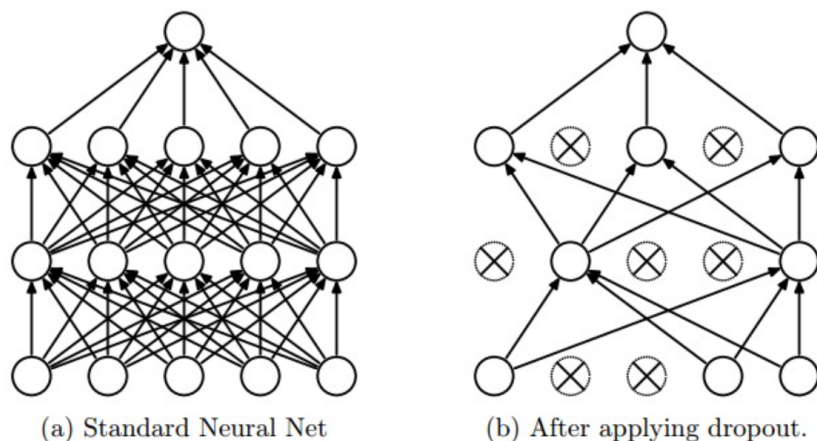


Dropout是一种简单有效的正则化方式

在**训练**的时候，随机失活的实现方法是让神经元以超参数 p 的概率被设置为0。



在训练过程中，随机失活可以被认为是对完整神经网络抽样出一些子集，每次基于输入数据值只更新子网络的参数（然而，数量巨大的子网络们并不是相互独立的，因为它们都共享参数）。

在测试（或者验证）时候不使用随机失活，可以理解为对数量巨大的子网络们做了模型集成，以此来计算出一个平均的预测。

理解了dropout的基本原理，下面我们来实现下dropout层。

主要内容如下：

1 Dropout层

1.1 前向传播

1.2 反向传播

2 训练

2.1 开始训练

2.2 结果分析

1 Dropout层

1.1 前向传播

代码如下：

```
1 def dropout_forward(x, dropout_param):
2
3     p, mode = dropout_param['p'], dropout_param['mode']
4     if 'seed' in dropout_param:
```

```

5      np.random.seed(dropout_param['seed'])
6
7      mask = None
8      out = None
9
10     if mode == 'train':
11
12         #####
13         ###
14         # TODO: Implement the training phase forward pass for inverted
15         dropout. #
16         # Store the dropout mask in the mask variable.
17         #
18
19         #####
20         ###
21         mask = (np.random.rand(*x.shape) >= p) / (1 - p)
22         # mask = (np.random.rand(x.shape[1]) >= p) / (1 - p)
23         out = x * mask
24         # pass
25
26         #####
27         ###
28         #
29         #
30         #
31         #
32         #
33         #
34         #
35         #
36         #
37         #
38         #
39         #
40         #
41         #
42         #
43         #
44         #
45         #
46         #
47         #
48         #
49         #
50         #
51         #
52         #
53         #
54         #
55         #
56         #
57         #
58         #
59         #
60         #
61         #
62         #
63         #
64         #
65         #
66         #
67         #
68         #
69         #
70         #
71         #
72         #
73         #
74         #
75         #
76         #
77         #
78         #
79         #
80         #
81         #
82         #
83         #
84         #
85         #
86         #
87         #
88         #
89         #
90         #
91         #
92         #
93         #
94         #
95         #
96         #
97         #
98         #
99         #
100        #
101        #
102        #
103        #
104        #
105        #
106        #
107        #
108        #
109        #
110        #
111        #
112        #
113        #
114        #
115        #
116        #
117        #
118        #
119        #
120        #
121        #
122        #
123        #
124        #
125        #
126        #
127        #
128        #
129        #
130        #
131        #
132        #
133        #
134        #
135        #
136        #
137        #
138        #
139        #
140        #
141        #
142        #
143        #
144        #
145        #
146        #
147        #
148        #
149        #
150        #
151        #
152        #
153        #
154        #
155        #
156        #
157        #
158        #
159        #
160        #
161        #
162        #
163        #
164        #
165        #
166        #
167        #
168        #
169        #
170        #
171        #
172        #
173        #
174        #
175        #
176        #
177        #
178        #
179        #
180        #
181        #
182        #
183        #
184        #
185        #
186        #
187        #
188        #
189        #
190        #
191        #
192        #
193        #
194        #
195        #
196        #
197        #
198        #
199        #
200        #
201        #
202        #
203        #
204        #
205        #
206        #
207        #
208        #
209        #
210        #
211        #
212        #
213        #
214        #
215        #
216        #
217        #
218        #
219        #
220        #
221        #
222        #
223        #
224        #
225        #
226        #
227        #
228        #
229        #
230        #
231        #
232        #
233        #
234        #
235        #
236        #
237        #
238        #
239        #
240        #
241        #
242        #
243        #
244        #
245        #
246        #
247        #
248        #
249        #
250        #
251        #
252        #
253        #
254        #
255        #
256        #
257        #
258        #
259        #
260        #
261        #
262        #
263        #
264        #
265        #
266        #
267        #
268        #
269        #
270        #
271        #
272        #
273        #
274        #
275        #
276        #
277        #
278        #
279        #
280        #
281        #
282        #
283        #
284        #
285        #
286        #
287        #
288        #
289        #
290        #
291        #
292        #
293        #
294        #
295        #
296        #
297        #
298        #
299        #
300        #
301        #
302        #
303        #
304        #
305        #
306        #
307        #
308        #
309        #
310        #
311        #
312        #
313        #
314        #
315        #
316        #
317        #
318        #
319        #
320        #
321        #
322        #
323        #
324        #
325        #
326        #
327        #
328        #
329        #
330        #
331        #
332        #
333        #
334        #
335        #
336        #
337        #
338        #
339        #
340        #
341        #
342        #
343        #
344        #
345        #
346        #
347        #
348        #
349        #
350        #
351        #
352        #
353        #
354        #
355        #
356        #
357        #
358        #
359        #
360        #
361        #
362        #
363        #
364        #
365        #
366        #
367        #
368        #
369        #
370        #
371        #
372        #
373        #
374        #
375        #
376        #
377        #
378        #
379        #
380        #
381        #
382        #
383        #
384        #
385        #
386        #
387        #
388        #
389        #
390        #
391        #
392        #
393        #
394        #
395        #
396        #
397        #
398        #
399        #
400        #
401        #
402        #
403        #
404        #
405        #
406        #
407        #
408        #
409        #
410        #
411        #
412        #
413        #
414        #
415        #
416        #
417        #
418        #
419        #
420        #
421        #
422        #
423        #
424        #
425        #
426        #
427        #
428        #
429        #
430        #
431        #
432        #
433        #
434        #
435        #
436        #
437        #
438        #
439        #
440        #
441        #
442        #
443        #
444        #
445        #
446        #
447        #
448        #
449        #
450        #
451        #
452        #
453        #
454        #
455        #
456        #
457        #
458        #
459        #
460        #
461        #
462        #
463        #
464        #
465        #
466        #
467        #
468        #
469        #
470        #
471        #
472        #
473        #
474        #
475        #
476        #
477        #
478        #
479        #
480        #
481        #
482        #
483        #
484        #
485        #
486        #
487        #
488        #
489        #
490        #
491        #
492        #
493        #
494        #
495        #
496        #
497        #
498        #
499        #
500        #
501        #
502        #
503        #
504        #
505        #
506        #
507        #
508        #
509        #
510        #
511        #
512        #
513        #
514        #
515        #
516        #
517        #
518        #
519        #
520        #
521        #
522        #
523        #
524        #
525        #
526        #
527        #
528        #
529        #
530        #
531        #
532        #
533        #
534        #
535        #
536        #
537        #
538        #
539        #
540        #
541        #
542        #
543        #
544        #
545        #
546        #
547        #
548        #
549        #
550        #
551        #
552        #
553        #
554        #
555        #
556        #
557        #
558        #
559        #
560        #
561        #
562        #
563        #
564        #
565        #
566        #
567        #
568        #
569        #
570        #
571        #
572        #
573        #
574        #
575        #
576        #
577        #
578        #
579        #
580        #
581        #
582        #
583        #
584        #
585        #
586        #
587        #
588        #
589        #
590        #
591        #
592        #
593        #
594        #
595        #
596        #
597        #
598        #
599        #
600        #
601        #
602        #
603        #
604        #
605        #
606        #
607        #
608        #
609        #
610        #
611        #
612        #
613        #
614        #
615        #
616        #
617        #
618        #
619        #
620        #
621        #
622        #
623        #
624        #
625        #
626        #
627        #
628        #
629        #
630        #
631        #
632        #
633        #
634        #
635        #
636        #
637        #
638        #
639        #
640        #
641        #
642        #
643        #
644        #
645        #
646        #
647        #
648        #
649        #
650        #
651        #
652        #
653        #
654        #
655        #
656        #
657        #
658        #
659        #
660        #
661        #
662        #
663        #
664        #
665        #
666        #
667        #
668        #
669        #
670        #
671        #
672        #
673        #
674        #
675        #
676        #
677        #
678        #
679        #
680        #
681        #
682        #
683        #
684        #
685        #
686        #
687        #
688        #
689        #
690        #

```

```

30 #####
31 ###
32     cache = (dropout_param, mask)
33     out = out.astype(x.dtype, copy=False)
34
35     return out, cache

```

我们来检验下，检验代码如下：

```

1  x = np.random.randn(500, 500) + 10
2
3  for p in [0.3, 0.6, 0.75]:
4      out, _ = dropout_forward(x, {'mode': 'train', 'p': p})
5      out_test, _ = dropout_forward(x, {'mode': 'test', 'p': p})
6
7      print ('Running tests with p = ', p)
8      print ('Mean of input: ', x.mean())
9      print ('Mean of train-time output: ', out.mean())
10     print ('Mean of test-time output: ', out_test.mean())
11     print ('Fraction of train-time output set to zero: ', (out ==
12     0).mean())
12     print ('Fraction of test-time output set to zero: ', (out_test ==
13     0).mean())
13     print()

```

输出结果如下：

```

1  Running tests with p = 0.3
2  Mean of input: 9.99902471947
3  Mean of train-time output: 9.98399347022
4  Mean of test-time output: 9.99902471947
5  Fraction of train-time output set to zero: 0.301064
6  Fraction of test-time output set to zero: 0.0
7
8  Running tests with p = 0.6
9  Mean of input: 9.99902471947
10 Mean of train-time output: 10.00544314
11 Mean of test-time output: 9.99902471947
12 Fraction of train-time output set to zero: 0.599708
13 Fraction of test-time output set to zero: 0.0
14

```

```

15 Running tests with p = 0.75
16 Mean of input: 9.99902471947
17 Mean of train-time output: 10.037004918
18 Mean of test-time output: 9.99902471947
19 Fraction of train-time output set to zero: 0.749104
20 Fraction of test-time output set to zero: 0.0

```

我们来对结果分析下，当 $p=0.3$ 时，在训练过程中，我们的神经元有0.3的概率失活，也就是被置0，我们可以看到神经元为0的概率为0.301064，约等于0.3；而在测试过程中，将所有神经元集成，也就是使用所有的神经元，所以神经元为0的概率为0。

1.2 反向传播

下面我们来看下dropout层的反向传播，代码如下：

```

1 def dropout_backward(dout, cache):
2
3     dropout_param, mask = cache
4     mode = dropout_param['mode']
5
6     dx = None
7     if mode == 'train':
8
9         #####
10        ###
11        # TODO: Implement the training phase backward pass for inverted
12        dropout. #
13
14        #####
15        ###
16
17        dx = dout * mask
18        # pass
19
20        #####
21        ###
22
23        #
24        #
25
26        #####
27        ###
28
29        elif mode == 'test':
30            dx = dout
31        return dx

```

同样，我们对梯度进行检验。

代码如下：

```
1 x = np.random.randn(10, 10) + 10
2 dout = np.random.randn(*x.shape)
3
4 dropout_param = {'mode': 'train', 'p': 0.8, 'seed': 123}
5 out, cache = dropout_forward(x, dropout_param)
6 dx = dropout_backward(dout, cache)
7 dx_num = eval_numerical_gradient_array(lambda xx: dropout_forward(xx,
8 dropout_param)[0], x, dout)
9
9 print ('dx relative error: ', rel_error(dx, dx_num))
```

得到结果如下：

```
1 dx relative error: 1.89290485329e-11
```

结果是 $1e-11$ ，误差还是很小的，不错。

2 训练

完成了对dropout层的设计，下面我们就使用下dropout层来对cifar10数据进行训练，来看下dropout层的效果。

2.1 开始训练

为了加快训练，这里我们只使用两层网络，即一个隐层。

代码如下：

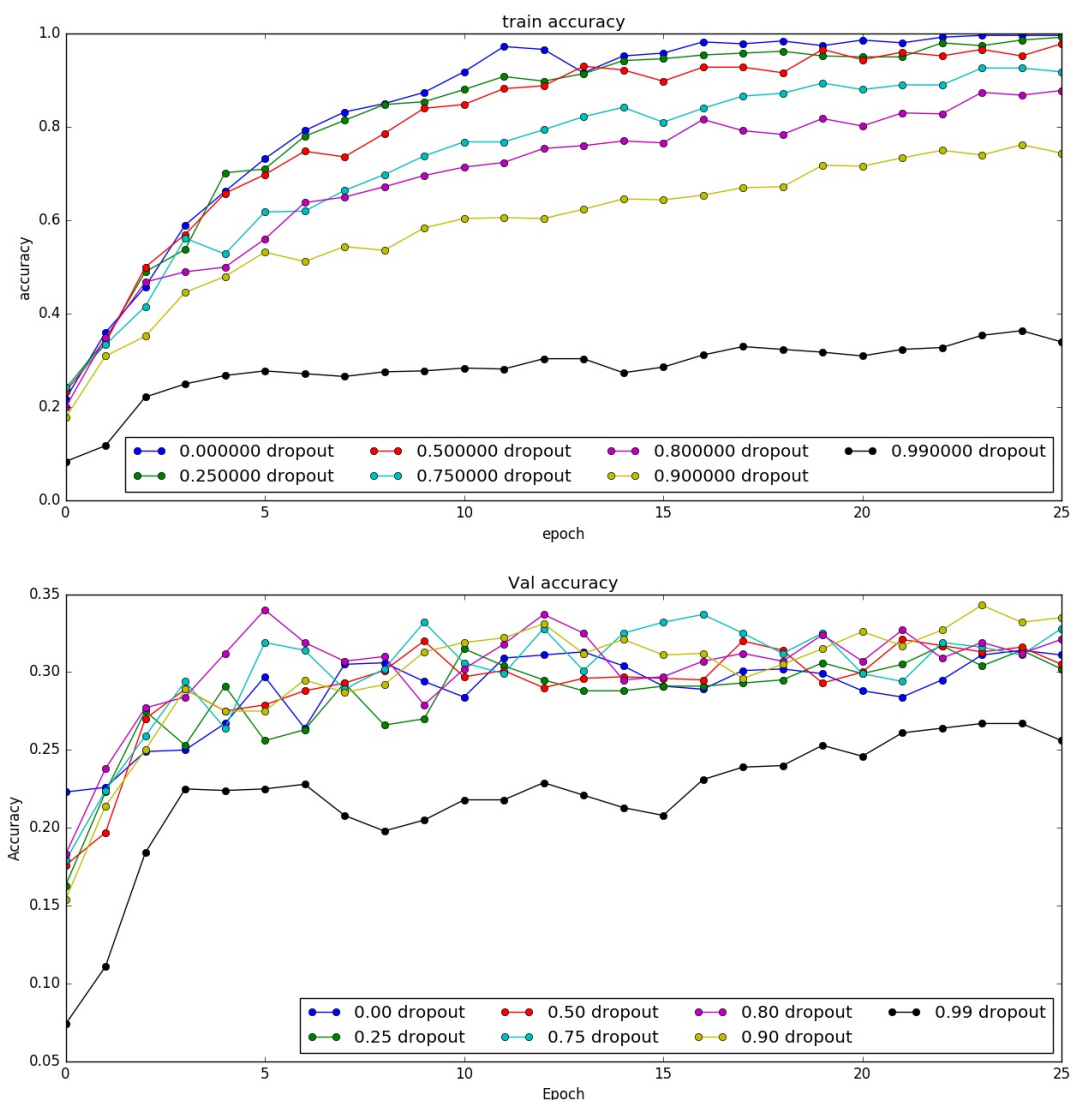
```
1 num_train=500
2
3 small_data={
4     'X_train':data['X_train'][:num_train],
5     'y_train':data['y_train'][:num_train],
6     'X_val':data['X_val'],
7     'y_val':data['y_val']
8 }
9 }
```

```

10
11 solvers={}
12 dropout_choices=[0,0.25,0.5,0.75,0.8,0.9,0.99]
13 for dropout in dropout_choices:
14     model=FullyConnectedNet([500],dropout=dropout)
15     print('dropout is:',dropout)
16
17     solver=Solver(model,small_data,num_epochs=25,batch_size=100,update_rule=
    'adam',
18                     optim_config={'learning_rate':5e-
19 4},verbose=True,print_every=100)
20     solver.train()
21     solvers[dropout]=solver
22
23 train_acc=[]
24 val_acc=[]
25 for dropout in dropout_choices:
26     solver=solvers[dropout]
27     train_acc.append(solver.train_acc_history[-1])
28     val_acc.append(solver.val_acc_history[-1])
29
30 plt.subplot(311)
31 for dropout in dropout_choices:
32     plt.plot(solvers[dropout].train_acc_history,'-o',label='%2f dropout'
    % dropout)
33 plt.title('train accuracy')
34 plt.xlabel('epoch')
35 plt.ylabel('accuracy')
36 plt.legend(ncol=4,loc='lower right')
37
38 plt.subplot(312)
39 for dropout in dropout_choices:
40     plt.plot(solvers[dropout].val_acc_history, 'o', label='%2f dropout' %
    dropout)
41 plt.title('Val accuracy')
42 plt.xlabel('Epoch')
43 plt.ylabel('Accuracy')
44 plt.legend(ncol=4, loc='lower right')
45
46 plt.gcf().set_size_inches(15, 15)
47 plt.show()

```

得到的可视化图如下：



2.2 结果分析

增加Dropout层可以防止过拟合，降低训练集和验证集之间的差距。但是随着dropout中p值的增加，模型的拟合能力也就变差，也就是降低了神经网络的容量，可能出现欠拟合。因此在实践中一般要谨慎选择p值。

同时，学习了BN层的读者，可能已经想到，在网络中增加BN层，可以减少对dropout层的依赖。