

Logistic Regression

Based on slides by Daniel Lowd, Vibhav Gogate,
Carlos Guestrin, Luke Zettlemoyer and Dan Weld.

Predicting Probabilities

- We have mainly talked about binary classification
- Our decision rule always give one type with 100% confidence for a given example (e.g., Spam nor Not)
- What if an email has a 60% chance of being spam?
- What if there is a 30% chance of rain?
- What if a mushroom has a 10% probability of being poisonous?
- **We want to be able to talk about uncertainty or the probability of a type given an example.**

Logistic Regression

- Predict class probabilities $P(Y|X)$ with a linear model: still compute a linear predictor (based on the linear combination of parameters), then transform it into a probability
- How do we turn $(w \cdot x + b)$ into a probability?

Linear models in general

- By linear models, we mean that the hypothesis function $h_w(x)$ is a linear function of the parameters w , not the linear function of the input vector x
- Generalized linear model

- Logistic regression is a special case of a **generalized linear model**:

$$E[Y | \mathbf{x}] = g^{-1}(\mathbf{w}^\top \mathbf{x}).$$

g is called the *link function*, it relates the mean of the response to the linear predictor.

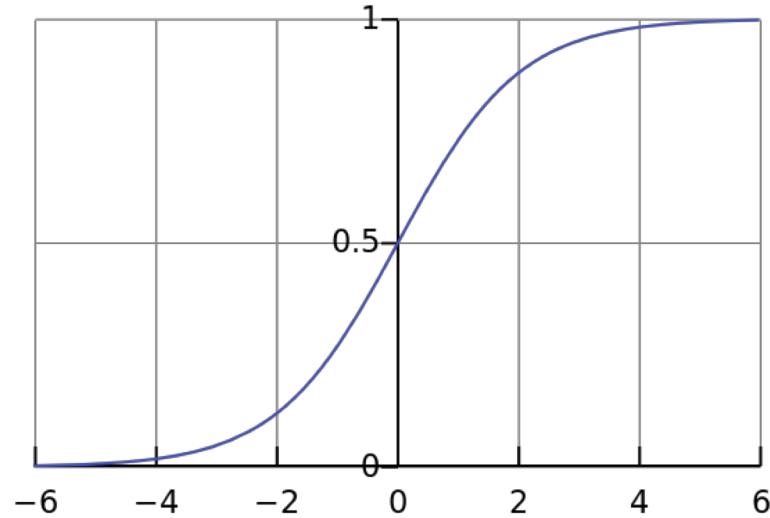
- Linear regression: $E[Y | \mathbf{x}] = E[\mathbf{w}^\top \mathbf{x} + \varepsilon | \mathbf{x}] = \mathbf{w}^\top \mathbf{x}$ (g is the identity).
- Logistic regression: $E[Y | \mathbf{x}] = P(Y = 1 | \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$
- Poisson regression: $E[Y | \mathbf{x}] = \exp(\mathbf{w}^\top \mathbf{x})$ (for count data)
- ...

The logit function $\text{Log } \frac{z}{1-z}$

The Logistic Function

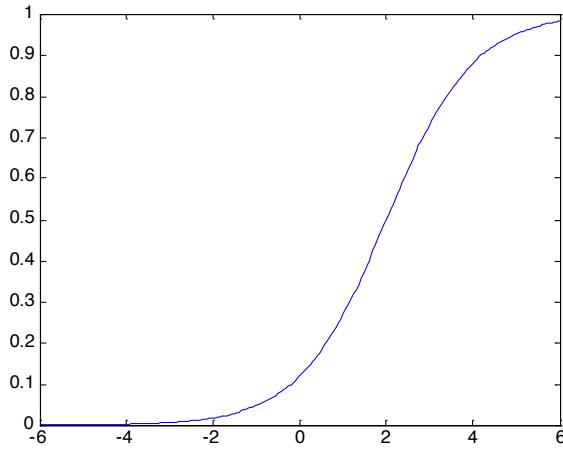
The **logistic function** squashes a real number in the range $(-\infty, +\infty)$ into the range $(0, 1)$ (inverse of the logit function).

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

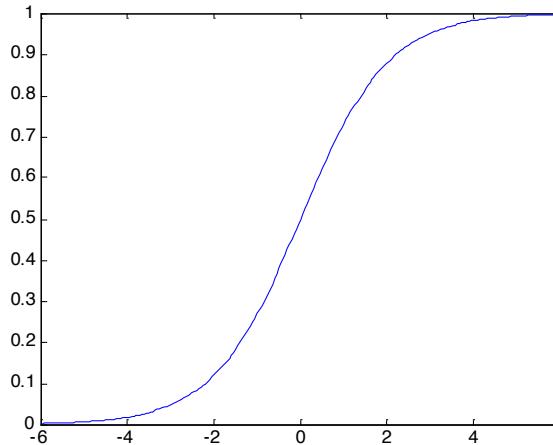


Understanding Sigmoids

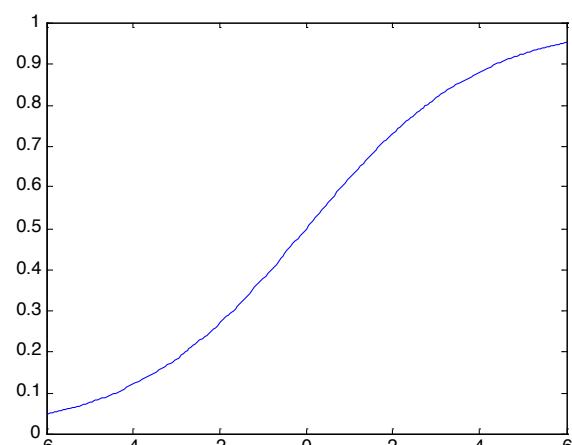
$$\sigma(w_1x_1 + b) = \frac{1}{1 + \exp(-(w_1x_1 + b))}$$



$w_1=1, b=-2$



$w_1=1, b = 0$

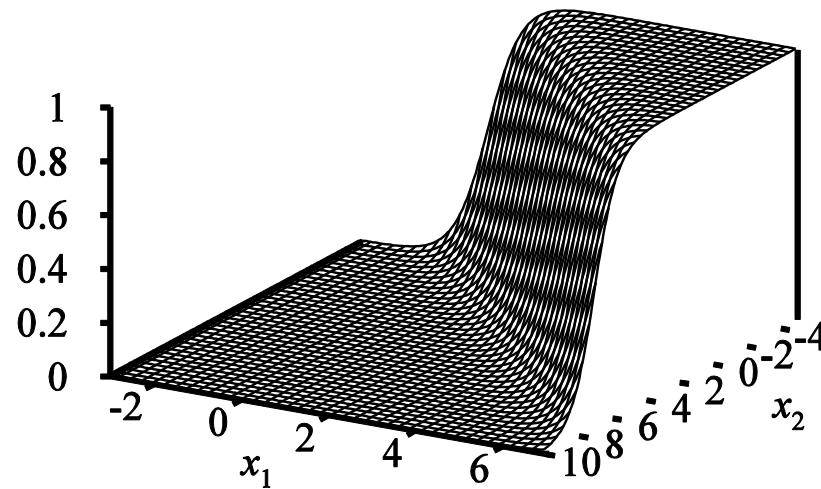


$w_1= 0.5, b = 0$

Logistic Regression

$$P(y|x) = \sigma(w^T x + b) = \frac{1}{1 + \exp(-(w^T x + b))}$$

Example: Logistic function for two input features x_1 and x_2 :



Very convenient for two classes

$$P(Y = 1|X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$P(Y = 0|X = \langle X_1, \dots, X_n \rangle) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i)$$

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$

linear classification rule!

$Y=0$ if the RHS > 0

Learning Logistic Regression

Key idea: Choose a model where reality is probable.

Probability of the true labels given the data:

$$\prod_{(x,y) \in D} \hat{P}(y|x) = \prod_{(x,y) \in D} \frac{1}{1 + \exp(-y(w^T x + b))}$$

$$\log \prod_{(x,y) \in D} \hat{P}(y|x) = \sum_{(x,y) \in D} -\log(1 + \exp(-y(w^T x + b)))$$

Conditional log-likelihood

(Negative) logistic loss

Maximize the log-likelihood
= Minimize negative log-likelihood
= Minimize logistic loss

Why logistic loss?

- How's about the 0/1 loss: $\min_{w,b} \sum_n \mathbf{1}[y_n(w \cdot x_n + b) > 0]$
- 0-1 loss is non-smooth – a small change in a parameter could lead to a BIG change in accuracy! (How?)
- What do we want beyond smoothness?

Convex functions

- A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if for all $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, $\lambda \in [0, 1]$:

$$f(\lambda\mathbf{a} + (1 - \lambda)\mathbf{b}) \leq \lambda f(\mathbf{a}) + (1 - \lambda)f(\mathbf{b}).$$

- If f and g are convex functions, $\alpha f + \beta g$ is also convex for any real numbers α and β .
- First-order* characterization:

$$f \text{ is convex} \Leftrightarrow \text{for all } \mathbf{a}, \mathbf{b} : f(\mathbf{a}) \geq f(\mathbf{b}) + \nabla f(\mathbf{b})^\top (\mathbf{a} - \mathbf{b})$$

(the function is globally above the tangent at \mathbf{b}).

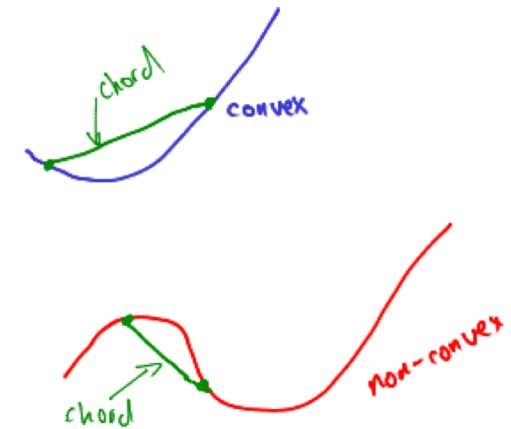
- Second-order* characterization:

$$f \text{ is convex} \Leftrightarrow \text{the Hessian of } f \text{ is positive semi-definite.}$$

The **Hessian** contains the second-order derivatives of f :

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

It is **positive semi-definite** if $\mathbf{a}^\top \mathbf{H} \mathbf{a} \geq 0$ for all $\mathbf{a} \in \mathbb{R}^d$.



Examples of Loss Functions

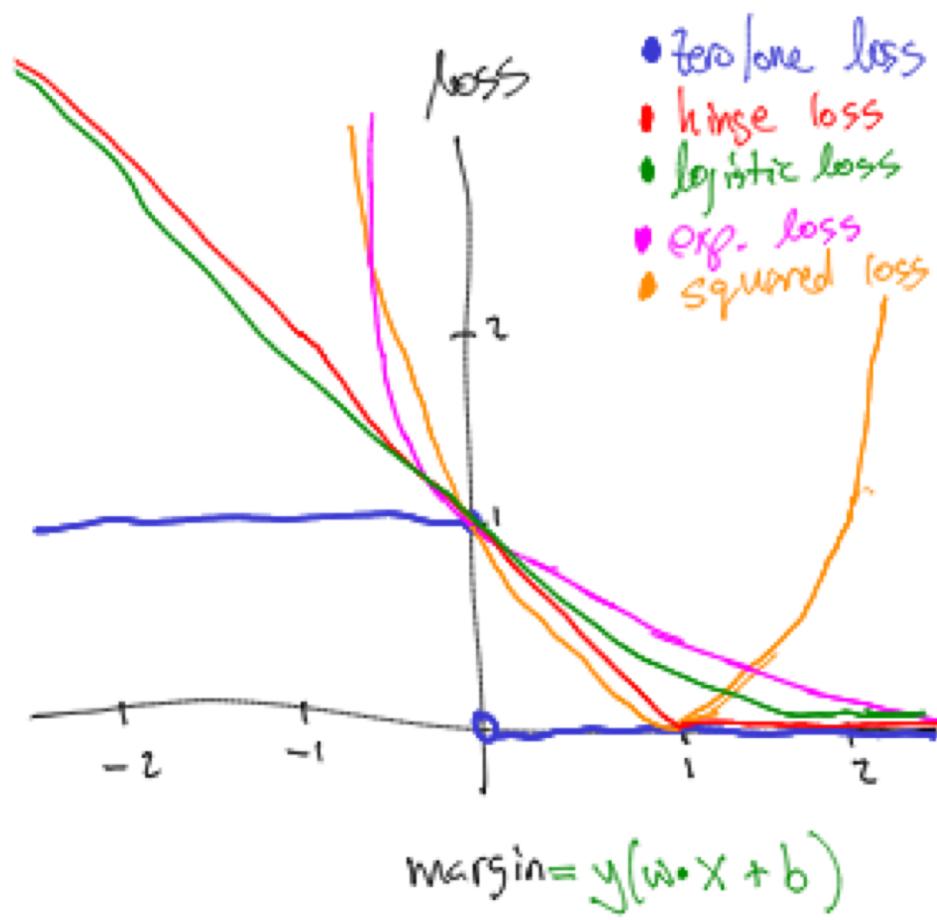
Zero/one: $\ell^{(0/1)}(y, \hat{y}) = \mathbf{1}[y\hat{y} \leq 0]$ (6.3)

Hinge: $\ell^{(\text{hin})}(y, \hat{y}) = \max\{0, 1 - y\hat{y}\}$ (6.4)

Logistic: $\ell^{(\log)}(y, \hat{y}) = \frac{1}{\log 2} \log (1 + \exp[-y\hat{y}])$ (6.5)

Exponential: $\ell^{(\text{exp})}(y, \hat{y}) = \exp[-y\hat{y}]$ (6.6)

Squared: $\ell^{(\text{sqr})}(y, \hat{y}) = (y - \hat{y})^2$ (6.7)



Learning Logistic Regression

- Given the training data, how can we find the model parameters that would minimize the negative log-likelihood?
- **Bad news:** no closed-form solution to maximize the negative log-likelihood
- **Good news:** The negative log-likelihood function is concave function of w !
 - No local minima
 - Concave functions easy to optimize
- Optimization techniques: finding the minimum/maximum of multivariate functions.
- So, machine learning is sort of
Model + Cost function + Optimization technique

Gradient Descent

Q: Suppose you want to avoid a flood. How do you find a high place?

A: Walk uphill!

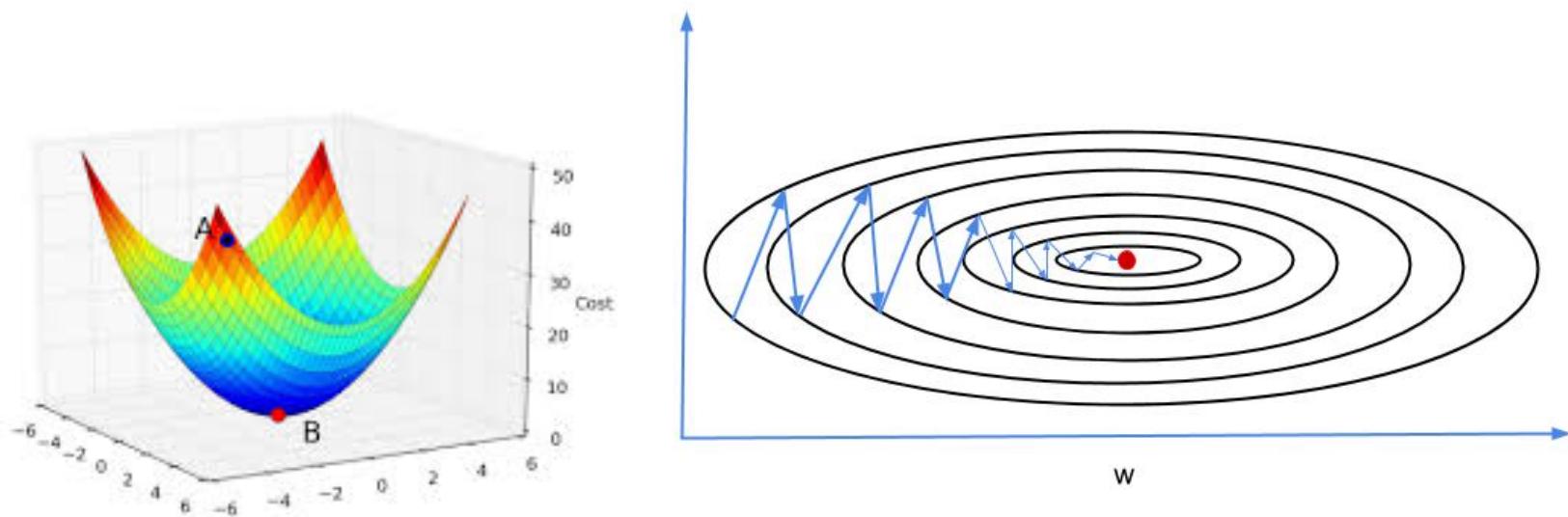
Q: And if you want to find a low place?

A: Walk downhill!

Q: And if you want to minimize a function?

A: Gradient descent!

Gradient Descent



Gradient Descent

Algorithm 22 GRADIENTDESCENT($\mathcal{F}, K, \eta_1, \dots$)

```

1:  $z^{(0)} \leftarrow \langle o, o, \dots, o \rangle$                                 // initialize variable we are optimizing
2: for  $k = 1 \dots K$  do
3:    $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$                           // compute gradient at current location
4:    $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$                       // take a step down the gradient
5: end for
6: return  $z^{(K)}$ 
  
```

For logistic regression:

$$\mathcal{F} = -\log \prod_{(x,y) \in D} \hat{P}(y|x) = \sum_{(x,y) \in D} \log(1 + \exp(-y(w^T x + b)))$$

Gradient of Logistic Loss

$$\frac{\partial \mathcal{L}}{\partial w_i} = -\sum_{(x,y) \in D} \frac{1}{1 + \exp(y(w^T x + b))} y x_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{(x,y) \in D} \frac{1}{1 + \exp(y(w^T x + b))} y$$

Try it!

Convergence Rate

For strongly convex functions:

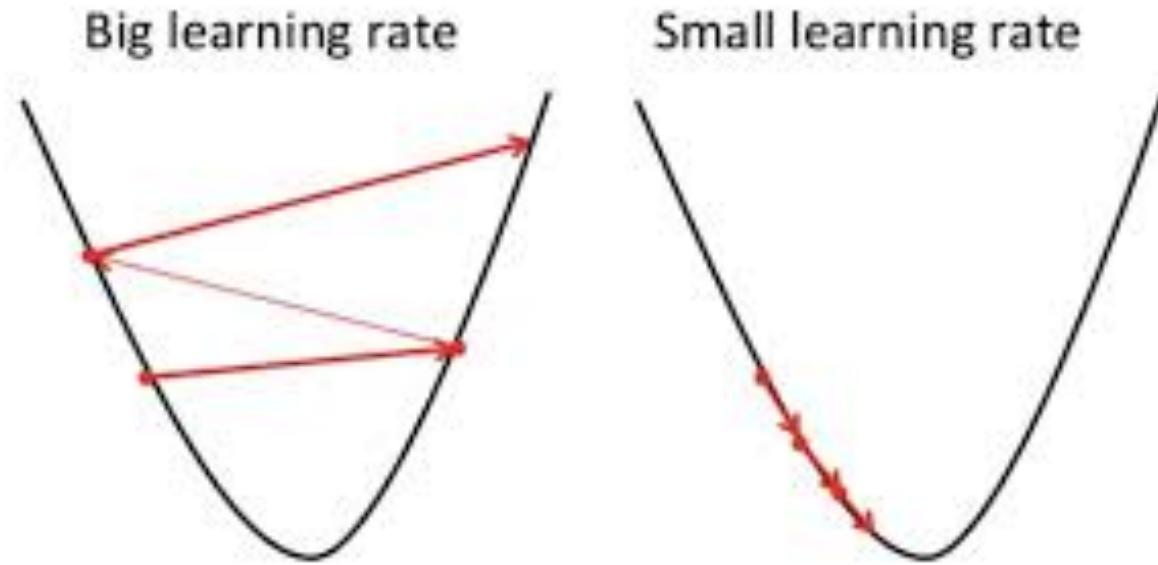
Theorem 7 (Gradient Descent Convergence). *Under suitable conditions¹, for an appropriately chosen constant step size (i.e., $\eta_1 = \eta_2, \dots = \eta$), the **convergence rate** of gradient descent is $\mathcal{O}(1/k)$. More specifically, letting z^* be the global minimum of \mathcal{F} , we have: $\mathcal{F}(z^{(k)}) - \mathcal{F}(z^*) \leq \frac{2\|z^{(0)} - z^*\|^2}{\eta k}$.* ← Doubling the number of iterations cuts the error in half.

The effect of learning rate

Q: What's an "appropriately chosen constant step size"?

A: $1/L$ where L is the curvature of the function.

Many variants – adaptive learning rates, line search, or momentum terms to speed up convergence.



The second order methods

- The Newton's method to find the zero of a function: $g: \mathbb{R} \rightarrow \mathbb{R}$

$$w^{i+1} = w^i - \frac{g(w^i)}{g'(w^i)}$$

- Applying to machine learning to minimize J , so finding the zero of the gradient J'

$$w^{i+1} = w^i - \frac{J'(w^i)}{J''(w^i)}$$

- So, no step size needed
- If the error function is quadratic, this will find the minimum in one step!
- But, we need to compute the second-order derivatives (the Hessian matrix)

$$\mathbf{H}_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1} \nabla_{\mathbf{w}} J$$

Stochastic Gradient

- **Perceptron algorithm:** Update model immediately after each misprediction.
- **Gradient descent:** Compute gradient over all examples before updating.

What if we instead compute the gradient on a randomly chosen subset of the data?

Advantages?

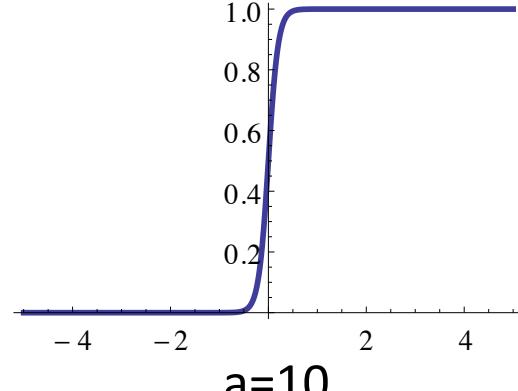
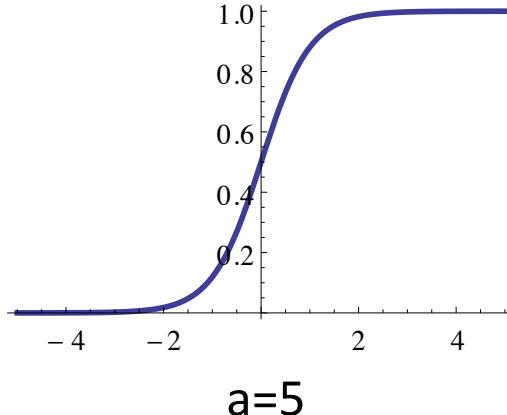
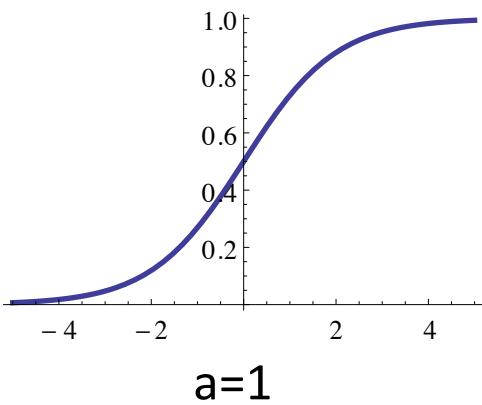
Disadvantages?

Mini batching

- A way to approximate the gradient computation over the whole training data
- Cons:
 - Not guaranteed to give us the steepest direction
- Pros:
 - Efficient (very often, you won't be able to load all your training data into memory for computation)
 - Less variance on gradient than stochastic gradient descent (batch size = 1)
 - Might be a way to jump out of shallow minima

Large parameters...

$$\frac{1}{1 + e^{-ax}}$$



- Maximum likelihood solution: prefers higher weights
 - higher likelihood of (properly classified) examples close to decision boundary
 - larger influence of corresponding features on decision
 - *can cause overfitting!!!*
- Regularization: penalize high weights

Why small weights correspond to simpler solution/function?

- Remember the Occam's razor?
- What does it mean by simple functions?
- Image a linear classifier:

$$a = \left[\sum_{d=1}^D w_d x_d \right] + b$$

- Large w_d might changes a a lot when x_d only changes a little, so high variance
- Simple functions mean small variance
- High variance has higher chance to overfit (the bias and variance tradeoff)

Weight Regularization

Q: How do we prevent overfitting for parametric models?

A: Adjust our inductive bias with a **regularization function** which penalizes large weights (or anything else we wish to avoid).

Regularized loss function:

$$\min_{w,b} \quad \sum_n \ell(y_n, w \cdot x_n + b) + \lambda R(w, b)$$

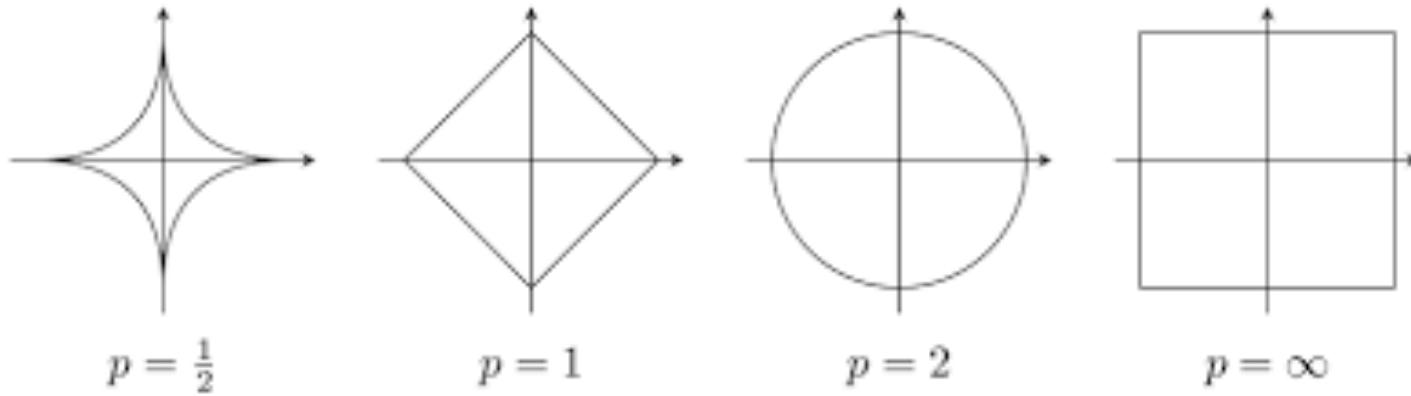
P-Norms

- Common loss functions:
 - L₂ norm of weight vector (square root of sum of squared weights)
 - L₁ norm of weight vector (sum of absolute weights)
- Generalizing these:

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

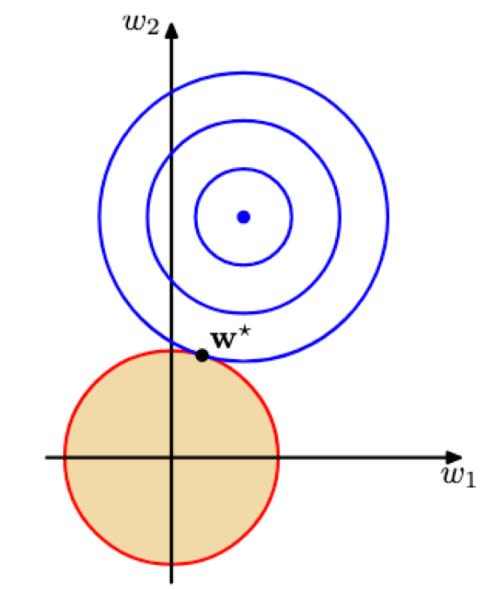
What if $p = 0$?

P-Norms



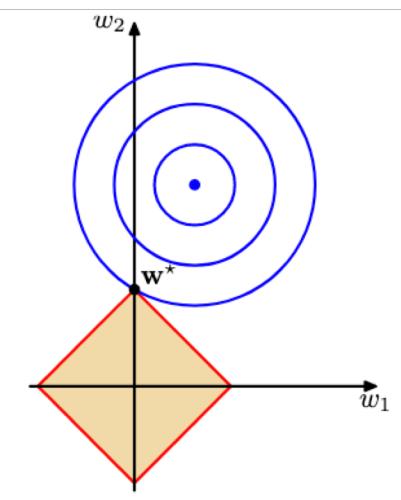
Isosurfaces: $\|w\|_p = \text{const}$

L2 Regularization (Ridge Regression)



- If λ is at a “good” value, regularization helps to avoid overfitting
- Choosing λ may be hard: cross-validation is often used
- If there are irrelevant features in the input (i.e. features that do not affect the output), L_2 will give them small, but non-zero weights.
- Ideally, irrelevant input should have weights exactly equal to 0.

L1 Regularization (Lasso)



- If λ is big enough, the circle is very likely to intersect the diamond at one of the corners
- This makes L_1 regularization much more likely to make some weights *exactly* 0

- If there are irrelevant input features, Lasso is likely to make their weights 0, while L_2 is likely to just make all weights small
- Lasso is biased towards providing *sparse solutions* in general
- Lasso optimization is computationally more expensive than L_2
- More efficient solution methods have to be used for large numbers of inputs (e.g. least-angle regression, 2003).

The p in L_p norm: a trade-off between convexity and sparsity

Justify L1 and L2 with Bayes Learning

$$h_{MAP} = \arg \max_{h \in \mathcal{H}} P(D|h)P(h)$$

- We assume all the hypothesis are equally likely to obtain the maximum likelihood principal with Bayes Learning
- If we impose some priors over $P(h)$, the solution will be different.

Maximum A Posteriori

$$h_{MAP} = \arg \max_{h \in \mathcal{H}} P(D|h)P(h)$$

- Common priors on \mathbf{w}

- Normal distribution, zero mean, diagonal covariance

$$p(\mathbf{w}) = \prod_i \frac{1}{\kappa\sqrt{2\pi}} e^{\frac{-w_i^2}{2\kappa^2}}$$

- Laplace distribution

$$p(w) = \frac{1}{2} \prod_i e^{-|w_i|}$$

- Normal Distribution: $\log p(w) = \frac{-w^2}{2\kappa^2} + \text{constants} = \text{L2 regularizer}$
- Laplace distribution: $\log p(w) = -|w| + \text{constants} = \text{L1 regularizer.}$