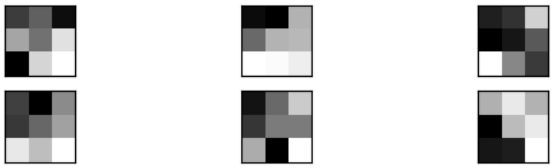
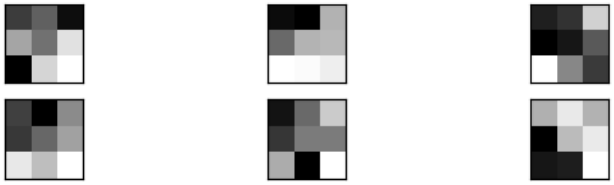
Report

The problem 1 (1), I visualize the filters in the first three convolution layers. First, I saved the cnn model as “cnn_model_new.hdf5”. And then getting the picture. I show the first six.



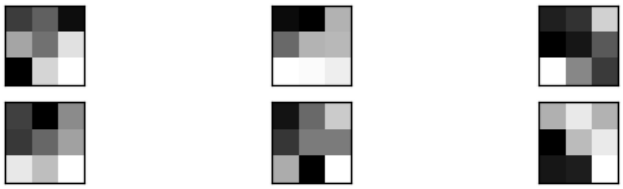
The model_1(3,3,1,32)

This is the filter after first convolution layer.



The model_2(3,3,1,32)

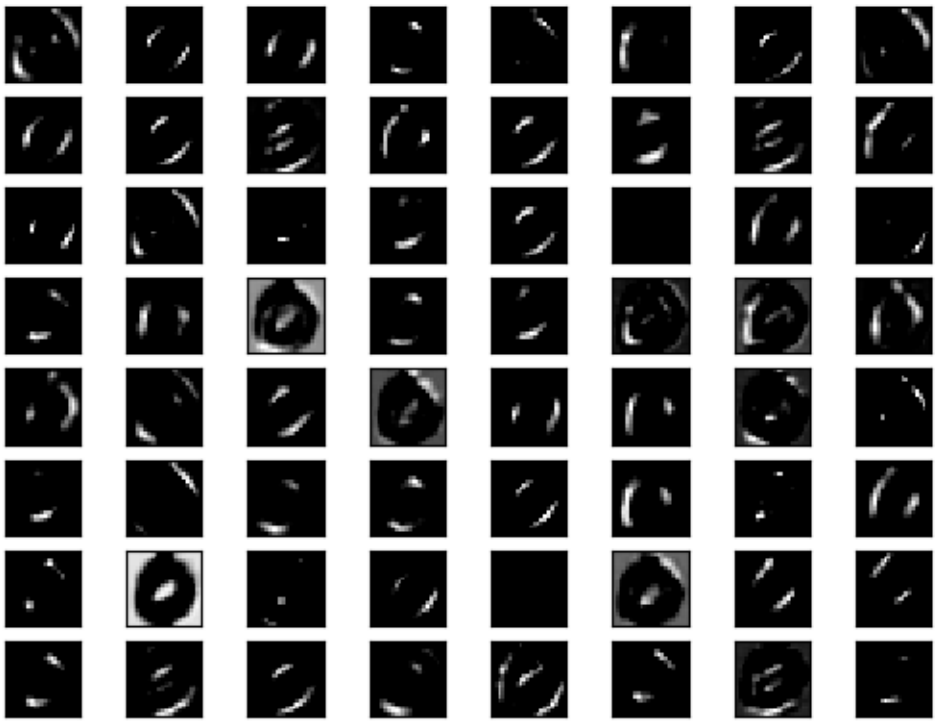
This is the filter after second convolution layer.

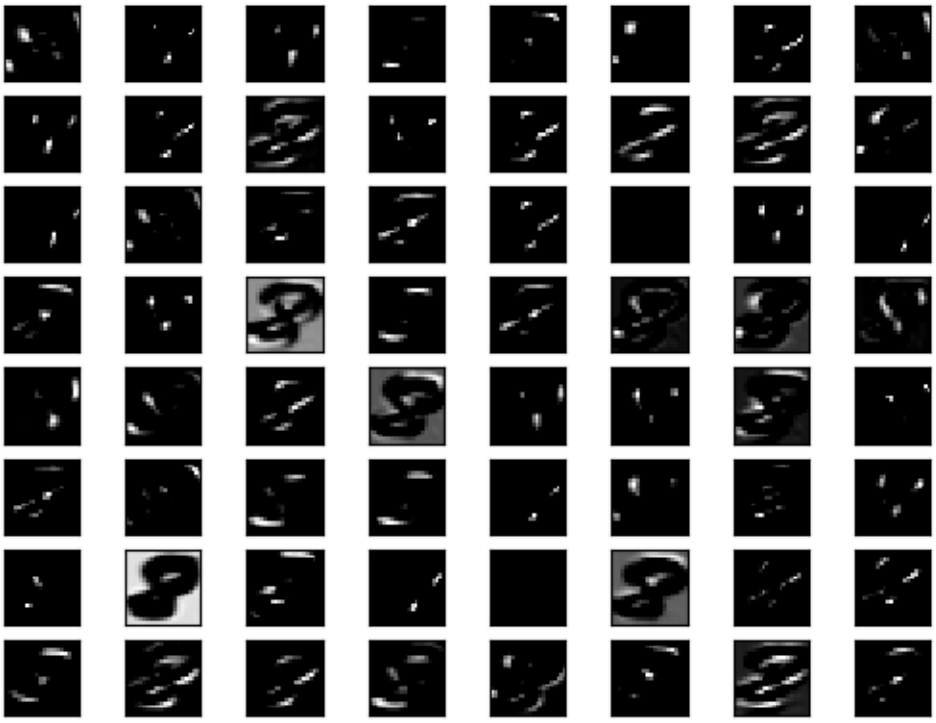


The model_3(3,3,1,32)

This is the filter of third convolution layer.

The problem1 (2):





After three convolution layers, it is difficult to recognize. But some in the picture can be easy to tell.

For others, we can look at the white area, that's the outline of the number. For ‘0’, in the center, there is no white area. But for ‘8’, there is some white in the center. Sometimes, it’s black area. And also, ‘8’ is more complicated than ‘0’ because the Eight has more strokes. I think the important part is the center.

The problem1 (3):

Although I shifted the pixels in ‘1’ and using valid 1 to fill, the prediction of result is not change. The little change is difficult to influence the network accuracy. Please running the ‘visualize_changepixels.py’ to see the result.

The problem2 (1):

the proba of 6:

[0.99945325, 0.9994661, 0.99906594, 0.9982463, 0.9982508, 0.99850345, 0.9964737, 0.99727076, 0.99831176, 0.99935406, 0.9994455, 0.99946207, 0.99929464, 0.99856323, 0.99827695, 0.9982461, 0.9983272, 0.99736005, 0.9972983, 0.99830353, 0.9993531, 0.99944526, 0.9994319, 0.99750805, 0.9969118, 0.99685574, 0.99534565, 0.99823785, 0.99860686, 0.9991015, 0.99946576, 0.99954057, 0.99944896, 0.9991805, 0.98160213, 0.98049176, 0.96752745, 0.96669346, 0.996669, 0.99877924, 0.99955314, 0.9995314, 0.99952805, 0.9995203, 0.99843615, 0.78756475, 0.77699524, 0.7641074, 0.7524222, 0.99700516, 0.9991812, 0.99941504, 0.9994209, 0.99944144, 0.99945194, 0.9981275, 0.46477073, 0.43364063, 0.23758478, 0.09806138, 0.9863885, 0.9964964, 0.9975424, 0.9986425, 0.99941015, 0.9995289, 0.99842584, 0.43606675, 0.4165155, 0.1705772, 0.07307172, 0.9773745, 0.99488527, 0.99697053, 0.9983298, 0.9992269, 0.9994886, 0.99849415, 0.7871121, 0.73029196, 0.5155482, 0.23038994, 0.9523996, 0.9914466, 0.99504435, 0.99831474, 0.99908984, 0.9994677, 0.9989837, 0.8883868, 0.81210536, 0.557346, 0.2385378, 0.95314896, 0.9913509, 0.99497366, 0.9983197, 0.9990896, 0.999468, 0.9994123, 0.9889237, 0.95895034, 0.8439641, 0.7600058, 0.9842552, 0.9982957, 0.99914694, 0.9994684, 0.9992907, 0.9994684, 0.9994553, 0.99920803, 0.99585295, 0.98354095, 0.9775375, 0.99193805, 0.9986694, 0.9995604, 0.99970263, 0.999519, 0.99941444]

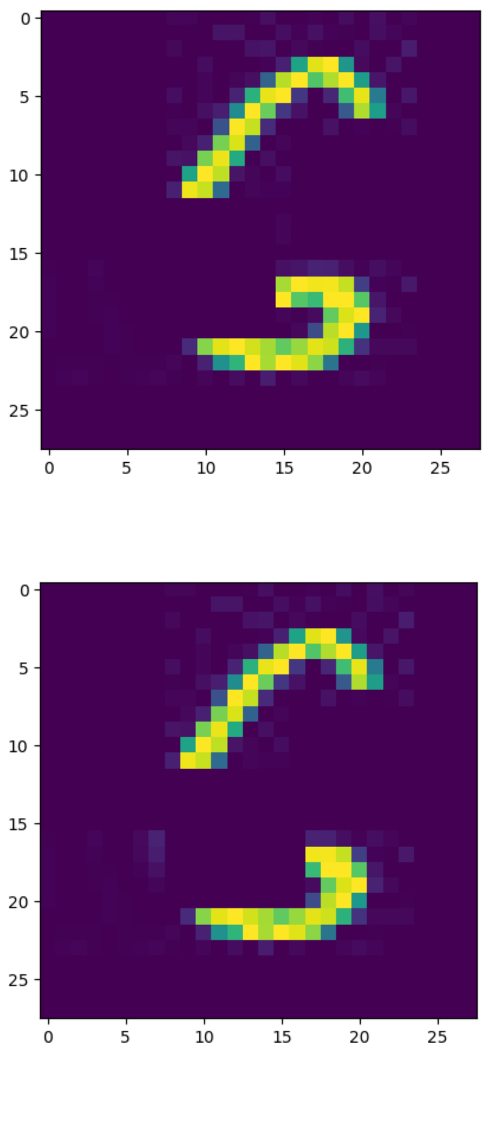
This is the probability of '6'. We can see there are only few numbers lower.

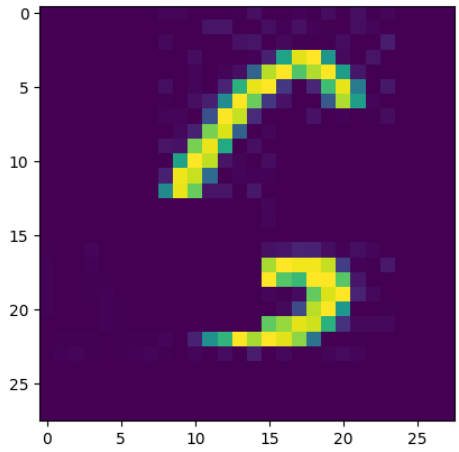
the highest prob:

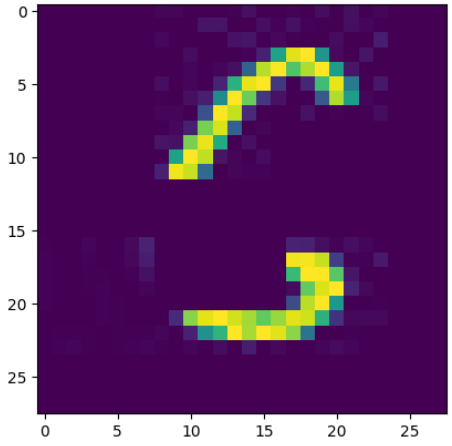
[0.99945325, 0.9994661, 0.99906594, 0.9982463, 0.9982508, 0.99850345, 0.9964737, 0.99727076, 0.99831176, 0.99935406, 0.9994455, 0.99946207, 0.99929464, 0.99856323, 0.99827695, 0.9982461, 0.9983272, 0.99736005, 0.9972983, 0.99830353, 0.9993531, 0.99944526, 0.9994319, 0.99750805, 0.9969118, 0.99685574, 0.99534565, 0.99823785, 0.99860686, 0.9991015, 0.99946576, 0.99954057, 0.99944896, 0.9991805, 0.98160213, 0.98049176, 0.96752745, 0.96669346, 0.996669, 0.99877924, 0.99955314, 0.9995314, 0.99952805, 0.9995203, 0.99843615, 0.78756475, 0.77699524, 0.7641074, 0.7524222, 0.99700516, 0.9991812, 0.99941504, 0.9994209, 0.99944144, 0.99945194, 0.9981275, 0.46477073, 0.43364063, 0.66887665, 0.8448167, 0.9863885, 0.9964964, 0.9975424, 0.9986425, 0.99941015, 0.9995289, 0.99842584, 0.43606675, 0.4165155, 0.74107355, 0.8575564, 0.9773745, 0.99488527, 0.99697053, 0.9983298, 0.9992269, 0.9994886, 0.99849415, 0.7871121, 0.73029196, 0.5155482, 0.68653536, 0.9523996, 0.9914466, 0.99504435, 0.99831474, 0.99908984, 0.9994677, 0.9989837, 0.8883868, 0.81210536, 0.557346, 0.7257384, 0.95314896, 0.9913509, 0.99497366, 0.9983197, 0.9990896, 0.999468, 0.9994123, 0.9889237, 0.95895034, 0.8439641, 0.7600058, 0.9842552, 0.9982957, 0.99914694, 0.9994684, 0.9992907, 0.9994684, 0.9994553, 0.99920803, 0.99585295, 0.98354095, 0.9775375, 0.99193805, 0.9986694, 0.9995604, 0.99970263, 0.999519, 0.99941444]

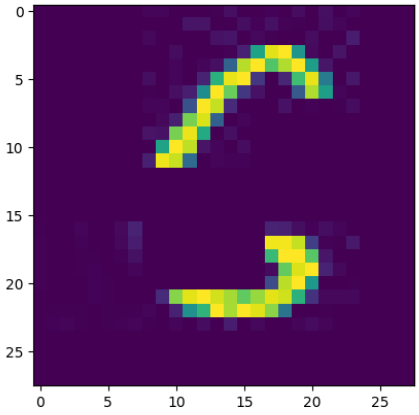
highest proba--label: [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]

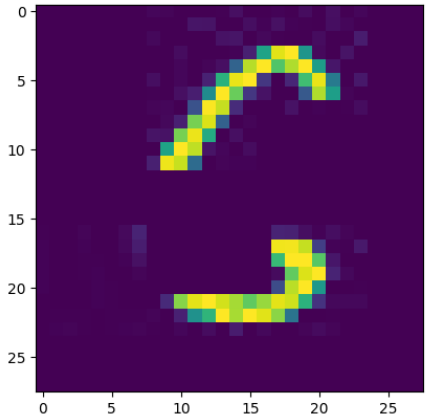
only six places cannot predict correctly. we can see the 59th 60th 60th 71st 82nd 93rd, the predict label is 5. I draw this picture.




0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25





The problem 2 (2):

Based on the pictures above, it's easy to see the important part is x:(6:16) y:(15:23). If covering this place, it is difficult to see which number it is if you don't know it is '6' before. If covering this place, it look like the '5'.

(3) I covered the important place in ‘6’ using ‘8’.

And predicting again.

[6 6 6 6 6 6]

This is the result. All of this can predict correctly. I think that when the important place is blank, RNN will analyses the existing part. For the number ‘5’, it looks like them. When the important place can be filled by other numbers, there are some in blank place. It totally cannot look like the ‘5’ because it should be blank in left bottom. l. So it predicts correct again.

Problem 3:

Result:

loss_b1_1 [3.00720699]

loss_b1_2 [3.00720898]

central_method_loss: [-0.00996139]

loss_b2_1 [3.00725138]

loss_b2_2 [3.00716458]

central_method_loss: [0.43400996]

y1 [0.94001124]

y2 [0.05998876]

old loss [3.00720799]

unfolding_loss_b1: [-0.00996139]

unfolding_loss_b2: [0.43400995]

new_b1, new_b2 [-0.99998008] [0.99913198]

new_loss [[3.0068308]]

(1) t =1: y1: [0.94921601]

y2: [0.05078399]

loss: [3.18196907]

t =2: y1: [0.95221995]

y2: [0.04778005]

loss: [3.24564995]

t=3: y1: [0.94001124]

y2: [0.05998876]

loss: [3.00720799]

(2) (3) (4) (5)

loss_b1_1 [3.00720699]

loss_b1_2 [3.00720898]

central_method_loss: [-0.00996139]

loss_b2_1 [3.00725138]

loss_b2_2 [3.00716458]

central_method_loss: [0.43400996]

y1 [0.94001124]

y2 [0.05998876]

old loss [3.00720799]

unfolding_loss_b1: [-0.00996139]

unfolding_loss_b2: [0.43400995]

new_b1, new_b2 [-0.99998008] [0.99913198]

new_loss [[3.0068308]]

problem 4:

(1)

I think we will have a lot of problems to learn the features using the long protein sequences. If the sequence gets longer, the learning will be difficult.

We know $h_i^{(t)} = \sigma(a_i^{(t)})$. σ is activation function.

Based on the chain rule in derivative, $dh_i^{(t)}/dw = dh_i^{(t)}/da_i^{(t)} * da_i^{(t)}/dw$.

And $da_i^{(t)}/dw = dh_i^{(t-1)}/dw$. We see the derivative of σ will be multiplied again. The time will be based on t. when the activation function is tanh, sigmoid, the derivative is less than 1 or larger than 1. If the derivative of activation function is less than 1, after multiple for many times, it will be very small. Because $dL/dw = dh_i^{(t)}/dw$, the dL/dw will be very small too. If you change the weights based on derivative of loss, the change in weights is very small. So it will cause gradient vanishing. The same thing,

If the derivative of activation is larger than 1, the change in weights will be very large so as to cause gradient exploding. The problem has to do with the size of time. The long sequence of protein will cause this problem. (vanishing or exploding). So, the long-term dependency variables result in the vanishing or exploding gradient problem.

(2) the recurrent weights and the input weights are some of the most difficult parameters to learn due to long-term dependencies. So using echo state networks to avoid this difficulty by fixing the recurrent weights and then learn only the output weights.

(3) There is no gradient vanishing or exploding in LSTM cell.

For LSTM, the equations: $C_t = f_t * C_{t-1} + i_t * \hat{C}_t$, and $\hat{C}_t = \sigma(W h_{t-1} + U X_t + b)$.

We don't have any chained multiplications of activation functions that can lower the value of all the values in the gradient because we only need to consider the previous state. So, there is no gradient vanishing or exploding in LSTM.