CPSC 340 Assignment 6 (due Friday November 29 at 11:55pm)

Zijia Zhang 42252965

1 Robust PCA

The function *example_RPCA* loads a dataset X where each row contains the pixels from a single frame of a video of a highway. The demo applies PCA to this dataset and then uses this to reconstruct the original image. It then shows the following 3 images for each frame of the first 50 frames (pausing for a tenth of a second on each frame):

1. The original frame.
2. The reconstruction based on PCA.
3. A binary image showing locations where the reconstruction error is non-trivial.

Recently, latent-factor models have been proposed as a strategy for "background subtraction": trying to separate objects from their background. In this case, the background is the highway and the objects are the cars on the highway. In this demo, we see that PCA does an ok job of identifying the cars on the highway in that it does tend to identify the locations of cars. However, the results aren't great as it identifies quite a few irrelevant parts of the image as objects.

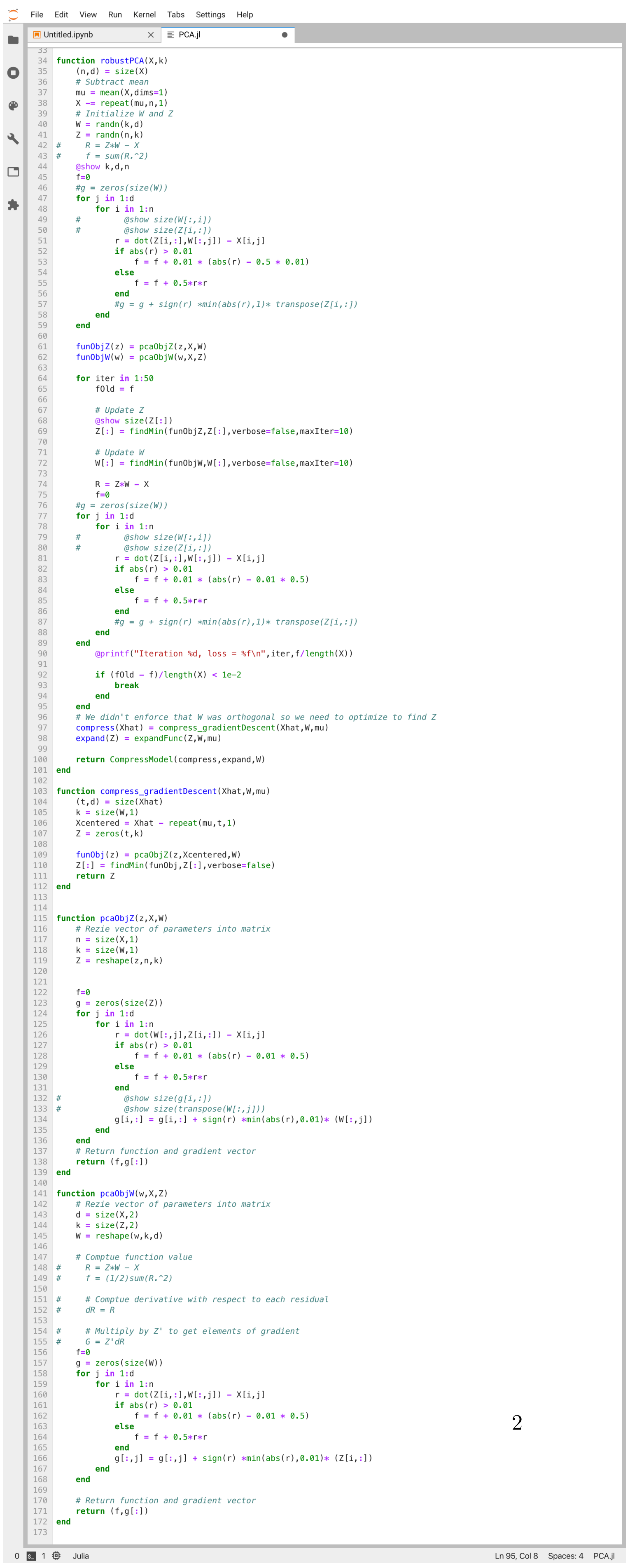
Robust PCA is a variation on PCA where we replace the L2-norm with the L1-norm,

$$f(Z,W) = \sum_{i=1}^{n}\sum_{j=1}^{d} |w_j^T z_i - x_{ij}|,$$

and it has recently been proposed as a more effective model for background subtraction. Write a new function, *robustPCA*, that uses the Huber loss to approximate the absolute value to implement robust PCA. Hand in your code.

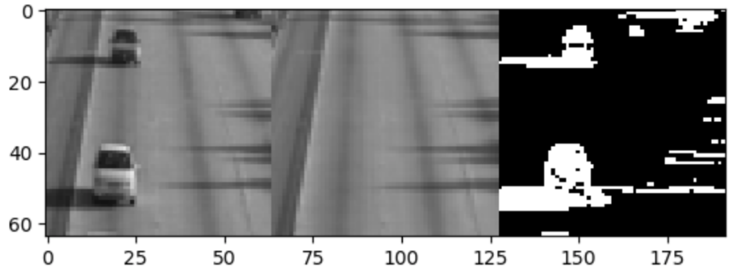
Hint: most of the work has been done for you in the function *PCA_gradient*. This function implements an alternating minimization approach to minimizing the PCA objective (without enforcing orthogonality). This gradient-based approach to PCA can be modified to use the Huber loss. A reasonable value of the hyper-parameter ϵ in the Huber loss might be 0.01 for this problem. You may want to use a smaller version of the dataset when debugging your objective/gradient code.

Answer:



```julia

function robustPCA(X,k)
    (n,d) = size(X)
    # Subtract mean
    mu = mean(X,dims=1)
    X -= repeat(mu,n,1)
    # Initialize W and Z
    W = randn(k,d)
    Z = randn(n,k)
#     R = Z*W - X
#     f = sum(R.^2)
    @show k,d,n
    f=0
    #g = zeros(size(W))
    for j in 1:d
        for i in 1:n
    #         @show size(W[:,i])
    #         @show size(Z[i,:])
            r = dot(Z[i,:],W[:,j]) - X[i,j]
            if abs(r) > 0.01
                f = f + 0.01 * (abs(r) - 0.5 * 0.01)
            else
                f = f + 0.5*r*r
            end
            #g = g + sign(r) *min(abs(r),1)* transpose(Z[i,:])
        end
    end

    funObjZ(z) = pcaObjZ(z,X,W)
    funObjW(w) = pcaObjW(w,X,Z)

    for iter in 1:50
        fOld = f

        # Update Z
        @show size(Z[:])
        Z[:] = findMin(funObjZ,Z[:],verbose=false,maxIter=10)

        # Update W
        W[:] = findMin(funObjW,W[:],verbose=false,maxIter=10)

        R = Z*W - X
        f=0
    #g = zeros(size(W))
    for j in 1:d
        for i in 1:n
    #         @show size(W[:,i])
    #         @show size(Z[i,:])
            r = dot(Z[i,:],W[:,j]) - X[i,j]
            if abs(r) > 0.01
                f = f + 0.01 * (abs(r) - 0.01 * 0.5)
            else
                f = f + 0.5*r*r
            end
            #g = g + sign(r) *min(abs(r),1)* transpose(Z[i,:])
        end
    end
        @printf("Iteration %d, loss = %f\n",iter,f/length(X))

        if (fOld - f)/length(X) < 1e-2
            break
        end
    end
    # We didn't enforce that W was orthogonal so we need to optimize to find Z
    compress(Xhat) = compress_gradientDescent(Xhat,W,mu)
    expand(Z) = expandFunc(Z,W,mu)

    return CompressModel(compress,expand,W)
end

function compress_gradientDescent(Xhat,W,mu)
    (t,d) = size(Xhat)
    k = size(W,1)
    Xcentered = Xhat - repeat(mu,t,1)
    Z = zeros(t,k)

    funObj(z) = pcaObjZ(z,Xcentered,W)
    Z[:] = findMin(funObj,Z[:],verbose=false)
    return Z
end


function pcaObjZ(z,X,W)
    # Rezie vector of parameters into matrix
    n = size(X,1)
    k = size(W,1)
    Z = reshape(z,n,k)


    f=0
    g = zeros(size(Z))
    for j in 1:d
        for i in 1:n
            r = dot(W[:,j],Z[i,:]) - X[i,j]
            if abs(r) > 0.01
                f = f + 0.01 * (abs(r) - 0.01 * 0.5)
            else
                f = f + 0.5*r*r
            end
#             @show size(g[i,:])
#             @show size(transpose(W[:,j]))
            g[i,:] = g[i,:] + sign(r) *min(abs(r),0.01)* (W[:,j])
        end
    end
    # Return function and gradient vector
    return (f,g[:])
end

function pcaObjW(w,X,Z)
    # Rezie vector of parameters into matrix
    d = size(X,2)
    k = size(Z,2)
    W = reshape(w,k,d)

    # Comptue function value
#     R = Z*W - X
#     f = (1/2)sum(R.^2)

#     # Comptue derivative with respect to each residual
#     dR = R

#     # Multiply by Z' to get elements of gradient
#     G = Z'dR
    f=0
    g = zeros(size(W))
    for j in 1:d
        for i in 1:n
            r = dot(Z[i,:],W[:,j]) - X[i,j]
            if abs(r) > 0.01
                f = f + 0.01 * (abs(r) - 0.01 * 0.5)
            else
                f = f + 0.5*r*r
            end
            g[:,j] = g[:,j] + sign(r) *min(abs(r),0.01)* (Z[i,:])
        end
    end

    # Return function and gradient vector
    return (f,g[:])
end

```

```
(k, d, n) = (5, 4096, 440)
size(Z[:]) = (2200,)
Iteration 1, loss = 0.135620
size(Z[:]) = (2200,)
Iteration 2, loss = 0.122954
size(Z[:]) = (2200,)
Iteration 3, loss = 0.121714
```



2 Multi-Dimensional Scaling

The function *example_MDS* loads the animals dataset and then applies gradient dsecent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$f(Z) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\|z_i - z_j\| - \|x_i - x_j\|)^2. \tag{1}$$

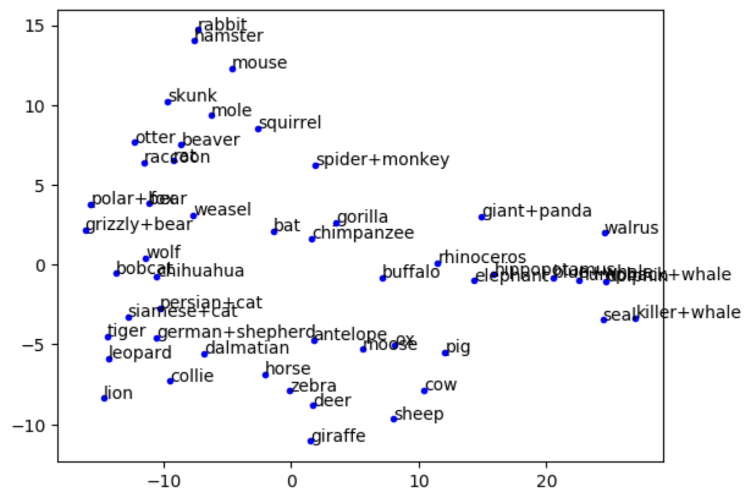
The result of applying MDS is shown below. Although this visualization isn't perfect (with "gorilla" being placed close to the dogs and "otter" being placed close to two types of bears), this visualization does organize the animals in a mostly-logical way.

2.1 ISOMAP

Euclidean distances between very different animals are unlikely to be particularly meaningful. However, since related animals tend to share similar traits we might expect the animals to live on a low-dimensional manifold. This suggests that ISOMAP may give a better visualization. Make a new function *ISOMAP* that computes the approximate geodesic distance (shortest path through a graph where the edges are only between nodes that are k-nearest neighbour) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Hand in your code and the plot of the result when using the 3-nearest neighbours.

Answer:

```
function ISOMAP(X)
    n,d = size(X);
    G = zeros(n,n);
    D = distancesSquared(X,X);
    D = sqrt.(abs.(D));
    for i in 1:n
        G[i,:] = fourMin(D[i,:])
        G[i,i] = Inf
    end
    for i in 1:n
        for j in 1:n
            if G[i,j] != Inf
                G[j,i] = G[i,j]
            end
        end
    end
    #@show G
    r = zeros(n,n)
    for i in 1:n
        for j in 1:i
            r[i,j] = dijkstra(G,i,j)
            #@show r[i,j]
            r[j,i] = r[i,j]
        end
    end
    #@show r
    model = PCA(X,2)
    Z = model.compress(X)

    funObj(z) = stress(z,r)

    Z[:] = findMin(funObj,Z[:])

    return Z

end

function fourMin(V)
    n = size(V,1)
    min1Index = 1
    min2Index = 1
    min3Index = 1
    min4Index = 1
    for i =1:n
        if V[i] < V[min1Index]
            min4Index = min3Index
            min3Index = min2Index
            min2Index = min1Index
            min1Index = i
        elseif V[i] < V[min2Index]
            min4Index = min3Index
            min3Index = min2Index
            min2Index = i
        elseif V[i] < V[min3Index]
            min4Index = min3Index
            max3Index = i
        elseif V[i] < V[min4Index]
            min4Index = i
        end
    end
    r = zeros(size(V))
    for i = 1:n
        r[i] = Inf
    end
    r[min1Index] = V[min1Index]
    r[min2Index] = V[min2Index]
    r[min3Index] = V[min3Index]
    r[min4Index] = V[min4Index]
    return r
end
```



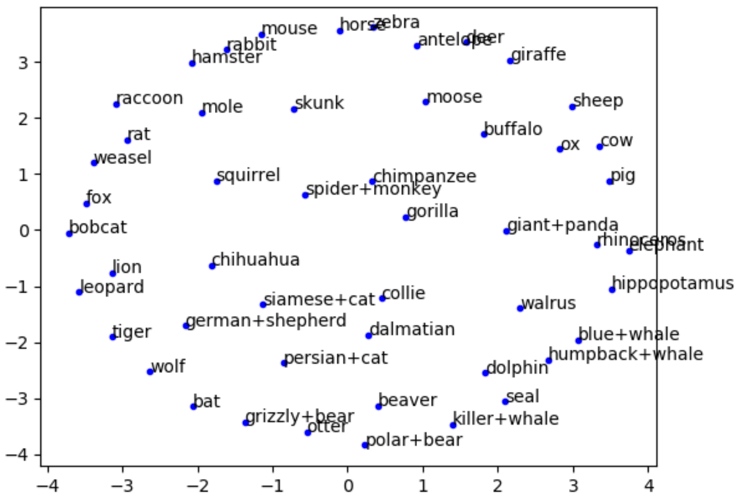
Hint: the function *dijskstra* (in *misc.jl*) can be used to compute the shortest (weighted) distance between two points in a weighted graph. This function requires an n by n matrix giving the weights on each edge (use ∞ as the weight for absent edges). Note that ISOMAP uses an undirected graph, while the k-nearest neighbour graph might be asymmetric. You can use the usual heuristics to turn this into an undirected graph of including an edge i to j if i is a KNN of j or if j is a KNN of i. (Also, be careful not to include the point itself in the KNN list).

2.2 ISOMAP with Disconnected Graph

An issue with measuring distances on graphs is that the graph may not be connected. For example, if you run your ISOMAP code with 2-nearest neighbours then some of the distances are infinite. One heuristic to address this is to set these infinite distances to the maximum distance in the graph (i.e., the maximum geodesic distance between any two points that are connected), which will encourage non-connected points to be far apart. Modify your ISOMAP function to implement this heuristic. Hand in your code and the plot of the result when using the 2-nearest neighbours.

Answer:

```
function ISOMAP(X)
    n,d = size(X);
    G = zeros(n,n);
    D = distancesSquared(X,X);
    D = sqrt.(abs.(D));
    for i in 1:n
        G[i,:] = fourMin(D[i,:])
        G[i,i] = Inf
    end
    largest = 0;
    for i in 1:n
        for j in 1:n
            if G[i,j]!= Inf && G[i,j] > largest
                largest = G[i,j]
            end
        end
    end

    for i in 1:n
        for j in 1:n
            if G[i,j] != Inf
                G[j,i] = G[i,j]
            else
                G[i,j] = largest
            end
        end
    end
    #@show G
    r = zeros(n,n)
    for i in 1:n
        for j in 1:i
            r[i,j] = dijkstra(G,i,j)
            #@show r[i,j]
            r[j,i] = r[i,j]
        end
    end
    #@show r
    model = PCA(X,2)
    Z = model.compress(X)

    funObj(z) = stress(z,r)

    Z[:] = findMin(funObj,Z[:])

    return Z

end

function fourMin(V)
    n = size(V,1)
    min1Index = 1
    min2Index = 1
    min3Index = 1
    min4Index = 1
    for i =1:n
        if V[i] < V[min1Index]
            min4Index = min3Index
            min3Index = min2Index
            min2Index = min1Index
            min1Index = i
        elseif V[i] < V[min2Index]
            min4Index = min3Index
            min3Index = min2Index
            min2Index = i
        elseif V[i] < V[min3Index]
            min4Index = min3Index
            max3Index = i
        elseif V[i] < V[min4Index]
            min4Index = i
        end
    end
    r = zeros(size(V))
    for i = 1:n
        r[i] = Inf
    end
    r[min1Index] = V[min1Index]
    r[min2Index] = V[min2Index]
    r[min3Index] = V[min3Index]
    #r[min4Index] = V[min4Index]
    return r
end
```



3 Neural Networks

3.1 Neural Networks by Hand

Suppose that we train a neural network with sigmoid activations and one hidden layer and obtain the following parameters (assume that we don't use any bias variables):

$$W = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix}, v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

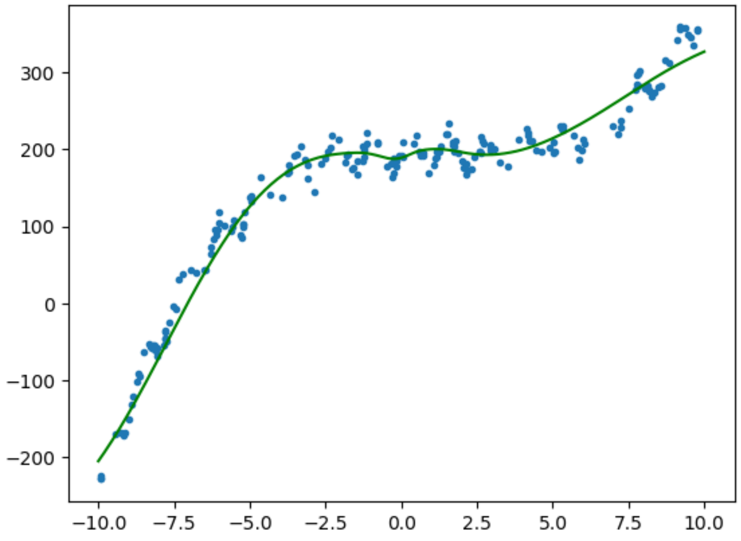
Assuming that we are doing regression, for a training example with features $x_i^T = \begin{bmatrix} -3 & -2 & 2 \end{bmatrix}$ what are the values in this network of z_i, $h(z_i)$, and $\hat{y}_i$?

Answer: $z_i = Wx_i = [0, 1]$
$h(z_i) = [0.5, e/(e+1)]$
$y = 1.5 + e/(e+1)$

3.2 Neural Network Tuning - Regression

The file `example_nnet.jl` runs a stochastic gradient method to train a neural network on the *basisData* dataset from a previous assignment. However, in its current form it doesn't fit the data very well. Modify the training procedure to improve the performance of the neural network. Hand in your plot after changing the code to have better performance, and list the changes you made.

Answer: Change the number of layers to 10000. Change to 15000 steps and stepsize to 1e-5.



Hint: there are many possible strategies you could take to improve performance. Below are some suggestions, but note that the some will be more effective than others:

- Changing the network structure (*nHidden* is a vector giving the number of hidden units in each layer).
- Changing the training procedure (you can change the stochastic gradient step-size, use mini-batches, run it for more iterations, add momentum, switch to *findMin*, and so on).
- Transform the data by standardizing the feautres, standardizing the targets, and so on.
- Add regularization (L2-regularization, L1-regularization, dropout, and so on).
- Add bias variables within the hidden layers.
- Change the loss function or the non-linearities (right now it uses squared error and tanh to introduce non-linearity).
- Use mini-batches of data, possibly with batch normalization.

3.3 Neural Network Tuning - Classification

The file `example_usps.jl` runs a stochastic gradient method to train a neural network on a set of images of digits. Modify the training procedure to improve the performance of the neural network. List the changes you made and the best test performance that you were able to obtain.

Answer: Tried 2 ways to improve the performance:
1. Change the number of hidden unit to 1000. Achived a testing error of 0.131 with maxIter = 48000.
2. Perform PCA first with k = 100 and change the number of unit in the layer to 200. Changed step size to 1e-4 when reaches 30000 iterations and 1e-5 when reaches 50000 iterations. At max iterations of 150000, achived test error of 0.128.

4 Very-Short Answer Questions

1. Is the NMF loss function convex? What is an optimization method you could use to try to minimize it?

   Answer: No, we can use Projected-Gradient to solve this kind of problem.

2. Consider fitting a linear latent-factor model, using L1-regularization of the w_c values and L2-regularization of the z_i values,

$$f(Z,W) = \frac{1}{2}\|ZW - X\|_F^2 + \lambda_W \sum_{c=1}^{k} [\|w_c\|_1] + \frac{\lambda_Z}{2} \sum_{i=1}^{n} [\|z_i\|^2],$$

   (a) What is the effect of λ_Z on the two parts of the fundamental trade-off in machine learning?

   Answer: When λ_Z is big, the training error will increase and approximation error will decrease.

   (b) What is the effect of k on the two parts?

   Answer: When k increases, the training error will decrease, and approximation error will increase.

   (c) Would either of answers to the *previous two questions* change if $\lambda_W = 0$?

   Answer: (1) The λ_Z will not affect. (2) k will have the same effect.

3. Which is better for recommending movies to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.

   Answer: content-based filtering, because colaborative filtering need to depend on finding similarities in teh ratings, but new users don't have any.

4. Is ISOMAP mainly used for supervised or unsupervised learning? Is it parametric or non-parametric?

   Answer: Unsupervised, is non-parametric model.

5. What is the difference between Euclidean distance and geodesic distance?

   Answer: Geodesic distance requires the distance to go through neighbouring points, whereas the Euclidean distance does not enforce that.

6. Are neural networks mainly used for supervised or unsupervised learning? Are they parametric or nonparametric?

   Answer: Supervised, parametric

7. The loss for a neural network is typically non-convex. Give one set of hyperparameters for which the loss is actually convex.

   Answer: nLayers = [1] h(x) = x

8. Assuming we have some procedure that returns a random global minimum of a neural network objective, how does the depth of a neural network affect the fundamental trade-off? For a convolutional network, how would the width of the convolutions affect the fundamental trade-off?

   Answer: The deeper the Network is, the lower training error it gets, but the approximation error goes higher, For convolutional network, the wider the convolutions are, the bigger training error it gets and the lower the approximation it gets.

9. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?

   Answer: When the sigmoid function is far away from 0, the gradiant will be approach 0, which might cause underflow, so the gradiant becomes 0.

10. List 3 forms of regularization we use to prevent overfitting in neural networks.

    Answer: L2 regularization, Early Stopping, SGD

11. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?

    Answer: Because it can represents the relationship between features, so they are not independent to each other. So it can reduce the time of training and overfitting.