Ex 7.1

a) As we know,

$$y = wx + b$$

we can see b, the bias is just the intercept whereas w is the slope that needs to be smoothened in order to minimize error. Here, the inputs are also constant so we need to penalize the weights in a way that it shrinks to smaller values minimizing the error while keeping the computational complexity minimal.

b) L1 regularization encourages sparser weights by driving the weights as near or equal to 0 if possible. Whereas L2 doesn't result in such sparse weights. Therefore, it allows us to explore the impact of the penalty with the shrinked weights. In cases where we want to select some features, we can use L1 regularizations. We should prefer L2 when we want to explore the relation between co-dependant features as L2 wouldn't lead any of them to zero.

c) It is possible to combine both L1 and L2 so we can receive both the benefit of sparsity generated by L1 and the nuance of L2 to analyze the impact of penalty on

the weights. This can eliminate less predictive features while well-describing the correlated features.

d) The statement is true. During the training process weights can get very large to accomodate the unseen input data. Therefore, we modify the learning algorithm to generate smaller weights while reducing training error. This stabilizes the network better and leads to faster generalization of the model. Appropriate regularization can thus reduce overfitting and output a better model.

e) Linear regression model with 1 feature:

$$y = wx + b$$

where the cost function

$$\text{Loss} = \text{error}(y, \hat{y})$$

For L1 regularization it becomes:

$$\text{Loss} = \text{error}(y, \hat{y}) + \lambda |w|$$

Therefore the derivative is:

$$\frac{\partial \text{Loss}}{\partial w} = \frac{\partial}{\partial w} \text{error}(y, \hat{y}) + \lambda$$

As the derivative loses the w term, it becomes a constant and in a few iterations our w can become 0.

for L2:

$$\text{Loss} = \text{error}(y, \hat{y}) + \lambda w^2$$

$$\frac{\partial \text{Loss}}{\partial w} = \frac{\partial}{\partial w} \text{error}(y, \hat{y}) + 2\lambda w$$

Here, the derivative isn't constant so convergence will be much slower. As the w term gets smaller, it will tend to be closer to 0 but never become 0.

Ex. 7.2

a) Given, $\tilde{J}(w) = J(w) + \frac{\lambda}{2} w^T . w$

$$\frac{\partial \tilde{J}(w)}{\partial w} = \frac{\partial J(w)}{\partial w} + \frac{\lambda}{2} \cdot 2w$$

$$= \frac{\partial J(w)}{\partial w} + \lambda w \qquad \left[\frac{\partial}{\partial x}(x^T . x) = 2x\right]$$

So weight update rule:

$$w_{i+1} = w_i - \eta \left(\frac{\partial J(w)}{\partial w} + \lambda w \right)$$

The L2 regularisation is also known as weight decay as it shrinks the weights into smaller weights. As seen in 7.2 (e), L2 regularization has the w term in the derivative of the loss function similar to this case. This indicates the given function will also shrink the weights and regularize the network similar to weight decay approach.

b) Given the relationship of $\tilde{w}$ and w^* as:

$$\tilde{w} = (H + \alpha I)^{-1} H w^*$$

where,

$$H = Q \Lambda Q^T$$

Since eigenvalues λ_i of H are scaled as $\frac{\lambda_i}{\lambda_i + \alpha}$,

in case 1 when $\lambda_i >> \alpha$

$$\Rightarrow \lambda_i \approx 1$$

$$\Rightarrow \Lambda \approx I$$

$$\therefore H \approx Q I Q^T \approx Q Q^T \approx I$$

$$\tilde{w} \approx (I + \alpha I)^{-1} I w^*$$

$$\approx (I(1+\alpha))^{-1} \cdot w^*$$

$$\approx \left(\frac{1}{1+\alpha}\right) w^*$$

in case 2 when $\lambda_i << \alpha$

$$\Rightarrow \lambda_i \approx 0$$

$$\Rightarrow \Lambda \approx 0$$

$$\therefore H \approx 0$$

Hence, $\tilde{w} \approx 0$.

c) The loss function with L2 regularization:

$$L = \sum_{n=1}^{N} (\hat{y} - y)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

Using Bayes rule we have,

$$P(w|D) = \frac{P(D|w) \cdot P(w)}{P(D)}$$

$$\propto P(D|w) \cdot P(w)$$

$$\propto \left[\prod_{n=1}^{N} \mathcal{N}(\hat{y}; y, \sigma_y^2)\right] \cdot \mathcal{N}(w; 0, \sigma_w^2)$$

$$\propto \left[\prod_{n=1}^{N} \mathcal{N}(\hat{y}; y, \sigma_y^2)\right] \cdot \prod_{k=1}^{k} \mathcal{N}(w_i; 0, \sigma_w^2)$$

Taking the negative log probability

we can write:

$$-\log\left[p(w|D)\right] = -\sum_{n=1}^{N} \log\left[\mathcal{N}(\hat{y}; y, \sigma_y^2)\right]$$

$$-\sum_{i=1}^{k} \log\left[\mathcal{N}(w_i; 0, \sigma_w^2\right] + c$$

Here we can see the log probability of posterior distribution is equal to L2 regularized square loss where gaussian prior is denoted as $\mathcal{N}(w; 0, \sigma_w^2)$.

d) As known,

$$\hat{J}(\theta) = J(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*)$$

$$\nabla_w \hat{J}(w) = H(w - w^*)$$

since $w^* = \arg\min_w J(w)$

$$\nabla_w \hat{J}(w) = 0 = H(w - w^*)$$

At any given iteration k

$$w^k = w^{k-1} - \epsilon \nabla_w \hat{J}(w^{k-1})$$
$$= w^{k-1} - \epsilon H(w^{k-1} - w^*)$$

Hence, $w^k - w^* = (I - \epsilon H)(w^{k-1} - w^*)$

$$w^k - w^* = (I - \epsilon Q \Lambda Q^T)(w^{k-1} - w^*)$$
$$Q^T(w^k - w^*) = (I - \epsilon \Lambda) Q^T (w^{k-1} - w^*)$$

As the α is chosen very small and initial $w(0) = 0$:

$$Q^T w^K = \left[I - (I - \epsilon\Lambda)^K\right] Q^T w^* \quad \text{— (1)}$$

Weights obtained by L2 regularization is known as per lecture:

$$\tilde{w} = (\Lambda + \alpha I)^{-1} Q \Lambda Q^T w^*$$

$$Q^T \tilde{w} = \left[I - (\Lambda + \alpha I)^{-1} \alpha\right] Q^T w^* \quad \text{— (2)}$$

Comparing eq. 1 and 2:

$$(I - \epsilon\Lambda)^K = (\Lambda + \alpha I)^{-1} \alpha$$

e) The formula for label smoothing adding noise is:

$$l_i' = (1 - \epsilon) * l_i + \frac{\epsilon}{c}$$

where ϵ is the noise and c is the number of classes.

Given, $[0, 1, 0, 0]$, the smoothened labels with a noise of 0.1 would be:

$$l' = (1 - 0.1) * 0 + 0.1$$

$$l_1' = (1-0.1) * 0 + \frac{-1}{4}$$

$$= 0.025$$

$$l_2' = (1-0.1) * 1 + \frac{0.1}{4}$$

$$= 0.925$$

$$l_3' = 0.025$$

$$l_4' = 0.025$$

Ex 7.3

a) Dropout is a regularization method that randomly drops some neurons and creates a different configuration to be trained by the neural network. This allows training multiple architectures in paralled forcing the network to learn or even fix previous mistakes of previous layers. This makes the model more robust and reduces overfitting.

b) Both Bagging and dropouts train the model using variation of training data. But bagging uses subsamples of training data whereas dropout introduces new

data by randomly activating/ deactivating neurons from the same input. Both offers the model to learn on different configurations but bagging due to subsampling specializes the network in different parts whereas dropout helps to train the model as a whole with subnetworks.

- Inverted dropout considers a probability p such that the activations are scaled by $q = 1 - p$

This allows the activations to stay small and helps during inference as no major change is required. But traditional dropout requires scaling before inference.