Part A

1. One of the vectors that determine the decision boundary of an SVM, or intuitively the points that are closest to the decision boundary, or points associated with a non-zero dual parameter.

2. Regularization. Prevent the network weights from getting too large. Prevent overfitting.

3. Parallel training: train the same classifier on different (bootstrapped) datasets in parallel.
   Sequential training: train the same classifier on weighted datasets. The weights used in one iteration depends on the performance of the previous classifiers in the previous iteration.
   Bagging - parallel training, boosting - sequential training.

4. Generative classifiers model the input distributions for different classes, discriminative classifiers don’t (they model the decision surface directly).
   Discriminative classifiers: SVM, logistic regression, neural networks, k-nearest neighbors.
   Generative classifiers: Naive Bayes classifier, Gaussian Bayes classifier, and other Bayes classifiers.

5. In both cases, the decision boundary is piecewise linear. Decision trees do axis-aligned splits while 1-NN gives a voronoi diagram.

Part B

1. (A)

   \[ p(\{x_1, ..., x_n\}|\theta) = \prod_{i=1}^{n} p(x_i|\theta) = \prod_{i=1}^{n} \sum_{k=1}^{K} p(z=k|\theta) \frac{1}{(2\pi\sigma_k^2)^{d/2}} \exp \left( -\frac{|x - \mu_k|^2}{2\sigma_k^2} \right) \]

   (B) For \( K = 1 \), yes. The log-likelihood of data has a simple form

   \[ \log p(\{x_1, ..., x_n\}|\theta) = \sum_{i=1}^{n} \log \left[ p(z=1|\theta) \frac{1}{(2\pi\sigma_1^2)^{d/2}} \exp \left( -\frac{|x - \mu_1|^2}{2\sigma_1^2} \right) \right] \]

   \[ = \sum_{i=1}^{n} -\frac{|x_i - \mu_1|^2}{2\sigma_1^2} - \frac{d}{2} \log(2\pi\sigma_1^2), \]

   where \( p(z=1|\theta) = 1 \) as \( K = 1 \). This is easy to optimize with gradient ascent, it even has an analytic solution.

   For \( K = 2 \), the log-likelihood is

   \[ \log p(\{x_1, ..., x_n\}|\theta) = \sum_{i=1}^{n} \log \left[ p(z=1|\theta) \frac{1}{(2\pi\sigma_1^2)^{d/2}} \exp \left( -\frac{|x - \mu_1|^2}{2\sigma_1^2} \right) \right] + p(z=2|\theta) \frac{1}{(2\pi\sigma_2^2)^{d/2}} \exp \left( -\frac{|x - \mu_2|^2}{2\sigma_2^2} \right), \]

   which can still be optimized with gradient ascent, but it’s a lot more complicated. A better approach is to use the EM algorithm.

   (C) Increasing the number of components by one gives the model more capacity and should fit the training data better, so the training data should have higher likelihood under the new model. The likelihood of test data may be better or worse depending on whether the model is underfitting or overfitting the data.
2. (A) The decision rule: ‘+’ if \(x + y > 1\), ‘−’ if \(x + y < 1\) Decision boundary is shown in Figure 1. Data cases 1, 3, and 4 are support vectors.

(B) Kernel trick is the the trick of replacing all dot-products of form \(\phi(x)^T \phi(y)\) with a kernel \(k(x, y)\). It directly works on \(x\), avoids computing and defining \(\phi\), and implicitly maps \(x\) to a high dimensional space.

(C) Non-linear SVMs have nonlinear feature transformation functions, and have nonlinear decision boundaries.

(D) Non-linear SVMs use fixed feature transformations (implicitly defined using the kernel), neural networks use learnable feature transformations. Non-linear SVM optimization is convex and has a unique global optimum, neural networks are non-convex and have a lot of local optima. SVMs scale with number of data examples, neural nets scale with the number of input dimensions.

3. (A) All activation functions must be \(L\) as shown below.

(B) See below

(C) The function represented by the neural network in (B) is
\[
f(x) = S(w_5c(w_1x_1 + w_3x_2) + w_6c(w_2x_1 + w_4x_2)) = S(c(w_1w_5 + w_2w_6)x_1 + c(w_3w_5 + w_4w_6)x_2).\]

The sigmoid signed function \(S\) is equivalent to the standard sign function (1 when \(a > 0\) and -1 when \(a < 0\)). Compare this to the binary logistic regression classifier. In a binary logistic
regression classifier, when $\beta_1 x_1 + \beta_2 x_2 > 0$ the output is $y = 1$, otherwise the output is $y = -1$. We therefore have

\[
\beta_1 = c(w_1 w_5 + w_2 w_6) \\
\beta_2 = c(w_3 w_5 + w_4 w_6)
\]

(D) See below

(E) Here each of the two first layer hidden units corresponds to one logistic regression classifier, and the last layer combines the two. The combination weights $\alpha_1$ and $\alpha_2$ directly corresponds to $w_5$ and $w_6$. $\alpha_1 = w_5, \alpha_2 = w_6$ or $\alpha_1 = w_6, \alpha_2 = w_5$.

4. (A) See below

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>left: 0</td>
<td>right: +8.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>right: +7.29</td>
<td>down: 0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>up: +6.56</td>
<td>down: 0</td>
<td>right: -9</td>
<td>up: +8.1</td>
</tr>
<tr>
<td>C</td>
<td>up: +5.90</td>
<td>left: +5.31</td>
<td>left: 0</td>
<td>right: 0</td>
</tr>
</tbody>
</table>

(B) Policy is shown in Figure 2. Optimal must mean shortest path to upper right, since reward worth less the further in future it occurs, as the objective is average reward $\frac{1}{T}\sum r_t$, bigger $T$ means less average reward. Need more exploration, should go right at C2.

(C) Strengths – not spending lots of time exploring useless actions; exploit knowledge; saves computation.

Weaknesses – may miss/never find optimal policy; sensitive to initial conditions.

(D) We can represent the $Q$ function as a parametric model instead of a table, and the same model is used everywhere, that maps a state representation and an action to a value. This same model can therefore generalize across states. The states can be represented by, for example, the coordinates, distance to terminal states, etc.
Figure 2: Learned policy

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>+10</td>
</tr>
<tr>
<td>B</td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>-10</td>
</tr>
<tr>
<td>C</td>
<td>↑</td>
<td>←</td>
<td>↑</td>
<td>←</td>
</tr>
</tbody>
</table>