# Lecture 4: Types of errors. Bayesian regression models. Logistic regression

- A Bayesian interpretation of regularization
- Bayesian vs maximum likelihood fitting more generally

# The anatomy of the error of an estimator

- Suppose we have examples $\langle \mathbf{x}, y \rangle$ where $y = f(\mathbf{x}) + \epsilon$ and $\epsilon$ is Gaussian noise with zero mean and standard deviation $\sigma$
- We fit a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$, such as to minimize sum-squared error over the training data:

$$\sum_{i=1}^{m} (y_i - h(\mathbf{x}_i))^2$$

- Because of the hypothesis class that we chose (hypotheses linear in the parameters) for some target functions $f$ we will have a *systematic prediction error*
- Even if $f$ were truly from the hypothesis class we picked, depending on the data set we have, the parameters $\mathbf{w}$ that we find may be different; this *variability* due to the specific data set on hand is a different source of error

# Bias-variance analysis

- Given a new data point $\mathbf{x}$, what is the *expected prediction error*?
- Assume that the data points are drawn *independently and identically distributed (i.i.d.)* from a unique underlying probability distribution $P(\langle \mathbf{x}, y \rangle) = P(\mathbf{x})P(y|\mathbf{x})$
- The goal of the analysis is to compute, for an arbitrary given point $\mathbf{x}$,

$$E_P\left[(y - h(\mathbf{x}))^2 | \mathbf{x}\right]$$

  where $y$ is the value of $\mathbf{x}$ in a data set, and the expectation is over all training sets of a given size, drawn according to $P$
- For a given hypothesis class, we can also compute the *true error*, which is the expected error over the input distribution:

$$\sum_{\mathbf{x}} E_P\left[(y - h(\mathbf{x}))^2 | \mathbf{x}\right] P(\mathbf{x})$$

  (if $\mathbf{x}$ continuous, sum becomes integral with appropriate conditions).
- We will decompose this expectation into three components

# Recall: Statistics 101

- Let $X$ be a random variable with possible values $x_i, i = 1 \ldots n$ and with probability distribution $P(X)$
- The *expected value* or *mean* of $X$ is:

$$E[X] = \sum_{i=1}^{n} x_i P(x_i)$$

- If $X$ is continuous, roughly speaking, the sum is replaced by an integral, and the distribution by a density function
- The *variance* of $X$ is:

$$\begin{aligned} Var[X] &= E[(X - E(X))^2] \\ &= E[X^2] - (E[X])^2 \end{aligned}$$

# The variance lemma

$$
\begin{aligned}
Var[X] &= E[(X - E[X])^2] \\
&= \sum_{i=1}^{n} (x_i - E[X])^2 P(x_i) \\
&= \sum_{i=1}^{n} (x_i^2 - 2x_i E[X] + (E[X])^2) P(x_i) \\
&= \sum_{i=1}^{n} x_i^2 P(x_i) - 2E[X] \sum_{i=1}^{n} x_i P(x_i) + (E[X])^2 \sum_{i=1}^{n} P(x_i) \\
&= E[X^2] - 2E[X]E[X] + (E[X])^2 \cdot 1 \\
&= E[X^2] - (E[X])^2
\end{aligned}
$$

We will use the form:

$$E[X^2] = (E[X])^2 + Var[X]$$

# Bias-variance decomposition

- Simple algebra:

$$\begin{aligned} E_P\left[(y - h(\mathbf{x}))^2 | \mathbf{x}\right] &= E_P\left[(h(\mathbf{x}))^2 - 2yh(\mathbf{x}) + y^2 | \mathbf{x}\right] \\ &= E_P\left[(h(\mathbf{x}))^2 | \mathbf{x}\right] + E_P\left[y^2 | \mathbf{x}\right] - 2E_P[y|\mathbf{x}]E_P\left[h(\mathbf{x})|\mathbf{x}\right] \end{aligned}$$

- Let $\bar{h}(\mathbf{x}) = E_P[h(\mathbf{x})|\mathbf{x}]$ denote the *mean prediction* of the hypothesis at $\mathbf{x}$, when $h$ is trained with data drawn from $P$
- For the first term, using the variance lemma, we have:

$$E_P[(h(\mathbf{x}))^2|\mathbf{x}] = E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2|\mathbf{x}] + (\bar{h}(\mathbf{x}))^2$$

- Note that $E_P[y|\mathbf{x}] = E_P[f(\mathbf{x}) + \epsilon|\mathbf{x}] = f(\mathbf{x})$ (because of linearity of expectation and the assumption on $\epsilon \sim \mathcal{N}(0, \sigma)$)
- For the second term, using the variance lemma, we have:

$$E[y^2|\mathbf{x}] = E[(y - f(\mathbf{x}))^2|\mathbf{x}] + (f(\mathbf{x}))^2$$

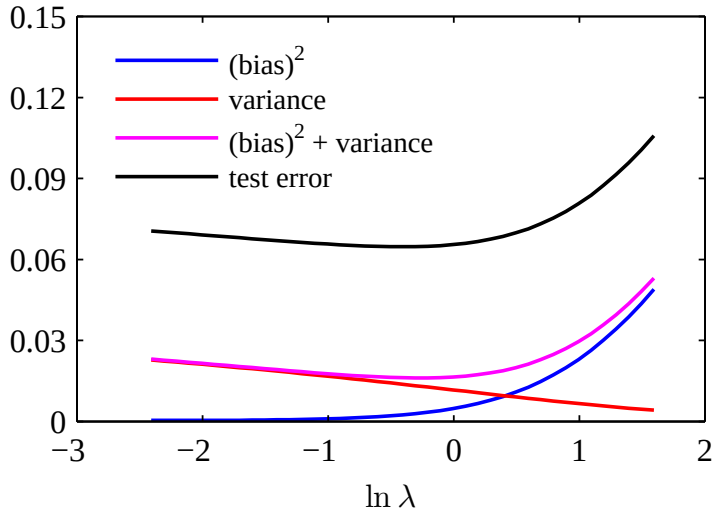
# Bias-variance decomposition (2)

- Putting everything together, we have:

$$\begin{aligned} E_P\left[(y - h(\mathbf{x}))^2 | \mathbf{x}\right] &= E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}] + (\bar{h}(\mathbf{x}))^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) \\ &+ E_P[(y - f(\mathbf{x}))^2 | \mathbf{x}] + (f(\mathbf{x}))^2 \\ &= E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}] + (f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 \\ &+ E[(y - f(\mathbf{x}))^2 | \mathbf{x}] \end{aligned}$$

- The first term, $E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 | \mathbf{x}]$, is the *variance* of the hypothesis $h$ at $\mathbf{x}$, when trained with finite data sets sampled randomly from $P$
- The second term, $(f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2$, is the *squared bias* (or systematic error) which is associated with the class of hypotheses we are considering
- The last term, $E[(y - f(\mathbf{x}))^2 | \mathbf{x}]$ is the *noise*, which is due to the problem at hand, and cannot be avoided

# Error decomposition



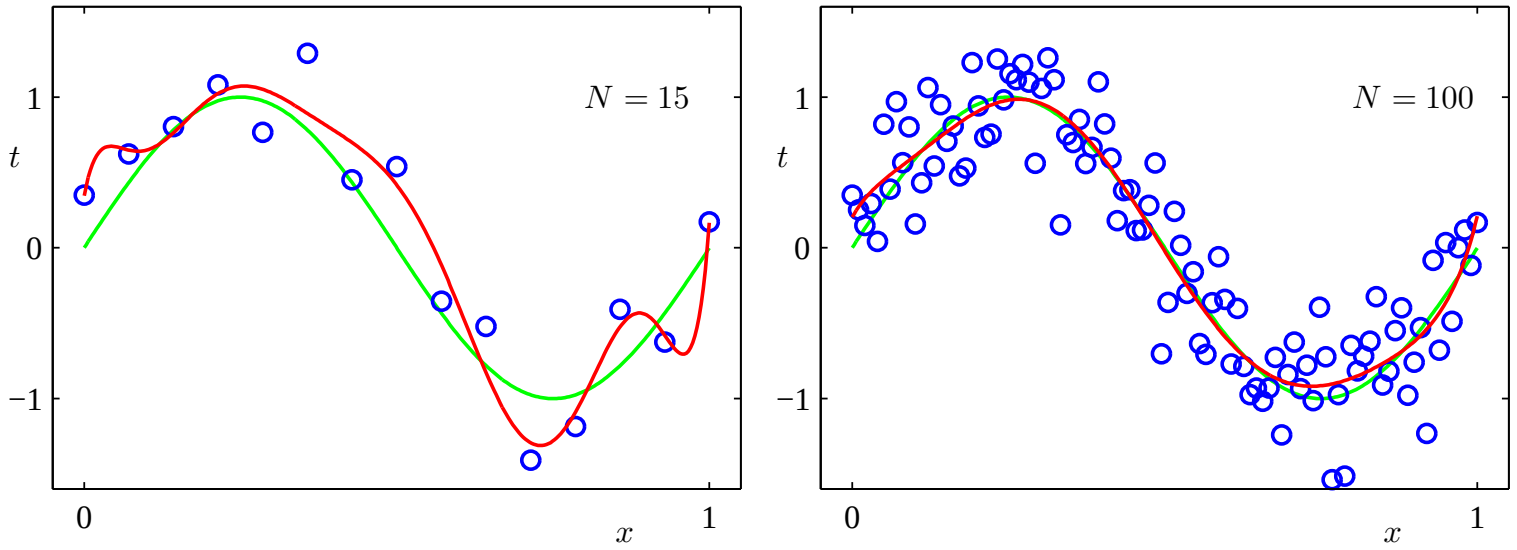

- The bias-variance sum approximates well the test error over a set of 1000 points
- x-axis measures the hypothesis complexity (decreasing left-to-right)
- Simple hypotheses usually have high bias (bias will be high at many points, so it will likely be high for many possible input distributions)
- Complex hypotheses have high variance: the hypothesis is very dependent on the data set on which it was trained.

# Bias-variance trade-off

- Typically, bias comes from not having good hypotheses in the considered class
- Variance results from the hypothesis class containing "too many" hypotheses
- MLE estimation is typically unbiased, but has high variance
- Bayesian estimation is biased, but typically has lower variance
- Hence, we are faced with a *trade-off*: choose a more expressive class of hypotheses, which will generate higher variance, or a less expressive class, which will generate higher bias
- Making the trade-off has to depend on the amount of data available to fit the parameters (data usually mitigates the variance problem)

# More on overfitting

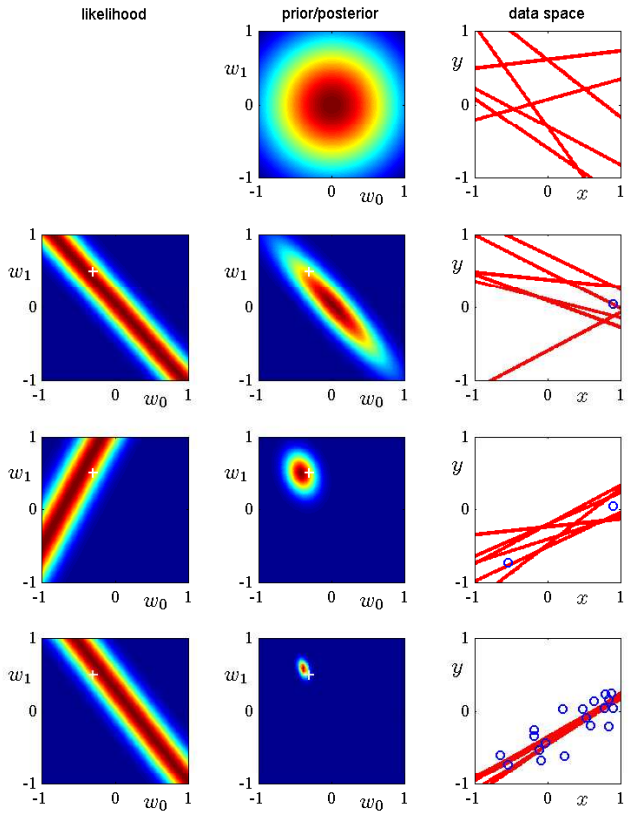
- Overfitting depends on the amount of data, relative to the complexity of the hypothesis
- With more data, we can explore more complex hypotheses spaces, and still find a good solution



# Bayesian view of regularization

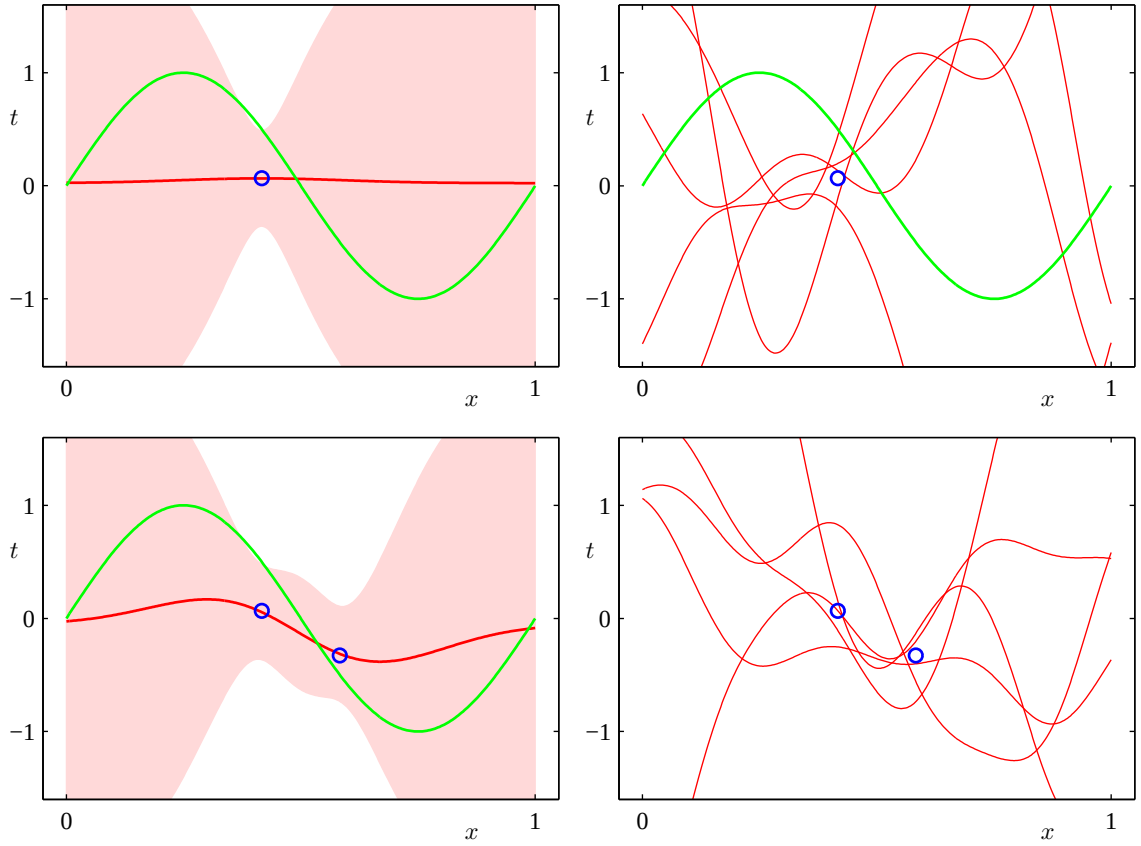
- Start with a *prior distribution* over hypotheses
- As data comes in, compute a *posterior distribution*
- We often work with *conjugate priors*, which means that when combining the prior with the likelihood of the data, one obtains the posterior in the same form as the prior
- Regularization can be obtained from particular types of prior (usually, priors that put more probability on simple hypotheses)
- E.g. $L_2$ regularization can be obtained using a circular Gaussian prior for the weights, and the posterior will also be Gaussian
- E.g. $L_1$ regularization uses double-exponential prior (see (Tibshirani, 1996))

# Bayesian view of regularization



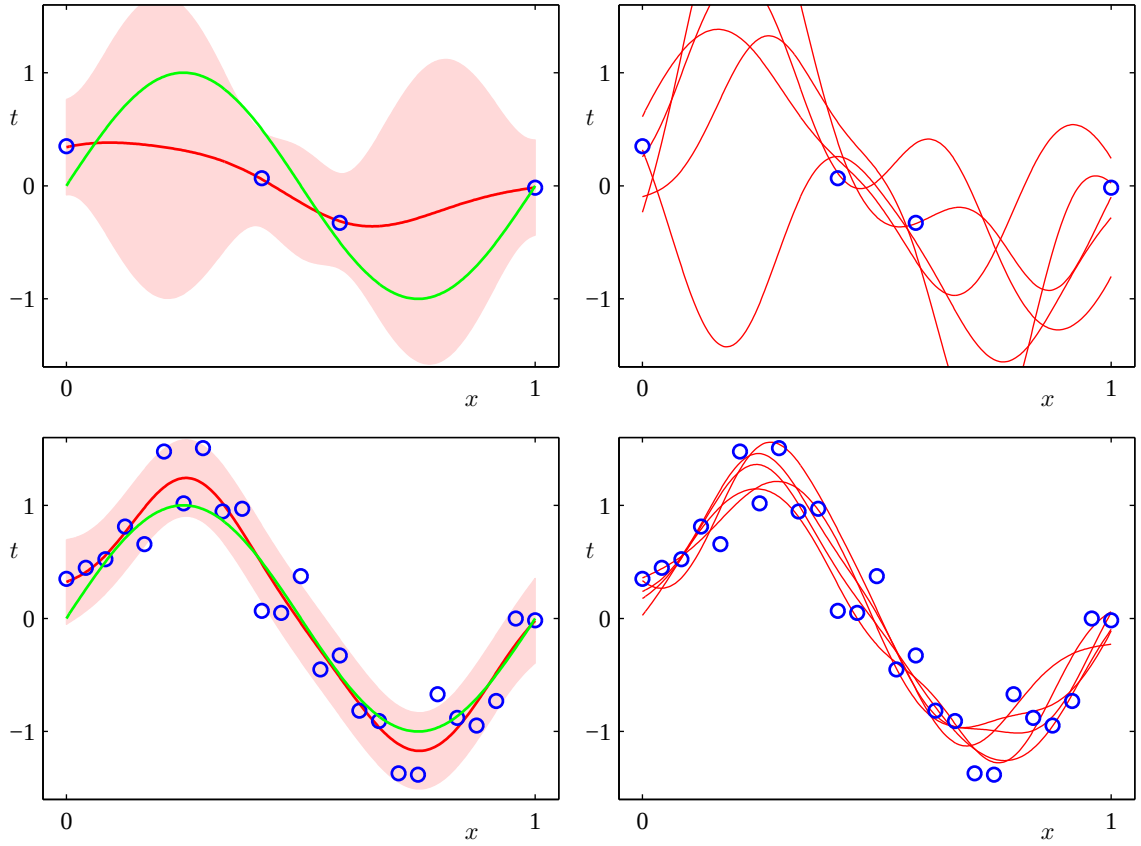

- Prior is round Gaussian
- Posterior will be skewed by the data

# What does the Bayesian view give us?




- Circles are data points
- Green is the true function
- Red lines on right are drawn from the posterior distribution

# What does the Bayesian view give us?




- Functions drawn from the posterior can be very different
- Uncertainty decreases where there are data points

# What does the Bayesian view give us?

- Uncertainty estimates, i.e. how sure we are of the value of the function
- These can be used to guide active learning: ask about inputs for which the uncertainty in the value of the function is very high
- In the limit, Bayesian and maximum likelihood learning converge to the same answer
- In the short term, one needs a good prior to get good estimates of the parameters
- Sometimes the prior is overwhelmed by the data likelihood too early.
- Using the Bayesian approach does NOT eliminate the need to do cross-validation in general
- More on this later...

# Logistic regression

- Suppose we represent the hypothesis itself as a logistic function of a linear combination of inputs:

$$h(\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}$$

  This is also known as a *sigmoid neuron*
- Suppose we interpret $h(\mathbf{x})$ as $P(y = 1|\mathbf{x})$
- Then the log-odds ratio,

$$\ln \left( \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} \right) = \mathbf{w}^T \mathbf{x}$$

  which is linear (nice!)
- The optimum weights will maximize the *conditional likelihood* of the outputs, given the inputs.

# The cross-entropy error function

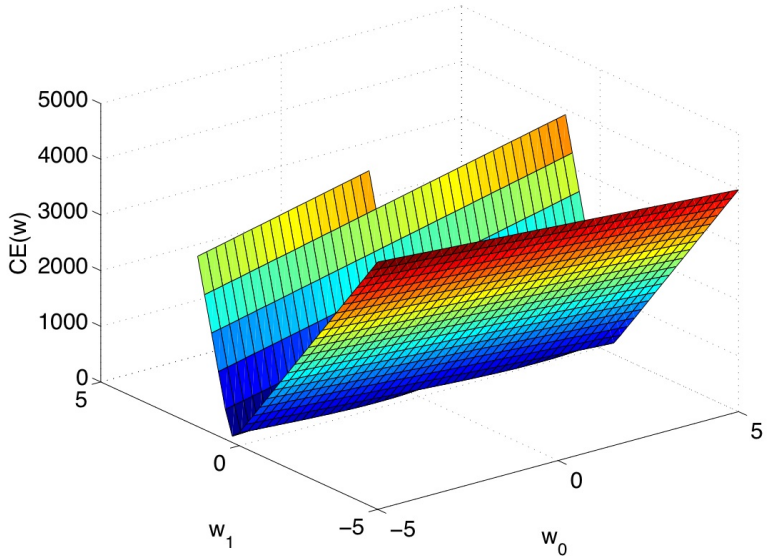
- Suppose we interpret the output of the hypothesis, $h(\mathbf{x}_i)$, as the probability that $y_i = 1$
- Then the log-likelihood of a hypothesis $h$ is:

$$\begin{aligned}\log L(h) &= \sum_{i=1}^{m} \log P(y_i|\mathbf{x}_i, h) = \sum_{i=1}^{m} \begin{cases} \log h(\mathbf{x}_i) & \text{if } y_i = 1 \\ \log(1 - h(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases} \\ &= \sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i))\end{aligned}$$

- The *cross-entropy error function* is the opposite quantity:

$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i))\right)$$

# Cross-entropy error surface for logistic function

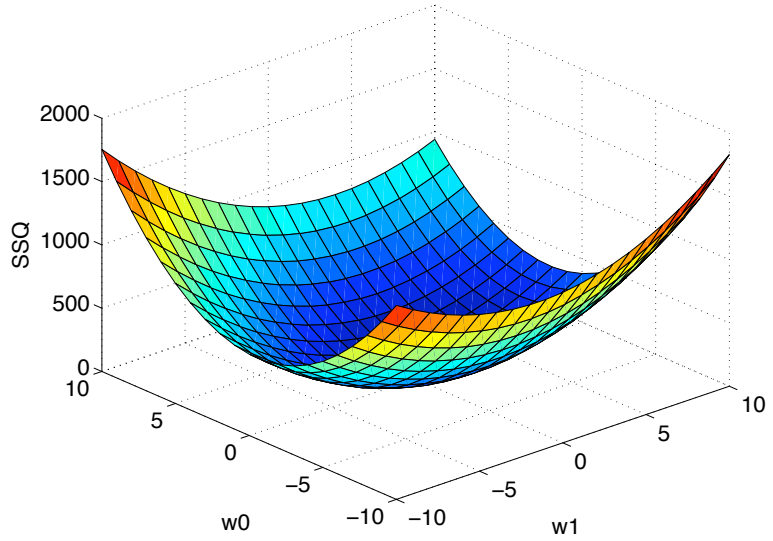


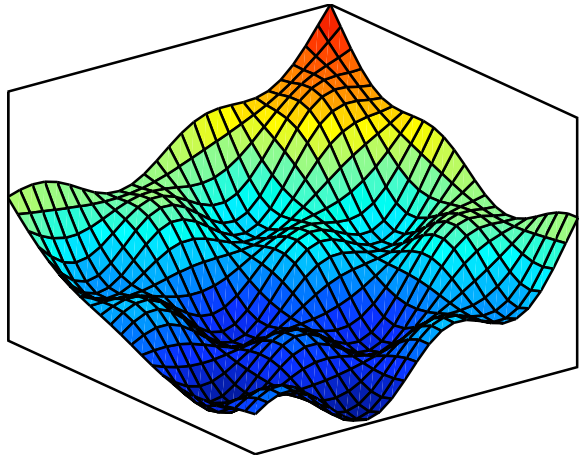

$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))\right)$$

Nice error surface, unique minimum, but *cannot solve in closed form*

# Gradient descent

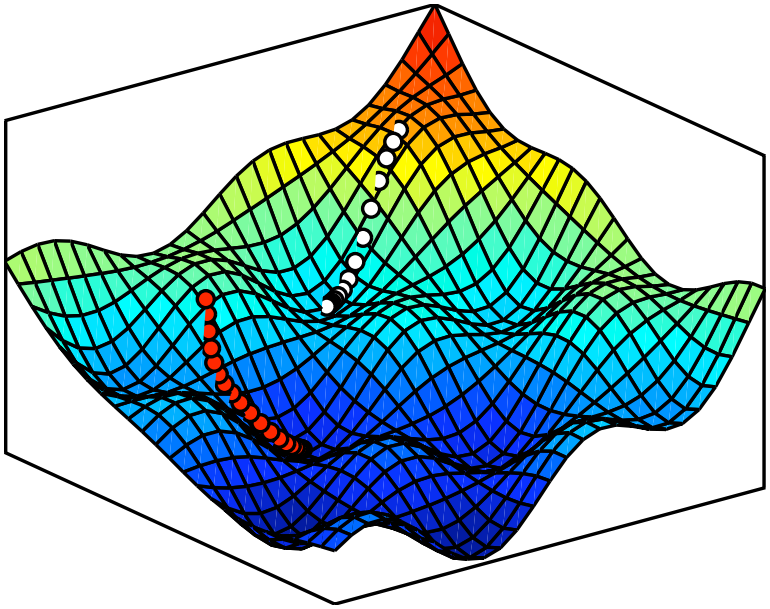
- The gradient of $J$ at a point $\mathbf{w}$ can be thought of as a vector indicating which way is "uphill".



- If this is an error function, we want to move "downhill" on it, i.e., in the direction opposite to the gradient

# Example gradient descent traces



- For more general hypothesis classes, there may be may local optima
- In this case, the final solution may depend on the initial parameters

# Gradient descent algorithm

- The basic algorithm assumes that $\nabla J$ is easily computed
- We want to produce a sequence of vectors $\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3, \ldots$ with the goal that:
  - $J(\mathbf{w}^1) > J(\mathbf{w}^2) > J(\mathbf{w}^3) > \ldots$
  - $\lim_{i \to \infty} \mathbf{w}^i = \mathbf{w}$ and $\mathbf{w}$ is locally optimal.
- The algorithm: Given $\mathbf{w}^0$, do for $i = 0, 1, 2, \ldots$

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \alpha_i \nabla J(\mathbf{w}^i) \ ,$$

where $\alpha_i > 0$ is the *step size* or *learning rate* for iteration $i$.

# Maximization procedure: Gradient ascent

- First we compute the gradient of $\log L(\mathbf{w})$ wrt $\mathbf{w}$:

$$\nabla \log L(\mathbf{w}) = \sum_i y_i \frac{1}{h_{\mathbf{w}}(\mathbf{x}_i)} h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i$$

$$+ (1 - y_i) \frac{1}{1 - h_{\mathbf{w}}(\mathbf{x}_i)} h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i(-1)$$

$$= \sum_i \mathbf{x}_i(y_i - y_i h_{\mathbf{w}}(\mathbf{x}_i) - h_{\mathbf{w}}(\mathbf{x}_i) + y_i h_{\mathbf{w}}(\mathbf{x}_i)) = \sum_i (y_i - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i$$

- The update rule (because we maximize) is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla \log L(\mathbf{w}) = \mathbf{w} + \alpha \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x_i}))\mathbf{x_i} = \mathbf{w} + \alpha \mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}})$$

where $\alpha \in (0, 1)$ is a step-size or learning rate parameter

- This is called *logistic regression*
- If one uses features of the input, we have:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}})$$

# Another algorithm for optimization

- Recall Newton's method for finding the zero of a function $g : \mathbb{R} \to \mathbb{R}$
- At point $w^i$, approximate the function by a straight line (its tangent)
- Solve the linear equation for where the tangent equals $0$, and move the parameter to this point:

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

# Application to machine learning

- Suppose for simplicity that the error function $J$ has only one parameter
- We want to optimize $J$, so we can apply Newton's method to find the zeros of $J' = \frac{d}{dw}J$
- We obtain the iteration:

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

- Note that there is *no step size parameter*!
- This is a *second-order method*, because it requires computing the second derivative
- But, if our error function is quadratic, this will find the global optimum in one step!

# Second-order methods: Multivariate setting

- If we have an error function $J$ that depends on many variables, we can compute the *Hessian matrix*, which contains the second-order derivatives of $J$:

$$H_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

- The inverse of the Hessian gives the "optimal" learning rates
- The weights are updated as:

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

- This is also called Newton-Raphson method for logistic regression, or Fisher scoring

# Which method is better?

- Newton's method usually requires significantly fewer iterations than gradient descent
- Computing the Hessian requires a batch of data, so there is no natural on-line algorithm
- Inverting the Hessian explicitly is expensive, but almost never necessary
- Computing the product of a Hessian with a vector can be done in linear time (Schraudolph, 1994)

# Newton-Raphson for logistic regression

- Leads to a nice algorithm called *iterative recursive least squares*
- The Hessian has the form:

$$\mathbf{H} = \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi}$$

  where $\mathbf{R}$ is the diagonal matrix of $h(\mathbf{x}_i)(1 - h(\mathbf{x}_i))$ (you can check that this is the form of the second derivative.
- The weight update becomes:

$$\mathbf{w} \leftarrow (\mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{R} (\mathbf{\Phi}\mathbf{w} - \mathbf{R}^{-1}(\mathbf{\Phi}\mathbf{w} - \mathbf{y}))$$

# Regularization for logistic regression

- One can do regularization for logistic regression just like in the case of linear regression
- Recall regularization makes a statement about the weights, so does not affect the error function
- Eg: $L_2$ regularization will have the optimization criterions:

$$J(\mathbf{w} = J_D(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$