Homework 3

Yu Zheng

1 Problem 1

(1) The expected side effects of dropout includes:

1. lower convergence speed;
2. reduce the capacity the network during training;
3. learn a sparse representation.

(2) Bagging and dropout are both the ways to improve the generalization accuracy. However, bagging is to ensemble several networks equally with shared weights, while dropout is an operation on the feature space of one network where certain features are dropped with a probability. Thus, through dropout, the neurons could not reply on one input totally, which cause the representations in the network to be more distributed and avoid overfit. Bagging does not have this feature, it is to improve the generalization accuracy by considering multiple different learning models.
(3) The expected side effects of batch normalization includes:

1. it would explode gradients due to the repeated re-scaling
2. unstable if batch size is small
3. increase training time per iteration

(4) The test result might be worse than training performance if the batch size is not enough for testing.
(5) Usually, we don't use dropout and batch normalization together, especially don't use dropout before batch normalization, since the statistics used to normalize the activation of the prior layer may become noisy given the random dropping out of nodes during the dropout procedure. This is called "variance shift".

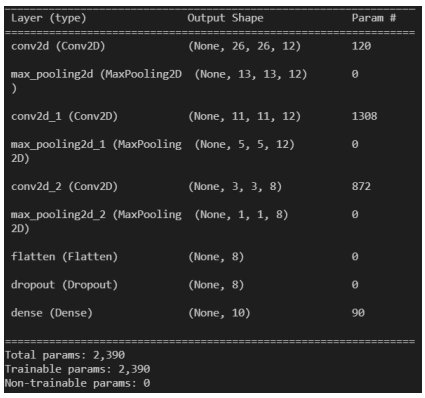
2 Problem 2

(1) Convolution is the operation used to computer a shift invariant linear system's response to any input. Bu setting a smaller kernel than input, the convolution layer has sparse connectivity. Fewer parameters have to be stored, which

both reduces the memory requirements of the model and improves its statistical efficiency. As for parameter sharing, the value of one input is tied to the value of a weight applied elsewhere. The parameter sharing will not affect the runtime of forward propagation but it will reduce the storage requirements of the model to all parameters. In summary, convolution layers is dramatically more efficient than dense layers with respect to the memory capability and statistical efficiency.
(2) Pooling layer is to obtain the output of net at a certain location through a summary statistic of the nearby outputs. Pooling helps to make the representation approximately invariant to small translations of the input. Thus, with pooling layer, it is more statistically efficient if we care more about whether some feature show than exactly where it is. For many tasks, pooling is essential for handling inputs of varying size.
(3) After training, the learned weights of a CNN is represented by the form of kernel which could be used to be applied to input and get the outputs.
(4) The deeper the convolution network is, the wider the receptive fields of the units are. This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image. Thus, the sparse connect would not affect the representation of the input image if the depth of network is enough.

3 Problem 3

I trained the following convolution network, shown in figure 1, with 60000 training dataset and 10000 testing dataset from MNIST dataset. This network has 3 convolution layers.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 12)	120
max_pooling2d (MaxPooling2D)	(None, 13, 13, 12)	0
conv2d_1 (Conv2D)	(None, 11, 11, 12)	1308
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 12)	0
conv2d_2 (Conv2D)	(None, 3, 3, 8)	872
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 8)	0
flatten (Flatten)	(None, 8)	0
dropout (Dropout)	(None, 8)	0
dense (Dense)	(None, 10)	90

Total params: 2,390
Trainable params: 2,390
Non-trainable params: 0

Figure 1: The convolution network's structure

The network achieves 95.62% accuracy on training dataset and 95.92% accuracy on testing dataset. Notice the accuracy can be improved future if I improve the size of convolution layers, but for the sake of presenting the following figures of

filter and feature maps, I am using small size. The loss and accuracy curve is shown in Figure 2.

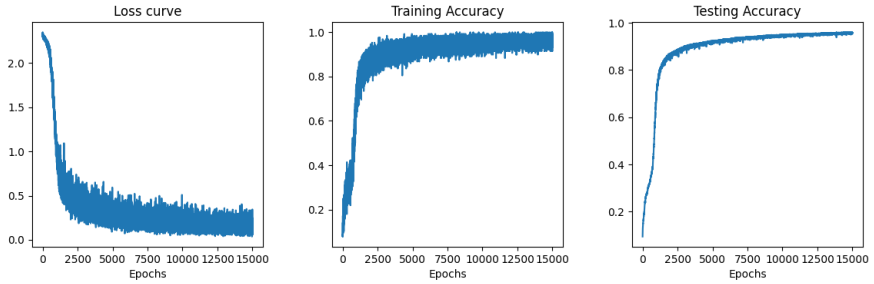


Figure 2: The loss and accuracy curves of the convolution network we use

(1) The filters are shown in figure 3. At different layers the network learns to detect stuff at different levels of abstraction by using different filters. There are 12, 12×12 and 8×12 filters in layer 1, layer 2 and layer 3 respectively. It is seen that the filters at layer 1 are trying to detect the edges, the filters at layer 2 are trying to obtain more abstract information, while the filters at layer 3 are trying to obtain the partial information of the target.

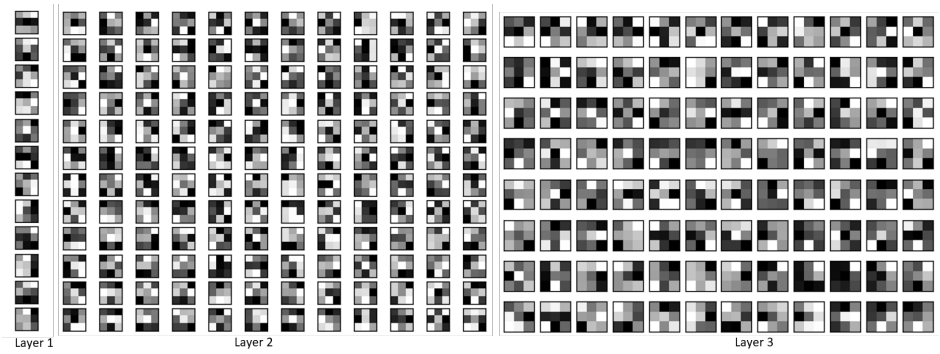


Figure 3: Visualization of filters at conv2D layer 1, layer 2 and layer 3

(2) The feature maps of digit 0 are shown in figure 4, and the feature maps of digit 8 are shown in figure 5. The important discriminant features include

1. digit 8 has a black pixel in the middle, while digit 0 has white pixels in the middle.

2. digit 8 has longer horizontal edges than digit 0 at the top and bottom.

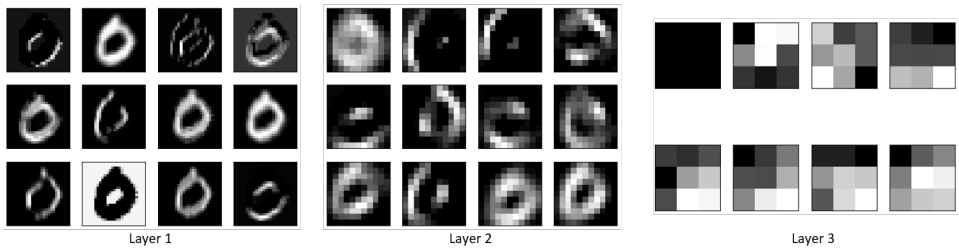


Figure 4: The feature maps of digit 0 at first conv2D layer

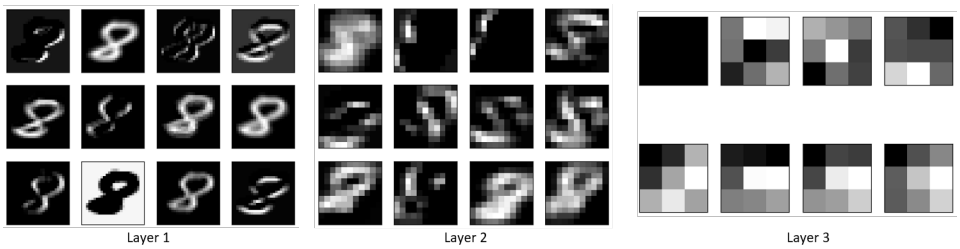


Figure 5: The feature maps of digit 8 at first conv2D layer

(3) Through shifting left and right back, it is like adding adversarial pixels instead of lose partial information. So whether the prediction change or not depends how resilient the convolutional neural network is. The digit 7 after shifting is shown in Figure 6, it is still classified successfully.

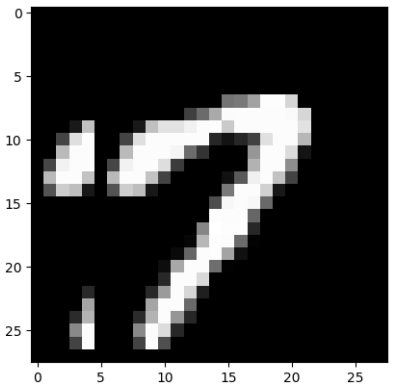


Figure 6: The digit 7 after shifting

4 Problem 4

(1) The three maps are shown in Figure 7, 8, 9.

prob_of_6	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	0.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0
1.0	0.999459	0.999459	0.999459	0.999234	0.998565	0.99805	0.995353	0.996661	0.999459	0.998486	0.998331	0.997917	0.998538	0.999578	0.999806	0.999746	0.999331	0.999038	0.999368	0.999459	0.999459
2.0	0.999459	0.999459	0.999058	0.994458	0.96962	0.980849	0.976682	0.990358	0.999459	0.995630	0.996836	0.997156	0.997894	0.999565	0.999806	0.999746	0.999331	0.999038	0.999368	0.999459	0.999459
3.0	0.999459	0.999512	0.999072	0.961327	0.671204	0.882957	0.966753	0.989686	0.999459	0.994401	0.995777	0.997398	0.997907	0.999592	0.999772	0.999750	0.999331	0.999149	0.999417	0.999459	0.999459
4.0	0.999473	0.99939	0.998361	0.758552	0.139425	0.391397	0.84126	0.9906	0.999459	0.994025	0.995830	0.998560	0.999130	0.999838	0.999899	0.999871	0.999522	0.999427	0.999564	0.999469	0.999459
5.0	0.999464	0.999233	0.996682	0.543808	0.037379	0.112461	0.351033	0.8035	0.999459	0.835295	0.938062	0.980538	0.993392	0.999713	0.999879	0.999927	0.999875	0.999682	0.999115	0.999246	0.999360
6.0	0.999351	0.997067	0.995679	0.79451	0.209979	0.206482	0.35569	0.698368	0.999459	0.640922	0.251700	0.583962	0.715085	0.991234	0.997870	0.999616	0.999706	0.998897	0.998618	0.999207	0.999376
7.0	0.999165	0.993644	0.993821	0.704579	0.151546	0.041124	0.055394	0.09778	0.999452	0.114831	0.071096	0.617561	0.506208	0.980509	0.995986	0.998803	0.999359	0.998729	0.998702	0.999363	0.999482
8.0	0.997694	0.947785	0.976266	0.527083	0.050624	0.015476	0.019854	0.027074	0.999345	0.041911	0.038228	0.153015	0.405679	0.949007	0.991191	0.996343	0.998670	0.998569	0.997919	0.999099	0.999501
0.0	0.999459	0.999459	0.999459	0.999437	0.999561	0.99967	0.999237	0.999184	0.999459	0.999595	0.999516	0.999509	0.999519	0.999606	0.999796	0.999746	0.999331	0.999038	0.999368	0.999459	0.999459
9.0	0.99584	0.904652	0.964549	0.394445	0.028286	0.019249	0.019494	0.023317	0.999261	0.163549	0.746141	0.623893	0.721554	0.942376	0.986232	0.995989	0.997902	0.997603	0.996000	0.998771	0.999439
10.0	0.991584	0.808998	0.881335	0.121965	0.028246	0.025391	0.018563	0.043425	0.999204	0.135816	0.959878	0.821668	0.494843	0.857809	0.988102	0.994877	0.997068	0.997686	0.996091	0.998651	0.999439
11.0	0.991216	0.664538	0.387384	0.027595	0.013216	0.003704	0.002528	0.00966	0.999228	0.152657	0.883544	0.803813	0.296983	0.548407	0.980027	0.995529	0.997089	0.996418	0.995915	0.998642	0.999439
12.0	0.992521	0.613443	0.243067	0.056887	0.028864	0.010827	0.004151	0.013795	0.999250	0.417606	0.931870	0.881145	0.445542	0.515215	0.957512	0.995409	0.994021	0.993652	0.995654	0.998631	0.999439
13.0	0.992968	0.74685	0.433487	0.148998	0.035754	0.01385	0.008182	0.010817	0.999248	0.243212	0.844864	0.968361	0.944011	0.934239	0.995630	0.999423	0.998794	0.998332	0.998827	0.999120	0.999458
14.0	0.994307	0.797582	0.647821	0.558886	0.253279	0.133602	0.117051	0.35638	0.999248	0.893720	0.955239	0.989591	0.987604	0.990747	0.998739	0.999584	0.999051	0.998310	0.999107	0.999352	0.999540
15.0	0.995452	0.919506	0.883962	0.858949	0.781894	0.409413	0.442821	0.580175	0.999256	0.938844	0.976074	0.989217	0.988340	0.996180	0.999007	0.999358	0.998741	0.998345	0.999237	0.999429	0.999575
16.0	0.997547	0.978938	0.98196	0.982986	0.978428	0.83887	0.828984	0.888669	0.999387	0.982048	0.995674	0.997312	0.996354	0.996889	0.998468	0.998819	0.998277	0.998437	0.999123	0.999434	0.999502
17.0	0.998859	0.997812	0.99834	0.997729	0.992929	0.957442	0.957223	0.962101	0.999520	0.992159	0.997823	0.998425	0.998120	0.998399	0.999068	0.999504	0.999303	0.999175	0.999122	0.999409	0.999459
18.0	0.999408	0.999381	0.999409	0.999042	0.997288	0.992394	0.992402	0.991337	0.999553	0.995512	0.997952	0.998578	0.998936	0.999265	0.999598	0.999760	0.999631	0.999418	0.999433	0.999459	0.999459
19.0	0.999453	0.999263	0.999199	0.999162	0.998435	0.996134	0.994938	0.992733	0.999500	0.995747	0.997653	0.998095	0.998278	0.999292	0.999503	0.999675	0.999589	0.999466	0.999459	0.999459	0.999459
20.0	0.999522	0.999492	0.999458	0.999471	0.999353	0.999051	0.998889	0.998649	0.999485	0.998587	0.998811	0.998953	0.999052	0.999146	0.999239	0.999466	0.999474	0.999459	0.999459	0.999459	0.999459

Figure 7: Probability of digit "6" when using 8×8 patch to occlude part of the image

highest_prob	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	0.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0
1.0	0.999459	0.999459	0.999459	0.999234	0.998565	0.99805	0.995353	0.996661	0.999459	0.998486	0.998331	0.997917	0.998538	0.999578	0.999806	0.999746	0.999331	0.999038	0.999368	0.999459	0.999459
2.0	0.999459	0.999459	0.999058	0.994458	0.96962	0.980849	0.976682	0.990358	0.999459	0.995630	0.996836	0.997156	0.997894	0.999565	0.999806	0.999746	0.999331	0.999038	0.999368	0.999459	0.999459
3.0	0.999459	0.999512	0.999072	0.961327	0.671204	0.882957	0.966753	0.989686	0.999459	0.994401	0.995777	0.997398	0.997907	0.999592	0.999772	0.999750	0.999331	0.999149	0.999417	0.999459	0.999459
4.0	0.999473	0.99939	0.998361	0.758552	0.75875	0.52652	0.84126	0.9906	0.999459	0.994025	0.995830	0.998560	0.999130	0.999838	0.999899	0.999871	0.999522	0.999427	0.999564	0.999469	0.999459
5.0	0.999464	0.999233	0.996682	0.543808	0.823683	0.755186	0.405042	0.8035	0.999459	0.835295	0.938062	0.980538	0.993392	0.999713	0.999879	0.999927	0.999875	0.999682	0.999115	0.999246	0.999360
6.0	0.999351	0.997067	0.995679	0.79451	0.65203	0.571914	0.35569	0.698368	0.999459	0.640922	0.668018	0.583962	0.715085	0.991234	0.997870	0.999616	0.999706	0.998897	0.998618	0.999207	0.999376
7.0	0.999165	0.993644	0.993821	0.704579	0.360061	0.664338	0.646397	0.487158	0.999452	0.745326	0.915532	0.617561	0.506208	0.980509	0.995986	0.998803	0.999359	0.998729	0.998702	0.999363	0.999482
8.0	0.997694	0.947785	0.976266	0.527083	0.321351	0.608339	0.575425	0.402625	0.999345	0.884959	0.958104	0.846681	0.593942	0.949007	0.991191	0.996343	0.998670	0.998569	0.997919	0.999099	0.999501
0.0	0.999459	0.999459	0.999459	0.999437	0.999561	0.99967	0.999237	0.999184	0.999459	0.999595	0.999516	0.999509	0.999519	0.999606	0.999796	0.999746	0.999331	0.999038	0.999368	0.999459	0.999459
9.0	0.99584	0.904652	0.964549	0.394445	0.487986	0.593504	0.655079	0.393699	0.999261	0.715833	0.746141	0.623893	0.721554	0.942376	0.986232	0.995989	0.997902	0.997603	0.996000	0.998771	0.999439
10.0	0.991584	0.808998	0.881335	0.359677	0.55321	0.805167	0.846017	0.511524	0.999204	0.826935	0.959878	0.821668	0.505099	0.857809	0.988102	0.994877	0.997068	0.997686	0.996091	0.998651	0.999439
11.0	0.991216	0.664538	0.43654	0.537569	0.460314	0.668582	0.722569	0.563323	0.999228	0.770407	0.883544	0.803813	0.702993	0.548407	0.980027	0.995529	0.997089	0.996418	0.995915	0.998642	0.999439
12.0	0.992521	0.613443	0.677312	0.748287	0.49672	0.700437	0.791207	0.585034	0.999250	0.417606	0.931870	0.881145	0.554428	0.515215	0.957512	0.995409	0.994021	0.993652	0.995654	0.998631	0.999439
13.0	0.992968	0.74885	0.494219	0.657584	0.428734	0.610205	0.491826	0.557662	0.999248	0.313003	0.844864	0.968361	0.944011	0.934239	0.995630	0.999423	0.998794	0.998332	0.998827	0.999120	0.999458
14.0	0.994307	0.797582	0.647821	0.558886	0.310623	0.513505	0.460039	0.421544	0.999248	0.893720	0.955239	0.989591	0.987604	0.990747	0.998739	0.999584	0.999051	0.998310	0.999107	0.999352	0.999540
15.0	0.995452	0.919506	0.883962	0.858949	0.781894	0.409413	0.442821	0.580175	0.999256	0.938844	0.976074	0.989217	0.988340	0.996180	0.999007	0.999358	0.998741	0.998345	0.999237	0.999429	0.999575
16.0	0.997547	0.978938	0.98196	0.982986	0.978428	0.83887	0.828984	0.888669	0.999387	0.982048	0.995674	0.997312	0.996354	0.996889	0.998468	0.998819	0.998277	0.998437	0.999123	0.999434	0.999502
17.0	0.998859	0.997812	0.99834	0.997729	0.992929	0.957442	0.957223	0.962101	0.999520	0.992159	0.997823	0.998425	0.998120	0.998399	0.999068	0.999504	0.999303	0.999175	0.999122	0.999409	0.999459
18.0	0.999408	0.999381	0.999409	0.999042	0.997288	0.992394	0.992402	0.991337	0.999553	0.995512	0.997952	0.998578	0.998936	0.999265	0.999598	0.999760	0.999631	0.999418	0.999433	0.999459	0.999459
19.0	0.999453	0.999263	0.999199	0.999162	0.998435	0.996134	0.994938	0.992733	0.999500	0.995747	0.997653	0.998095	0.998278	0.999292	0.999503	0.999675	0.999589	0.999466	0.999459	0.999459	0.999459
20.0	0.999522	0.999492	0.999458	0.999471	0.999353	0.999051	0.998889	0.998649	0.999485	0.998587	0.998811	0.998953	0.999052	0.999146	0.999239	0.999466	0.999474	0.999459	0.999459	0.999459	0.999459

Figure 8: Highest probability outputted by network when using 8×8 patch to occlude part of the image

(2) From Figure 9, it is obvious that the range of pixels which are important is: $[\text{row}(4, 22), \text{column}(5, 20)]$. Thus, the circle part of digit 6 is important for recognition.
(3) As suggested form (2), We could use a small black patch to block the circle part of digit 6, such in Figure 10. The pixel $[\text{row}(11, 19), \text{column}(5, 13)]$ are blocked by a 8×8 black patch. It is recognized as digit 5. All feasible adversarial images with 8×8 black patch are saved in the folder `'/Result_prob4/adv_img'`.

	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	0.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0
label																					
1.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
2.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
3.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
4.0	6	6	6	6	0	0	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
5.0	6	6	6	6	0	0	0	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
6.0	6	6	6	6	0	0	6	6	6.0	6.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
7.0	6	6	6	6	5	5	5	5	6.0	0.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
8.0	6	6	6	6	5	5	5	5	6.0	0.0	0.0	0.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
0.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
9.0	6	6	6	6	5	5	5	5	6.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
10.0	6	6	6	5	5	5	5	5	6.0	0.0	6.0	6.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
11.0	6	6	8	8	5	5	5	5	6.0	0.0	6.0	6.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
12.0	6	6	8	8	8	5	5	5	6.0	6.0	6.0	6.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
13.0	6	6	8	8	8	9	9	9	6.0	0.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
14.0	6	6	6	6	8	9	5	5	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
15.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
16.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
17.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
18.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
19.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
20.0	6	6	6	6	6	6	6	6	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0

Figure 9: Classified labels when using 8×8 patch to occlude part of the image

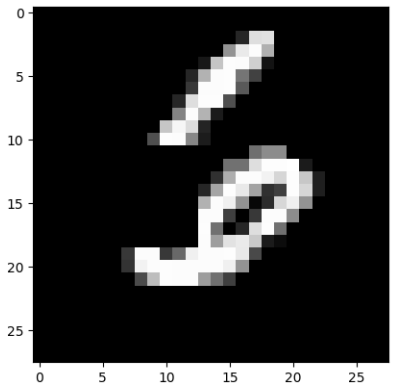


Figure 10: An example adversarial image, it is recognized as digit 5

5 Problem 5

(1)

$$y^{(1)} = \begin{bmatrix} 0.94921601 \\ 0.05078399 \end{bmatrix}, y^{(2)} = \begin{bmatrix} 0.95221995 \\ 0.04778005 \end{bmatrix}, y^{(3)} = \begin{bmatrix} 0.94001124 \\ 0.05998876 \end{bmatrix},$$

$$L = 0.255473337380186$$

(2)

$$\frac{\partial L}{\partial b_1} = 0.00010432302510743341, \quad \frac{\partial L}{\partial b_2} = -0.0045452732375795055$$

(3)

$$\frac{\partial L}{\partial y^{(3)}} = \begin{bmatrix} -1.06381706 & -0.88002248 \end{bmatrix},$$

$$\frac{\partial y^{(3)}}{\partial h^{(3)}} = \begin{bmatrix} -0.05639011 & 0.05639011 \\ 0.05639011 & -0.05639011 \end{bmatrix},$$

$$\frac{\partial h^{(3)}}{\partial b^{(3)}} = \begin{bmatrix} 0.01002062 & 0 \\ 0 & 0.42731798 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{(3)}} = \frac{\partial L}{\partial y^{(3)}} \frac{\partial y^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial b^{(3)}} = \begin{bmatrix} 0.00010386 & -0.00442881 \end{bmatrix}$$

$$\frac{\partial h^{(3)}}{\partial h^{(2)}} = \begin{bmatrix} 0.01002062 & -0.01002062 \\ 0 & 0.85463595 \end{bmatrix},$$

$$\frac{\partial h^{(2)}}{\partial b^{(2)}} = \begin{bmatrix} 0.00420315 & 0 \\ 0 & 0.01138416 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial y^{(3)}} \frac{\partial y^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial b^{(2)}} = \begin{bmatrix} 4.36520935e-07 & -1.02018799e-04 \end{bmatrix}$$

$$\frac{\partial h^{(2)}}{\partial h^{(1)}} = \begin{bmatrix} 0.01002062 & -0.01002062 \\ 0 & 0.85463595 \end{bmatrix},$$

$$\frac{\partial h^{(2)}}{\partial b^{(2)}} = \begin{bmatrix} 0.07065082 & 0 \\ 0 & 0.07065082 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial y^{(3)}} \frac{\partial y^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial b^{(1)}} = \begin{bmatrix} 3.08405641e-08 & -1.44462651e-05 \end{bmatrix}$$

Therefore, the gradients of the loss with respect to b is:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial b^{(1)}} + \frac{\partial L}{\partial b^{(2)}} + \frac{\partial L}{\partial b^{(3)}} = \begin{bmatrix} 0.00010432 & -0.00454527 \end{bmatrix}$$

(4) Updated b through one step of gradient descent is

$$b = \begin{bmatrix} -1.00000021 & 1.00000909 \end{bmatrix}$$

(5) The loss by using the new b:

$$L = 0.25547329603987234$$

6 Problem 6

(1) Input weights and recurrent weights W, U are most difficult parameters to learn due to the long-term dependency. They are relatively further away from the loss in each layer of of the network. If some of parameters are large, then gradient exploding problem will show up due to long-time term. If some of parameters are small, then gradient vanishing problem will show up due to long-time term.
(2) The recurrent weights W, U should be fixed, and only output weights c, V should be learned.
(3) LSTM has three gates: forget gate, input gate and output gate. Forget gate uses sigmoid activation function and controls what information in the cell state to forget.

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

Input gate uses tanh and sigmoid function, and controls what new information will be propagated in the cell state.

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t]), i_t = \sigma(W_i[h_{t-1}, x_t])$$

Output gate uses sigmoid activation function, and controls what information in cell state till be sent to the network. Thus, the

$$o_t = \sigma(W_o[h_{t-1}, x_t]), h_t = o_t \otimes \tanh(c_t)$$

Thus, the cell state captures all long-term dependencies, and it has the following propagation law:

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$

Then, the loss gradient with respect to the furthest recurrent weight becomes:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial c^{(3)}} \frac{\partial c^{(3)}}{\partial c^{(2)}} \frac{\partial c^{(2)}}{\partial c^{(1)}} \frac{\partial c^{(1)}}{\partial W}$$

Usually, this part $\frac{\partial c^{(3)}}{\partial c^{(2)}} \frac{\partial c^{(2)}}{\partial c^{(1)}}$ will result in gradient vanishing or exploding. However, in LSTM, this part is an **addictive gradient function**:

$$\frac{\partial c^{(t+1)}}{\partial c^{(t)}} = A_t + B_t + C_t + D_t$$

where

$$\begin{aligned}
A_t &= \sigma^{'}(W_f[h_{t-1}, x_t])W_f o_{t-1} \otimes \tanh^{'}(c_{t-1}) \\
B_t &= f_t \\
C_t &= \sigma^{'}(W_f[h_{t-1}, x_t])W_i o_{t-1} \otimes \tanh^{'}(c_{t-1})\tilde{c}_t \\
D_t &= \sigma^{'}(W_f[h_{t-1}, x_t])W_c o_{t-1} \otimes \tanh^{'}(c_{t-1}) i_t
\end{aligned}$$

Notice that A_t, B_t contains the forget term which we could adjust to make sure the addictive gradients not vanish or explode.

7 Problem 7

I implement the LSTM cell described in equations $(10.40) - (10.44)$ in the deep learning book. I use the same U, W, b in problem 5 for all three gates of LSTM cell, i.e. $W_o = W_i = W_f = W$, $U_o = U_i = U_f = U$, $b_o = b_i = b_f = b$. And I choose the initial LSTM internal state as $s_0 = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$. The outputs and the cutomized loss are

$$y^{(1)} = \begin{bmatrix} 0.83367573 \\ 0.16632427 \end{bmatrix}, y^{(2)} = \begin{bmatrix} 0.86175934 \\ 0.13824066 \end{bmatrix}, y^{(3)} = \begin{bmatrix} 0.83104223 \\ 0.16895777 \end{bmatrix},$$

$$L = 0.2946636237723147$$

For calculating the gradients, we only need to replace the recurrent part in the Problem 5 by the LSTM cell. The gradients in LSTM cell are:

$$\begin{aligned}
\frac{\partial h^{(t)}}{\partial s^{(t)}} &= \mathsf{tanh}^{'}(s^{(t)}), \\
\frac{\partial s^{(t)}}{\partial f^{(t)}} &= s^{(t-1)}, \\
\frac{\partial f^{(t)}}{\partial bf^{(t)}} &= \mathsf{sigmoid}^{'}(W_f h^{(t-1)} + U_f x^{(t)} + b_f), \\
\frac{\partial f^{(t)}}{\partial h^{(t-1)}} &= W_f \mathsf{sigmoid}^{'}(W_f h^{(t-1)} + U_f x^{(t)} + b_f)
\end{aligned}$$

Thus,

$$\begin{aligned}
\frac{\partial h^{(t)}}{\partial bf^{(t)}} &= \mathsf{tanh}^{'}(s^{(t)}) \otimes s^{(t-1)} \otimes \mathsf{sigmoid}^{'}(W_f h^{(t-1)} + U_f x^{(t)} + b_f), \\
\frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \mathsf{tanh}^{'}(s^{(t)}) \otimes s^{(t-1)} \otimes W_f \mathsf{sigmoid}^{'}(W_f h^{(t-1)} + U_f x^{(t)} + b_f)
\end{aligned}$$

where $\otimes$ denote piecewise multiplication. By coding this, I got the gradients as follows:

$$\begin{aligned}
\frac{\partial L}{\partial bf^{(3)}} &= \begin{bmatrix} 0.00013102 & -0.01754358 \end{bmatrix} \\
\frac{\partial L}{\partial bf^{(2)}} &= \begin{bmatrix} -2.91351203e-07 & -8.19807080e-04 \end{bmatrix} \\
\frac{\partial L}{\partial bf^{(1)}} &= \begin{bmatrix} -2.74254195e-10 & -1.07033861e-05 \end{bmatrix}
\end{aligned}$$

Thus, the total gradient of loss with respect to forget gate bias is:

$$\frac{\partial L}{\partial bf} = \begin{bmatrix} 0.00013073 & -0.01837409 \end{bmatrix}$$