ECE 498 ICC Spring 2020 Jin Yucheng (yucheng9)

Homework 1 Problem 1

Solution:

(1) There are mainly four reasons,

- **Parallelism**: tf.Graph explicitly shows data dependencies, making it easier for system to figure out operations that can be executed in parallel.
- **Distributed Execution**: tf.Graph helps system to partition excution into multiple devices.
- **Compilation**: tf.Graph can enable compiler optimization (e.g. fusion).
- **Portability**: tf.Graph has the language-independent property, which makes it easier for users to transform models between different platforms.

(2) For Cloud Computing,

- **Advantages**: 1. Cloud Computing is a centralized design where data producers deliver data to the data centers (cloud), and data consumers request data from these data centers, which makes it easier to control and monitor data, while gives end devices remote access to the data. 2. Cloud Computing is capable of storing a huge amount of data, and enables global collaboration via networks.
- **Disadvantages**: 1. Sending all data to cloud is not efficient, since data redundancy, bandwidth occupation, and slow response time could all significantly influence the performance of Cloud Computing. 2. Data often need some changes from producers to consumers (for example, for Youtube users, videos can be adjusted before uploading to cloud), and it requires more functions at the edge, which makes Cloud Computing an expensive choice.

For Edge Computing,

- **Advantages**: 1. Edge Computing pushes intelligence and processing capabilities to the network edge, which eliminates lag-time and saves bandwidth. 2. Connectivity, data migration, bandwidth, and latency features are pretty expensive in Cloud Computing, but in Edge Computing, data is from near sources instead of cloud, which makes it much less expensive compared to Cloud Computing.
- **Disadvantages**: 1. Edge Computing requires higher standards for end devices, for example, end devices should have both high computational efficiency and low power consumption to give users a good experience. 2. Edge Computing requires more robust security protocols, such as strong authentication methods to proactively tackle with attacks.

(3)

- Cloud Computing. Because Apple and Fitbit are big companies with a lot of users, so they need to collect a huge amount of data, and therefore Cloud Computing is preferable. Also, by uploading data to cloud, your doctor, your family can access your health data. Additionaly, Edge Computing plays a role before uploading your data to cloud, for example, the Apple watch may first perform some computations based on the data collected by sensors.
- Edge Computing. Because the data from temperature sensors can be directly transmitted to the processing unit, and it is expensive to first upload data to cloud then access data from cloud.
- Edge Computing. Because under this situation, less latency is important since no one wants to wait too long, and Edge Computing directly computes the data from sensors which makes the entire process faster.
- I think the answer is both Cloud Computing and Edge Computing. Because this wearable device should send an emergency alert to local hosptials (or your families and friends), it needs to upload your data to cloud and enable other people to know your situation. Edge Computing is also required since this device should know you are falling based on computations on the data from sensors.

■

ECE 498 ICC Spring 2020 Jin Yucheng (yucheng9)

Homework 1 Problem 2

Solution:

(1) The loss function can be rewrite as,

$$J(x) = \begin{cases} x^2 & x \leq 0 \\ 0 & 0 < x < 2 \\ -x^2 + 2x & x \geq 2 \end{cases}$$

the gradient of $J(x)$ is,

$$J^{'}(x) = \begin{cases} 2x & x \leq 0 \\ 0 & 0 < x < 2 \\ -2x + 2 & x \geq 2 \end{cases}$$

because the learning rate is very small, if we start from $x = -2$, since the loss function is monotonously decreasing from -2 to 0, and when x reaches 0 the gradient also becomes 0, so finally x is stuck to 0, and the value of loss function is 0, which is a local minimum.

If we start from $x = 3$, since the loss function is monotonously decreasing from 3 to ∞, and in theory, gradient descent brings x to ∞ with $J(x) = -\infty$.

(2) No. Because $x^2 + x + 1 = 0$ has no real roots, if we use gradient descent, the loss function will be (if the value of $J(x)$ is smaller, then $x^2 + x$ is nearer to -1),

$$J(x) = |-1 - (x^2 + x)|$$

to minimize $J(x)$, the gradient is,

$$J^{'}(x) = -2x - 1$$

which will bring x to $-\frac{1}{2}$ as $J'(x) = 0$, but $x = -\frac{1}{2}$ is not a real root of $x^2 + x + 1 = 0$. ■

ECE 498 ICC Spring 2020 Jin Yucheng (yucheng9)

Homework 1 Problem 3

Solution:

(1)

$$S_1 = w_1 x_1 + w_2 x_2 = -0.23 \quad h_1 = \frac{1}{1+e^{-S_1}} = 0.44275 \quad \|\hat{y}-y\| = 0.18851$$

$$S_2 = w_3 x_3 + w_4 x_4 = -1.92 \quad h_2 = \frac{1}{1+e^{-S_2}} = 0.12786$$

$$S_3 = w_5 h_1 + w_6 h_2 = 0.23511 \quad \hat{y} = \frac{1}{1+e^{-S_3}} = 0.55851$$

① $$\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial S_3} \frac{\partial S_3}{\partial w_6} = 2\|y-\hat{y}\| \frac{e^{-S_3}}{(1+e^{-S_3})^2} h_2 = 0.01189$$

② $$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial S_3} \frac{\partial S_3}{\partial w_5} = 2\|y-\hat{y}\| \frac{e^{-S_3}}{(1+e^{-S_3})^2} h_1 = 0.04116$$

③ $$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial S_3} \frac{\partial S_3}{\partial h_2} \overset{\partial h_2/\partial S_2 \ \partial S_2/\partial w_4}{\cancel{\frac{\partial h_2}{\partial w_4}}} = 2\|y-\hat{y}\| \frac{e^{-S_3}}{(1+e^{-S_3})^2} w_6 \frac{e^{-S_2}}{(1+e^{-S_2})^2} x_4 = -0.00498$$

④ $$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial S_3} \frac{\partial S_3}{\partial h_2} \overset{\partial h_2/\partial S_2 \ \partial S_2/\partial w_3}{\cancel{\frac{\partial h_2}{\partial w_3}}} = 2\|y-\hat{y}\| \frac{e^{-S_3}}{(1+e^{-S_3})^2} w_6 \frac{e^{-S_2}}{(1+e^{-S_2})^2} x_3 = 0.00746$$

⑤ $$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial S_3} \frac{\partial S_3}{\partial h_1} \frac{\partial h_1}{\partial S_1} \frac{\partial S_1}{\partial w_2} = 2\|y-\hat{y}\| \frac{e^{-S_3}}{(1+e^{-S_3})^2} w_5 \frac{e^{-S_1}}{(1+e^{-S_1})^2} x_2 = 0.00963$$

⑥ $$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial S_3} \frac{\partial S_3}{\partial h_1} \frac{\partial h_1}{\partial S_1} \frac{\partial S_1}{\partial w_1} = 2\|y-\hat{y}\| \frac{e^{-S_3}}{(1+e^{-S_3})^2} w_5 \frac{e^{-S_1}}{(1+e^{-S_1})^2} x_1 = 0.00206$$

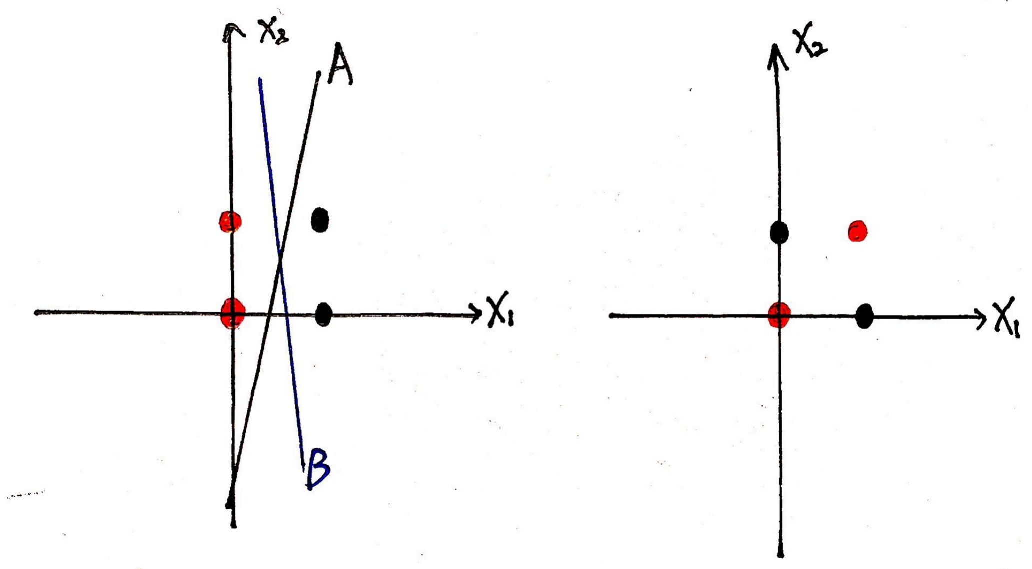
(2) Vanishing and exploding gradients occur when training some deep neural networks,

- **Vanishing Gradients**: when computing gradients backward through the hidden layers, the gradients tend to be smaller in the earlier layers, since the gradients are products of terms of weights and other expressions, they may decay exponentially if the terms are less than 1.
- **Exploding Gradients**: is the opposite of vanishing gradients, that the gradients tend to be larger in the earlier layers, since the terms of the gradients are larger than 1.

To solve these problems, we should initialize weights and activation functions carefully such that the products will not increase or decay exponentially. ■

ECE 498 ICC Spring 2020 Jin Yucheng (yucheng9)
Homework 1 Problem 4

Solution: (1) Linear classifiers can learn patterns in Problem A but cannot learn patterns in Problem B. The red dots in the following figure label "0", while the black dots label "1".



For Problem A, we can see that both line A and line B can separate red and black dots (also, for example, $x_1 = 0.5$), but for Problem B, there is no such line can separate red and black dots.

A formal proof by contradiction,

Suppose there exists a line $x_2 = ax_1 + b$ that separates two categories in Problem B, then,

For label "0", $-b$ $(0,0)$ and $1-a-b$ $(1,1)$ must both be non-negative or non-positive,

respectively, for label "1", $1-b$ $(0,1)$ and $-1-b$ $(1,0)$ must both be negasitive or positive.

If $-b \geq 0$ and $1-a-b \geq 0$, then $1-b > 0$, which contradicts the fact that $1-b < 0$.

If $-b \leq 0$ and $1-a-b \leq 0$, then $-1-b < 0$, which contradicts the fact that $-1-b > 0$.

In conclusion, our assumption contradicts the given conditions, and dots in Problem B are not linearly separable.

(2) For the first neural network,

- The number of neurons: $16 + 32 \times 8 + 16 = 288$.
- The number of weights: $16 \times 32 + 32 \times 32 \times 7 + 32 \times 16 = 8192$.
- The number of biases: $32 \times 8 + 16 = 272$

So the memory requirement is $288 + 8192 + 272 = 8752$.

For the second neural network,

- The number of neurons: $16 + 128 \times 2 + 16 = 288$.
- The number of weights: $16 \times 128 + 128 \times 128 + 128 \times 16 = 20480$.
- The number of biases: $128 \times 2 + 16 = 272$

So the memory requirement is $288 + 20480 + 272 = 21040$.

I prefer the first one, because it requires less memory, and it is deeper, in practice, deep neural networks are usually better than shallow neural networks.

(3)

- The accuracy of the Multilayer Perceptron will decrease a lot, since it uses linear activation functions and is sensitive to the change in input.
- The accuracy of the CNN will decrease a little compared to Multilayer Perceptron, because CNN is capable of extracting features thus the change of their positions will not influence accuracy much.

(4) Here is a comparison of ReLU, Tanh, and Unit Step Function,

Activation Function	Equation	Range	Advantages	Disadvantages
ReLU	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	(0, +∞)	1. It is able to converge quickly, and is easy to compute; 2. It is not linear in nature and is a good estimator in practice; 3. It speeds up the training process.	1. Learning is not performed in the zero-valued region (*Dead ReLU Problem*), or in other words, some neurons will never be activated.
Tahn	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1, 1)	1. Its derivative can take more values compared to sigmoid, and it has a wider range for faster learning and grading.	1. It is not easy to compute; 2. It has the problem of vanishing gradients.
Unit Step Function	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$	{0, 1}	1. It takes a binary value and is widely used as a binary classifier, often preferred in the output layers; 2. Easy to compute.	1. It is not recommended to use in hidden layers, since its derivative is always 0 (and at $x = 0$, its derivative is even not defined).

(5) Yes. The most important disadvantage is that because classification problem deals with discrete labels, but regression problem deals with continuous labels, the conversion between continuous and discrete variables is difficult to accomplish. ■

ECE 498 ICC Spring 2020 Jin Yucheng (yucheng9)

Homework 1 Problem 5

Solution:

(1) For matrices A, B, and $C \in R^{2^n \times 2^n}$, such that $C = AB$, in the naive algorithm,

$$C_{ij} = \Sigma_{k=1}^{k=2^n} A_{ik} B_{kj}$$

which is equivalent to,

$$C_{ij} = A_i B_j$$

where A_i is a row vector, representing the i^{th} row of A,

$$A_i = \begin{bmatrix} A_{i1} & A_{i2} & \dots & A_{i2^n} \end{bmatrix}$$

B_j is a column vector, representing the j^{th} column of B,

$$B_j = \begin{bmatrix} B_{1j} \\ B_{2j} \\ \vdots \\ B_{2^n j} \end{bmatrix}$$

(2) The naive algorithm: 112 (64 multiplications + 48 additions), Strassen's algorithm: 214 (49 multiplications + 165 additions) if optimized, 247 (49 multiplications + 198 additions) if not optimized.

If A, B, and $C \in R^{4\times4}$, there are totally 16 elements in A, B, and C. For the naive algorithm, for one element C_{ij}, since $C_{ij} = \Sigma_{k=1}^{k=2^n} A_{ik} B_{kj}$, it takes 4 multiplications ($2^n, n = 2$) and 3 addition (Σ). Therefore, the naive algorithm takes $(4+3)\times4\times4 = 112$ multiplications and additions.

The original algorithm discovered by V. Strassen in 1969 requires 7 multiplications and 18 additions and subtractions to compute $C \in R^{2\times2}$, but later, the number of additions and subtractions was reduced to 15 [1], and the number of multiplications and additions has the following recurrence,

$$S(1) = 1,\ S(n) = 7S(n/2) + 15(n/2)^2, \text{ for } n > 1 \text{ and } n \text{ is a power of } 2$$

for $n = 4$, $S(4) = 7S(2) + 15(4/2)^2 = 7 \times (7 + 15) + 15 \times 4 = 214$.

Among 214 operations, 49 are multiplications, since if we can multiply 2 by 2 matrices using only 7 multiplications, then as for multiplying 4 by 4 matrices, we can use 7 multiplications of 2 by 2 matrices, and for each 2 by 2 matrices multiplication, we need 7 multiplications of numbers, so totally 49 multiplications. Further, we can find that for matrices of dimension $2^n \times 2^n$, Strassen's algorithm takes 7^n multiplications.

[1]UTSA, CS 3343/3341: Analysis of Algorithms. "Strassen's Method for Multiplying Matices". Web. <http://www.cs-.utsa.edu/ wagner/CS3343/strassen/strassen.html>. Accessed March 24, 2020.

For the original version of Strassen's algorithm, it takes $6 \times (7^n - 4^n)$ additions, because at level 2^n there will be 18 additions each of 2^{2n-2} matrix elements; at the next level, there will $\frac{7}{4}$ as many sums: the 7 comes from there being 7 times as many matrix products that we have to perform, the 4 comes from the fact that they each have half the size and hence a quarter as many elements. So the number of additions is $18 \times 2^{2n-2} \times (1 + \frac{7}{4} + (\frac{7}{4})^2 + \ldots + (\frac{7}{4})^{n-1})$, which works out to be $6 \times (7^n - 4^n)$[2].

(3) For the naive algorithm, it takes $2^n \times 2^n \times 2^n + 2^n \times 2^n$ multiplications and additions, the asymptotic complexity is $\Theta(N^3)$, for $N = 2^n$. For Strassen's algorithm, according to the recurrence, for $N = 2^n$, its time complexity is $O([7 + o(1)]^n) = O(N^{log_2 7 + o(1)}) \approx O(N^{2.8074})$ [3].

So in theory, because Strassen's algorithm reduces the number of multiplications, it is faster for calculations of large matrices than the naive algorithm. However, the reduction in the number of arithmetic operations also costs numerical stability, and requires significantly more memory compared to the naive algorithm.

(4) Strassen's algorithm computes M_1 to M_7 as,

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}$$

and computes C_{ij} as,

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

since,

$$A = \begin{bmatrix} A_{11} \\ \\ A_{12} \\ \\ A_{21} \\ \\ A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} \\ \\ B_{12} \\ \\ B_{21} \\ \\ B_{22} \end{bmatrix}$$

[2] MIT Math. "25. Strassen's Fast Multiplication of Matrices Algorithm, and Spreadsheet Matrix Multiplications". Web. <http://www-math.mit.edu/ djk/18.310/18.310F04/Matrix_%20Multiplication.html>. Accessed March 24, 2020.

[3] "Strassen's Algorithm". *wikipedia.org*. Web. <https://en.wikipedia.org/wiki/Strassen_algorithm>. Accessed March 24, 2020.

and $C = F \otimes ((G^T \otimes A) * (H^T \otimes B))$, clearly,

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

because $G^T \otimes A$ must correspond to the sum of A_{ij} which appears on the left side of M_k,

$$H = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

because $H^T \otimes B$ must correspond to the sum of B_{ij} which appears on the right side of M_k.
$((G^T \otimes A) * (H^T \otimes B))$ gives us a vector with 7 elements, corresponding to M_1 to M_7, then,

$$F = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

because every row corresponds to the coefficients of M_1 to M_7. For example, the first row calculates C_{11}, and $C_{11} = M_1 + M_4 - M_5 + M_7$, so the coefficient of M_1 is 1, of M_2 is 0, of M_3 is 0, of M_4 is 1, of M_5 is -1, of M_6 is 0, and of M_7 is 1.

Observations:

- Each column of G encodes the coefficients of A_{ij} which appear on the left side of M_k.
- Each column of H encodes the coefficients of B_{ij} which appear on the right side of M_k.
- Each row of F encodes the coefficients of M_k that express C_{ij}.

■

ECE 498 ICC Spring 2020 Jin Yucheng (yucheng9)

Homework 1 Problem 6

Solution:

(1) The curse of dimensionality refers to "various phenomena that arise when analyzing and organizing data in high-dimensional spaces" [4]. For example, when the dimensionality increases, the availabel data may become sparse as the volume of the space explodes, and with a fixed number of training samples, the performance of the ML algorithm becomes worse if we do not apply feature selection and dimensionality reduction techniques.

(2) Principal Component Analysis (PCA) is a statistical procedure that "uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components" [5]. PCA transforms the original data into a new subspace such that the projected features are orthonormal, thus enabling us to reduce features that are highly correlated, because we can choose the first n principal components such that most information is maintained. In a statistical sense, variations along the axes of these n principal components are higher than variations of other projected features, and are decreasing from the 1^{st} principal component to the n^{th} principal component.

(3) The input data is expressed as a $n \times d$ matrix, each feature vector $\vec{x_i}$ is of dimension d,

$$X = \begin{bmatrix} -\vec{x_1}- \\ \\ -\vec{x_2}- \\ \vdots \\ -\vec{x_n}- \end{bmatrix}$$

the objective is to find a k-dimensional subspace that captures the most information, such that every projected feature in the k-dimensional subspace is orthonormal (linearly uncorrelated) to each other,

$$V = \begin{bmatrix} -\vec{v_1}- & -\vec{v_2}- & \dots & -\vec{v_k}- \end{bmatrix}$$

notice that in V, every column vector is an orthonormal basis in the new subspace.

The relationship between X and V is shown by the eigenvector decomposition,

$$X^T X = USU^T$$

since the first k columns of U are the first k principal components, which just consist V.

The Frobenius norm of a matrix X is a "measure" of its length, defined as,

$$||X||_F = \sqrt{\Sigma_{ij} X_{ij}^2} = \sqrt{tr[X^T X]}$$

[4] Deming Chen, Jinjun Xiong, V. Kindratenko. "Lecture 4: Data Analytics for IoT Basics". p. 34.

[5] "Principal Component Analysis". *wikipedia.org*. Web. <https://en.wikipedia.org/wiki/Principal_component_analysis>. Accessed March 24, 2020.

it is equal to the sum of the singular values,

$$||X||_F = \Sigma_{j=1}^{d} s_j$$
$$s_j = \Sigma_{i=1}^{N} (\vec{u}_j \cdot \vec{x}_i)^2$$

Therefore, to find the most important k principal components, we must maximizes the Frobenious norm of the data projected onto V,

$$\hat{V}_{pca} = argmax_V ||XV||_F^2$$

equivalently,

$$\hat{V}_{pca} = argmin_V ||X - XVV^T||_F^2$$

such that $V^T V = I$.

This objective function says that the principal components define an orthonormal basis such that the distance between the original data and the data projected onto that subspace is minimal[6].

Notice that it is uncommon to do PCA on uncentered data, the standard procedure is to first zero-center X, which means replacing each $\vec{x}_i$ with $\vec{x}_i - \overline{x}$, $\overline{x} = \frac{1}{N}\Sigma\vec{x}_i$.

(4) Here is the proof that $\hat{V}_{pca} = argmax_V ||XV||_F^2$ and $\hat{V}_{pca} = argmin_V ||X - XVV^T||_F^2$ are equivalent,

$$\begin{aligned}
||X - XVV^T||_F^2 &= tr((X - XVV^T)^T(X - XVV^T)) && \text{because } ||A||_F^2 = tr(A^T A) \\
&= tr((X - XVV^T)(X - XVV^T)^T) && \text{because } tr(A^T A) = tr(AA^T) \\
&= tr((X - XVV^T)(X^T - VV^T X^T)) \\
&= tr(XX^T - XVV^T X^T - XVV^T X^T + XVV^T VV^T X^T) \\
&= tr(XX^T - XVV^T X^T - XVV^T X^T + XVV^T X^T) && \text{because } V^T V = I \\
&= tr(XX^T - XVV^T X^T) \\
&= tr(XX^T) - tr(XVV^T X^T) && \text{because } tr(A+B) = tr(A) + tr(B) \\
&= tr(XX^T) - ||XV||_F^2
\end{aligned}$$

Therefore, to minimize $||X - XVV^T||_F^2$ is equivalent to maximize $||XV||_F^2$.

(5) The eigendecomposition of $X^T X$ exists and the eigenvectors can form an orthonormal basis of the d-dimensional space because $X^T X$ forms a symmetric matrix $\in R^{d\times d}$, furthermore, we already have the assumption that $X^T X = QDQ^T$.

Since Q is orthonormal, $Q^T Q = I$ and $QQ^T = I$, $||XV||_F^2$ can be transformed as,

$$\begin{aligned}
||XV||_F^2 &= tr(V^T X^T XV) \\
&= tr((QZ)^T QDQ^T QZ) \\
&= tr(Z^T Q^T QDQ^T QZ) \\
&= tr(Z^T DZ)
\end{aligned}$$

■

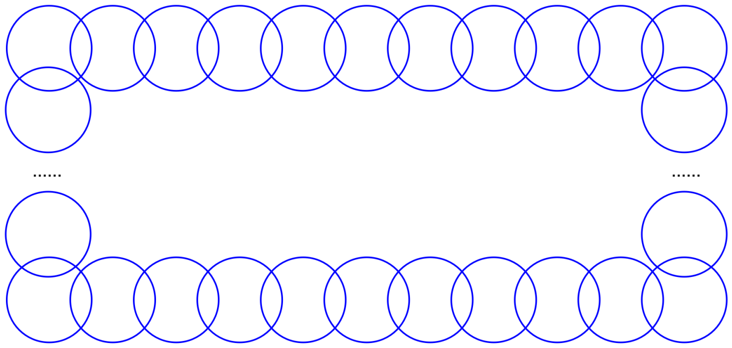
[6] Jonathan Pillow. "Statistical Modeling and Analysis of Neural Data (NEU 560)". Princeton University. Web. <http://pillowlab.princeton.edu/teaching/statneuro2018/slides/notes05_PCA2.pdf>. Accessed March 25, 2020.

ECE 498 ICC Spring 2020 — Jin Yucheng (yucheng9)

Homework 1 Problem 7

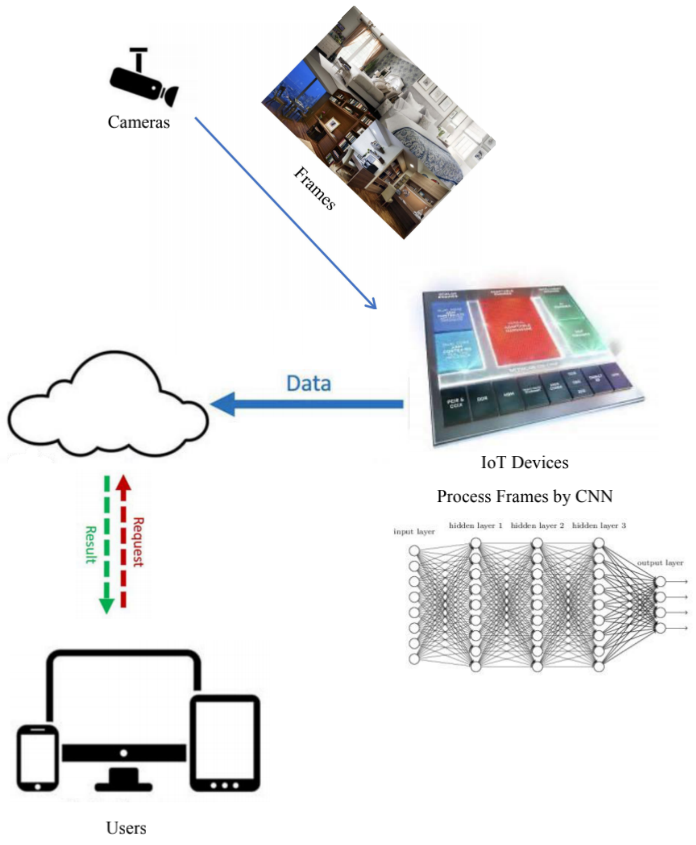
Solution:

(1) To build a security system that is free of dead corner, we should make sure the viewing region of cameras covers all edges, therefore, for a single floor, the layout of cameras will be like,



There are 4 cameras at 4 corners, covering 100 ft of the 3000 ft edge and 100 ft of the 2000 ft edge respectively, more cameras are on the edges such that there is no dead corner.

The above configuration applies to all 3 floors, and for each floor there is one set of IoT devices to detect intrusions, the workflow of my security system is as follows,



Frames collected by cameras are immediately passed to IoT devices, then IoT devices will use CNN to extract information from the input frames and report any result to cloud. Apps in users' devices (such as mobile phones, tablets, & laptops) will request data from cloud regularly, and if there is intrusion, users will receive emergency alert to inform the possible intrusion.

(2) For one floor, there are $4+2\times(2000-100\times2)/200+2\times(3000-100\times2)/200=50$ cameras, so for three floors, 150 cameras in total.

(3) I will choose one set of processing engines per floor, for one floor there are 50 cameras, and each generates 10 frames per second, so there are totally 500 frames per second (about 1.5 MB/s, for three floors about 4.5 MB/s, while the bandwidth is 10MB/s). There are several possible plans, for example, two Xilinx Cloud FPGA ($\$4,000$, 700FPS), one Xilinx Cloud FPGA and six Nvidia Jetson TX1 ($\$4,100$, 560FPS), and other plans. I prefer two Xilinx Cloud FPGA, each coupled with 25 cameras.

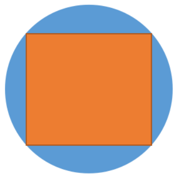
Therefore, for three floors, there will be six Xilinx Cloud FPGA ($\$12,000$, $2,100$FPS), each coupled with 25 cameras.

(4) 150 cameras cost $\$1,500$, six Xilinx Cloud FPGA cost $\$12,000$, the total cost is $\$13,500$.

(5) First, we must look at 2 seconds ($2,000$ milliseconds) of footage; second, as cameras capture frames, it takes about 300 to 600 milliseconds for Xilinx Cloud FPGA to access frames. Since one Xilinx Cloud FPGA is coupled with 25 cameras and is able to process 350 frames per second, it takes about 80 to 200 milliseconds for Xilinx Cloud FPGA to process frames. Therefore, the average worst-case latency for detecting an intrusion is about $2,400$ to $2,800$ milliseconds.

Above is the discussion under the assumption that only covering edges is enough. If we want to fill in the whole area, then we need,

(2) For one floor, there are $3000\times2000/(200/\sqrt{2})^2=300$ cameras, so for three floors, 900 cameras in total. This may not be the optimal solution, but according to the figure below, if we cover the entire area with the orange inscribed squares within the viewing region, then we can ensure that there is no dead corner left.



(3) For one floor, there are 300 cameras, and each generates 10 frames per second, so there are totally 3000 frames per second, we need 8 Xilinx Cloud FPGA ($\$16,000$, $2,800$FPS) and 10 Xilinx PYNQ Z1 ($\$2,000$, 200FPS). Each Xilinx Cloud FPGA is coupled with 35 cameras and each Xilinx PYNQ Z1 is coupled with 2 cameras.

Therefore, for three floors, there will be 24 Xilinx Cloud FPGA and 30 Xilinx PYNQ Z1, each Xilinx Cloud FPGA is coupled with 35 cameras and each Xilinx PYNQ Z1 is coupled with 2 cameras.

(4) 900 cameras cost $\$9,000$, 24 Xilinx Cloud FPGA cost $\$48,000$, 30 Xilinx PYNQ Z1 cost $\$6,000$, the total cost is $\$63,000$.

(5) Because each Xilinx Cloud FPGA and each Xilinx PYNQ Z1 are computing in full capacity, now it takes about 450 to 900 milliseconds for them to access frames, and about 100 to 240 milliseconds for them to process frames. Therefore, the average worst-case latency for detecting an intrusion is about $2,600$ to $3,200$ milliseconds.

■