Value Function Approximation

Mark Schmidt

March 10, 2007

Outline

- Context
- Linear Models
- Neural Networks
- Kernel Machines
- Advanced Probabilistic Methods

Notation for ADP

- time: t
- state: x_t
- noise: w_t
- decision: u_t
- probability of transitioning to state y if decision u is made in state x: $P_{xy}(u)$
- policy: $\pi = \{u_1, u_2, ..., u_n\}$
- cost: $g(x_t, u_t(x_t), w_t)$
- goal: $\min_{\pi} \mathbf{E}_w [\sum_t \gamma_t g(x_t, u_t(x_t), w_t)]$
- state value/cost-to-go: $J_\pi(x_0) = \mathbf{E}_w[\sum_{t=0}^{\infty} \gamma^t g_t(x_t, u_t(x_t), w_t)]$
- state-action value/cost-to-go function:

$$Q_\pi(x_0, u(x_0)) = \mathbf{E}_w[\sum_{t=0}^{\infty} \gamma^t g_t(x_t, u_t(x_t), w_t)]$$

Where we are now...

	{P,g} Known:	{P,g} Unknown:	
Prediction:	DP	TD	
Control:	DP	On-Policy:	Off-Policy:
		Sarsa	Q-Learn

- {P,g} Unknown methods can be applied in {P,g} Known cases
- Q only needed in P-Unknown/Control scenarios (unless can't compute expectation)
- TD: Evaluation of Fixed Policy (can later run TD w/ revised policy)
- Sarsa: On-Policy but NOT fixed (depends on Q)
- Sarsa vs. Q: When updating Q, Sarsa computes TD w/ action from new state under current policy, Q maximizes over next action
- TD(lambda), Sarsa(lambda), Q(lambda): used to update past few states/state-actions

Motivating Function Approximation

- In many applications, space of states for $J_\pi(x)$ or state-action pairs for $Q_\pi(x, u(x))$ is too large (curse of dimensionality)
- In this case, make use an approximate value/Q function parametrized in terms of a (smaller) vector r:
- $\tilde{J}(x, r) \approx J^*(x)$
- $\tilde{Q}(x, u(x), r) \approx Q^*(x, u(x))$
- Example:
  - $\tilde{J}(x, r) = r^T x$ (approximate value function is bilinear in state representation x and parameters r)
  - Approximate value $\tilde{J}(x, r)$ generated only when needed

FA for ADP

- Two Issues:
  - (1) Decide general structure of $\tilde{J}(x, r)$ (approximation architecture)
  - (2) Calculate r so as to minimize some measure of error between $J^*(x)$ and $\tilde{J}(x, r)$ (parameter estimation)
- This talk will focus on issue (1), next slide sketches how to address (2).

- One possible error function is the (weighted) squared error:
  - $\sum_{s \in \mathcal{S}} p(x_s)[J^*(x_s) - \tilde{J}(x_s, r)]^2$
  - $p(x_s)$ might be 'on-policy' distribution
- Problem 1: we may be going through data set sequentially:
  - First-Order Update for differentiable $\tilde{J}(x_s, r)$:
  - $$\begin{aligned} r_{t+1} &= r_t - \frac{\alpha}{2} \nabla_r [J^*(x_t) - \tilde{J}(x_t, r_t)]^2 \\ &= r_t - \alpha [J^*(x_t) - \tilde{J}(x_t, r_t)] \nabla_r \tilde{J}(x_t, r_t) \end{aligned}$$
- Problem 2: we don't have $J^*(x_s)$:
  - Approximate update: $$r_{t+1} = r_t - \frac{\alpha}{2} \nabla_r [\bar{J}(x_t) - \tilde{J}(x_t, r_t)]^2$$
  - Examples of $\bar{J}(x_t)$:
    - DP: $\mathbf{E}_w [g(x_t, u_t(x_t), w_t) + \gamma J_{t+1}(f_t(x_t, u_t(x_t), w_t))]$
    - TD(0): $g(x_t, u_t(x_t), w_t) + \gamma J_{t+1}(f_t(x_t, u_t(x_t), w_t))$
    - Monte Carlo: $G(x_t)$ (average cost of following policy after x_t)

Biased vs. Unbiased

- Unbiased (Monte Carlo):
  - can find local optimum of MSE
- Biased, on-policy (TD, Sarsa):
  - in some cases, can bound distance to MSE, and decrease to 0
- Biased, off-policy (DP, Q):
  - may diverge

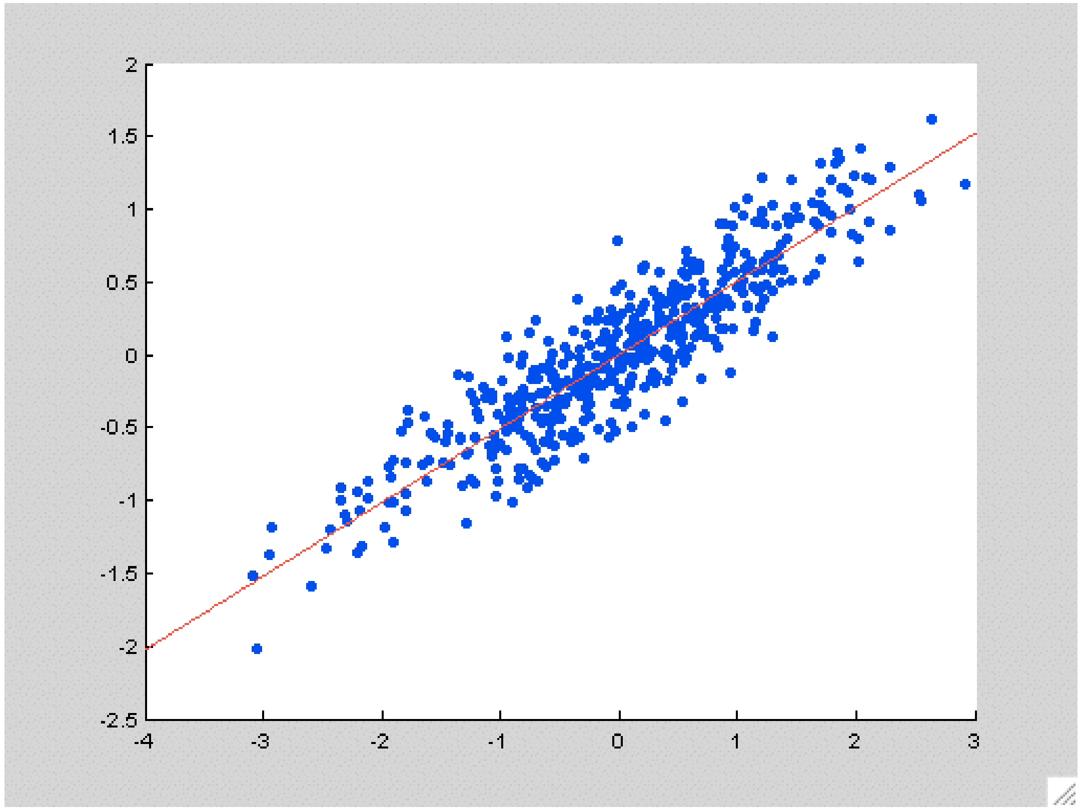
Outline

- Context
- <u>Linear Models</u>
- Neural Networks
- Kernel Machines
- Advanced Probabilistic Methods

Least Squares

- We are now able to formulate the MSE problem as:
  - $\min_r \sum_t (\bar{J}(x_t) - \tilde{J}(x_t, r))^2$
  - with bilinear model $\tilde{J}(x_t, r) := r^T x_t$, we can turn this into a standard Least Squares problem:
    - $y_t := \bar{J}(x_t)$
    - $w := r$
    - $x_t := x_t$
  - Least Squares:
    - $\min_w \sum_t (w^T x_t - y_t)^2$
  - Weighted Least Squares:
    - $\min_w \sum_t p(x_t)(w^T x_t - y_t)^2$

Least Squares Regression



Least Squares fit to data

Standard Least Squares Solution

- Re-write in matrix notation:

$$f(x,w,y) = (Xw-y)^T(Xw-y) = w^TX^TXw - 2w^TX^Ty + y^Ty$$

  - Use first optimality conditions: $\nabla_w f(X,w,y) = 0$

$$0 = 2X^TXw - 2X^Ty$$

$$X^TXw = X^Ty$$ (Normal equations)

- Show that this is min using second order condition:

$$\nabla_w^2 f(X,w,y) = 2X^TX \geq 0$$

- (adding weights is is easy)
- Nice theoretical properties (CLT, consistent, CR-b, MVUE, etc.)
- In many scenarios we will NOT be using the Normal equations

Stochastic Gradient Descent

- We may be accessing the pairs (x_t, y_t) sequentially, we may have an immense/infinite amount of data, or dynamics may change over time
- In these cases, we may want to build FA as we go:
  - Stochastic Gradient Update:

    $$w := w - \alpha \nabla_w F(x_t, w, y_t)$$

    - For convergence, need SA conditions:

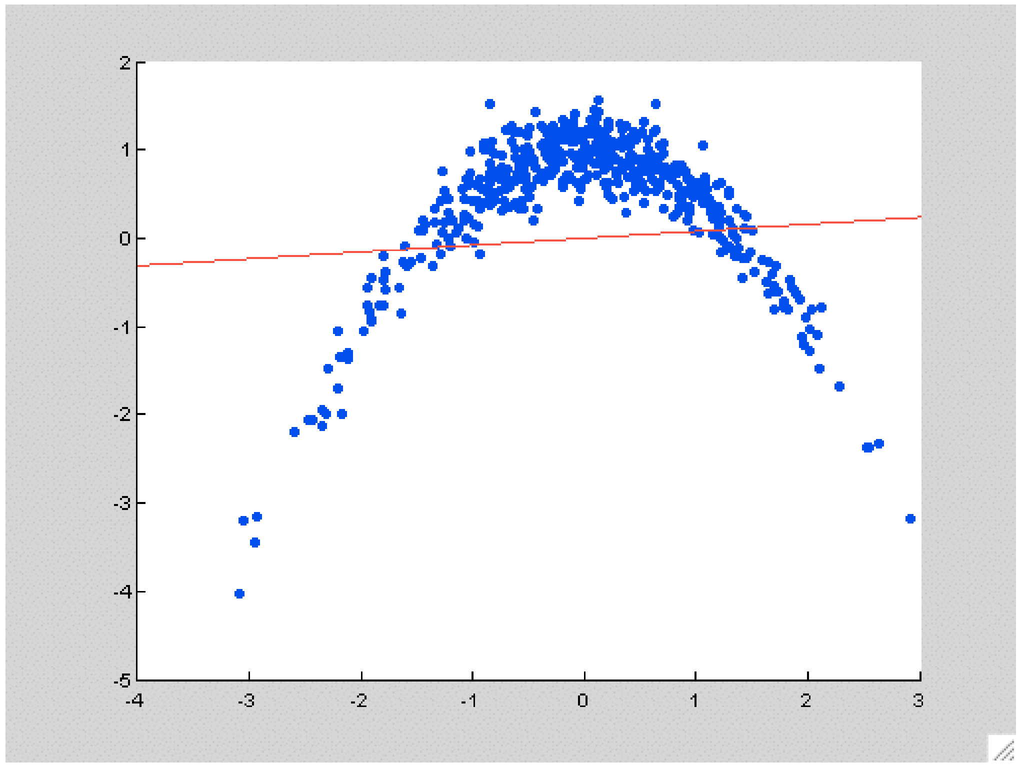
      $$\sum_{i=1}^{\infty} \alpha_i = \infty \quad \sum_{i=1}^{\infty} \alpha_i^2 < \infty$$

    - in practice, alpha chosen by heuristically (one method: test an alpha out for a while and see how well it works)
    - steepest descent, but different behaviour from on-line updates

Basis Functions

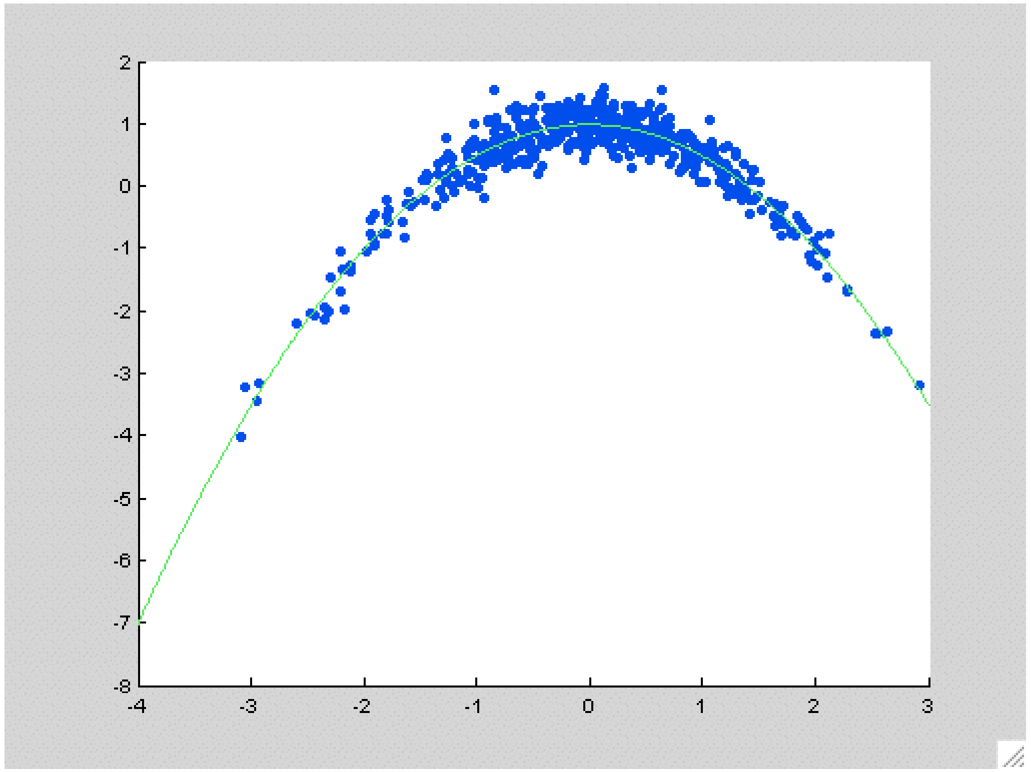
- What if relation $(x + \epsilon)Ry$ is non-linear?
- Model non-linear effects in a linear model using change of basis ('basis functions')
- Example: instead of x_t, use $[1 \quad x_t \quad x_t^2]^T$
- Still linear in w and nothing changes in solving for w
- Basically the same as having a different representation for the state
- Some common basis functions: polynomials, radial basis functions, splines, wavelets, etc.

Basis Functions



Least Squares on original basis

x_t



Least Squares on expanded basis

$[1 \quad x_t \quad x_t^2]^T$

Regularization

- If we use a sufficiently expressive basis, we can approximate any function
- If we have a basis that is too expressive (relative to amount of data), we can fit not only the desired function but noise in process

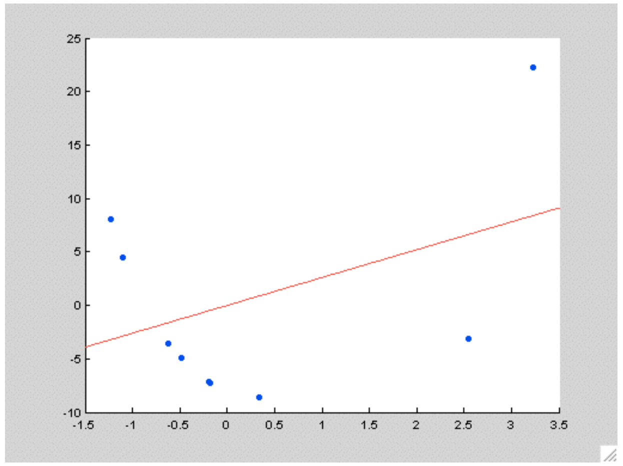
- Regularization: assign a penalty function $R(w)$ to each w:

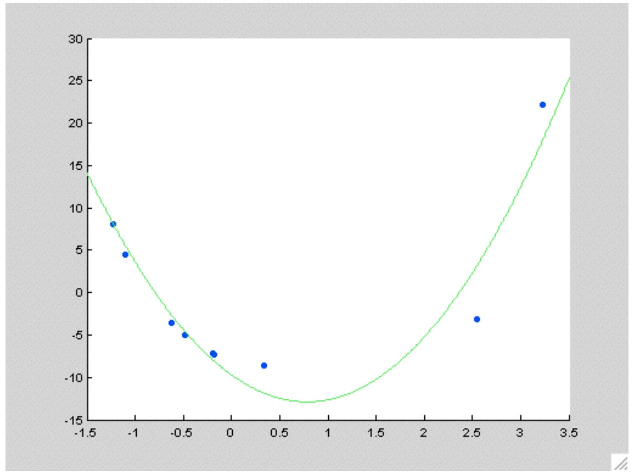
$$\min_w F(X, w, y) + R(w)$$

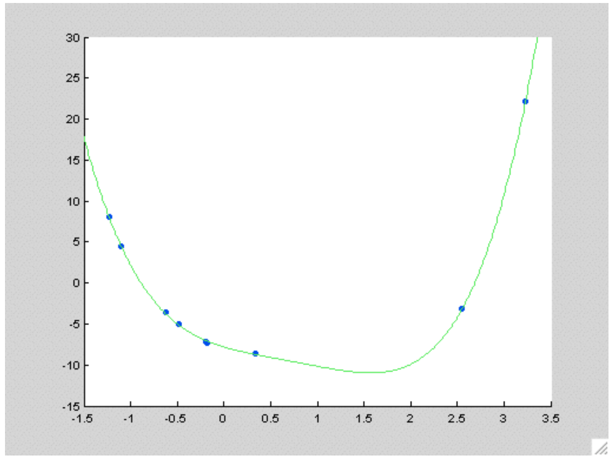
- Example of Regularization: penalizes weights by squared Euclidean distance from 0: $R(w) := \lambda \sum_{i=1}^{p} w_i^2$
- forces minimization to find balance between growing weights and fitting data

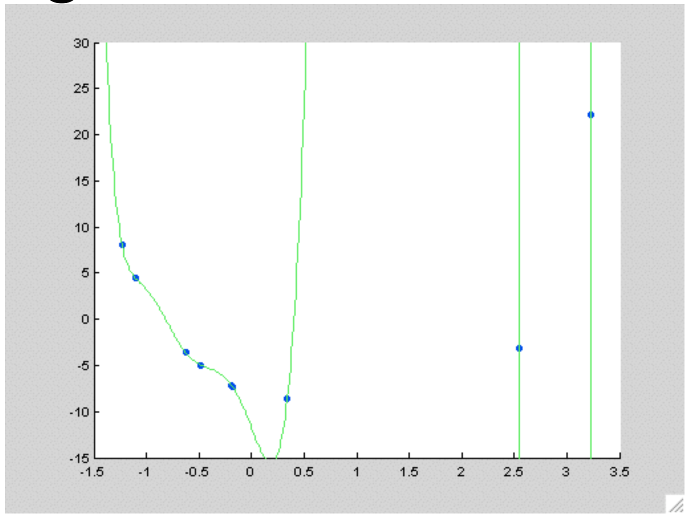
- various strategies to choose lambda (independent data samples, degrees of freedom, ∇F vs. ∇R , etc.)

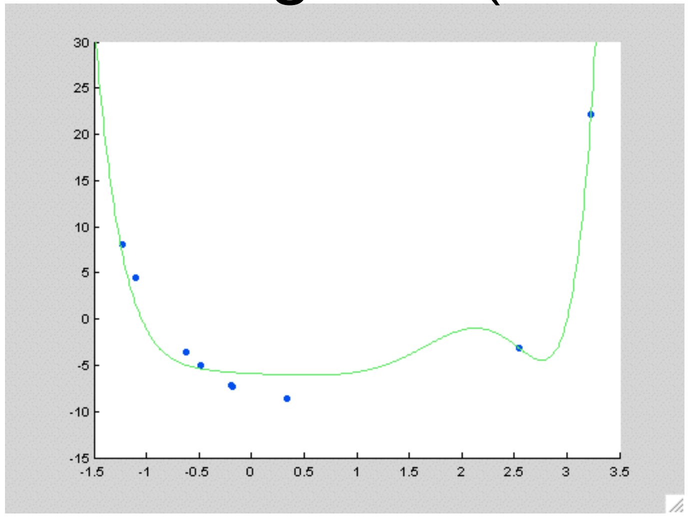
Regularizaiton


Original Basis


Degree 2


Degree 4 (correct)


Degree 8


Degree 8 (regularized)

Probabilistic View

- Add a constant $\frac{1}{2\sigma^2}$ to all terms and an additional constant Z:

$$Z + \sum_t \frac{1}{2\sigma^2}(w^T x_t - y_t)^2$$

- Take negation and exponentiate:

$$\frac{1}{Z}\prod_t exp(\frac{1}{2\sigma^2}(w^T x_t - y_t)^2)$$

- Least Squares w corresponds to max likelihood of model:

$$P(y|X,w) = \prod_t N(y_t|w^T x_t, \sigma^2)$$

- With regularizer, corresponds to MAP estimate:

$$P(y|X,w) = P(y,X|w)P(w) = \prod_t [N(y_t|w^T x_t, \sigma^2)] \prod_i [N(w|0,\lambda^2)]$$

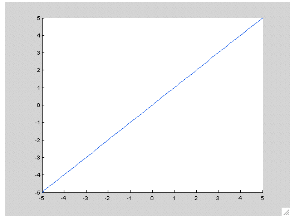
- Dual View:
  - min (Loss Function) + Regularizer
  - max (Likelihood)(Prior)
  - (exists third view based on maximum entropy)

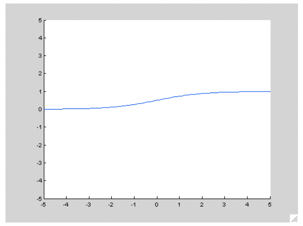
Other Losses/Regularizers

- The combination of a Least Squares error and Tikhonov regularization is not the only possiblity.
- Can mix and match various loss-regularizer/likelihood-prior combinations:
  - $L1$ (sparse, robust to outliers): $\sum_t |w^T x_t - y_t|$ $\sum_i |w_i|$
  - $L\infty$ (worse-case): $\max_t |w^T x_t - y_t|$ $\max_i |w_i|$
  - Student T (robust to outliers): $\sum_t -log(T(w^T x_t - y_t, \eta))$ $\sum_i -log(T(w_i, \eta))$
  - Min/Max Entropy methods, etc.

Discrete Output

- Suppose target is binary: $y_t \in \{0, 1\}$
- Rather than having $\hat{y} \in (\infty, \infty)$, we can use a sigmoid function to force output to be in range $(0, 1)$
- Logistic function is one example:





$$p(y_t = 1 | x_t, w) = \frac{1}{1 + \exp(-w^T x_t)}$$

$$p(y_t = 0 | x_t, w) = 1 - p(y_t = 1 | x_t, w)$$

- Logistic Regression: maximum likelihood w, or MAP w with a regularizer on w

Outline

- Context
- Linear Models
- <u>Neural Networks</u>
- Kernel Machines
- Advanced Probabilistic Methods

Learning the Basis

- We have assumed that output can be modeled as linear combination of basis
- What if it can't? What if we don't know the right basis? What if we do know the right basis but we can't compute/store it?
- Basic Idea behind Neural Networks:
  - Try to estimate a good basis!
  - Do this by composing linear models

Composing Linear Models

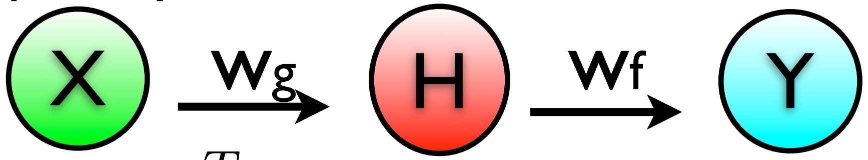
- Composing 1 function:

  $$f(g(x, r_g), r_f)$$

  - f, g are both parameterized linear models, where g has output in a fixed range (such as a sigmoid function)

- Example:
  - $f(x, w_f) := w_f x$
  - $g(x, w_g) := \frac{1}{1 + \exp(-w_g^T x)}$
  - $f(g(x, w_g), w_f) := w_f(\frac{1}{1 + \exp(-w_g^T x)})$

- Graphically:



- Weights $[w_g^T \quad w_f]$ optimized jointly

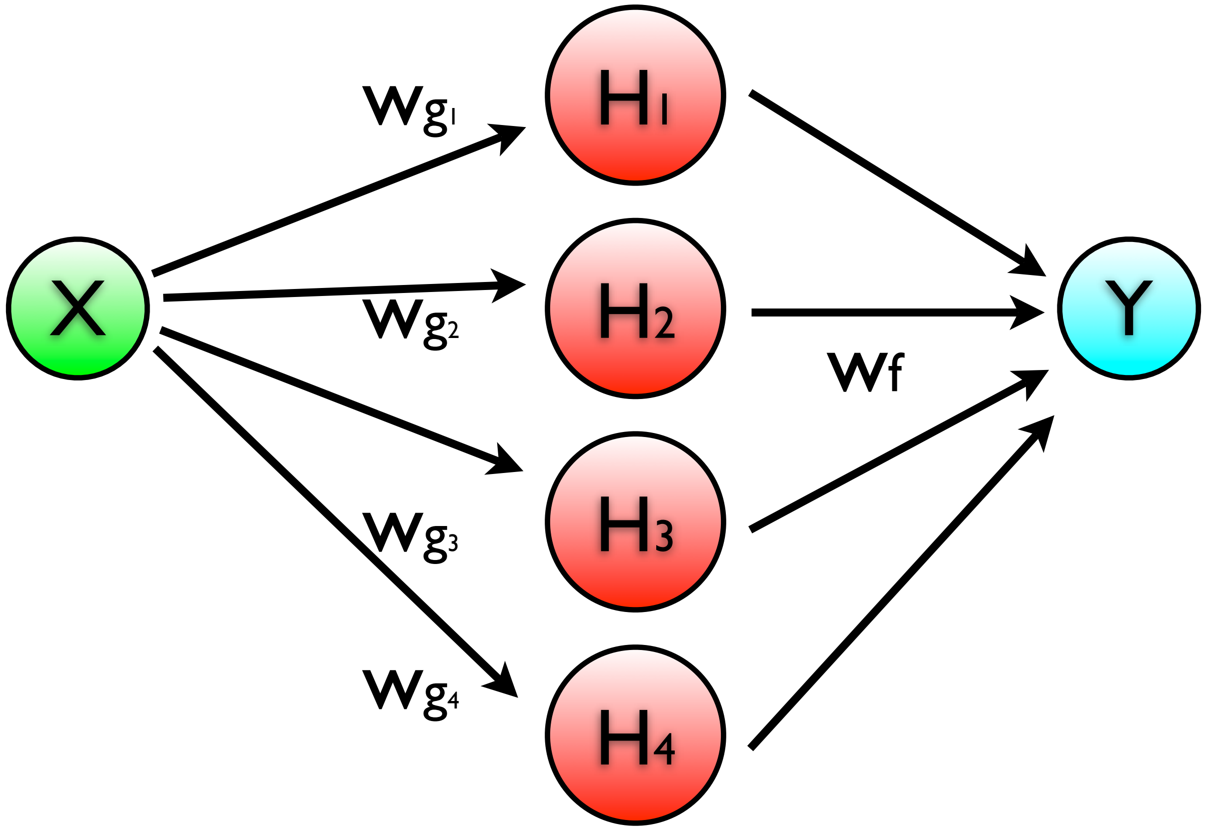
Neural Networks

- In some sense, 'H' compresses what the linear model g knows about the function into a scalar feature
- In Neural Networks, we seek to jointly learn multiple h values, that might form a better basis for representing our function than the original basis:

$$f(g_1(x, w_{g_1}), g_2(x, w_{g_2}), ..., g_n(x, w_{g_n}), w_f)$$

- Jointly optimize $\begin{bmatrix} w_f & w_{g_*} \end{bmatrix}$
- Early work motivated by ideas from neuroscience
- Also known as 'multi-layer perceptron' (perceptron: linear model)
- Specific instance of a model known in Stats as a 'mixture model'
- If g, f both give probabilistic output, then forms a (Sigmoid) 'Belief Network'

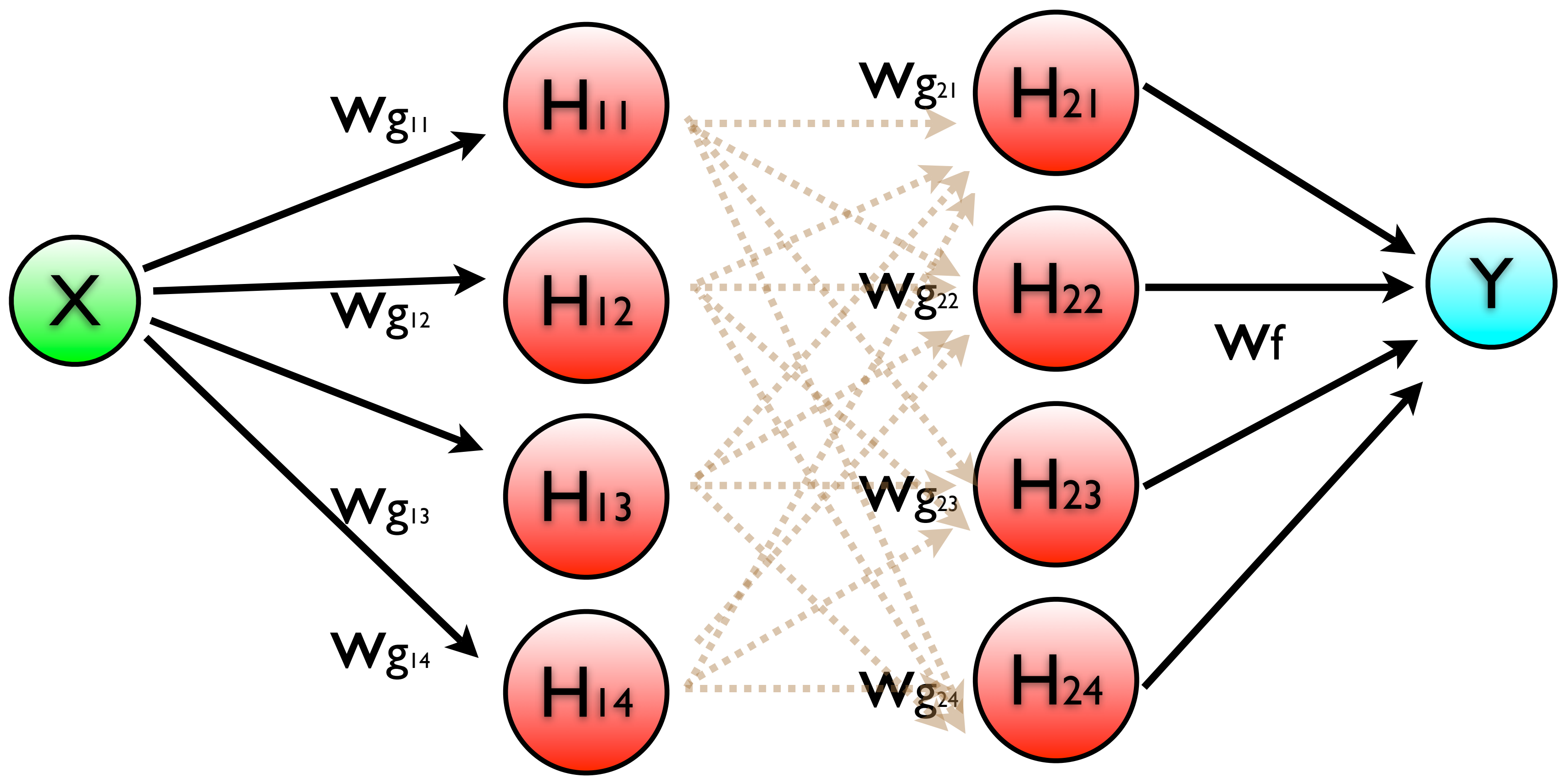
Neural Networks



Neural Networks

- Optimize parameters jointly using SGD, as before
- Optimization is now non-convex (and h unidentifiable)
- Computing composition called ‘forward-propagation’
- Using chain rule to compute gradient called ‘back-propagation’
- Applying a regularizer to weights called ‘weight decay’
- Alternative form of regularization: early stopping
- Can make multiple layers of composition (often difficult to get working, with some exceptions like ‘convolutional neural networks’)

Neural Networks



Recent Work

- ‘Deep Belief Networks’
    - use undirected models and stochastic approximations to build one layer of hidden units at a time, training each layer to generate outputs produced by previous layer

Outline

- Context
- Linear Models
- Neural Networks
- <u>Kernel Machines</u>
- Advanced Probabilistic Methods

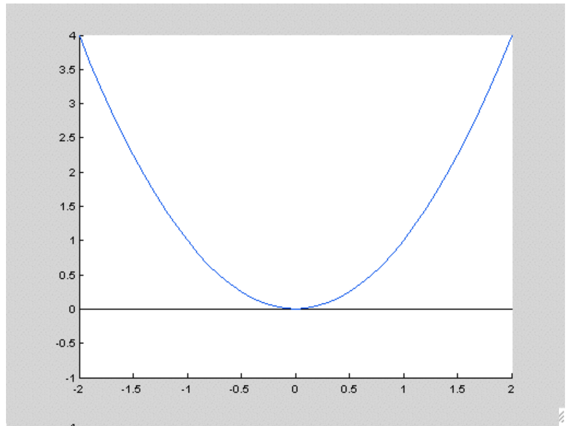
Kernel Methods

- Basic Idea:
  - Use a large basis and regularize
- Kernel ‘trick’
  - Lets us use a large set of basis functions without storing them
  - If we can define an appropriate similarity metric, we may not even need to know the basis
- Nice theory behind SVMs. I don’t have time to go over this (previous talk: 1.5 hours) and will just give an overview of a Support Vector regression model

Support Vector Regression

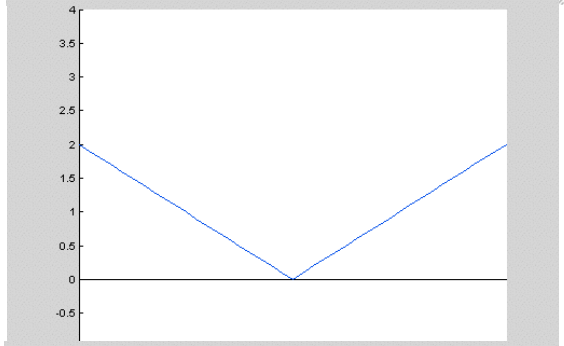
- Least Squares Loss:

$$\sum_t (w^T x_t - y_t)^2$$



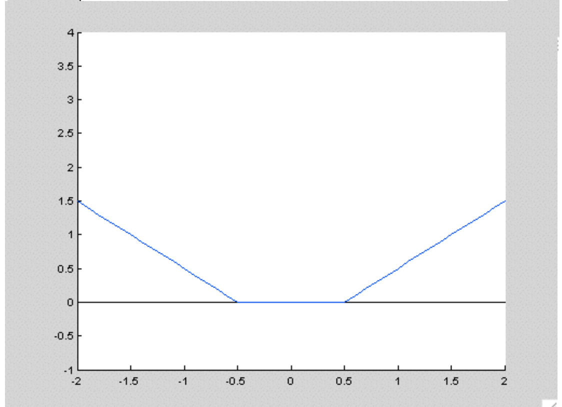
- Least Absolute Deviation Loss:

$$\sum_t |w^T x_t - y_t|$$

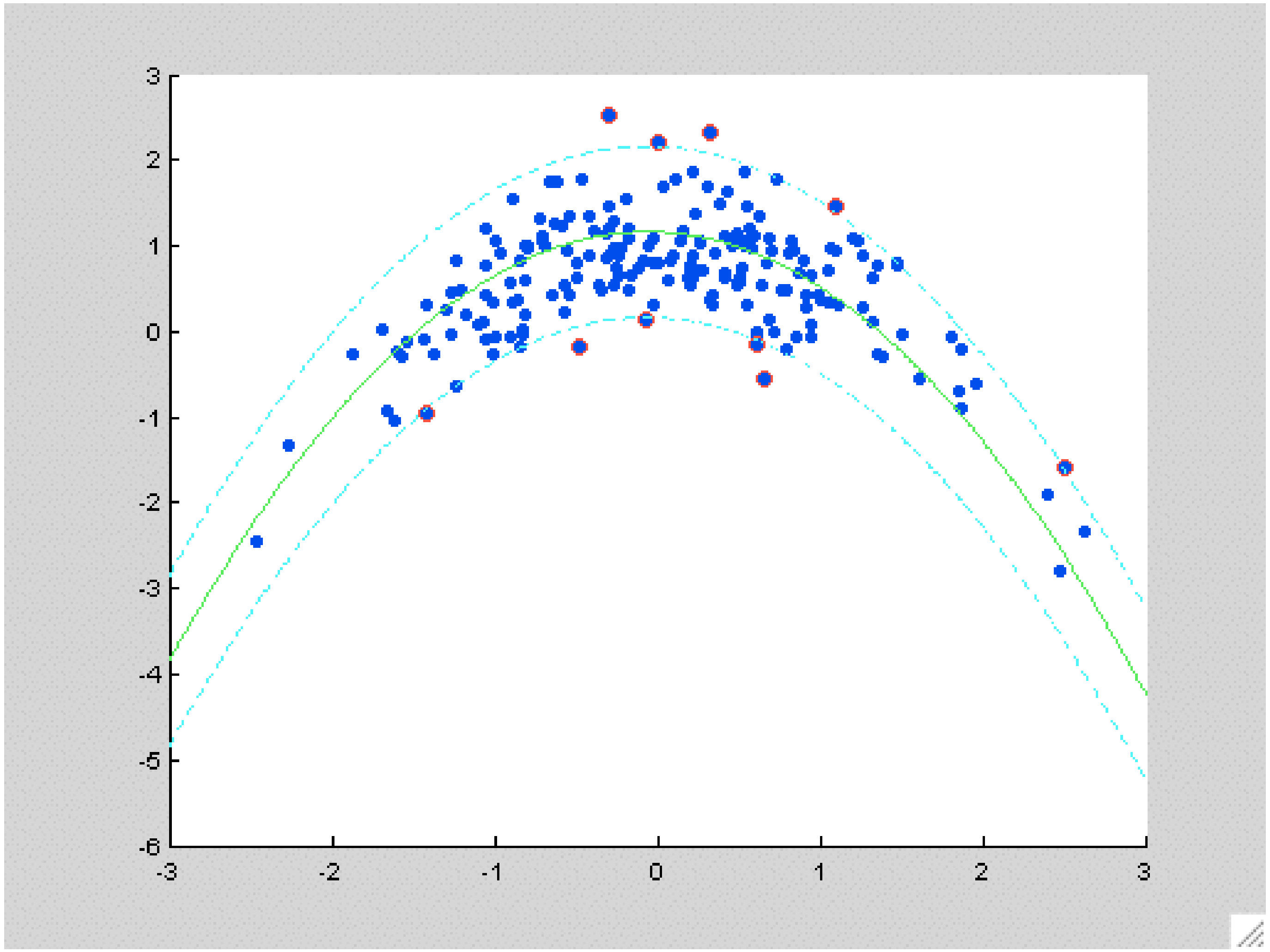


- SVR 'eps-Insensitive' Loss:

$$\sum_t [|w^T x_t - y_t| - \epsilon]^+$$



Support Vector Regression



Support Vector Regression

- We will consider the eps-Insensitive loss with the Quadratic Regularizer:

$$\sum_t [[|w^T x_t - y_t| - \epsilon]^+] + \frac{\lambda}{2} \sum_i w_i^2$$

- Optimal parameters can be found by solving a Quadratic Program with the same minimum

$$\min_{w,b,z,\hat{z}} \sum_t [z_t + \hat{z}_t] + \frac{\lambda}{2} \sum_i w_i^2$$

$$s.t. \forall_t z_t \geq 0, \hat{z}_t \geq 0$$

$$\forall_t z_t \geq w^T x_t - y_t - \epsilon \,, \hat{z}_t \geq y_t - w^T x_t - \epsilon$$

- ‘Support Vectors’ are points where either of the slacks z is non-zero, all other points are within the ‘eps-tube’ and are ‘good enough’

Dual Problem

- After introducing Lagrange multipliers and some algebra, we obtain the following dual minimization:

$$-\sum_t y_t \alpha_t + \epsilon \sum_t |\alpha_t| + \frac{1}{2} \sum_t \sum_{t'} \alpha_t \alpha_{t'} x_t^T x_{t'}$$

$$s.t. \sum_t \alpha_t = 0, \forall_t \frac{1}{\lambda} \leq \alpha_t \leq \frac{1}{\lambda}$$

- Predictions made using: $f(x_s) = \sum_t \alpha_t x_t^T x_s$
- Note: Only inner product between features is relevant

Kernel-Defined Basis

- Kernel 'trick': replace $x_s^T x_t$ with kernel function $k(x_s, x_t)$
- Example:

$$(x^T z)^2 = x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 x_2 z_2$$

http://www.youtube.com/watch?v=3liCbRZPrZA

- Kernel Functions:
  - polynomial kernel: $(x^T z + 1)^d$
  - rbf kernel: $exp(-\gamma||x - z||^2)$
  - more generally: some similarity metric between vectors/graphs/text/images/etc.
- Restriction: 'Gram Matrix' $k(X^T X) \geq 0$
- Related to Covariance functions in GMRFs/Kriging, and optimization in Reproducing Kernel Hilbert Spaces

General Advantages of SV Methods

- convexity
- regularization
- sparsity
- kernels
- efficient and large-scale training
- computational learning theory
  (none of these is unique to SV methods)

Recent Work

- Learning the kernel:
  - Linear combination of kernels
  - Semi-definite programming

Outline

- Context
- Linear Models
- Neural Networks
- Kernel Machines
- <u>Advanced Probabilistic Methods</u>

Advanced Probabilistic Methods

- Hierarchical Bayesian:
  - eg. Gaussian Process: regularized kernel linear regression, where we integrate over w and optimize parameters of kernel based on marginal likelihood (or approximate integral over kernel parameters)
- Non-Parametric Bayesian:
  - eg. Dirichlet Process on mixture coefficients in mixture model: integrates over all possible values of the number of mixture components
- Structured Output:
  - eg. Conditional Random Field: models multiple targets y, including individual costs and costs based on joint configurations, conditioned on a set of features
- Sequential Monte Carlo:
  - eg. Particle Filter: model-free filtering for (non-linear) dynamic systems