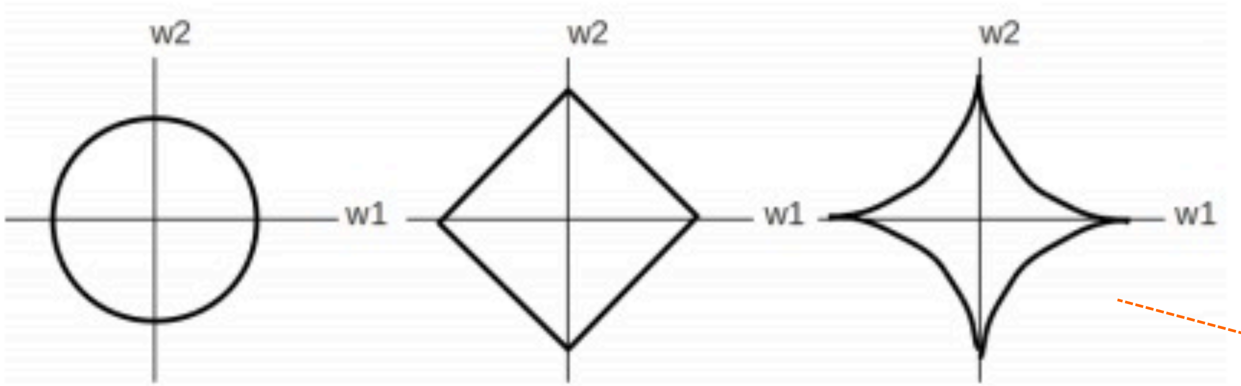
- A very popular choice of regularization term is to minimize the norm of the weight vector
  - For example $||\mathrm{w}|| = \sqrt{\sum_d \mathrm{w}_d^2}$
  - For convenience: ½ squared norm $\min_{\mathrm{w}} = \sum_n loss(y_n, \mathrm{w}_n) + \frac{\lambda}{2}||\mathrm{w}||^2$
    - *What is the update gradient of the loss function?*
    - *At each iteration we subtract the weights by λ*w*
- In general, we can pick other norms (p-norm)
  - *Referenced as L-p norm*
  - Different p will have a different effect!
  - *What will the norm for p=1 and p=0 minimize?*

$$||\mathrm{w}||_p = \left(\sum_d |\mathrm{w}_d|^p\right)^{\frac{1}{p}}$$

Different Regularization functions

- To understand the difference between different norms consider their iso-surface $||w||_p = C$ (*c is a constant value)

$$||w||_p = \left(\sum_d |w_d|^p\right)^{\frac{1}{p}}$$



P=2 **P=1** **P<1**

p<1: peaked: large values of one weight comes at the "expense" of another

- As the value of p gets closer to 0, the norm is closer to a *counting norm*: count non-zero parameters (ignoring values) →directly minimizes the number of active features
- In general: smaller values of p encourage sparsity

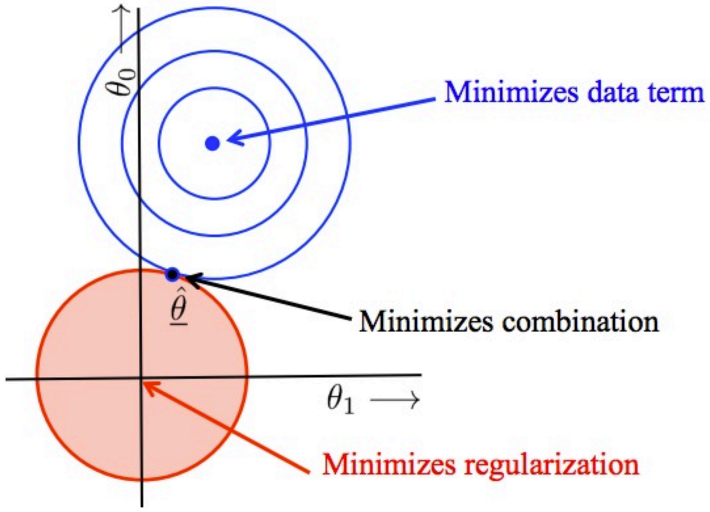
How would the iso surface of p=0 look?

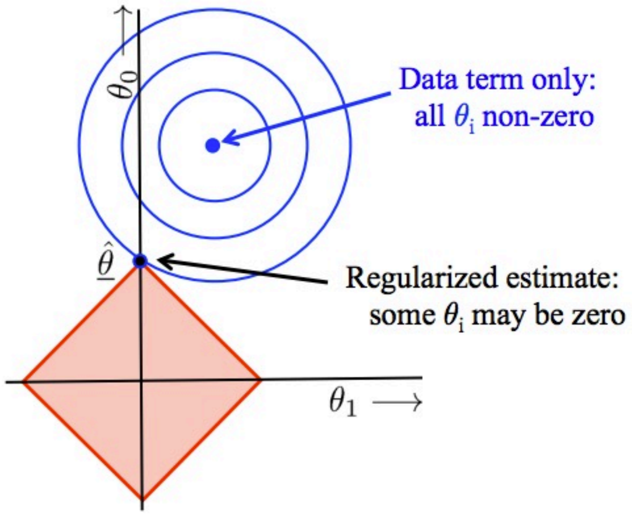
L1 and L2 norms

- Common choices: L1 and L2 are both convex ($L_{p<1}$ *is not convex*)
- *Regularized objective*: balance between minimizing the error and the regularization cost

L2 optimum will be sparse, ONLY if the data loss term is minimized at the axis

*L1 norm contour are **sharp**, will intersect with the contour of data loss term even when the data loss term min point is not at the axis*
➔ L1 encourages sparsity





Classification

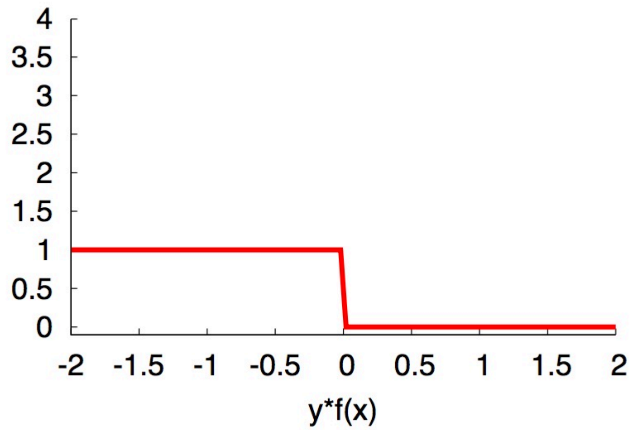
- **So far**:
  - General optimization framework for learning
  - Minimize regularized loss function

$$\min_{\mathrm{w}} = \sum_n loss(y_n, \mathrm{w}_n) + \frac{\lambda}{2}||\mathrm{w}||^2$$

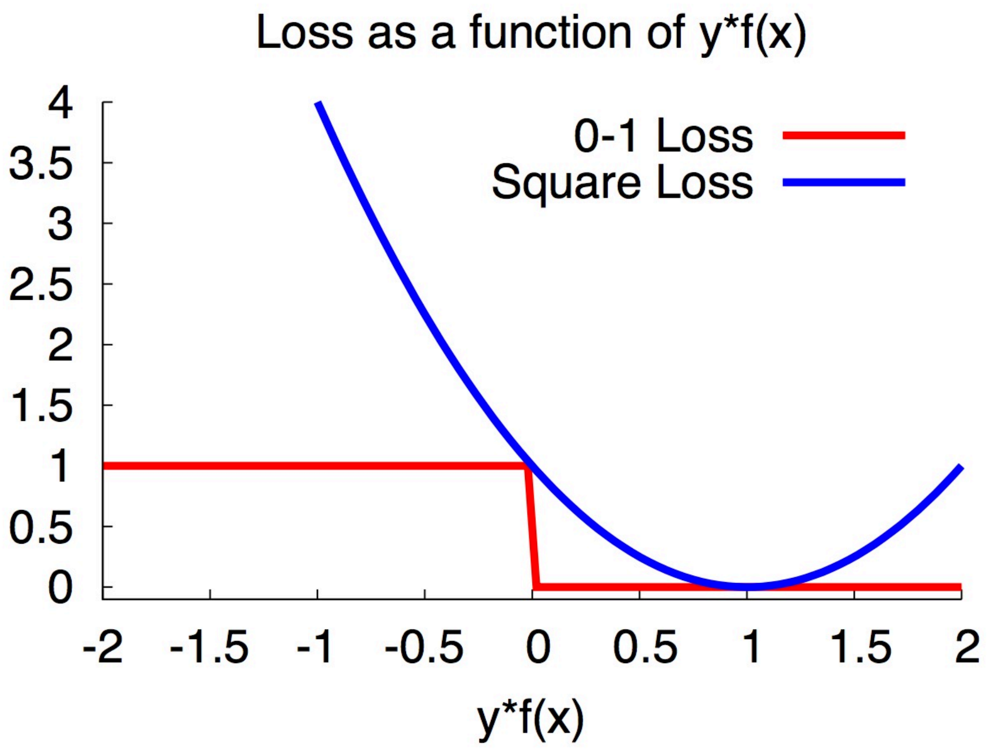
  - Gradient descent is an all purpose tool
    - Computed the gradient of the *square loss function*
- *Moving to classification should be very easy*
  - Simply replace the loss function
- **…**

Classification

- Can we minimize the empirical error directly?
    - Use the 0-1 loss function
- ***Problem**: Cannot be optimized directly*
    - *Non convex and non differentiable*
- **Solution**: define a smooth loss function, an upper bound to the 0-1 loss function



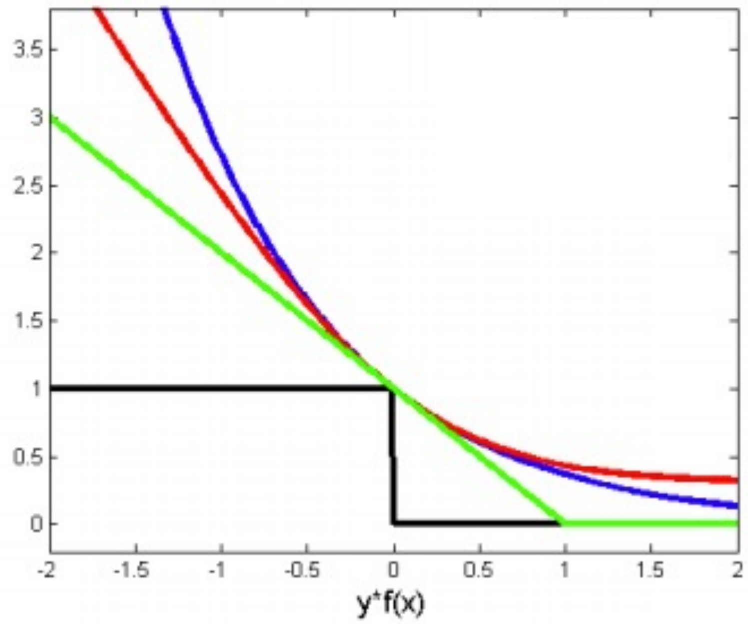
Square Loss is an **Upper bound** to 0/1 Loss



Is the square hinge loss a good candidate to be a surrogate loss function for 0-1 loss?

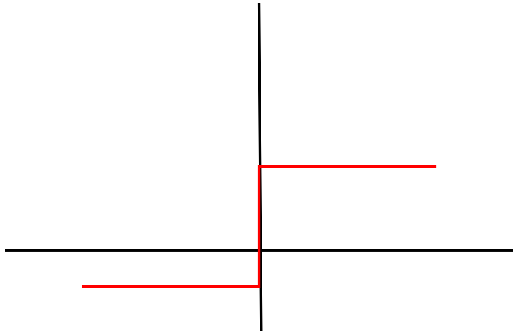
Surrogate Loss functions

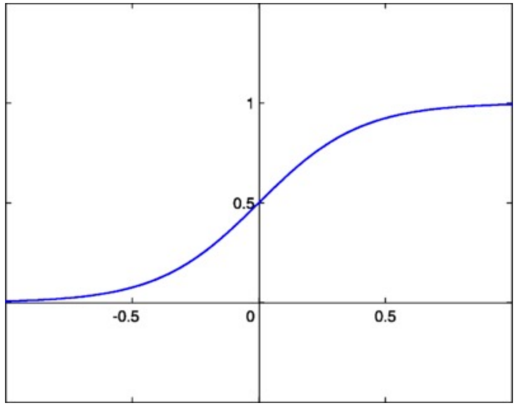
- Surrogate loss function: smooth approximation to the 0-1 loss
  - Upper bound to 0-1 loss



Logistic Function

- Smooth version of the threshold function





$$h_{\mathrm{w}}(x) = g(\mathrm{w}^T x)$$

$$z = \mathrm{w}^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid (logistic) function

- Known as a sigmoid/logistic function
  - Smooth transition between 0-1
- Can be interpreted as the conditional probability
- Decision Boundary
  - y=1: h(x) >0.5 ➔ $w^t x >= 0$
  - Y=0: h(x) <0.5 ➔ $w^t x < 0$

Logistic Regression

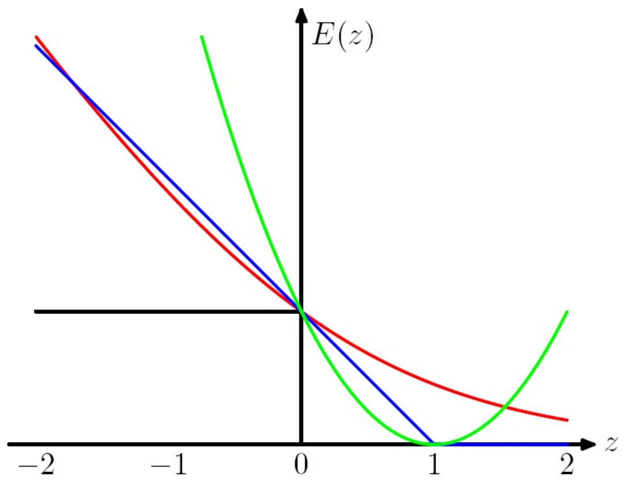
- **Learning**: optimize the likelihood of the data
  - **Likelihood**: *probability of our data under current parameters*
  - For easier optimization, we look into the log likelihood (*negative*)
- **Cost Function**
  - If y = 1: - log P(y=1| x, w) = - log (g(w,x))
    - *If the model gives very high probability to y=1, what is the cost?*
  - If y = 0: - log P(y=0| x, w) = - log 1- P(y=1| x, w) = - log (1- g(w,x))
- Or more succinctly (with l_2 regularization)

$$Err(\mathrm{w}) = -\sum_i y^i log(e^{g(w,x_i)}) + (1 - y^i) log(1 - e^{g(w,x_i)}) + \frac{1}{2}\lambda||w||^2$$

- This function is convex and differentiable
  - Compute the gradient, minimize using gradient descent

Question

- 0-1 loss has a steep change when $y(w^Tx)>0$ and $y(w^Tx)<0$, while the logistic loss function has a smooth transition.
- This means that even correct classification will incur some cost.
- *How does this fact bias the learning process?*

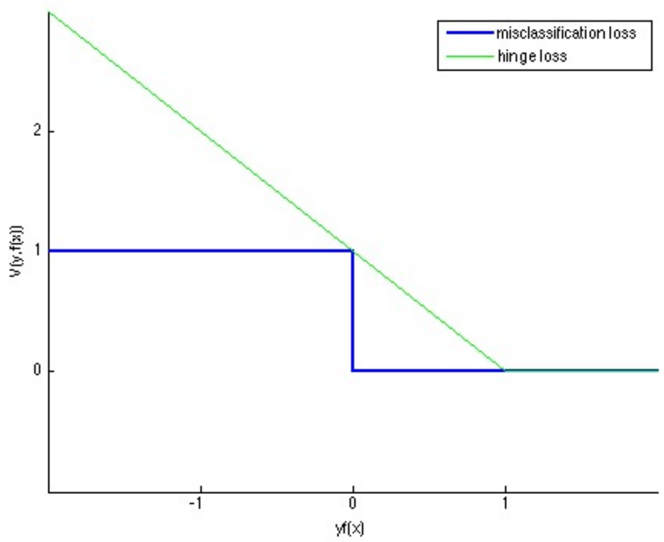


Hinge Loss

- Another popular choice for loss function is the hinge loss

$$L(y, f(x)) = \max(0, 1 - y\, f(x))$$

  - We will discuss in the context of support vector machines (SVM)



It's easy to observe that:

(1) The hinge loss is an upper bound to the 0-1 loss

(2) The hinge loss is a good approximation for the 0-1 loss

(3) BUT …

It is not differentiable at $y(w^Tx)=1$

Solution: Sub-gradient descent

Summary

- Introduced an optimization framework for learning:
  - Minimization problem
  - Objective: data loss term and regularization cost term
  - Separate learning objective from learning algorithm
    - Many algorithms for minimizing a function
- Can be used for regression and classification
  - Different loss function
  - GD and SGD algorithms
- Classification: use surrogate loss function
  - Smooth approximation to 0-1 loss