Machine Learning 1: Lecture 2

Ted Meeds, tmeeds@gmail.com

Linear Regression with Basis Functions

See Bishop section 3.1

Recall: the goal of regression is **prediction** of target t given input $\mathbf{x}$. Learning a linear regression model is a procedure for modifying parameters $\boldsymbol{\theta}$ to optimize some pre-defined objective function. Because we describe the parameters as *weights*, we use $\mathbf{w}$ instead of $\boldsymbol{\theta}$ for clarity.

Linear regression is *linear* in the parameter / weight vector $\mathbf{w}$, but is not necessarily linear in the input space since we can use non-linear basis functions to add flexibility (aka prediction power) to the model class. In this case the model still a linear function of the basis functions, but a non-linear function of the input space.

For simple linear regression, without basis functions, there are D parameters in $\mathbf{w}$, one weight per dimension of the input vector $\mathbf{x}$. A **bias** parameter w_0 is added as an offset added to each prediction:

$$y\left(\mathbf{x}, \mathbf{w}, w_0\right) = \sum_{d=1}^{D} w_d x_d + w_0 \tag{1}$$

For convenience, we augment $\mathbf{x}$ with a 1 (now has dimension $D+1$) to write the prediction as an inner product:

$$y\left(\mathbf{x}, \mathbf{w}\right) = \sum_{d=0}^{D} w_d x_d = \mathbf{w}^T \mathbf{x} \tag{2}$$

Basis Functions

In Figure 1 are plots of three different types of basis functions. The x-axis is the value of a 1-dimensional input x and the y-axis shows output of the basis function as a function of x. The polynomial basis function $\phi_m x = x^m$ is a **global** function because every prediction is influenced, via the weights, by every polynomial function.

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_M(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^M \end{bmatrix} \tag{3}$$

Gaussian basis functions, e.g. $\phi_m(x) = \exp\left(-\left(x-\mu_m\right)^2 / 2\sigma_m^2\right)$, and sigmoid functions $\phi_m(x) = 1/\left(1+\exp\left(-a_m x\right)\right)$ are **local** functions because they are "turned on" in local regions of the input space.

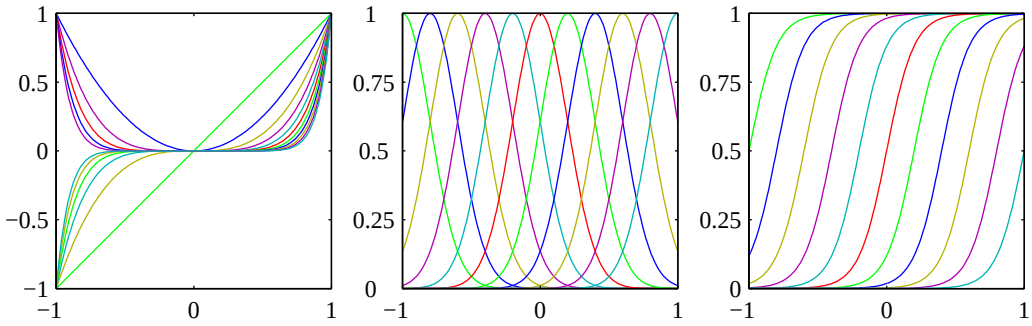

Figure 1. Basis Functions (Bishop Figure 3.1). Left: polynomial basis functions. Center: Gaussian basis functions. Right: Sigmoid basis functions.

We assume, for now, that these are **fixed** basis functions, i.e. the values of μ_m or a_m are fixed in advance and that learning adjusts $\mathbf{w}$ only. Later, we will see that a more powerful and general approach is to learn the parameters of the basis functions; e.g. with neural networks, Gaussian processes, and support vector machines.

How do fixed basis functions **scale** with the input dimension D?

Cost Functions and Regularization

The basis functions define part of the model class for linear regression. For example, we decide maximum degree polynomial M *a priori*. This defines the dimensionality of the weight vector $\mathbf{w}$ for which we optimize. We now define the **objective function** which we minimize with the aid our training data.

For linear regression, the sum-of-squares error function is:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y\left(x_n, \mathbf{w}\right) - t_n\right)^2 \tag{4}$$

$$= \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^T\boldsymbol{\phi}_n - t_n\right)^2 \tag{5}$$

$$\tag{6}$$

where we have substituted $y\left(x_n, \mathbf{w}\right) = \mathbf{w}^T\boldsymbol{\phi}_n$ and $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x})$.

Given our objective $\mathbb{E}(\mathbf{w})$, we can now optimize it by finding its minimum. Note that $\mathbb{E}(\mathbf{w})$ is a quadratic function that at its minimum the gradient is 0:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}\underbrace{\left(\mathbf{w}^T\boldsymbol{\phi}_n - t_n\right)}_{\text{error for } \boldsymbol{\phi}_n}\cdot\underbrace{\boldsymbol{\phi}_n}_{\text{move to } \boldsymbol{\phi}_n} \tag{7}$$

We rewrite this slightly to get a form that we can solve for analytically:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} \left(\mathbf{w}^T \boldsymbol{\phi}_n - t_n\right) \cdot \boldsymbol{\phi}_n^T \tag{8}$$

$$= \sum_{n=1}^{N} \mathbf{w}^T \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T - \sum_{n=1}^{N} t_n \boldsymbol{\phi}_n^T = 0 \tag{9}$$

$$\sum_{n=1}^{N} \mathbf{w}^T \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T = \sum_{n=1}^{N} t_n \boldsymbol{\phi}_n^T \tag{10}$$

$$\mathbf{w}^T \sum_{n=1}^{N} \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T = \sum_{n=1}^{N} t_n \boldsymbol{\phi}_n^T \tag{11}$$

$$\mathbf{w}^T = \sum_{n=1}^{N} t_n \boldsymbol{\phi}_n^T \left(\sum_{n=1}^{N} \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T\right)^{-1} \tag{12}$$

$$\mathbf{w} = \sum_{n=1}^{N} \left(\sum_{n=1}^{N} \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T\right)^{-1} \boldsymbol{\phi}_n t_n \tag{13}$$

$$= \left(\sum_{n=1}^{N} \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T\right)^{-1} \sum_{n=1}^{N} \boldsymbol{\phi}_n t_n \tag{14}$$

Just as we introduces matrix notation for a data matrix $\mathbf{X}$, we can do the same for $\boldsymbol{\phi}$, which makes the solution very simple. Recall:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \qquad \boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_1^T \\ \boldsymbol{\phi}_2^T \\ \vdots \\ \boldsymbol{\phi}_N^T \end{bmatrix} \tag{15}$$

We can then write the solution for $\mathbf{w}$ in this form:

$$\mathbf{w} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \mathbf{t} \tag{16}$$

This is the **least-squares** solution. For generalized prediction, we would "plug-in" this $\mathbf{w}$ into y.

Example: polynomial regression with sinusoidal data

In Figure 2 we show the true underlying function (a green sinusoid) and a data set (blue circles) that is generated by randomly choosing $x \in \{0, 1\}$, then adding random Gaussian noise. Fixing this $\mathbf{X}$, the prediction for different polynomial models are shown

for $M \in \{0, 1, 3, 9\}$. Several remarks are worth noting: 1) the model becomes more flexible with more basis functions of higher order, 2) for $M = 0$ the model learns a single offset w_0, 3) with enough basis functions, the data can be interpolated *exactly*, 4) the *best* model is somewhere between $M = 1$ and $M = 9$ (maybe $M = 3$).

Regularization using Penalized Weights

The least-squares solution has only found the optimal parameters for a given model class (here polynomial of degree M), but recall we are most interested in **generalization** performance; in this case this is low (or minimal) error on unseen data. For this example we can generate test data from the same generating process and use them to approximate generalization performance. In Figure 3 (left) we see the train and test error as a function of M. As M increases, the polynomials can fit the training data exactly, but to do so they must use larger and larger weights. With larger weights, we say the model **overfits**. Idea: add a penalty to large weights to the objective function aka **regularization**.

$$E(\mathbf{w}, \lambda) = \underbrace{\frac{1}{2} \sum_{n=1}^{N} \left(\mathbf{w}^T \boldsymbol{\phi}_n - t_n \right)^2}_{E(\mathbf{w})} + \frac{\lambda}{2} \underbrace{\|\mathbf{w}\|^2}_{\mathbf{w}^T \mathbf{w}} \tag{17}$$

$$\tag{18}$$

where $\lambda \geq 0$ is an additional parameter that determines the trade-off between fitting the data and penalizing weights. It is straightforward to show that the penalized least-squares solution is:

$$\mathbf{w} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \mathbf{I} \right)^{-1} \boldsymbol{\Phi}^T \mathbf{t} \tag{19}$$

This form of regularization has several names: **penalized least-squares**, **shrinkage**, **ridge regression** (for quadratic form), and **weight decay**.

Example: penalized polynomial regression

Consider the same example above. This time, fix $M = 9$ and vary the penalty parameter λ. If $\lambda = 0$, then we get the same solution as in Figure 2 bottom-right. For each value of λ the solution is a trade-off between the error in fitting the training data and weight size. Figure 6 shows two models: one with a strong penalty (left, $\ln \lambda = 0$) and a weaker penalty (right, $\ln \lambda = -18$). In Figure 3 right we plot the generalization as a function of $\ln \lambda$; the best solution is somewhere between -35 and -25.

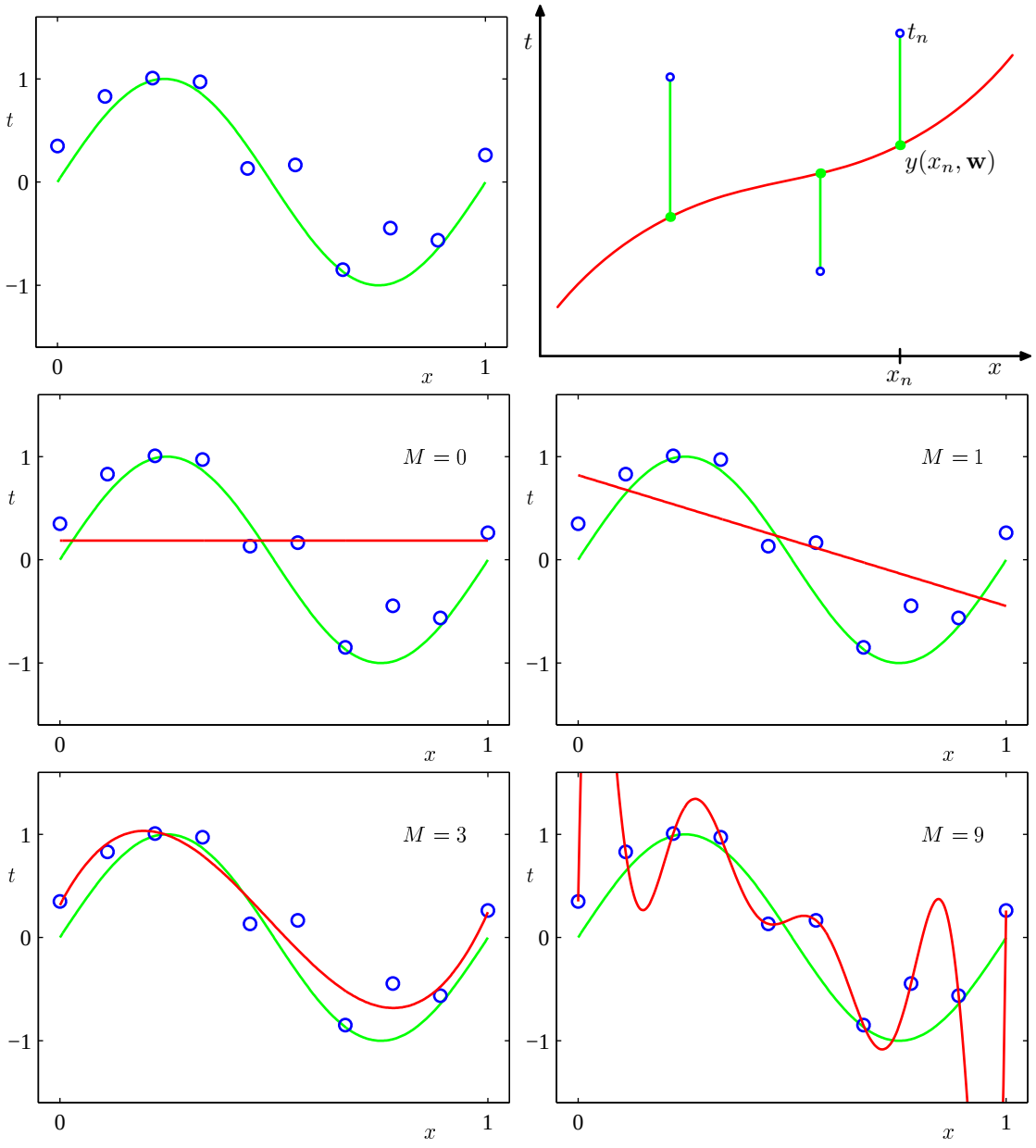

Figure 2. Polynomial regression for increasing degree M or model flexibility (Bishop Figures 1.2 and 1.4).

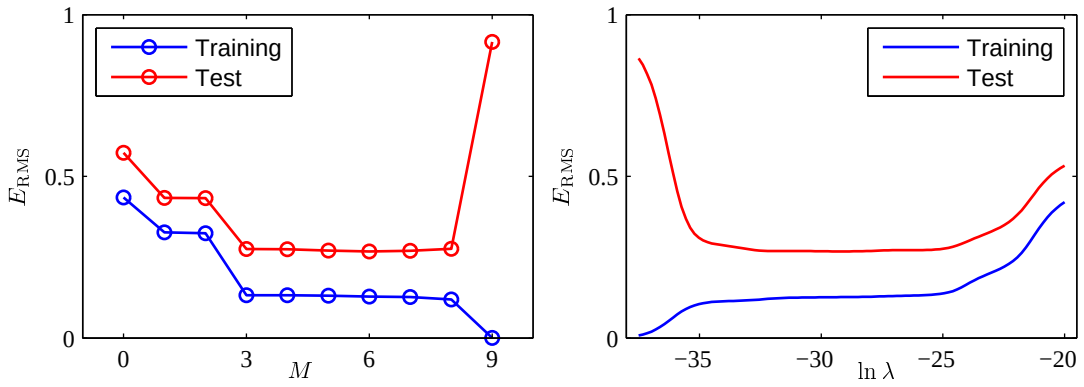

Figure 3. Left: Training and test performance as a function of the polynomial degree M (Bishop Figure 1.5). Right: Same, but for fixed $M = 9$ and as a function of $\ln \lambda$ (Bishop Figure 1.8).

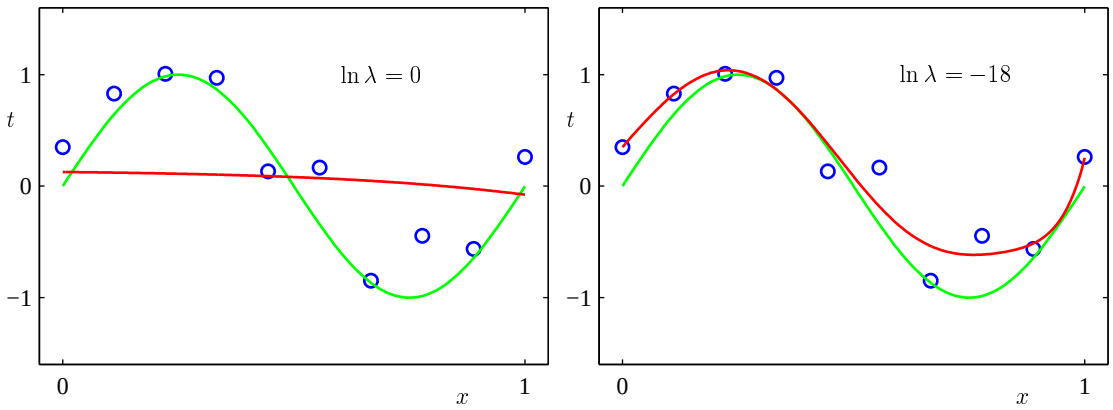

Figure 4. Fixing $M = 9$, the predictions with, left: $\ln \lambda = 0$ and right: $\ln \lambda = -18$. (Bishop Figures 1.7b, 1.7a).

Model Selection

We have seen how to optimize $\mathbf{w}$ and how to regularize the solution so that large weights are penalized. However, we have implicitly assumed we a fixed model, that is for our example, the degree of polynomial and the penalty parameter λ.

Some remarks:

- we can improve the training performance with increased M and small λ
- goal is generalization performance
- use "held-out" data to *estimate* generalization performance
- main approach: use **K-fold cross-validation** to select model

Cross-validation

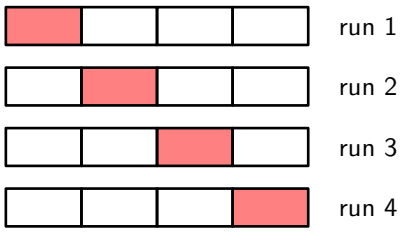

Figure 5. 4-fold cross-validation (Bishop Figure 1.18).

1. Create set of models from which we want to select (i.e. combinations of $M \in \{0, 1, 2, \ldots, L\}$ and $\ln \lambda \in \{0, -2, -4, -10, -20)$
2. For each model:
   (a) Splits training data $\mathcal{D}_{train}$ into K equally sized sub-sets. Treat each split as a test set and the other $K - 1$ sets as the training set.
   (b) For each K training and test dataset pairs, train and test.
   (c) Compute a loss over all the test sets.
3. Select model with smallest test loss.

In other words, we use the entire training set as held-out data by using the K-folds. This is a very powerful and common model selection technique: nearly all model selection procedures use some form of cross-validation in practice.

Some remarks:

- highly recommended if the number of models (aka **hyperparameters**)
- if the training set is very large, set aside a **validation** dataset and use this as a single test set
- computation issues: cross-validation is very expensive: $K \times |\Theta_1| \times |\Theta_2| \times \cdots \times |\Theta_J|$.

Other model selection methods

- AIC: penalize the model complexity, i.e. maximize $\ln p(\mathcal{D}|\mathbf{w}_M) - M$, where M is the number of adjustable parameters and $\mathbf{w}_M$ is a maximum likelihood estimator (Bishop p 33).
- BIC: similar to AIC, but maximizes $\ln p(\mathcal{D}|\mathbf{w}_M) - \frac{1}{2}M \ln N$
- problem: AIC and BIC tend to choose overly simple models.
- solution: do not limit complexity, but integrate over complexity. This tends to avoid overfitting. We will do this shortly for Bayesian linear regression.

Probabilistic Models for Linear Regression

See Bishop section 3.1.1

We can interpret the linear regression problem probabilistically. This allows us to find point estimates for $\mathbf{w}$ (both **maximum likelihood** and **maximum a posterior**) and to find a posterior distribution over $\mathbf{w}$.

Suppose the noise at each training point is Gaussian:

$$\underbrace{t}_{\text{observed target}} = \underbrace{y\left(\mathbf{x}, \mathbf{w}\right)}_{\text{deterministic}} + \underbrace{\epsilon}_{\text{additive Gaussian noise}} \tag{20}$$

i.e. $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$. What this means is: we assume that the observed targets are equal to a deterministic function (which we do not know) and some Gaussian noise centered at 0 with standard deviation σ (which we also do not know). We may instead use precision $\beta = 1/\sigma^2$ for convenience. We can **generate** data as follows: select input x_n, compute the true output y_n, add Gaussian noise ϵ_n to produce t_n; repeat.

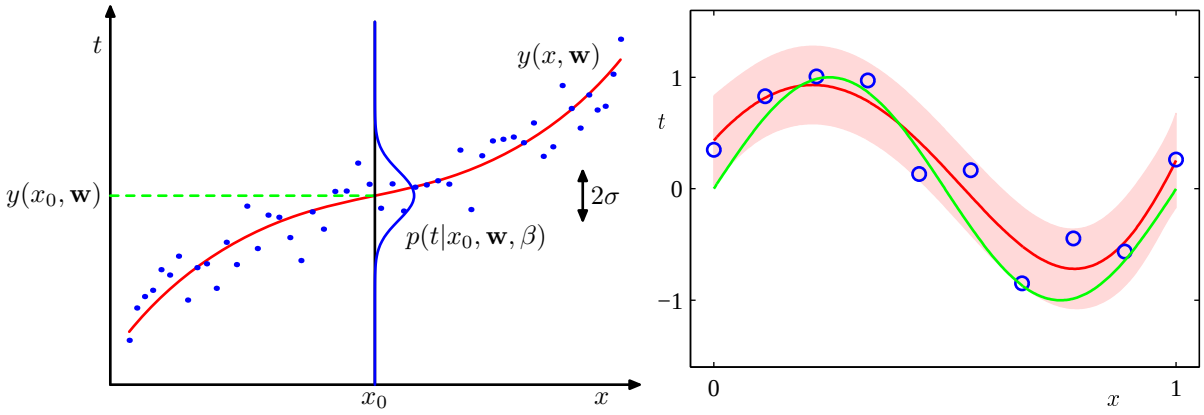

Figure 6. Probabilistic interpretation of linear regression model (Bishop Figures 1.17 and 1.18). Left: true function in red, data in blue. Right: pink region shows 2σ data noise.

We can now write the **conditional** predictive distribution for each data pair:

$$p\left(t_n|\mathbf{x}_n, \mathbf{w}, \sigma^2\right) = \mathcal{N}\left(t_n|y\left(\mathbf{x}_n, \mathbf{w}\right), \sigma^2\right) \tag{21}$$

Now consider a data set $\mathbf{X}$, $\mathbf{t}$. The **likelihood** function is

$$p\left(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2\right) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|y\left(\mathbf{x}_n, \mathbf{w}\right), \sigma^2\right) \tag{22}$$

Remarks. The likelihood function assumes data are generated independently and identically. The likelihood function is conditional, meaning we model the output distribution

conditioned on the input $p(t|\mathbf{x})$, and do not model the input distribution $p(\mathbf{x})$. Recall that the likelihood is one piece of a complete probabilistic model of the data and parameters, i.e. $p(\mathcal{D}|\boldsymbol{\theta})$.

Maximum Likelihood

We now find $\mathbf{w}$ that maximizes the likelihood function:

$$\mathbf{w} = \arg\max_{\mathbf{w}} \left(p\left(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2\right)\right) \tag{23}$$

Since we are maximizing, we can equivalently maximize over the **log-likelihood**, which is useful because it changes the maximization over a product to a maximization over a summation (gradients are easier), and the log-likelihood is much more stable numerically (in practice, do all probability computations in log-space, if possible). The log-likelihood is:

$$\ln p\left(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2\right) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n | y\left(\mathbf{x}_n, \mathbf{w}\right), \sigma^2\right) \tag{24}$$

$$= -\frac{N}{2}\ln(\sigma^2) - \frac{N}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}\left(t_n - y\left(\mathbf{x}_n, \mathbf{w}\right)\right)^2 \tag{25}$$

$$= -\frac{N}{2}\ln(\sigma^2) - \frac{N}{2}\ln(2\pi) - \frac{1}{\sigma^2}E(\mathbf{w}) \tag{26}$$

where $E(\mathbf{w})$ is the sum of squares error objective function we used previously. This maximization is equivalent to minimizing $E(\mathbf{w})$.

Maximum likelihood $\mathbf{w}_{\mathbf{MLE}}$

The objective function $O()$ is $\ln p\left(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2\right)$:

$$\mathbf{w}_{\mathrm{MLE}} = \arg\max_{\mathbf{w}} \left(-\frac{N}{2}\ln(\sigma^2) - \frac{N}{2}\ln(2\pi) - \frac{1}{\sigma^2}E(\mathbf{w})\right) \tag{27}$$

Find a maxima by setting the gradient to 0, and solving for $\mathbf{w}$:

$$\nabla O(\mathbf{w}) = -\nabla\left(\frac{N}{2}\ln(\sigma^2)\right) - \nabla\left(\frac{N}{2}\ln(2\pi)\right) - \frac{1}{\sigma^2}\sum_{n=1}^{N}\nabla\left(\left(t_n - y\left(\mathbf{x}_n, \mathbf{w}\right)\right)^2\right) \quad (28)$$

$$= 0 + 0 - \frac{1}{\sigma^2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}_n)\boldsymbol{\phi}_n^T = 0 \quad (29)$$

$$0 = \sum_{n=1}^{N}\mathbf{w}^T\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T - \sum_{n=1}^{N}t_n\boldsymbol{\phi}_n^T \quad (30)$$

$$\sum_{n=1}^{N}\mathbf{w}^T\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T = \sum_{n=1}^{N}t_n\boldsymbol{\phi}_n^T \quad (31)$$

$$\mathbf{w}^T\sum_{n=1}^{N}\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T = \sum_{n=1}^{N}t_n\boldsymbol{\phi}_n^T \quad (32)$$

$$\mathbf{w}^T = \sum_{n=1}^{N}t_n\boldsymbol{\phi}_n^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T\right)^{-1} \quad (33)$$

$$\mathbf{w} = \sum_{n=1}^{N}\left(\sum_{n=1}^{N}\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T\right)^{-1}\boldsymbol{\phi}_n t_n \quad (34)$$

$$= \left(\sum_{n=1}^{N}\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T\right)^{-1}\sum_{n=1}^{N}\boldsymbol{\phi}_n t_n \quad (35)$$

$$= \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t} \quad (36)$$

In other words, the maximum likelihood solution is exactly the same found by minimizing the sum of squares error objective, i.e. $\mathbf{w}_{\text{MLE}} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$.

Maximum likelihood $\beta^{\mathbf{ML}}$ Our predictions y_n only depend on $\mathbf{w}$ (or $\mathbf{w}_{\text{MLE}}$), not β, but it is still informative to solve for β^{ML}. Using the same objective, the log-likelihood, we can maximize over $\beta = 1/sigma^2$ given our maximum likelihood estimate $\mathbf{w}_{\text{MLE}}$:

$$\nabla O(\beta) = \frac{N}{2\beta} - \frac{1}{2}\sum_{n=1}^{N}\left(t_n - y\left(\mathbf{x}_n, \mathbf{w}_{\text{MLE}}\right)\right)^2 = 0 \quad (37)$$

$$\beta^{\text{ML}} = \frac{N}{\sum_{n=1}^{N}\left(t_n - y\left(\mathbf{x}_n, \mathbf{w}_{\text{MLE}}\right)\right)^2} \quad (38)$$

$$\sigma^2_{\text{ML}} = \frac{1}{N}\sum_{n=1}^{N}\left(t_n - y\left(\mathbf{x}_n, \mathbf{w}_{\text{MLE}}\right)\right)^2 \quad (39)$$

I.e. the maximum likelihood estimate for the variance is the average empirical residual squared error or residual variance.

Maximum a posterior, MAP

We can also assume a prior distribution over the weights $p(\mathbf{w})$. This allows us to write the posterior distribution over $\mathbf{w}$ as:

$$p(\mathbf{w}|\mathcal{D}) = p(\mathcal{D}|\mathbf{w})p(\mathbf{w})/Z \tag{40}$$

where Z is a normalizing constant that, importantly for MAP estimation, does not depend on $\mathbf{w}$. We can therefore find $\mathbf{w}_{\text{MAP}}$ that maximizes the posterior distribution (we will use the log-posterior):

$$\ln p\left(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2\right) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n|y\left(\mathbf{x}_n, \mathbf{w}\right), \sigma^2\right) + \ln(p(\mathbf{w})) - \ln(Z) \tag{41}$$

$$= C - \frac{1}{\sigma^2} E(\mathbf{w}) + \ln(p(\mathbf{w})) \tag{42}$$

where $C = -\frac{N}{2}\ln(\sigma^2) - \frac{N}{2}\ln(2\pi) - \ln(Z)$ (i.e. a constant that does not depend on $\mathbf{w}$). If we now assume a Gaussian prior over $\mathbf{w}$, centered at the zero vector and with diagonal covariance, i.e. $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{1}/\alpha\mathbf{I})$, we end up with exactly the same expression we had for penalized least-squares regression:

$$\ln p\left(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2\right) = -\frac{\beta}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}_n\right) - \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} \tag{43}$$

If we let $\lambda = \alpha/\beta$ then this is the same objective (except we are maximizing the log-posterior rather than minimizing the penalized sum of squares). With the same manipulation, we can solve for $\mathbf{w}_{\text{MAP}} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$.
TODO: rewrite product of univariate gaussians as joint distribution / multivariate gaussian.
TODO: penalty on bias is different.

Summary of Least Squares for Linear Regression

	Least Squares		Probabilistic	
Objective	$E(\mathbf{w})$	$E(\mathbf{w}, \lambda)$	$\ln(p(\mathcal{D}\|\mathbf{w}))$	$\ln(p(\mathcal{D}\|\mathbf{w})p(\mathbf{w}))$
Solution $\mathbf{w}$	$\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$	$\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$	$\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$	$\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$