MLPR Week 6a

Bayesian linear regression **prior**:

    draw weight, bias from gaussians

    = gaussian process (defining function value) with specific kernel

        specified by parameters σ2w and σb2 (hyperparameters)

weight view (W^T X) & function value view (fi fj)

$$\tilde{f}_i = f(\mathbf{x}^{(i)}) = \mathbf{w}^\top \mathbf{x}^{(i)} + b, \qquad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma_w^2 \mathbb{I}), \;\; b \sim \mathcal{N}(0, \sigma_b^2)$$

$$\begin{aligned}
\mathrm{cov}(\tilde{f}_i, \tilde{f}_j) &= \mathbb{E}[\tilde{f}_i \tilde{f}_j] - \overbrace{\mathbb{E}[\tilde{f}_i]\mathbb{E}[\tilde{f}_j]}^{0} \\
&= \mathbb{E}\left[(\mathbf{w}^\top \mathbf{x}^{(i)} + b)^\top (\mathbf{w}^\top \mathbf{x}^{(j)} + b)\right] \\
&= \sigma_w^2 \mathbf{x}^{(i)\top} \mathbf{x}^{(j)} + \sigma_b^2 = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}).
\end{aligned}$$

Since

    Line 3: Since expectation over w, could take out x

    also line 3: terms in "…" are combination of w and b,

        since w and b are independent, "…" = 0

$$\begin{aligned}
&= \mathbb{E}\left[(\underline{w}^T \underline{x}^{(i)} + b)^T (\underline{w}^T \underline{x}^{(j)} + b)\right] \\
&= \mathbb{E}\left[\underline{x}^{(i)T} \underline{w}\,\underline{w}^T \underline{x}^{(j)} + b^2 + \ldots\right] \\
&= \underline{x}^{(i)T} \underbrace{\mathbb{E}[\underline{w}\,\underline{w}^T]}_{\sigma_w^2 \mathbb{I}} \underline{x}^{(j)} + \underbrace{\mathbb{E}[b^2]}_{\sigma_b^2} + \underbrace{\ldots}_{0} \\
\Rightarrow k(\underline{x}^{(i)}, \underline{x}^{(j)}) &= \sigma_w^2 \underbrace{\underline{x}^{(i)T} \underline{x}^{(j)}}_{\underline{\phi}(\underline{x}^{(i)})^T \underline{\phi}(\underline{x}^{(j)})} + \sigma_b^2
\end{aligned}$$

kernel trick

Drawback of finite RBFs in Bay linear regression:

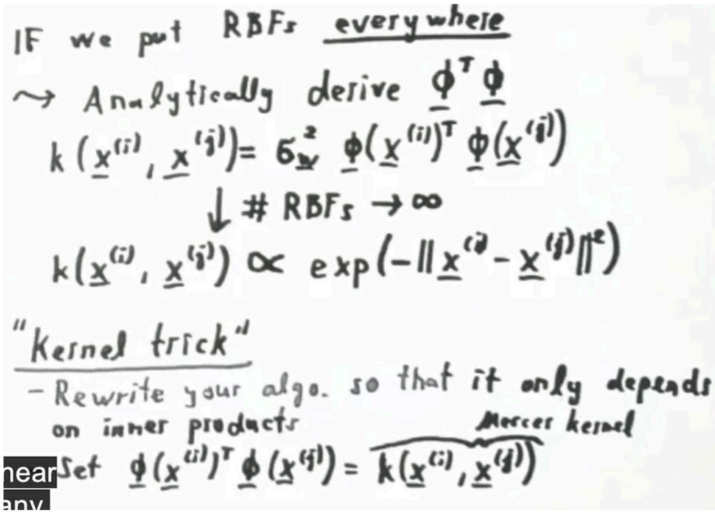
1. selected manually, not good for exploration
2. If x move away from origin, always zero ("under fitting")

put RBFs everywhere

    = **infinite RBFs, = mercer kernel**

    infinite dimension feature vector, but **take inner product, only get scalar value**

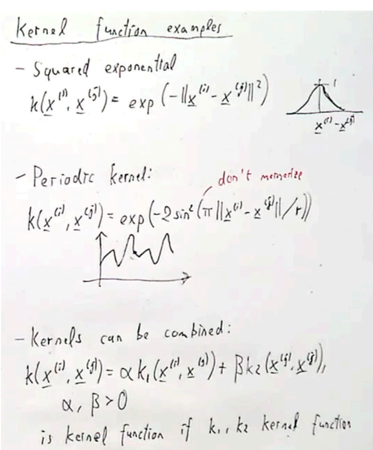
=> kernel trick / kernelized: Replacing an inner product with a kernel function



GP Prediction: in week 5 bottom

Can we be more uncertain at prediction in response to a surprising training label?
=> change GP kernel function, learn the parameters in kernel

Choosing a family of kernel functions



Another kernel function with learnable parameters: RBF kernel
hyperparameters σ_f (marginal variance), l_d (curve width)

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}(x_d^{(i)} - x_d^{(j)})^2/\ell_d^2\right)$$

The marginal variance of one function value $\tilde{f}_i$ is:

$$\mathrm{var}[\tilde{f}_i] = k(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}) = \sigma_f^2.$$

larger marginal variance = larger width in y dimension
smaller l_d, get more wiggly curve

Learning the hyperparameters: choose the parameters (σ_f and l_d) with largest marginal likelihood (calculate marginal likelihood = just copy values)

Pick parameters by (marginal) likelihood:

$$p(\underline{y} \mid X, \theta = \{\sigma_y^2, \sigma_f^2, \{l_d\}\})$$
$$= \mathcal{N}(\underline{y}; \underline{0}, K(X, X) + \sigma_y^2 \mathbb{I})$$

so log P(y | x, M) = log of the standard multivariate Gaussian pdf

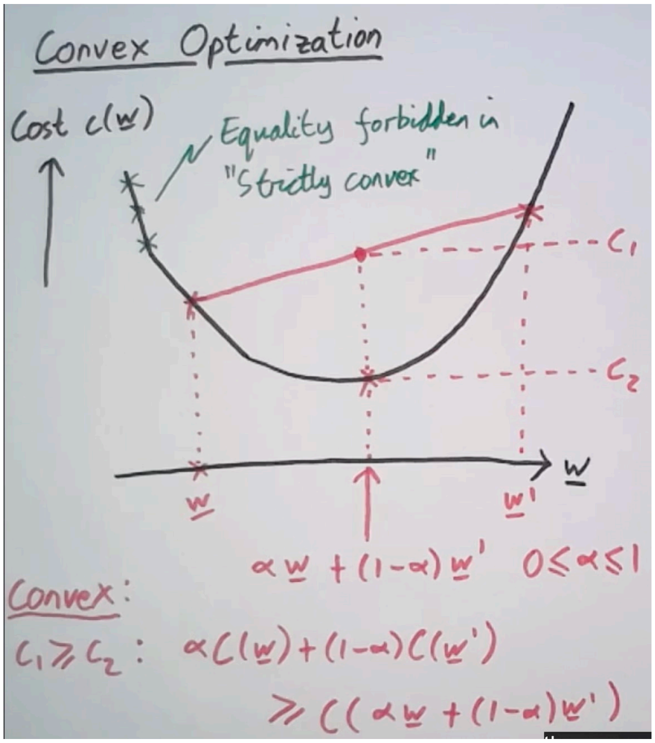
covariance = kernel matrix evaluated at the training inputs + observation noise

fully Bayesian approach integrating over all possible hyperparameters, but can’t be computed exactly

Week6b

gradient descent not constrained,
If step size not small enough ,could take weight into illegal value / not converge
=> if got constrained parameters (e.g. positive), take log (exponential)
obtain that derivative with the chain rule

Convex:
if convex, guarantee minimum
convex (could have straight line) VS strictly convex
norms are convex
sum of any convex functions is convex



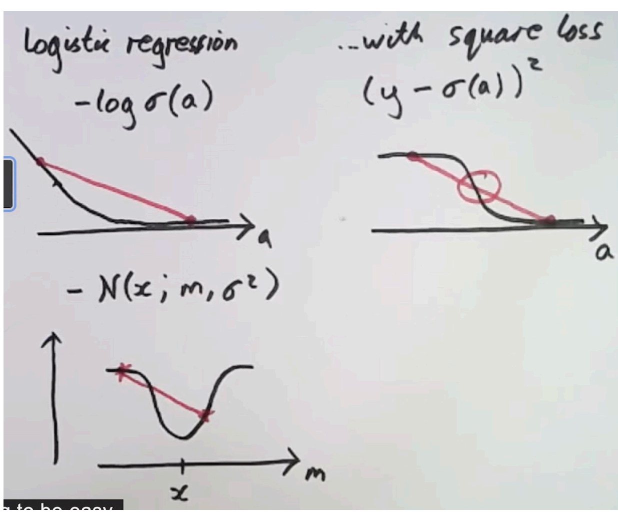
Not convex:

Square loss, not convex optimization problem, harder to optimize

why take log instead of just optimize probability:

gaussian is not convex (one reason)

++ (sum over examples, avoid numerical underflow)



if not convex, could still try to reduce this loss function using gradient methods

Week6c How to develop Softmax loss function (regression):

Define likelihood P(Y | X, W), want positive score, normalized to sum to 1

log-probability of a single observation:

C: y of the current sample
K: the current class when doing sum

Kronecker delta kc: if k and c the same, get 1

finally take x out

$$\log P(y_c\!=\!1\,|\,\mathbf{x}, W) = \log f_c = (\mathbf{w}^{(c)})^\top \mathbf{x} - \log \sum_{k'} e^{(\mathbf{w}^{(k')})^\top \mathbf{x}}$$

$$\nabla_{\mathbf{w}^{(k)}} \log f_c = \delta_{kc}\mathbf{x} - \frac{1}{\sum_{k'} e^{(\mathbf{w}^{(k')})^\top \mathbf{x}}}\,\mathbf{x}\, e^{(\mathbf{w}^{(k)})^\top \mathbf{x}}$$
$$= (y_k - f_k)\,\mathbf{x}.$$

Derivative: if label & function output different, move in the direction of the input

decrease the scorer of all other classes (in denominator of softmax function)

++ dimension of w: each row is w(k), each col is feature, total KxD

Redundant parameters: (when divided by numerator)

when only have 2 classes, = logistic regression
but only depends on difference between w1 and w2, could set w2 = 0

in multi class case, weight matrix is KxD, could set one row = 0

in neural network most parameters are redundant (over-parameterized)

when have over-parameterized model, don't try to interpret parameters

$$P(y\!=\!1\,|\,\mathbf{x}, W) = \frac{e^{(\mathbf{w}^{(1)})^\top \mathbf{x}}}{e^{(\mathbf{w}^{(1)})^\top \mathbf{x}} + e^{(\mathbf{w}^{(2)})^\top \mathbf{x}}}$$
$$= \frac{1}{1 + e^{(\mathbf{w}^{(2)} - \mathbf{w}^{(1)})^\top \mathbf{x}}} = \sigma\big((\mathbf{w}^{(1)} - \mathbf{w}^{(2)})^\top \mathbf{x}\big).$$

Week6d Robust logistic regression

have corrupted label, want to describe this mathematically, derive a different loss binary choice m (m=1, correct / m=0, corrupt label) for each observation

since true label hidden

epsilon = very small probability

Assume Epsilon is entirely random, make the choice independent of the input position and the weights (or…)

$$P(m \mid \epsilon) = \text{Bernoulli}(m; 1-\epsilon) = \begin{cases} 1-\epsilon & m=1 \\ \epsilon & m=0. \end{cases}$$

If correct label (m=1), do nothing, if corrupt label, output at random (Or…)

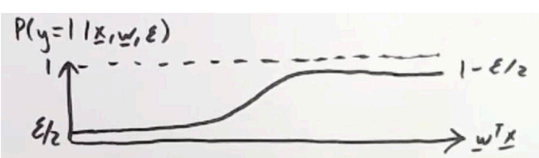
$$P(y=1 \mid \mathbf{x}, \mathbf{w}, m) = \begin{cases} \sigma(\mathbf{w}^\top \mathbf{x}) & m=1 \\ \frac{1}{2} & m=0. \end{cases}$$

Introduce m by sum & product rule

After product rule, cancel out x & w, since m does not depends on those

$$\begin{aligned} P(y=1 \mid \mathbf{x}, \mathbf{w}, \epsilon) &= \sum_{m \in \{0,1\}} P(y=1, m \mid \mathbf{x}, \mathbf{w}, \epsilon) \\ &= \sum_{m \in \{0,1\}} P(y=1 \mid \mathbf{x}, \mathbf{w}, m)\, P(m \mid \epsilon) \\ &= \underbrace{(1-\epsilon)\sigma(\mathbf{w}^\top \mathbf{x})}_{m=1} + \underbrace{\epsilon \frac{1}{2}}_{m=0}. \end{aligned}$$

Result: model could not be 100% confident in prediction



Get the Gradient, = correction term + normal logistic gradient

Tweak epsilon

get epsilon