Announcements

Class is now 176.

Matlab Grader homework, emailed Thursday,

1 and 2 (of less than 9) homeworks Due 21 April, Binary graded.

Homework 3 (nor released yet) due 28 April

Jupiter "GPU" home work released Wednesday. First part of class will focus on this. Presented by graduate student Emma Ozanich.

Today:

Stanford CNN

Linear models for regression

Wednesday 10 April

Stanford CNN, Linear models for classification (Bishop 4),

Projects

3-4 person groups preferred

Deliverables: Poster & Report & main code (plus proposal, midterm slide)

Topics your own or chose form suggested topics. Some **physics inspired**.

April 26 groups due to TA (if you don’t have a group, ask in piaza we can help). TAs will construct group after that.

May 5 proposal due. TAs and Peter can approve.

Proposal: One page: Title, A large paragraph, data, weblinks, references.

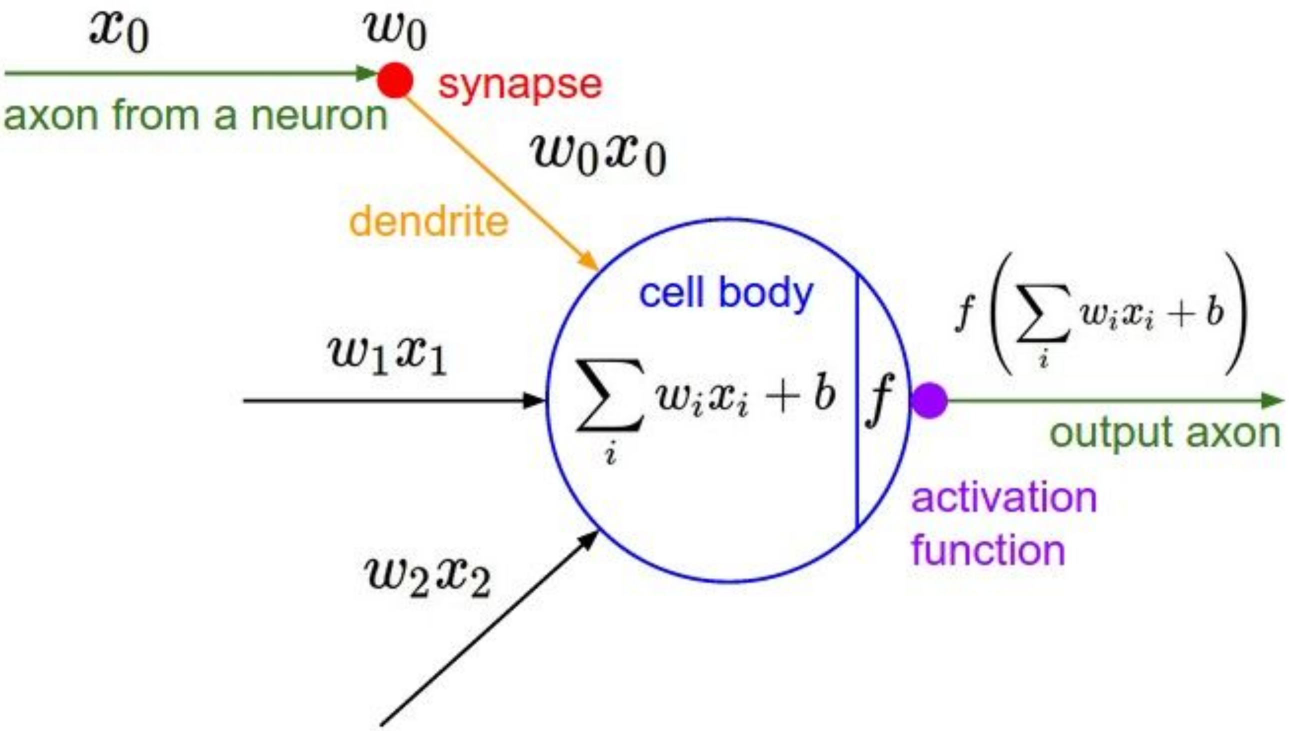
Something **physical**

May 20 Midterm slide presentation. Presented to a subgroup of class.

June 5 final poster. Uploaded June 3

Report and code due **Saturday 15 June.**

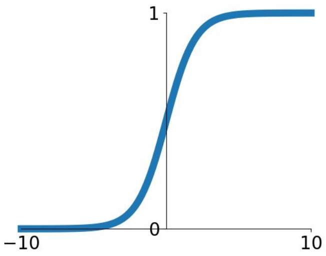
Activation Functions



Activation Functions

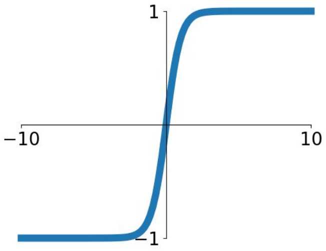
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



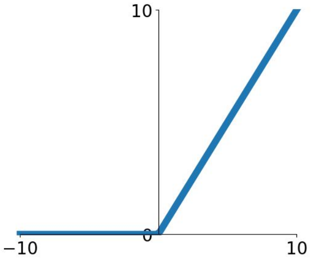
tanh

$\tanh(x)$



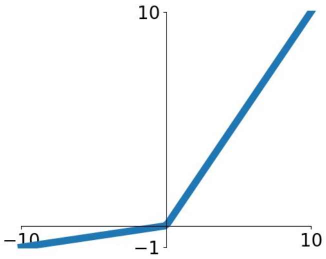
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

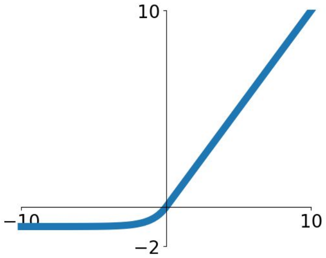


Maxout

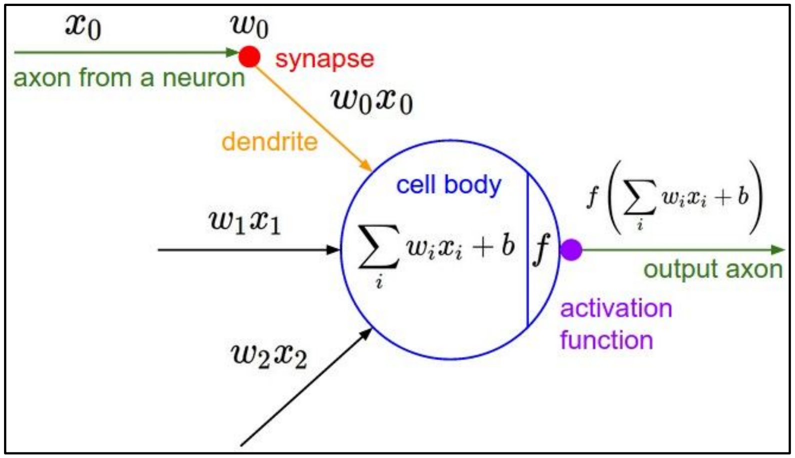
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Consider what happens when the input to a neuron (x) is always positive:

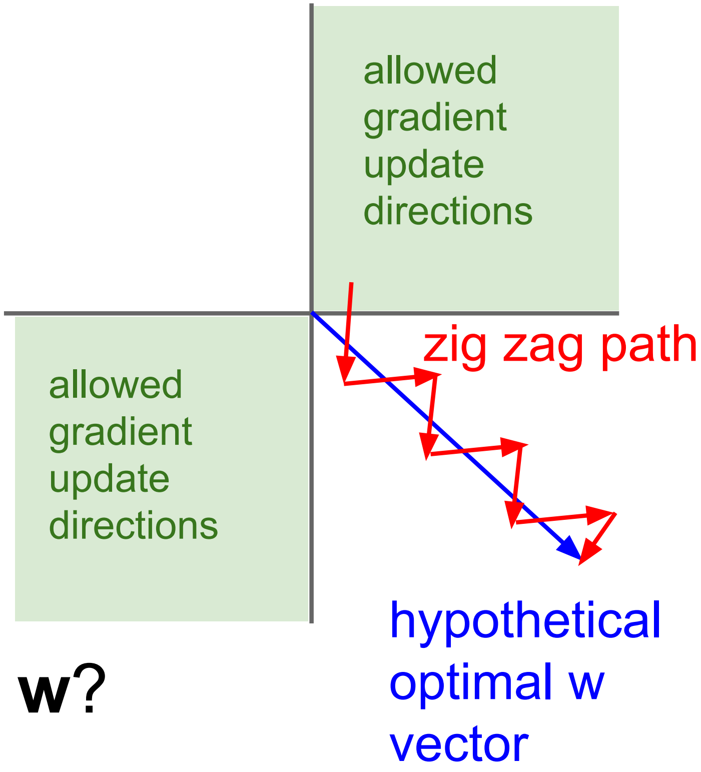


$$f\left(\sum_i w_i x_i + b\right)$$

What can we say about the gradients on **w**?

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$

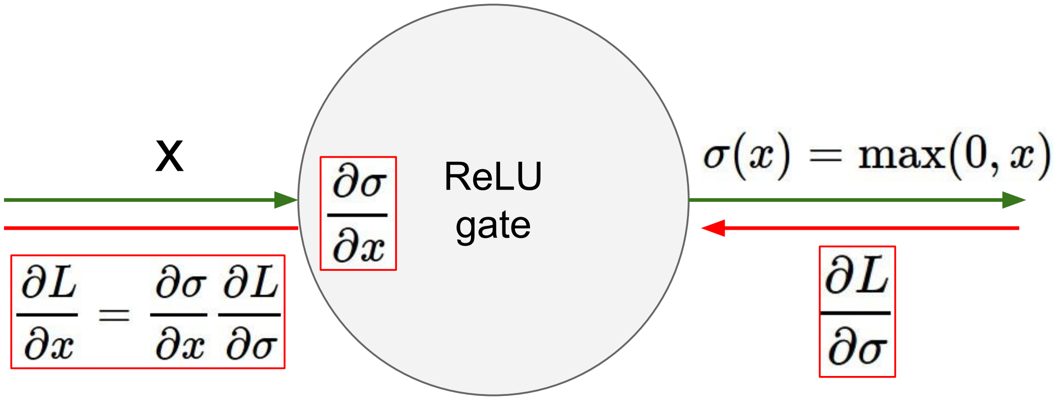

allowed gradient update directions
allowed gradient update directions
zig zag path
hypothetical optimal w vector

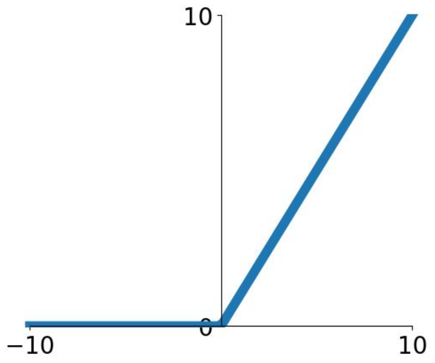
What can we say about the gradients on **w**?

Always all positive or all negative :(

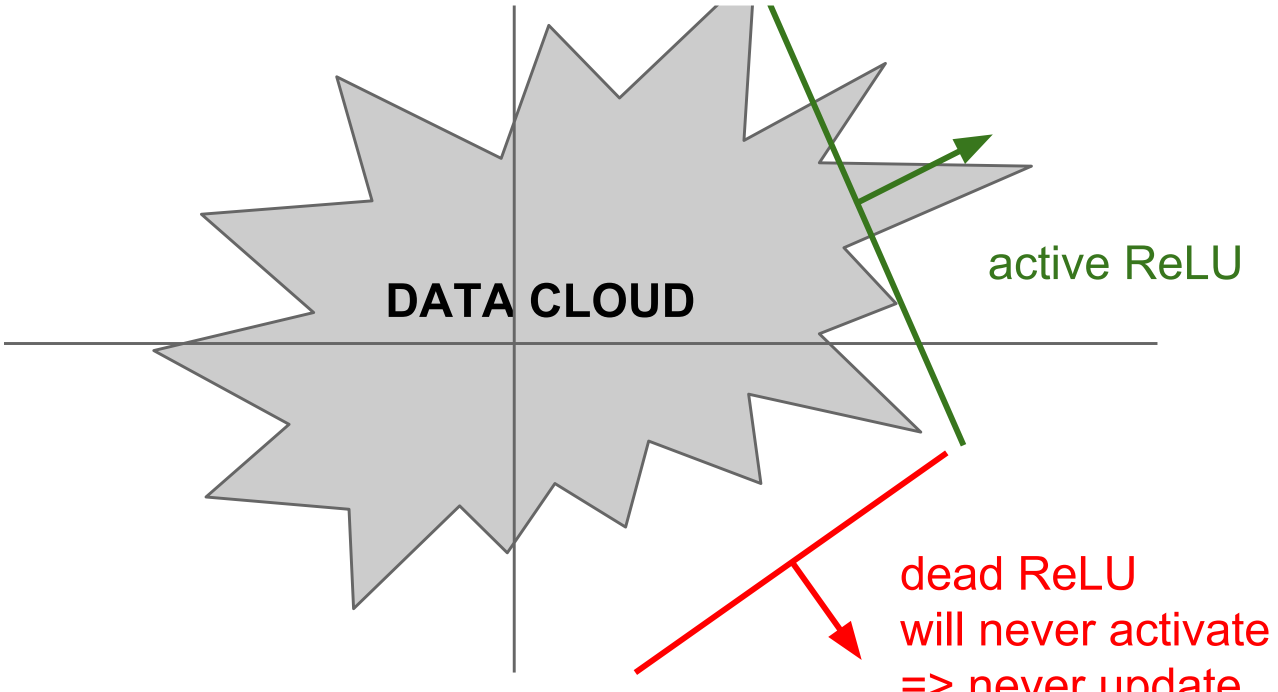
(this is also why you want zero-mean data!)

RELU


x
$\frac{\partial \sigma}{\partial x}$
ReLU
gate
$\sigma(x) = \max(0, x)$
$\frac{\partial L}{\partial \sigma}$
$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x} \frac{\partial L}{\partial \sigma}$



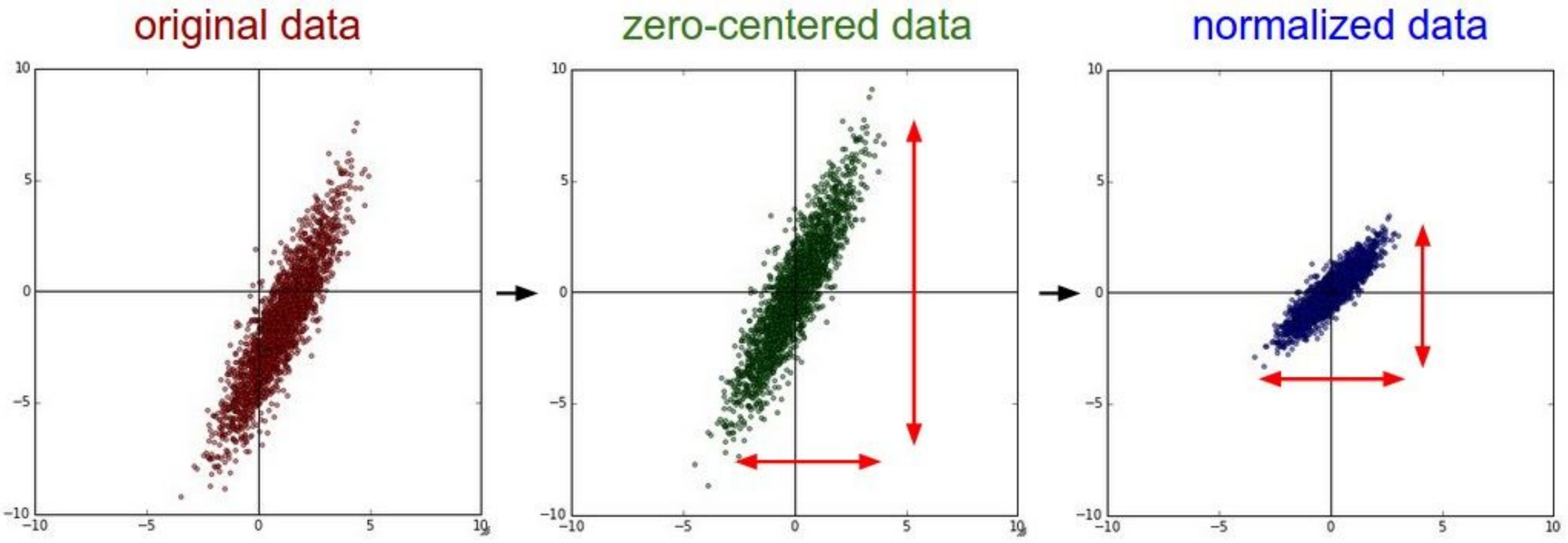
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?



TLDR: In practice:

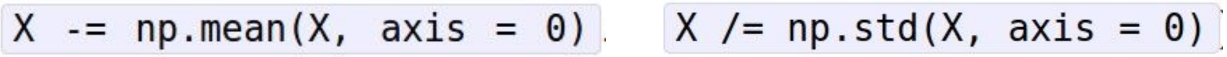
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

Step 1: Preprocess the data


original data

zero-centered data

normalized data


```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix,
each example in a row)

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

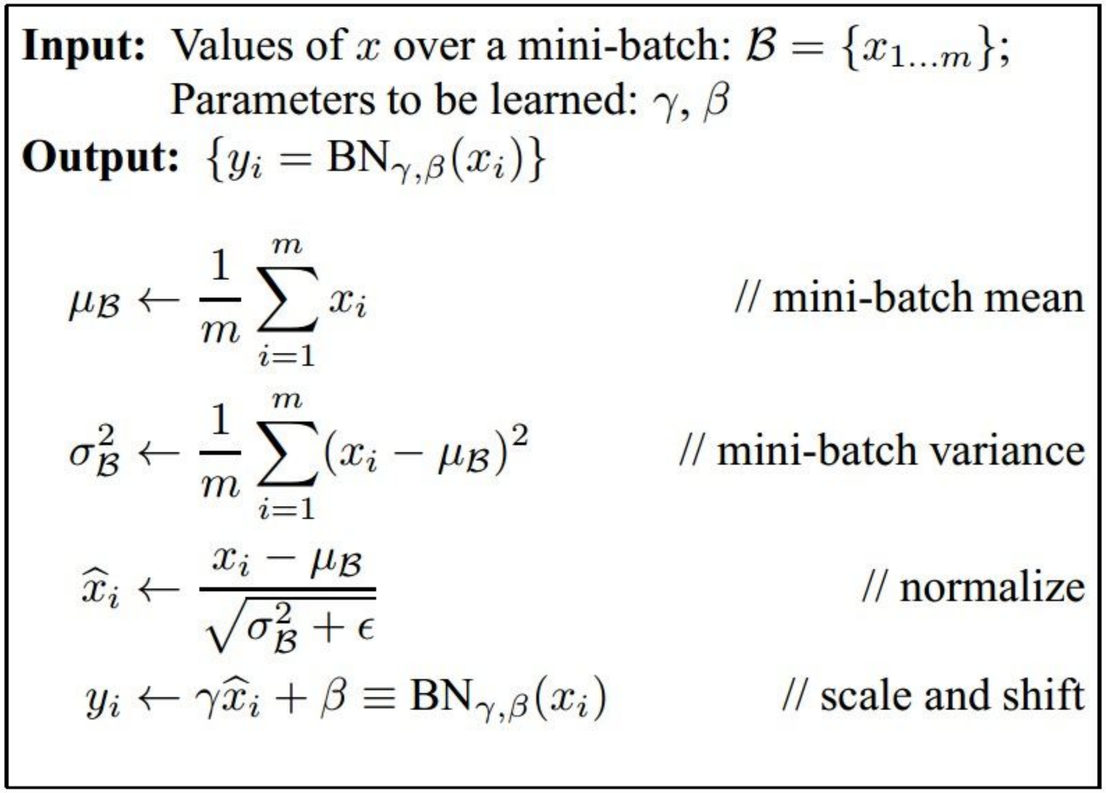
$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Batch Normalization

[Ioffe and Szegedy, 2015]



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

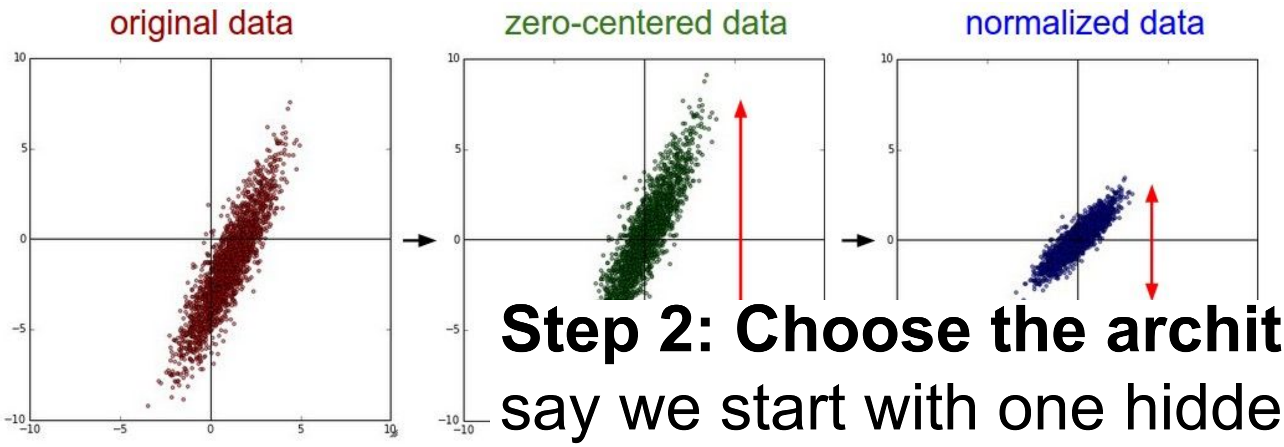
Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

Babysitting the Learning Process

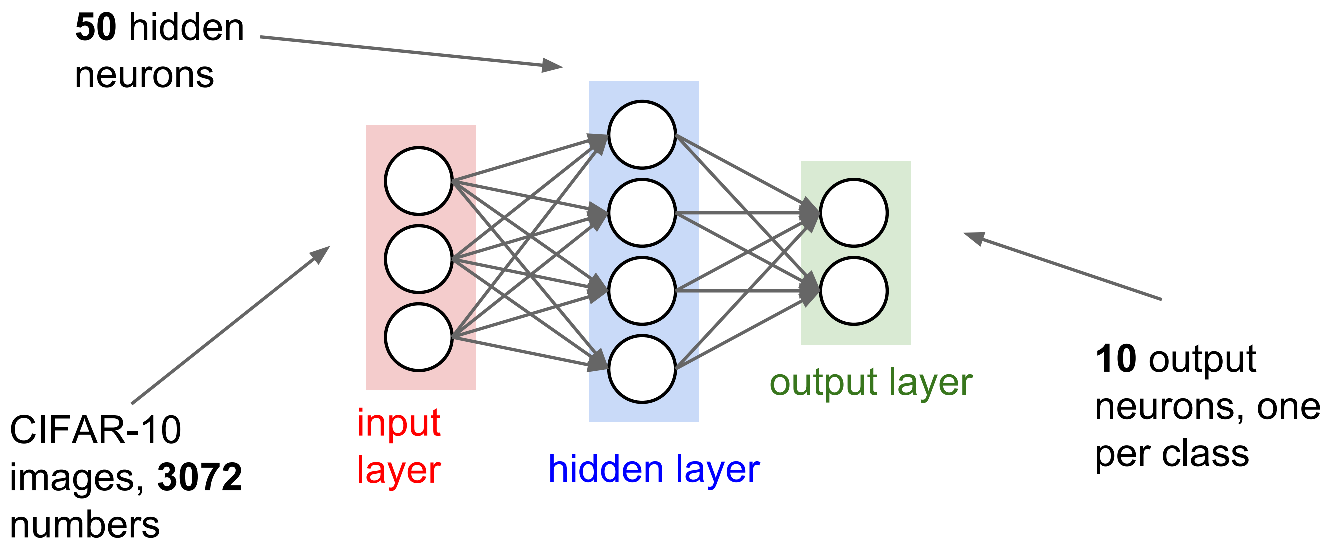
Step 1: Preprocess the data



```
X -=
```

Step 2: Choose the architecture:

say we start with one hidden layer of 50 neurons:



50 hidden neurons

input layer

hidden layer

output layer

CIFAR-10 images, **3072** numbers

10 output neurons, one per class

Hyperparameters

Cross-validation strategy

Hyperparameters to play with:

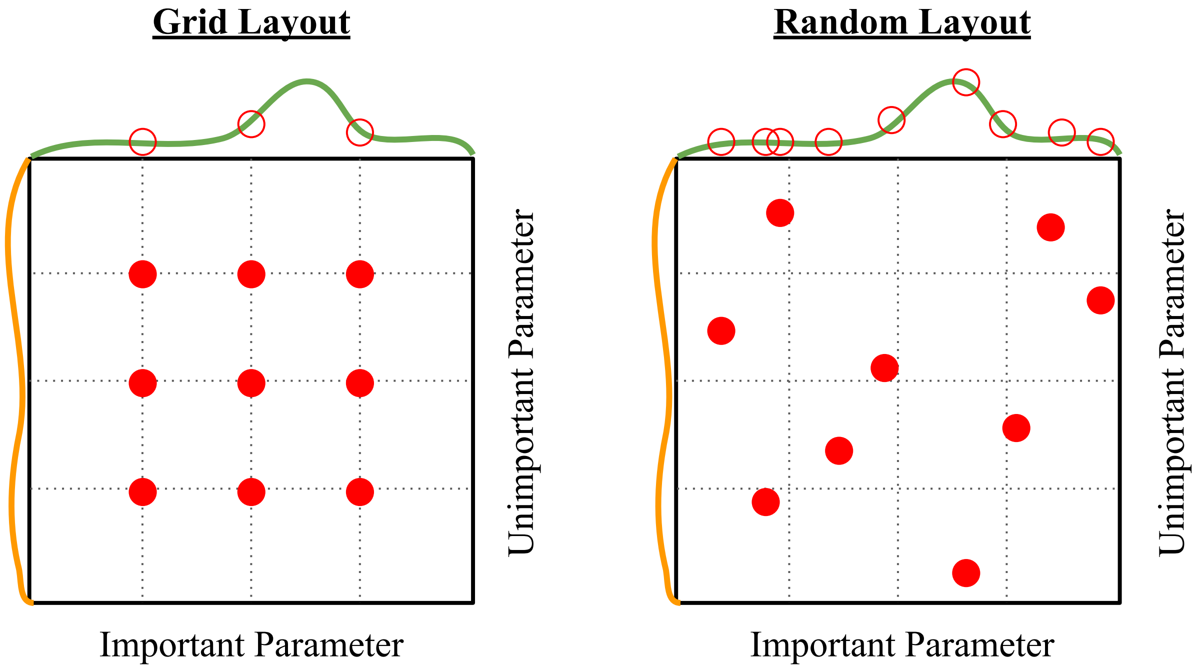
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

coarse -> fine cross-validation in stages

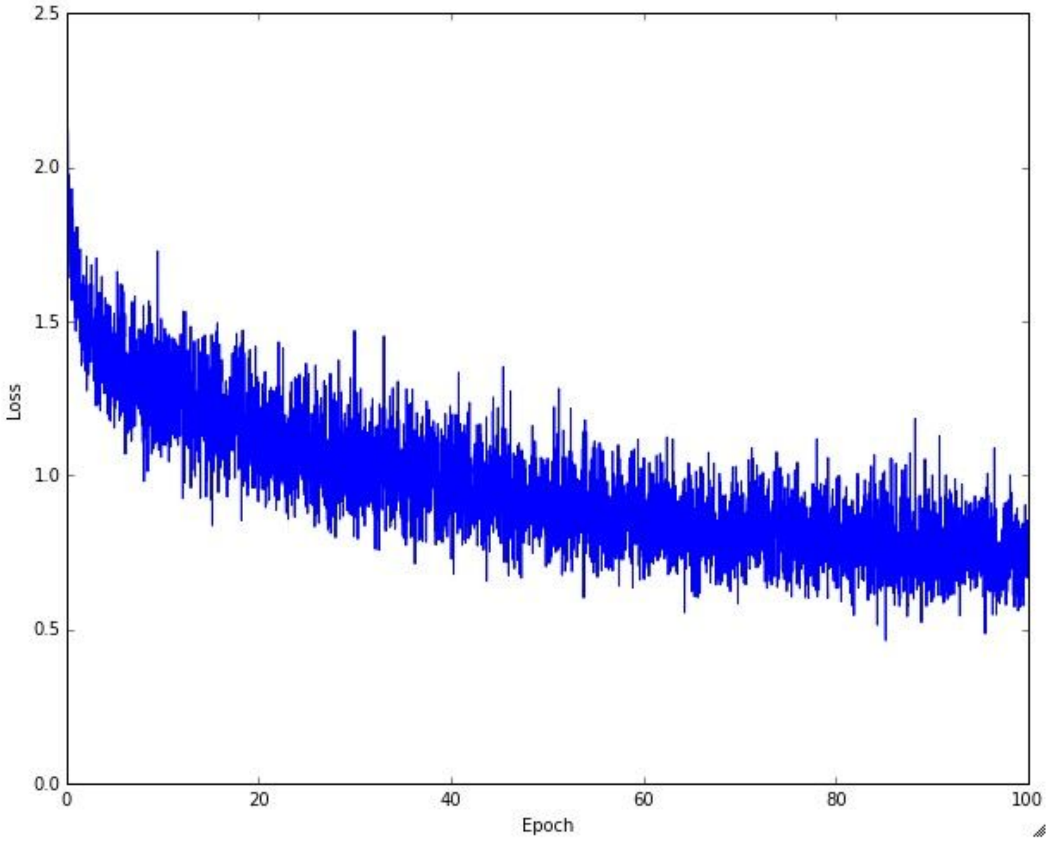
First stage: only a few epochs to get rough idea of what params work
Second stage: longer running time, finer search
… (repeat as necessary)

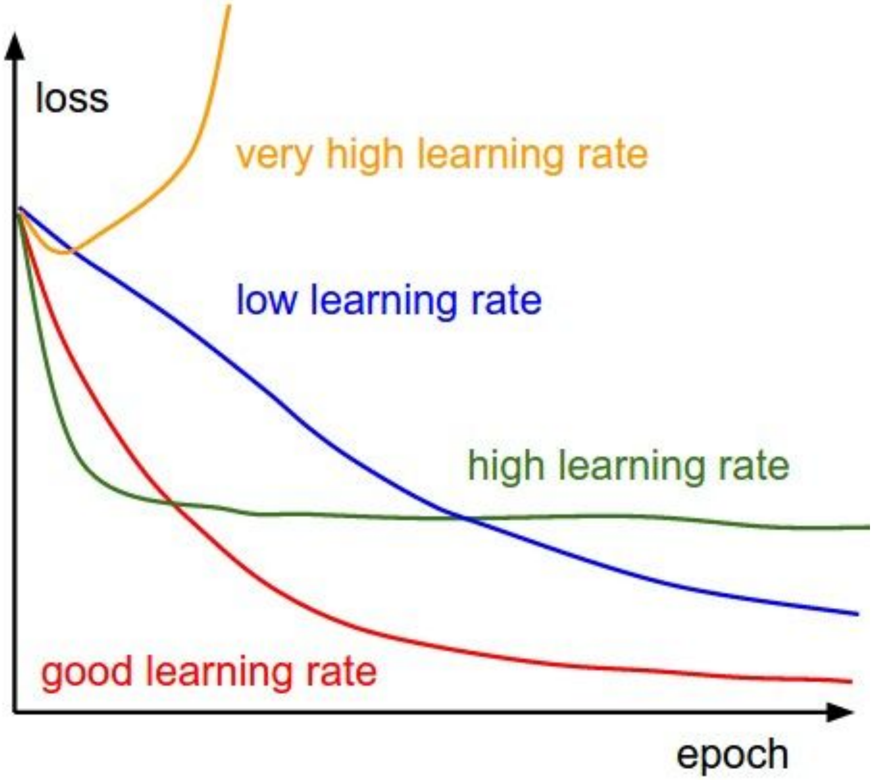
Random Search vs. Grid Search

Random Search for
Hyper-Parameter Optimization
Bergstra and Bengio, 2012



Monitor and visualize the loss curve





Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier init)
- Batch Normalization (use)
- Babysitting the Learning process
- Hyperparameter Optimization
  (random sample hyperparams, in log space when appropriate)

Maximum Likelihood and Least Squares (3)

Computing the gradient and setting it to zero yields

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

Solving for w,

$$\mathbf{w}_{\mathrm{ML}} = \overbrace{\left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}} \mathbf{t}$$

The Moore-Penrose pseudo-inverse, $\boldsymbol{\Phi}^{\dagger}$.

where

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

Geometry of Least Squares

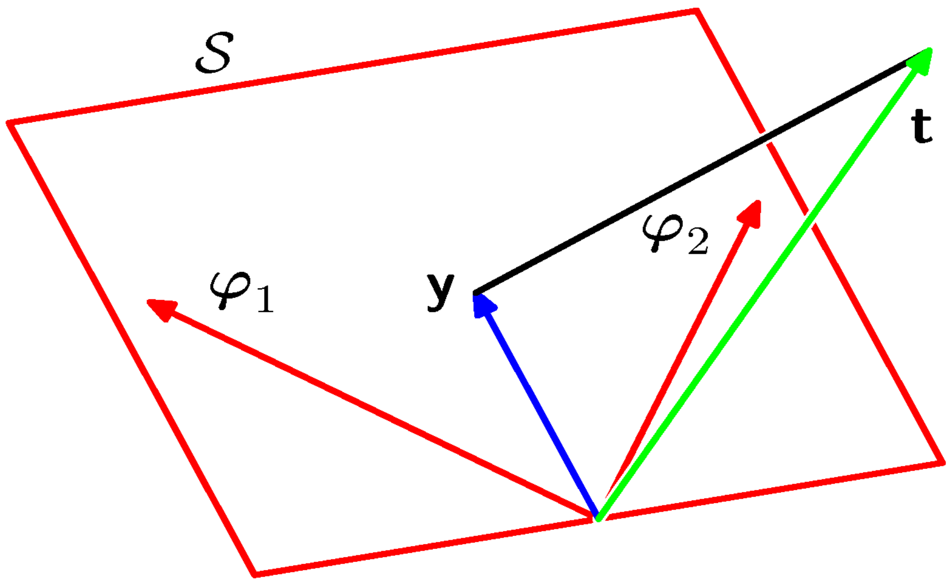
Consider

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{w}_{\mathrm{ML}} = [\varphi_1, \ldots, \varphi_M]\,\mathbf{w}_{\mathrm{ML}}.$$

$$\mathbf{y} \in \mathcal{S} \subseteq \mathcal{T} \qquad \mathbf{t} \in \mathcal{T}$$

N-dimensional

M-dimensional



S is spanned by $\varphi_1, \ldots, \varphi_M$

w_{ML} minimizes the distance between t and its orthogonal projection on S, i.e. y.

Least mean squares: An alternative approach for big datasets

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \boxed{\nabla E_{n(\tau)}}$$

weights after seeing training case tau+1

learning rate

squared error derivatives w.r.t. the weights on the training case at time tau.

This is **“on-line“ learning**. It is efficient if the dataset is redundant and simple to implement.

It is called **stochastic gradient descent** if the training cases are picked randomly.

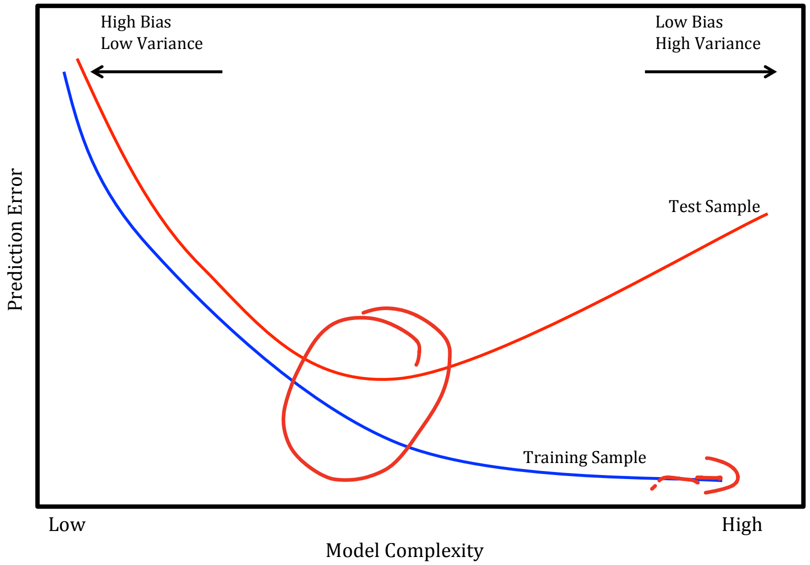
Care must be taken with the learning rate to prevent divergent oscillations. Rate must decrease with tau to get a good fit.

$$\frac{\partial E}{\partial w} = \sum_{n}^{N} \phi_n \left(t_n - w^T \phi_n\right)$$

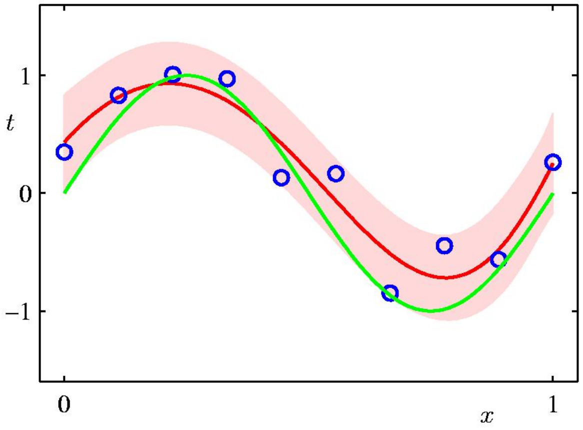
Bias-variance (from lecture 1)

Bias-variance tradeoff

Concept: Complex models can learn data-label relationships well, bu may not extrapolate to new cases.



$$p(t|x, \mathbf{x}, \mathbf{t}) = \int p(t|x, \mathbf{w})p(\mathbf{w}|\mathbf{x}, \mathbf{t})\,\mathrm{d}\mathbf{w}$$



We focus on Gaussians!

The **bias-variance** decomposition

model estimate for testcase n trained on dataset D

average target value for test case n

“Bias” term is the squared error of the average, over training datasets D, of the estimates.

Bias: average between prediction and desired.

$$\left\langle \left\{ y(\mathbf{x}_n;D) - \bar{t}_n \right\}^2 \right\rangle_D = \boxed{\left\{ \left\langle y(\mathbf{x}_n;D) \right\rangle_D - \bar{t}_n \right\}^2}$$

$$+ \boxed{\left\langle \left\{ y(\mathbf{x}_n;D) - < y(\mathbf{x}_n;D) >_D \right\}^2 \right\rangle_D}$$

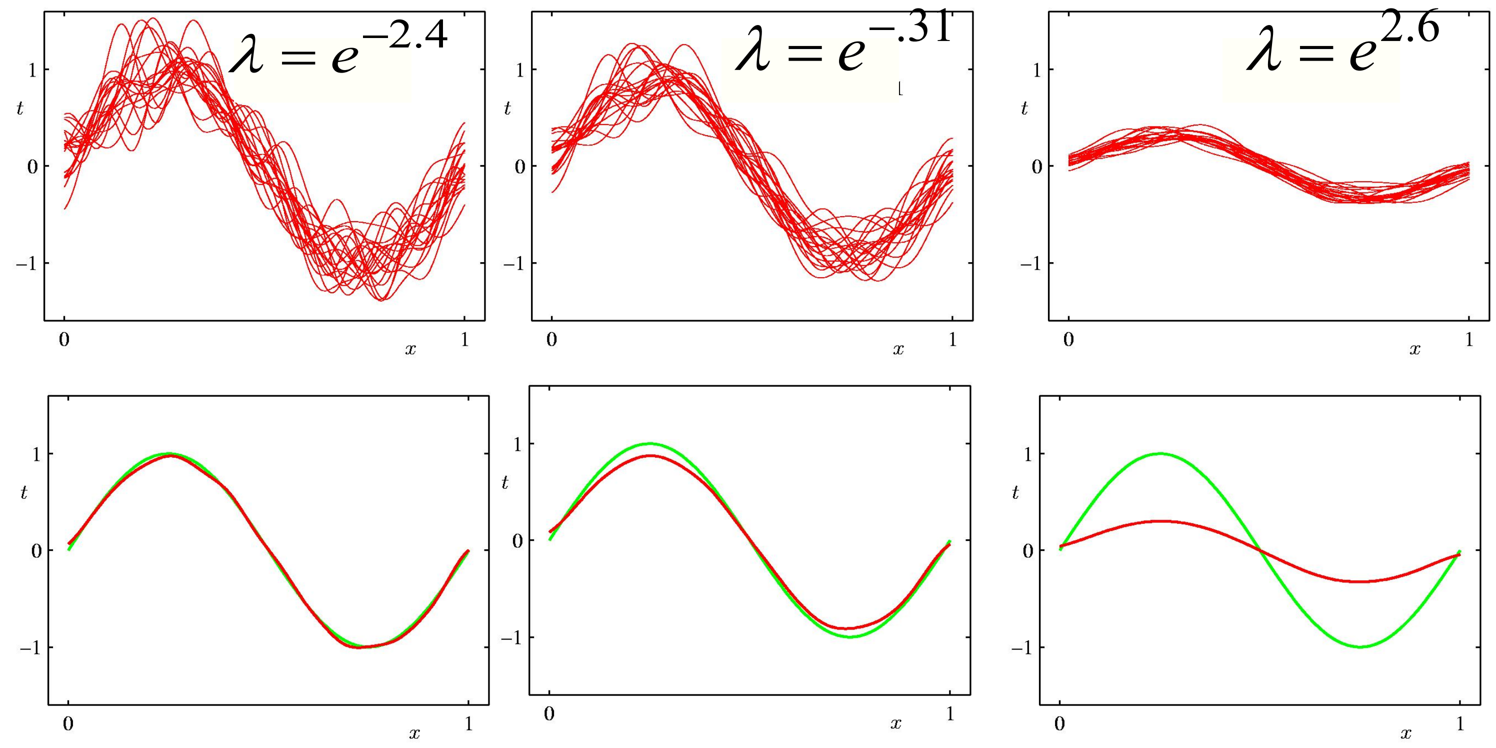
<. > means expectation over D

“Variance” term: variance over training datasets D, of the model estimate.

Regularization parameter affects the bias and variance

high variance

low variance



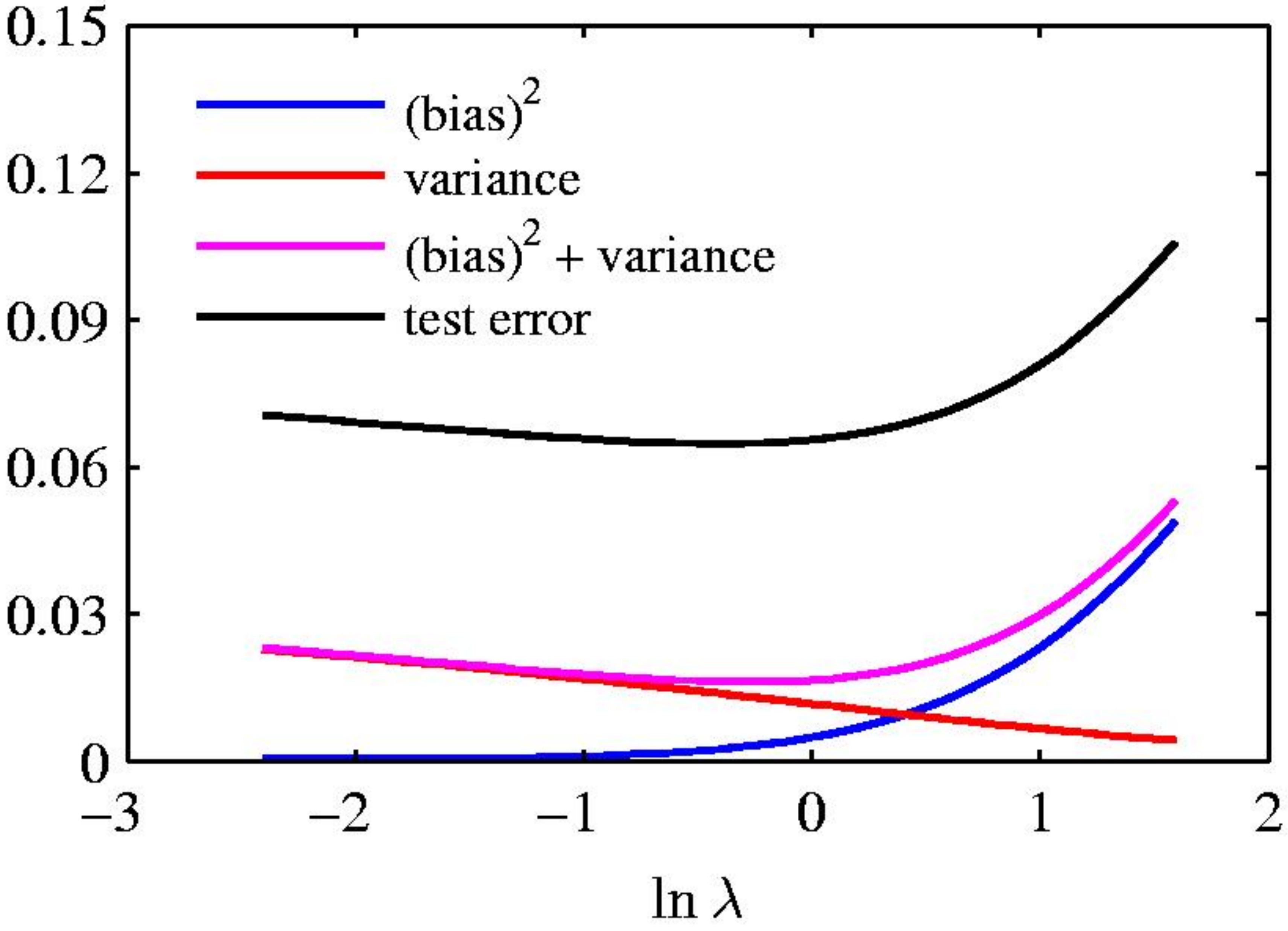
20 realizations

True model
average

low bias

high bias

An example of the bias-variance trade-off



Beating the bias-variance trade-off

Reduce the variance term by averaging lots of models trained on different datasets.

- Seems silly. For lots of different datasets it is better to combine them into one big training set.
  - More training data has much less variance.

Weird idea: We can create different datasets by bootstrap sampling of our single training dataset.

- This is called “bagging” and it works surprisingly well.

If we have enough computation its better doing it Bayesian:

- Combine the predictions of many models using the posterior probability of each parameter vector as the combination weight.

The bias-variance trade-off
(a figment of the frequentists lack of imagination?)

Imagine a training set drawn at random from a whole set of training sets.

The squared loss can be decomposed into a

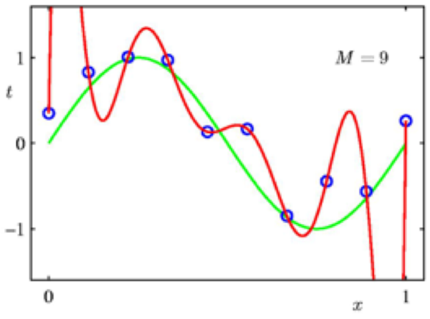
Bias = systematic error in the model’s estimates

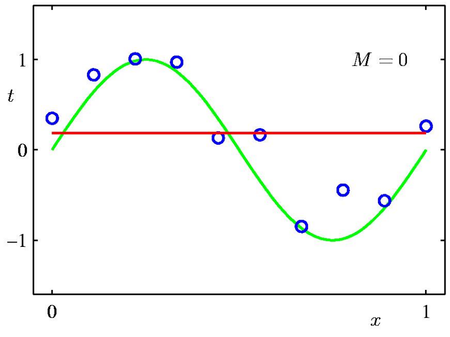
Variance = noise in the estimates cause by sampling noise in the training set.

There is also additional loss due to noisy target values.

We eliminate this extra, irreducible loss from the math by using the average target values (i.e. the unknown, noise-free values)

9 Order Polynomial





Bayesian Linear Regression (Bishop 3.3)

Define a conjugate prior over w

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0).$$

Combining this with the likelihood function and using results for multiplying Gaussians, gives the posterior

$$p(\mathbf{w}|\mathbf{t}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N)$$

$$\begin{aligned}\mathbf{m}_N &= \mathbf{S}_N \left(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}\right) \\ \mathbf{S}_N^{-1} &= \mathbf{S}_0^{-1} + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}.\end{aligned}$$

A common simpler prior

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

Which gives

$$\begin{aligned}\mathbf{m}_N &= \beta\mathbf{S}_N\mathbf{\Phi}^{\mathrm{T}}\mathbf{t} \\ \mathbf{S}_N^{-1} &= \alpha\mathbf{I} + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}.\end{aligned}$$

From lecture 3:

Bayes for linear model

$y = Ax + n$ $n \sim N(0, C_n)$ $y \sim N(Ax, C_n)$ prior: $x \sim N(0, C_x)$

$p(x|y) \sim p(y|x)p(x) \sim N(x_p, C_p)$ mean $x_p = C_p A^T C_n^{-1} y$

Covariance $C_p^{-1} = A^T C_n^{-1} A + C_x^{-1}$

$$\sim e^{-\frac{1}{2}(x-x_p)^T C_p^{-1}(x-x_p)} \leftarrow$$

$$= e^{-\frac{1}{2}(y-Ax)^T C_n^{-1}(y-Ax)} \, e^{-\frac{1}{2}x^T C_x^{-1} x}$$

$$= e^{-\frac{1}{2}(\underbrace{x^T A^T C_n^{-1} A x + x^T C_x^{-1} x}_{x^T C_p^{-1} x^T})} \, e^{-\frac{1}{2}\underbrace{x^T A^T C_n^{-1} y}_{x^T C_p^{-1} x_p}}$$

$$C_p^{-1} = A^T C_n^{-1} A + C_x^{-1}$$

$$x_p = C_p A^T C_n^{-1} y$$

Interpretation of solution

$$\begin{aligned}\mathbf{m}_N &= \beta \mathbf{S}_N \mathbf{\Phi}^{\mathrm{T}} \mathbf{t} \\ \mathbf{S}_N^{-1} &= \alpha \mathbf{I} + \beta \mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}.\end{aligned}$$

Draw it

Sequential, **conjugate prior**

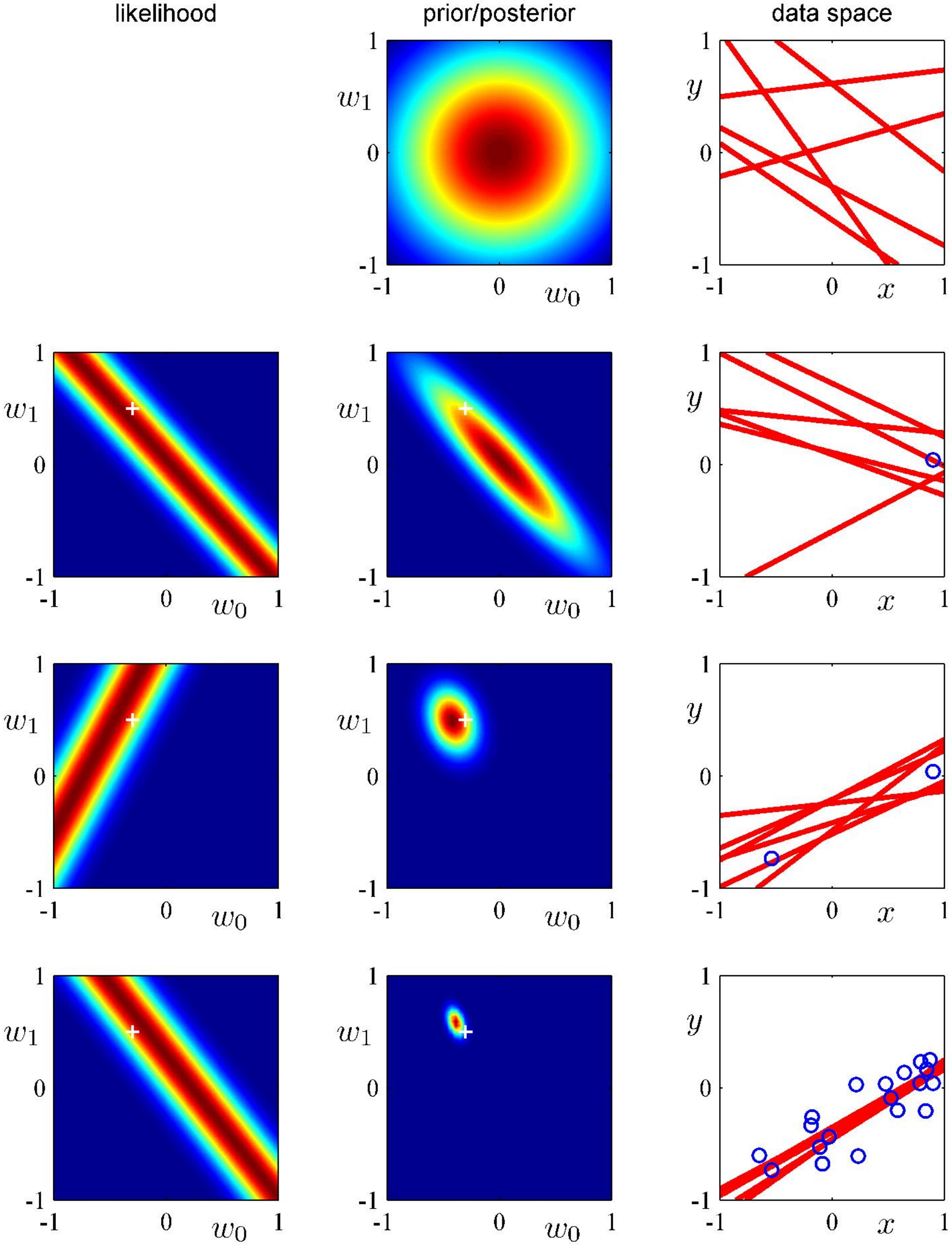
$$p(\boldsymbol{x}|\boldsymbol{y}) \sim p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x}) \sim \mathrm{N}(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{C}_n)\ \mathrm{N}(\boldsymbol{0}, \boldsymbol{C}_x) \sim N\left(\boldsymbol{x}_p, \boldsymbol{C}_p\right)$$

Covariance $\boldsymbol{C}_p^{-1} = \boldsymbol{A}^T \boldsymbol{C}_n^{-1} \boldsymbol{A} + \boldsymbol{C}_x^{-1}$

Likelihood, prior/posterior Bishop Fig 3.7

$$y = w_0 + w_1 x + N(0,0.2)$$

Data generated with. w_0=-0.3, w_1=0.5



With no data we sample lines from the prior.

With 20 data points, the prior has little effect

Predictive distributions

marginal

Prior predictive

Predictive Distribution

Predict t for new values of x by integrating over w (Giving the marginal distribution of t):

$$\begin{aligned} p(t|\mathbf{t},\alpha,\beta) &= \int p(t|\mathbf{w},\beta)p(\mathbf{w}|\mathbf{t},\alpha,\beta)\,\mathrm{d}\mathbf{w} \\ &= \mathcal{N}(t|\mathbf{m}_N^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}),\sigma_N^2(\mathbf{x})) \end{aligned}$$

training data

precision of prior

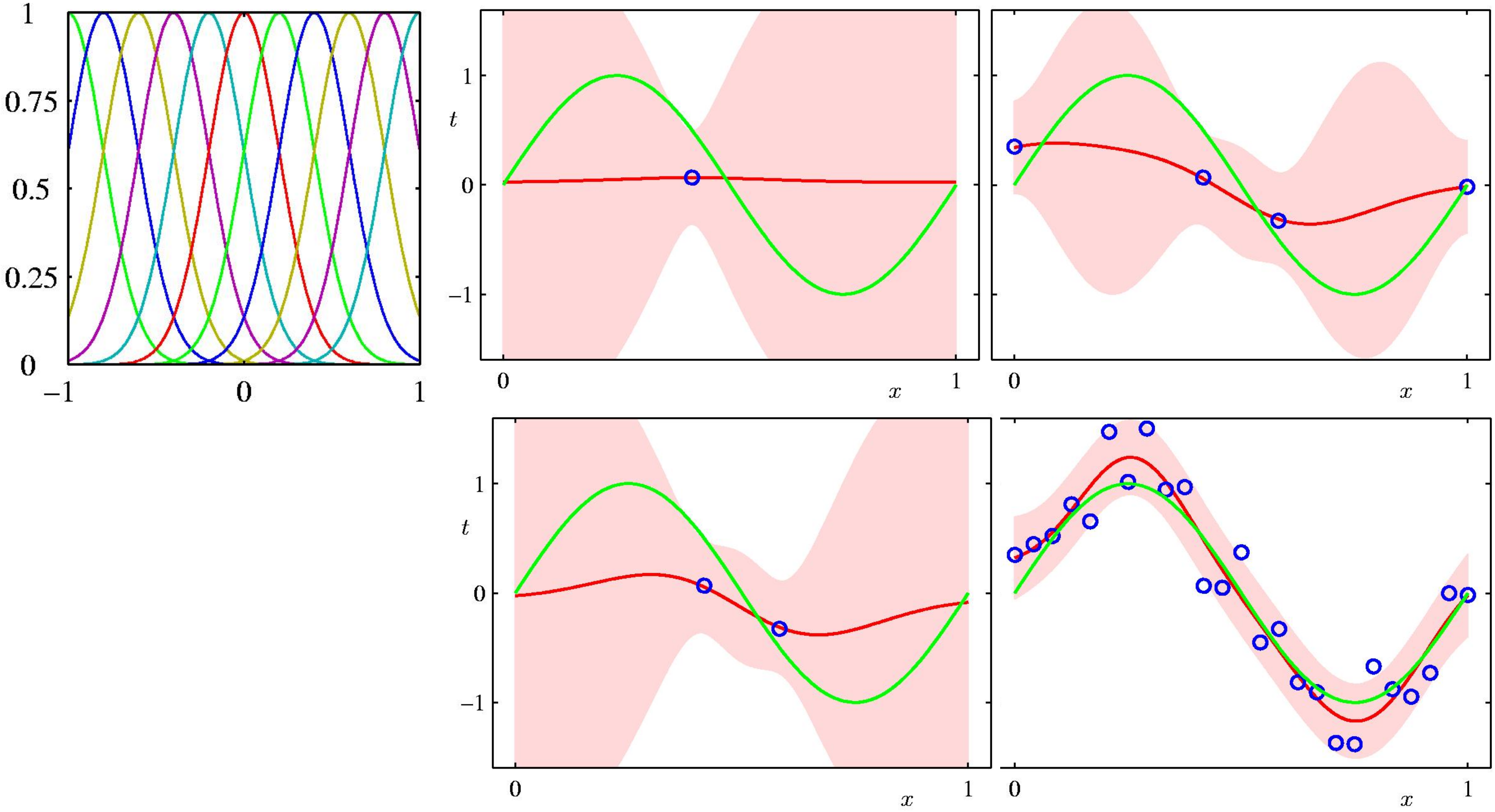
precision of output noise

$$\begin{aligned} \mathbf{m}_N &= \beta\mathbf{S}_N\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \\ \mathbf{S}_N^{-1} &= \alpha\mathbf{I}+\beta\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}. \end{aligned}$$

where

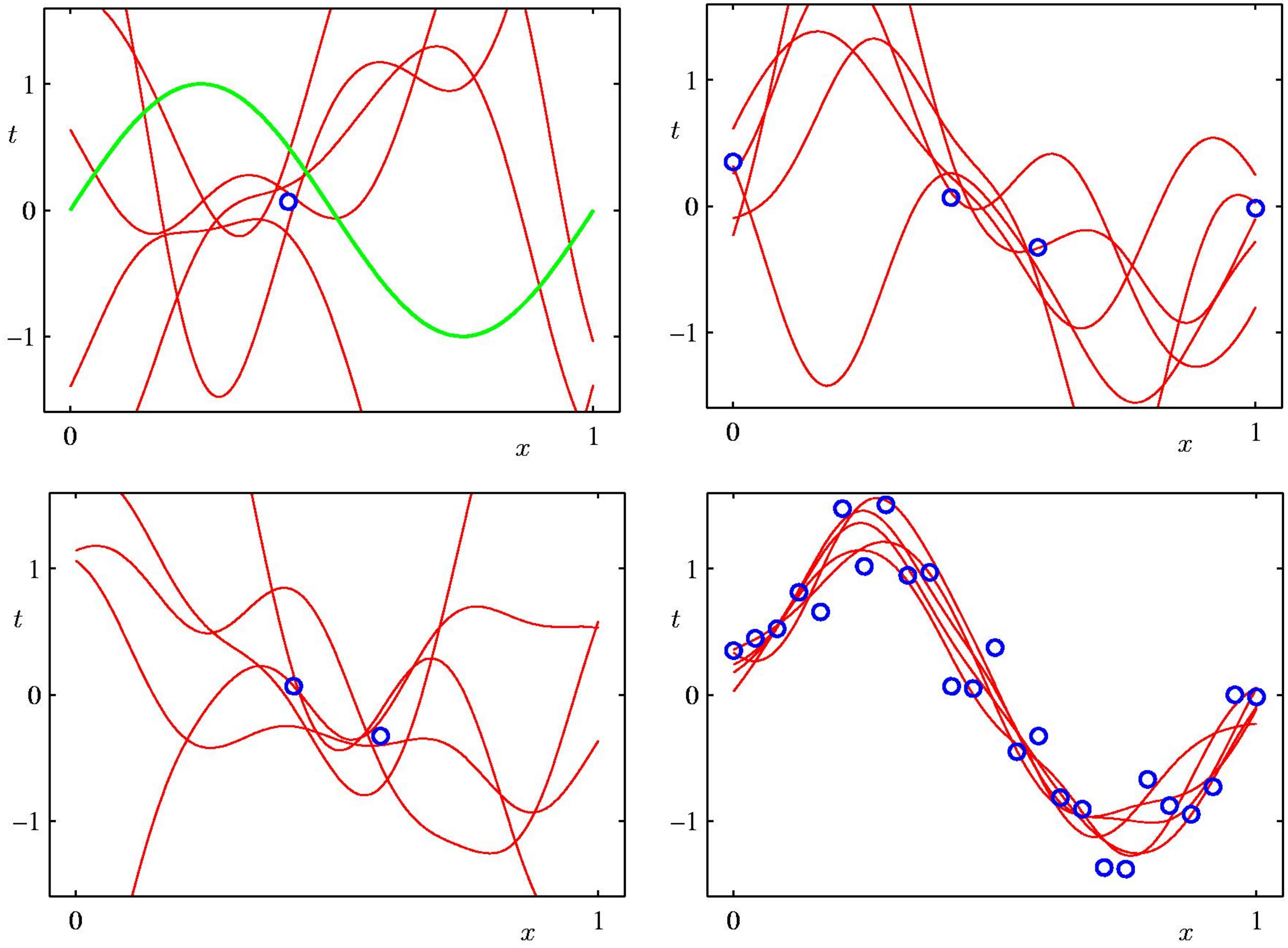
$$\sigma_N^2(\mathbf{x}) = \frac{1}{\beta}+\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{S}_N\boldsymbol{\phi}(\mathbf{x}).$$

Predictive distribution for noisy sinusoidal data modeled by linear combining 9 radial basis functions.



A way to see the covariance of predictions for different values of x

We sample models at random from the posterior and *show the mean* of each model’s predictions



Equivalent Kernel BISHOP 3.3.3

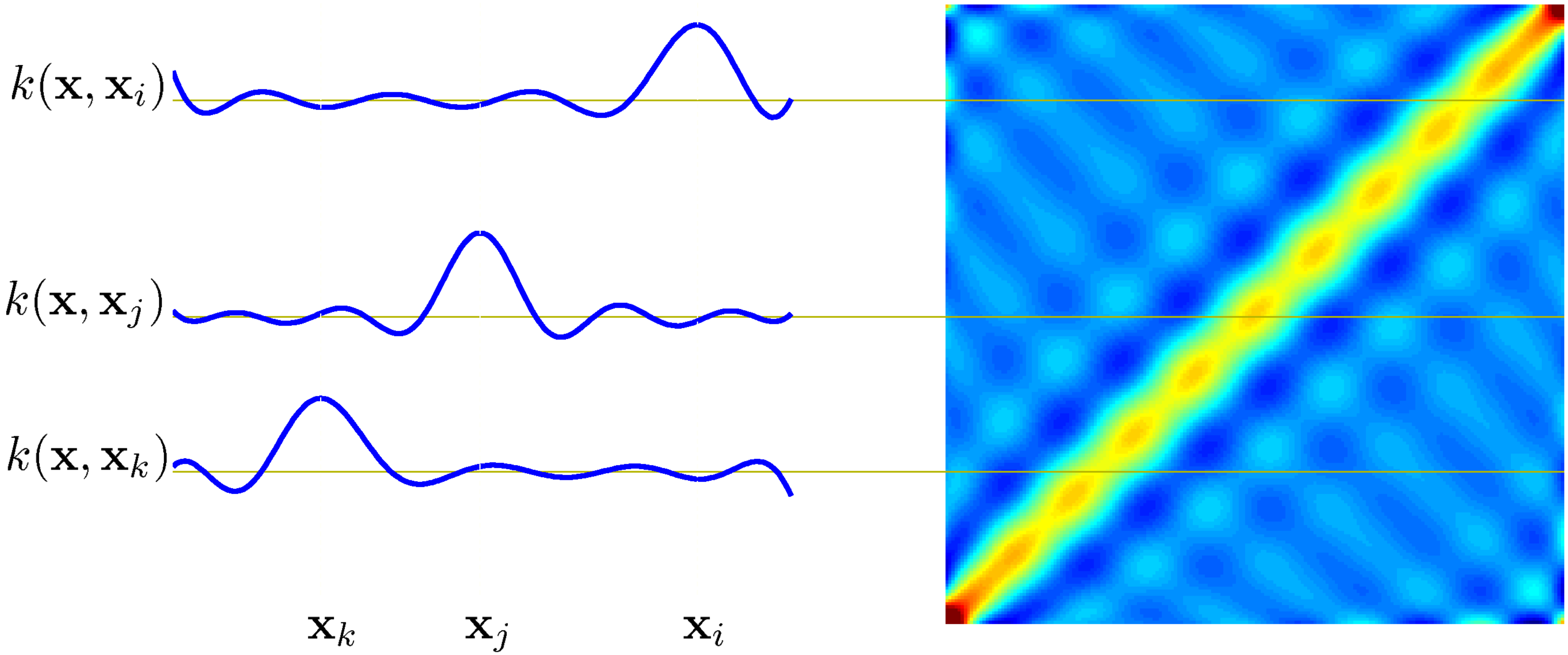
The predictive mean can be written

$$\begin{aligned} y(\mathbf{x},\mathbf{m}_N) &= \mathbf{m}_N^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = \beta\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{S}_N\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \\ &= \sum_{n=1}^{N} \underbrace{\beta\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{S}_N\boldsymbol{\phi}(\mathbf{x}_n)}\, t_n \\ &= \sum_{n=1}^{N} \overbrace{k(\mathbf{x},\mathbf{x}_n)}\, t_n. \end{aligned}$$

Equivalent kernel or *smoother matrix*.

This is a weighted sum of the training data target values, t_n.

Equivalent Kernel



Weight of t_n depends on distance between x and x_n; nearby x_n carry more weight.

Equivalent Kernel

The kernel as a covariance function: consider

$$\begin{aligned}\operatorname{cov}[y(\mathbf{x}), y(\mathbf{x}')] &= \operatorname{cov}[\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{w}, \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')] \\ &= \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{S}_N\boldsymbol{\phi}(\mathbf{x}') = \beta^{-1}k(\mathbf{x}, \mathbf{x}').\end{aligned}$$

We can avoid the use of basis functions and define the kernel function directly, leading to *Gaussian Processes* (Chapter 6).

No need to determine weights.

Like all kernel functions, the equivalent kernel can be expressed as an inner product:

$$k(\mathbf{x}, \mathbf{z}) = \boldsymbol{\psi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\psi}(\mathbf{z})$$

$$\boldsymbol{\psi}(\mathbf{x}) = \beta^{1/2}\mathbf{S}_N^{1/2}\boldsymbol{\phi}(\mathbf{x})$$