Rademacher_F1

June 18, 2020

[10]:
```
import numpy as np
import matplotlib.pyplot as plt
import compute_parameters
import pandas as pd
import statsmodels.api as sm
from scipy.stats import norm
from scipy.optimize import curve_fit
import pylab
```

[11]:
```
def get_actual_predicted(trace, trace_name, jvm, hidden_size, history_size=40,
                         model_type="fcn", output_file_location="/media/arjun/
↪Shared/chaos/output_files"):
    start_point = 10000
    n_points = 5000

    if jvm == "jikes":
        jvm_name = "JikesRVM"
    elif jvm == "j9":
        jvm_name = "J9"
    else:
        jvm_name = "HotSpot"

    offset = 3

    data = []

    predictions = np.load(
                 '{}/{}/{}/predictions_{}_{}_{}_{}.npy'.
↪format(output_file_location,
                                                                    trace_name,␣
↪jvm, history_size,
                                                                    1,␣
↪hidden_size, 1))
    predictions = np.argsort(predictions)
#     print(predictions.shape)
#     print(trace.shape)
    for idx, point in enumerate(trace):
```

```
        if idx >= len(trace)-offset-1:
            continue
#         print("{}: {}".format(np.argmax(point),␣
↪predictions[history_size-1+idx, -1]))
        actual = bin_to_val(int(np.argmax(point)))
        predicted = bin_to_val(predictions[history_size+idx+offset, -1])

        data.append((actual, predicted))

    return data
```

[12]:
```
def mse_function(y, y1):
    return np.mean((y-y1)**2)

## precision and recall
def relevance_function(x, sigma=1.0, mu=0):
    return np.exp(-((x-mu)**2)/(2*(sigma**2)))/(sigma*np.sqrt(2*np.pi))


def relevance_function1(x, mu=0):
    return 1


def alpha(y, y_pred, loss_function=mse_function, threshold=3e-1):
    return loss_function(y, y_pred) < threshold


def recall(data, y_ref, loss_function=mse_function, relevance_threshold=0.3):
    num = 0
    din = 0
    for y_actual, y_pred in data:
        phi_y = relevance_function(y_actual, mu=y_ref)
        if phi_y >= relevance_threshold:
            num += alpha(y_actual, y_pred, loss_function) * phi_y
            din += phi_y

    if din > 0:
        return num/din
    else:
        return 0



def existance_check(data, y_ref, loss_function=mse_function,␣
↪relevance_threshold=0.3):
    num = 0
    din = 0
```

```
    exists = False
    for y_actual, y_pred in data:
        phi_y = relevance_function(y_actual, mu=y_ref)
        if phi_y >= relevance_threshold:
            return True

    return False


def precision(data, y_ref, loss_function=mse_function, relevance_threshold=0.3):
    num = 0
    din = 0
    for y_actual, y_pred in data:
        phi_y1 = relevance_function(y_pred, mu=y_ref)
        if phi_y1 >= relevance_threshold:
            num += alpha(y_actual, y_pred, loss_function) * phi_y1
            din += phi_y1

    if din > 0:
        return num/din
    else:
        return 0


def f1_score(precision, recall, beta=1):
    if precision+recall > 0:
        return ((1+beta**2)*precision*recall)/(precision+recall)
    else:
        return 0


def bin_to_val(bin_idx):
    g_max = 1
    g_min = 3 * np.exp(-8)
    feature_dimension = 100
    multiplier = (np.log(g_max) - np.log(g_min))/feature_dimension # values␣
 ↪from preprocess cache file

    return bin_idx*multiplier + np.log(g_min)
```

[13]:

```
def sup(arr, mode="max"):
    if mode == "max":
        return (np.max(arr))
    elif mode == "999percentile":
        return np.mean(arr) + 5 * np.std(arr)
```

```
def get_rademacher(loss_array):
    rademacher = []

    n_sigma = 2000

    for i in range(n_sigma):
        sigma_arr = np.random.choice([1, -1], size=loss_array.shape)

        f = sigma_arr*loss_array
        f = np.sum(f, axis=1)/loss_array.shape[1]
        # print(f)
        rademacher.append(sup(f))

    return np.mean(rademacher)
```

[14]:
```
def get_loss_dict1(trace, output_file_location, hidden_sizes=None,␣
↪trace_name="pmd", plot_graphs=False):

    if hidden_sizes is None:
        hidden_sizes = [ 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000, 6000,␣
↪7000, 8000 ]

    history_sizes = [ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 ]
    history_size = 40
    n_seeds = 30
    n_sets = 1
    n_points = 5000
    chunk_size = int(n_points/n_sets)
    relevance_threshold = relevance_function(1e-1)
    loss_dict = {}
    min_val = np.log(3e-8)
    max_val = 0
    n_steps = 50

    print("relevance threshold: {}".format(relevance_threshold))

    for hidden_size in hidden_sizes:
        loss_array = np.zeros((n_seeds, n_sets))
        print("hidden_size: {}".format(hidden_size))

        if hidden_size in loss_dict.keys():
            continue

        for seed in range(n_seeds):
            data1 = get_actual_predicted(trace, trace_name, "jikes",␣
↪hidden_size=hidden_size, history_size=40,
```

```
                         model_type="lstm",␣
↪output_file_location=output_file_location)

          indices = np.arange(0, n_points, chunk_size)

          for index in indices:
    #            print(index, index+chunk_size)
    #            print(np.linspace(0, n_points, n_sets))
              data = data1[index:index+chunk_size]
              recall_vals = []
              precision_vals = []
              f1_vals = []
              for val in np.linspace(min_val, max_val, n_steps):
                  recall_val = recall(data, val,␣
↪relevance_threshold=relevance_threshold)
                  precision_val = precision(data, val,␣
↪relevance_threshold=relevance_threshold)
                  recall_vals.append(recall_val)
                  precision_vals.append(precision_val)

#                    print("precision: {}, recall: {}".format(precision_val,␣
↪recall_val))
                  f1_vals.append(f1_score(precision_val, recall_val))

              f1_avg_list=[]
              for idx, val in enumerate(np.linspace(min_val, max_val,␣
↪n_steps)):
                  if existance_check(data, val,␣
↪relevance_threshold=relevance_threshold):
                      f1_avg_list.append(f1_vals[idx])

              if plot_graphs and index==0 and seed==1:
                  ax1=plt.subplot(1, 3, 1)
                  ax2=plt.subplot(1, 3, 2)
                  ax3=plt.subplot(1, 3, 3)

                  ax1.figure.set_size_inches(10, 3)
                  ax2.figure.set_size_inches(10, 3)
                  ax3.figure.set_size_inches(10, 3)

                  ax1.set_title("precision")
                  ax2.set_title("recall")
                  ax3.set_title("f1")

                  ax1.plot(np.linspace(min_val, max_val, n_steps),␣
↪precision_vals, label="precision")
```

```
                    ax2.plot(np.linspace(min_val, max_val, n_steps),␣
↪recall_vals, label="recall")
                    ax3.plot(np.linspace(min_val, max_val, n_steps), f1_vals,␣
↪label="f1")

                    ax1.set_ylim((-0.1, 1.1))
                    ax2.set_ylim((-0.1, 1.1))
                    ax3.set_ylim((-0.1, 1.1))

                    plt.title("Hidden: {}, seed: {}".format(hidden_size, seed))
                    plt.legend()
                    plt.show()

                chunk_idx = int(index/chunk_size)

                loss_array[seed, chunk_idx] = 1-np.average(f1_avg_list)
        loss_dict[hidden_size] = loss_array

    return loss_dict
```

[23]:
```
def get_loss_dict2(trace, output_file_location, hidden_sizes=None,␣
↪trace_name="pmd", plot_graphs=False):

    if hidden_sizes is None:
        hidden_sizes = [ 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000, 6000,␣
↪7000, 8000 ]

    history_sizes = [ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 ]
    history_size = 40
    n_seeds = 30
    n_sets = 50
    n_points = 5000
    chunk_size = 100
    relevance_threshold = relevance_function(1e-1)
    loss_dict = {}
    min_val = np.log(3e-8)
    max_val = 0
    n_steps = 50

    print("relevance threshold: {}".format(relevance_threshold))

    for hidden_size in hidden_sizes:
        loss_array = np.zeros((n_seeds, n_sets))
        print("hidden_size: {}".format(hidden_size))

        if hidden_size in loss_dict.keys():
            continue
```

```
        for seed in range(n_seeds):
            data1 = get_actual_predicted(trace, trace_name, "jikes",␣
↪hidden_size=hidden_size, history_size=40,
                          model_type="lstm",␣
↪output_file_location=output_file_location)

            indices = np.random.choice(np.arange(n_points-chunk_size), n_sets)

            for index_idx, index in enumerate(indices):
    #             print(index, index+chunk_size)
    #             print(np.linspace(0, n_points, n_sets))
                data = data1[index:index+chunk_size]
                recall_vals = []
                precision_vals = []
                f1_vals = []
                for val in np.linspace(min_val, max_val, n_steps):
                    recall_val = recall(data, val,␣
↪relevance_threshold=relevance_threshold)
                    precision_val = precision(data, val,␣
↪relevance_threshold=relevance_threshold)
                    recall_vals.append(recall_val)
                    precision_vals.append(precision_val)

#                     print("precision: {}, recall: {}".format(precision_val,␣
↪recall_val))
                    f1_vals.append(f1_score(precision_val, recall_val))

                f1_avg_list=[]
                for idx, val in enumerate(np.linspace(min_val, max_val,␣
↪n_steps)):
                    if existance_check(data, val,␣
↪relevance_threshold=relevance_threshold):
                        f1_avg_list.append(f1_vals[idx])

                if plot_graphs and index_idx==0 and seed==1:
                    ax1=plt.subplot(1, 3, 1)
                    ax2=plt.subplot(1, 3, 2)
                    ax3=plt.subplot(1, 3, 3)

                    ax1.figure.set_size_inches(10, 3)
                    ax2.figure.set_size_inches(10, 3)
                    ax3.figure.set_size_inches(10, 3)

                    ax1.set_title("precision")
                    ax2.set_title("recall")
```

```
                    ax3.set_title("f1")

                    ax1.plot(np.linspace(min_val, max_val, n_steps),␣
↪precision_vals, label="precision")
                    ax2.plot(np.linspace(min_val, max_val, n_steps),␣
↪recall_vals, label="recall")
                    ax3.plot(np.linspace(min_val, max_val, n_steps), f1_vals,␣
↪label="f1")

                    ax1.set_ylim((-0.1, 1.1))
                    ax2.set_ylim((-0.1, 1.1))
                    ax3.set_ylim((-0.1, 1.1))

                    plt.title("Hidden: {}, seed: {}".format(hidden_size, seed))
                    plt.legend()
                    plt.show()

                chunk_idx = int(index/chunk_size)

                loss_array[seed, index_idx] = 1-np.average(f1_avg_list)
        loss_dict[hidden_size] = loss_array

    return loss_dict
```

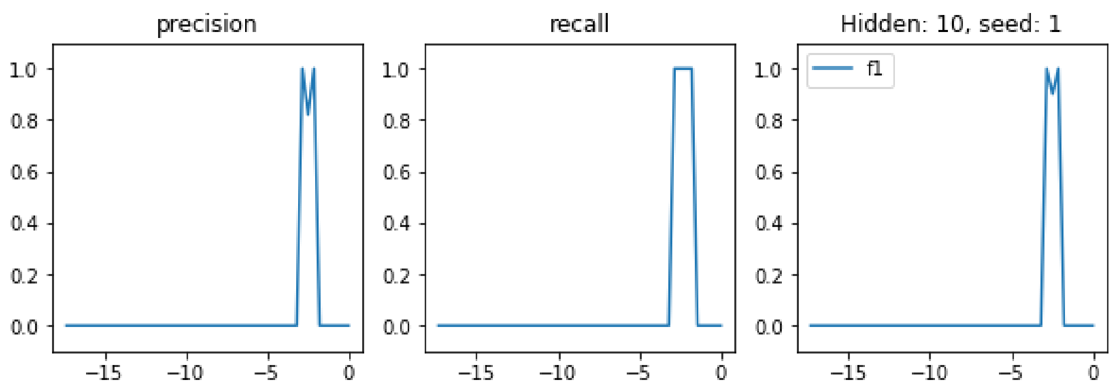
[16]:
```
trace_name = "pmd"
start_point = 10000
n_points = 5000
jvm = "jikes"

if jvm == "jikes":
    jvm_name = "JikesRVM"
elif jvm == "j9":
    jvm_name = "J9"
else:
    jvm_name = "HotSpot"

trace = pd.read_pickle(
                '../data/{}-small-{}-d-l64-p4096-w100000i.analyzed-1.pkl'.
↪format(trace_name, jvm_name)
            ).to_numpy()[start_point:start_point+n_points]
```

[17]:
```
loss_main = {}
# hidden_sizes=[ 1, 10, 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000 ]
# hidden_sizes_lstm = [ 99, 148, 198, 248, 298, 347, 845, 1342,
#                       1840, 2337, 2835, 3332, 3830, 4327, 4825 ]
```

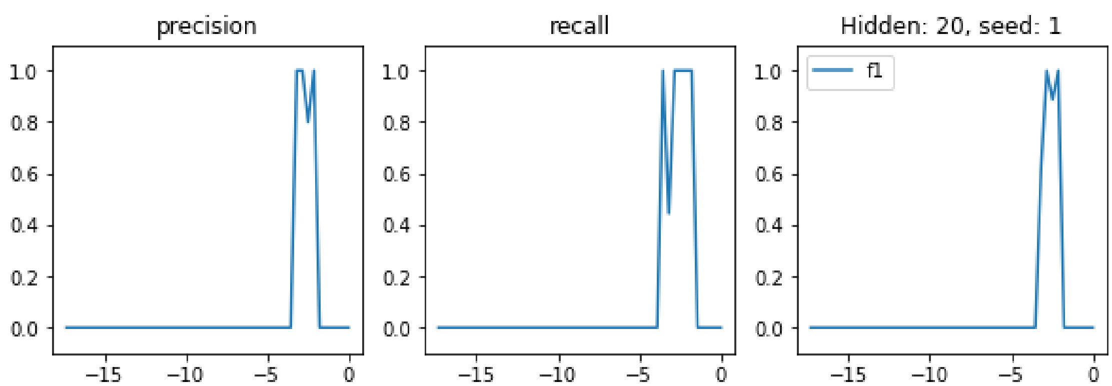
```
hidden_sizes_lstm = [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 99, 148, 198, 248,␣
↪298, 347, 845, 1342,
                     1840, 2337, 2835, 3332, 3830, 4327, 4825 ]
hidden_sizes_fcn = [ 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300,
                    400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000,
                   6000, 7000, 8000 ]
```

[ ]:
```
hidden_sizes = hidden_sizes_lstm

loss_main["lstm"] = get_loss_dict2(trace, "/media/arjun/Shared/chaos/
↪output_files_v2/lstm",
                                   hidden_sizes=hidden_sizes, plot_graphs=True)
```

```
relevance threshold: 0.3969525474770118
hidden_size: 10
```



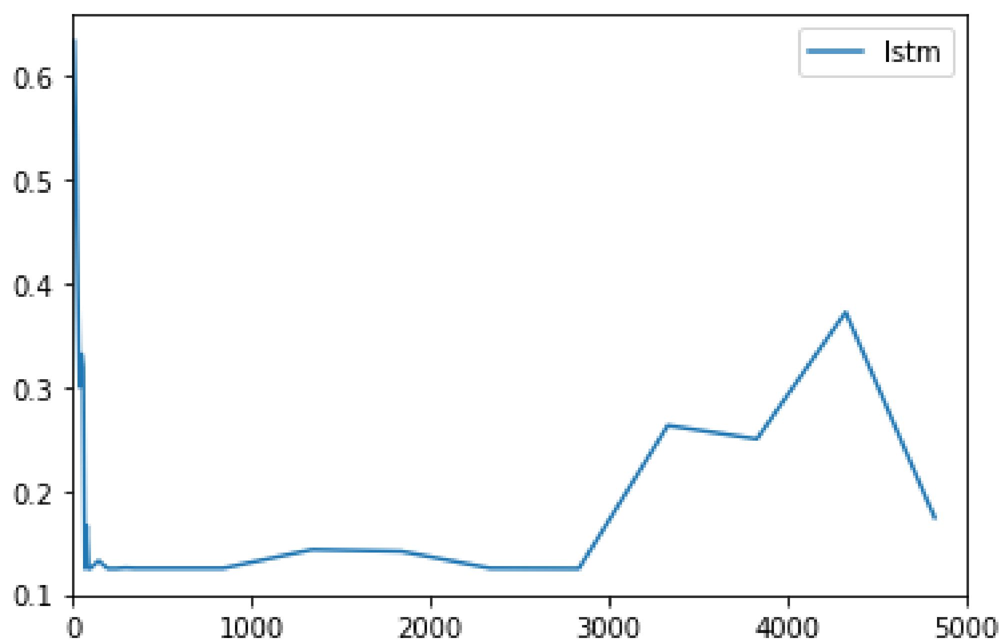

```
hidden_size: 20
```



[10]:
```
rademacher_list_lstm = []

for hidden_size in hidden_sizes_lstm:
    loss_array = loss_main["lstm"][hidden_size]
    rademacher_list_lstm.append(get_rademacher(loss_array))
```

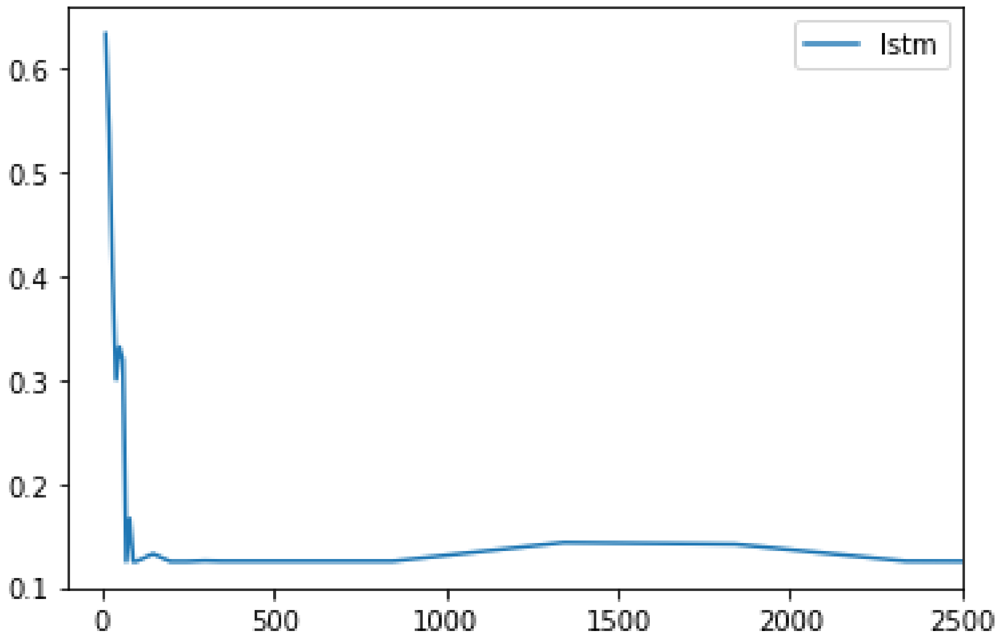
[11]:
```
# compute number of parameters
fcn_param_list = []
lstm_param_list = []

for idx, hidden_size in enumerate(hidden_sizes_fcn):
    fcn_param_list.append(compute_parameters.get_count(4000, 100, [␣
 ↪hidden_sizes_fcn[idx] ], [ 'fcn' ]))

for idx, hidden_size in enumerate(hidden_sizes):
    lstm_param_list.append(compute_parameters.get_count(100, 100, [100,␣
 ↪hidden_sizes_lstm[idx]], ["lstm", "fcn"]))
```

[12]:
```
# plt.plot(fcn_param_list, rademacher_list_fcn, label="fcn")
plt.plot(hidden_sizes_lstm, rademacher_list_lstm, label="lstm")
plt.legend()
# plt.ylim((0.8, 1.1))
plt.xlim((-0, 5000))
plt.show()
```



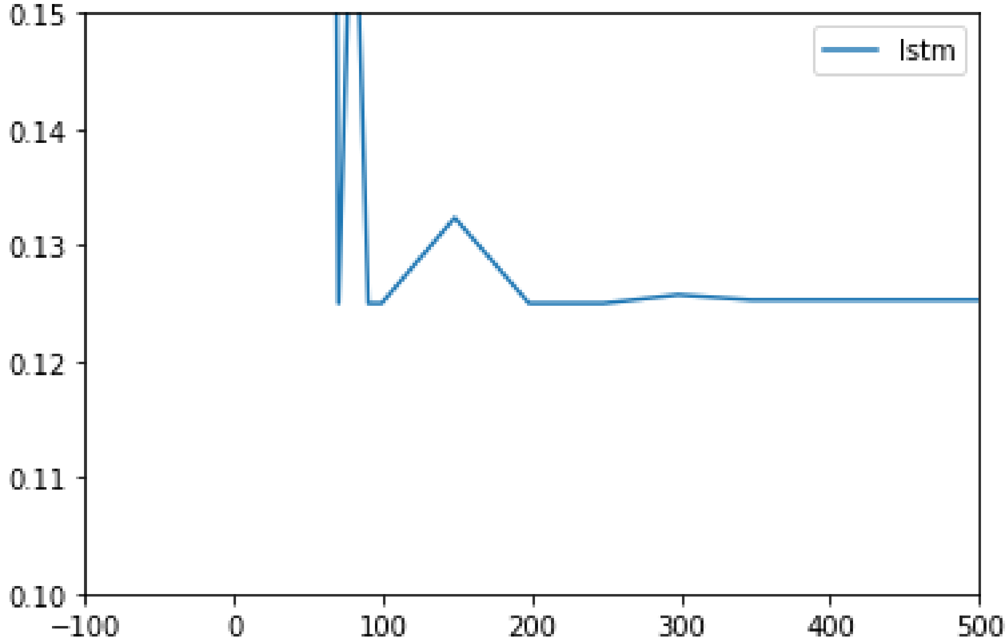
[13]:

```
# plt.plot(fcn_param_list, rademacher_list_fcn, label="fcn")
plt.plot(hidden_sizes_lstm, rademacher_list_lstm, label="lstm")
plt.legend()
# plt.ylim((0.8, 1.1))
plt.xlim((-100, 2500))
plt.show()
```



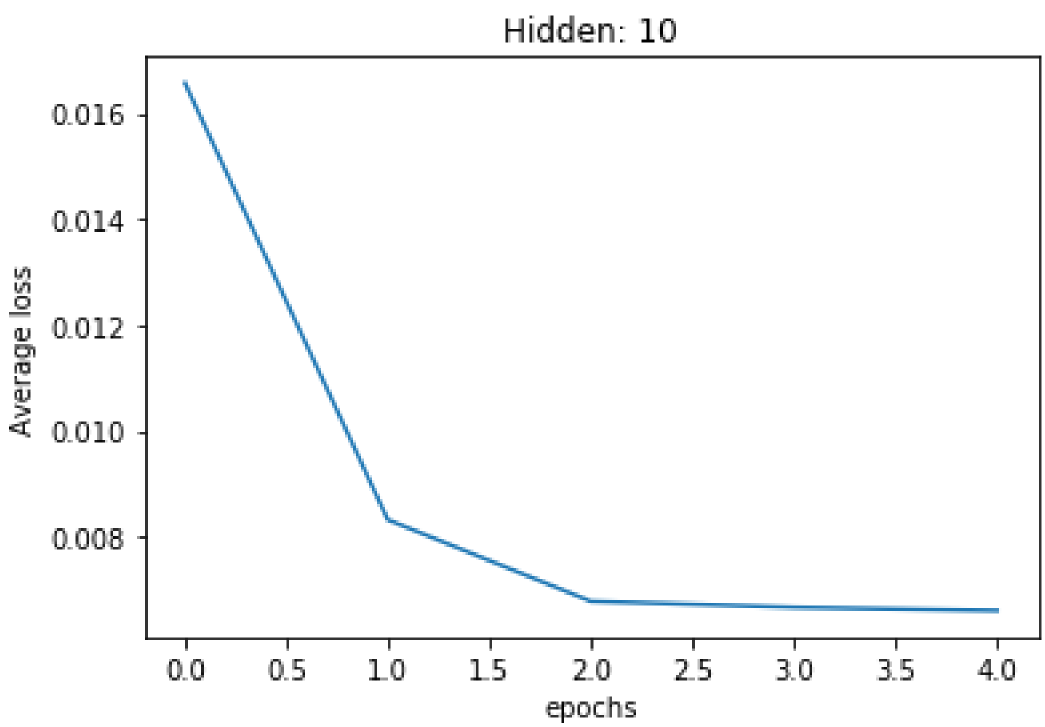
[25]:

```
# plt.plot(fcn_param_list, rademacher_list_fcn, label="fcn")
plt.plot(hidden_sizes_lstm, rademacher_list_lstm, label="lstm")

plt.legend()
plt.ylim((0.1, 0.15))
plt.xlim((-100, 500))
plt.show()

for idx, val in enumerate(rademacher_list_lstm):
    print(hidden_sizes_lstm[idx]," ", val)
```

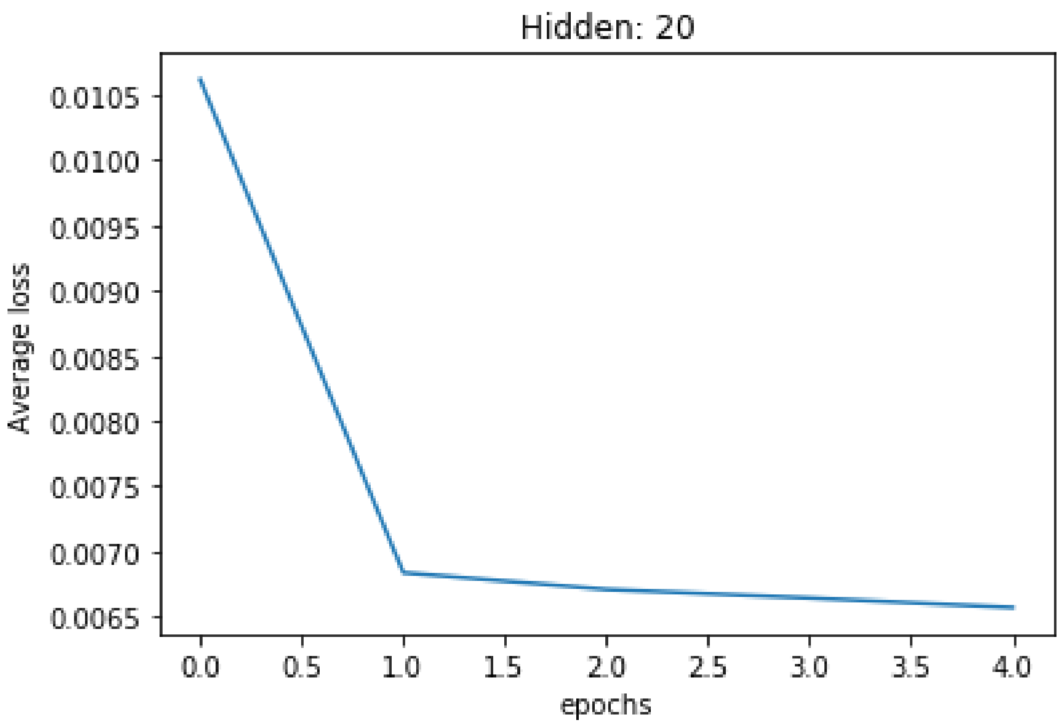


10	0.6333156038184563
20	0.538711162846022
30	0.4031134227186018
40	0.30017246823683535
50	0.33168431737696336
60	0.31972028403494684
70	0.125
80	0.16666581770387978
90	0.125
99	0.125
148	0.13237044062757694
198	0.125
248	0.125
298	0.12573297917419635
347	0.12529079198605203
845	0.12529072319261253
1342	0.1432662839673639
1840	0.1416085300044363
2337	0.12529072319261253
2835	0.125
3332	0.2630236827860713
3830	0.25
4327	0.37225972995745493
4825	0.1740853828770101

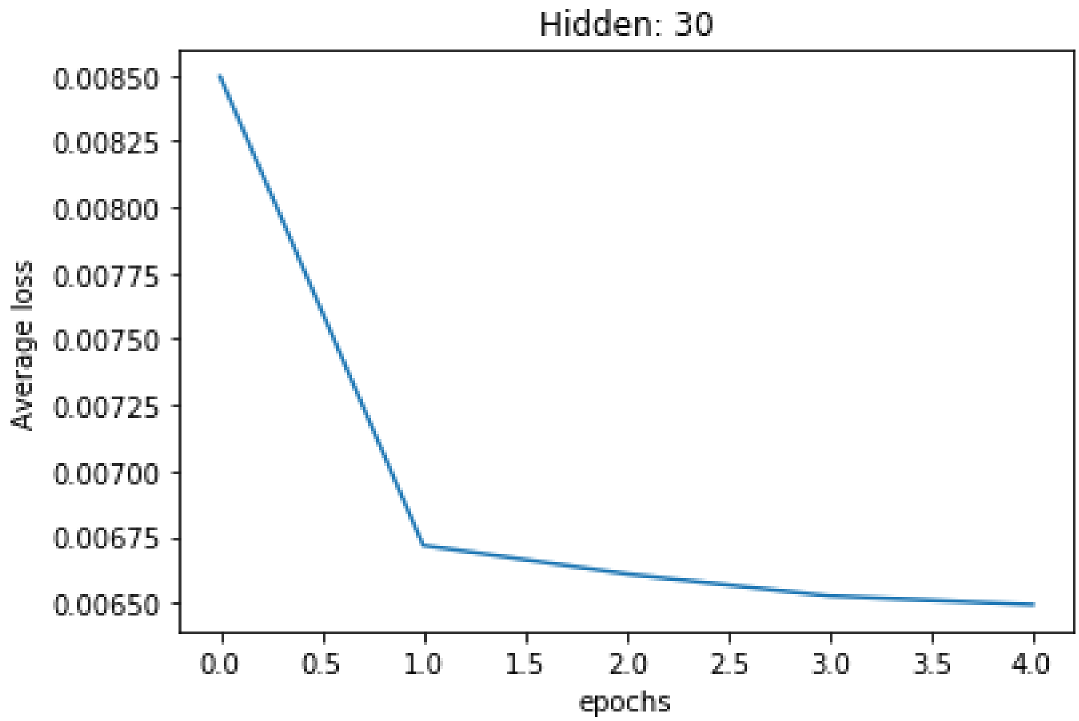
[15]:
```
loss_list = []

for hidden_size in hidden_sizes_lstm:
    epochTrace = np.load('{}/{}/{}/epochTrace_{}_{}_{}_{}.npy'.format(
                                                "/media/arjun/Shared/chaos/
 ↪output_files_v2/lstm",
                                                                    trace_name,␣
 ↪jvm, 40,
                                                                    1,␣
 ↪hidden_size, 1))
    plt.title("Hidden: {}".format(hidden_size))
    plt.plot(epochTrace)
    plt.xlabel("epochs")
    plt.ylabel("Average loss")
    plt.show()

    print("min: {}".format(np.min(epochTrace)))

    if abs(epochTrace[-2] - epochTrace[-1]) < 6e-5 and epochTrace[-2] -␣
 ↪epochTrace[-1] >= 0:
        print("converged")
        print(epochTrace[-2] - epochTrace[-1])
    else:
        print("not converged")
        print(epochTrace[-2] - epochTrace[-1])
```



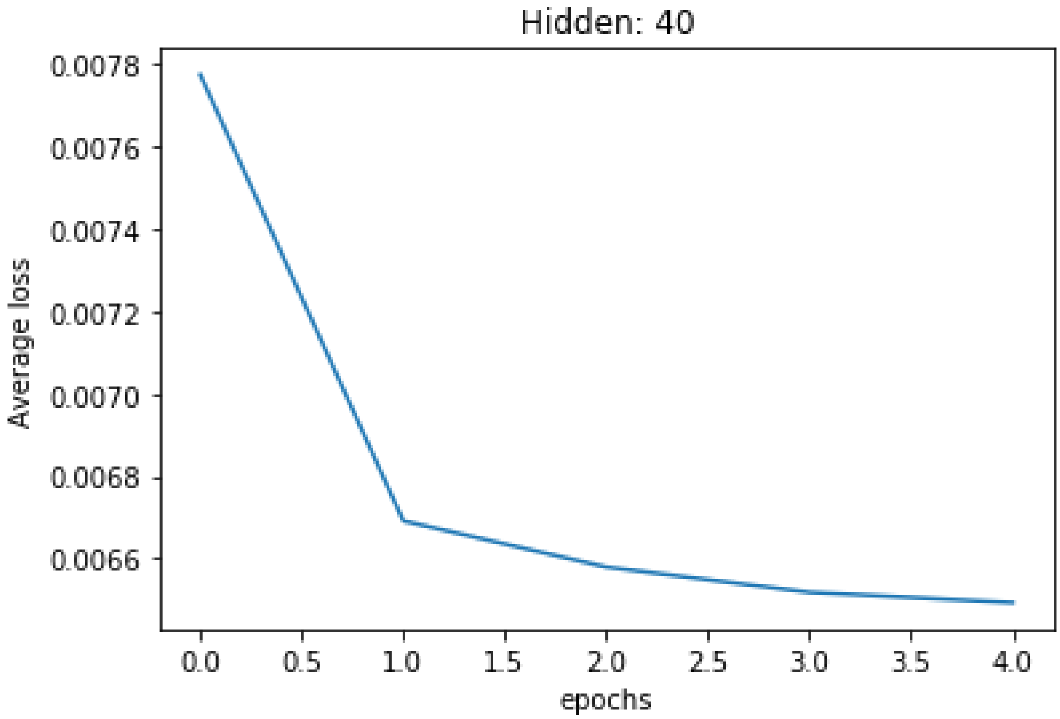
```
min: 0.006612486667779027
not converged
6.267511236424366e-05
```



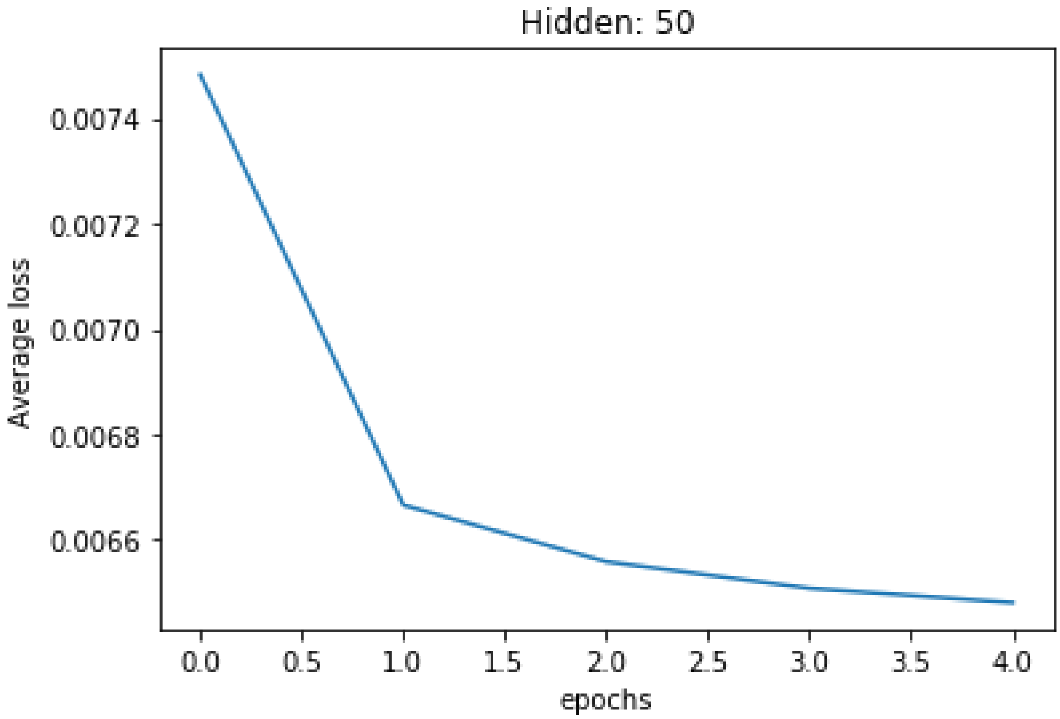

```
min: 0.006569464528743102
not converged
7.331176984066816e-05
```



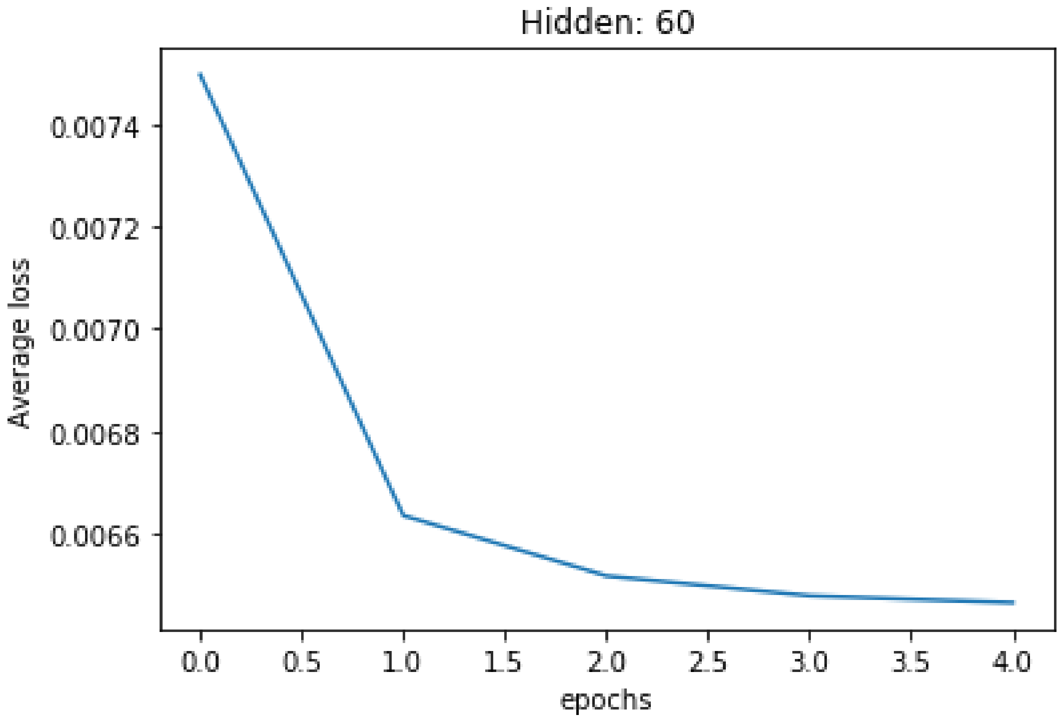

```
min: 0.006493579888526274
converged
3.288901460414117e-05
```



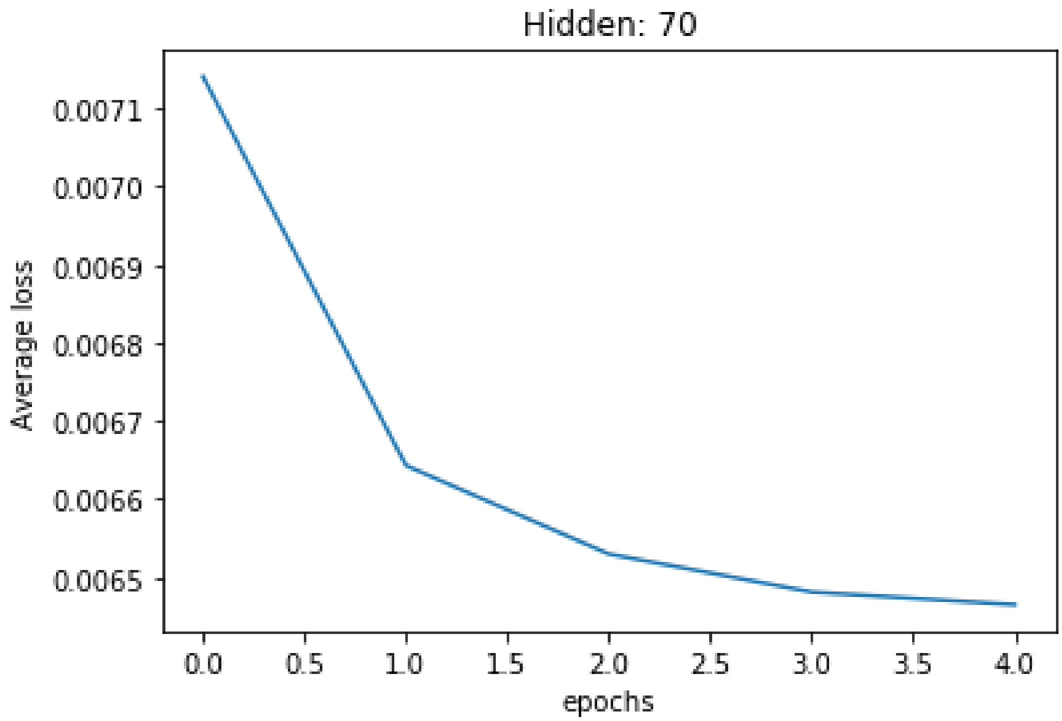

```
min: 0.0064923272090298785
converged
2.5082025905044658e-05
```



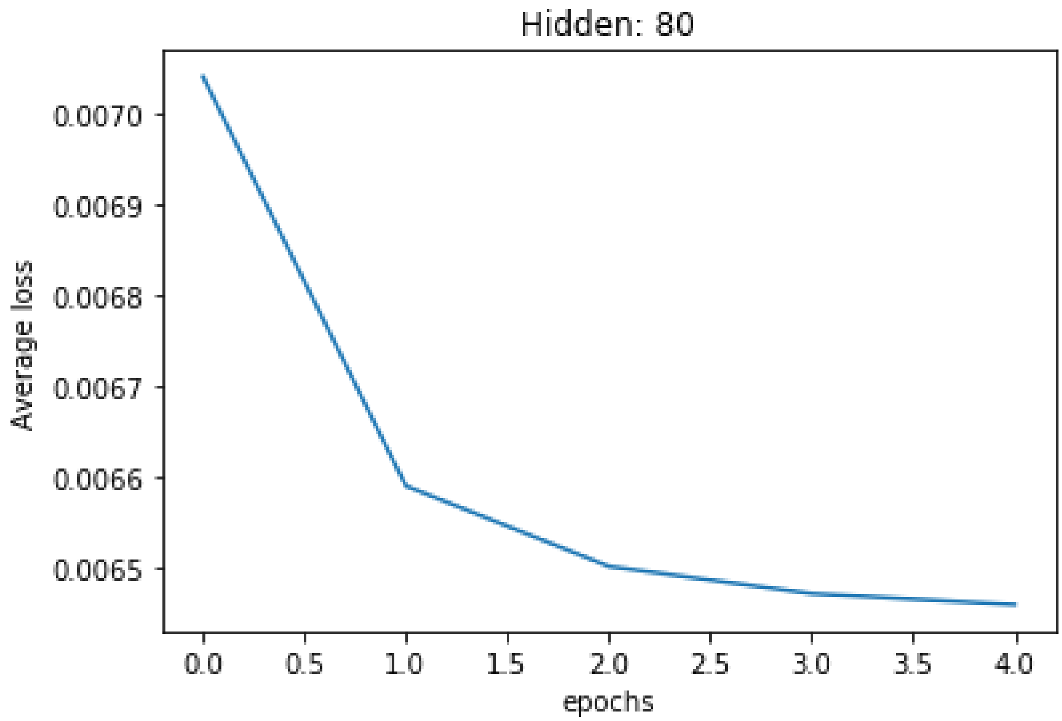

```
min: 0.006481095797553353
converged
2.7186418370326454e-05
```



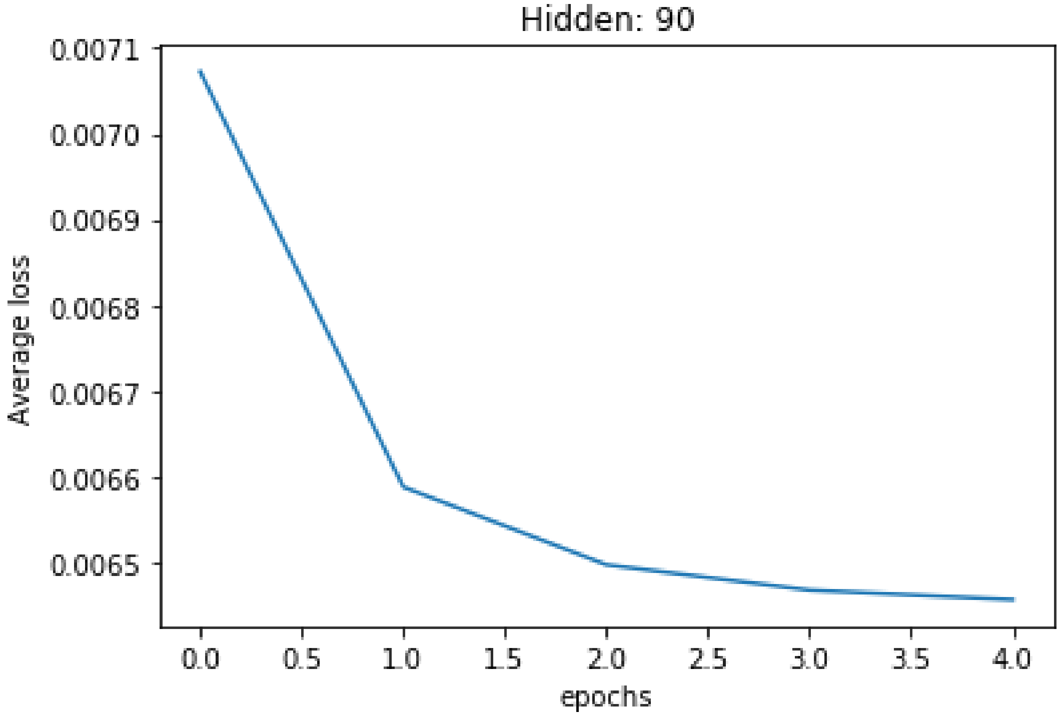

```
min: 0.006465050496921247
converged
1.3474917229341389e-05
```



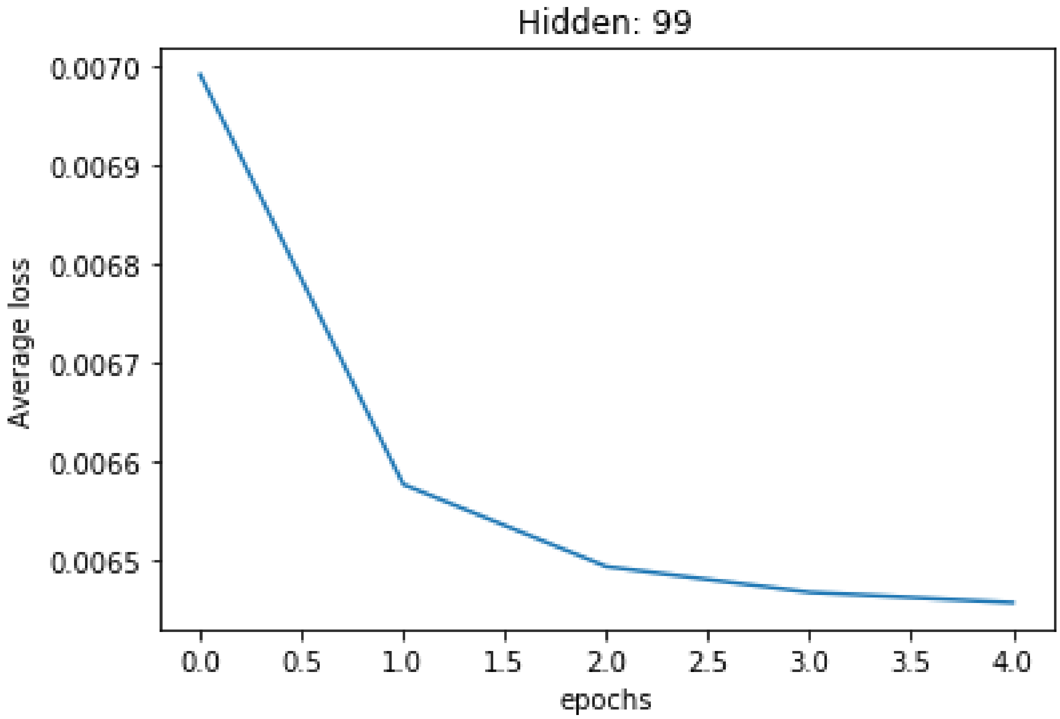

```
min: 0.006465155226539592
converged
1.6156085899898115e-05
```



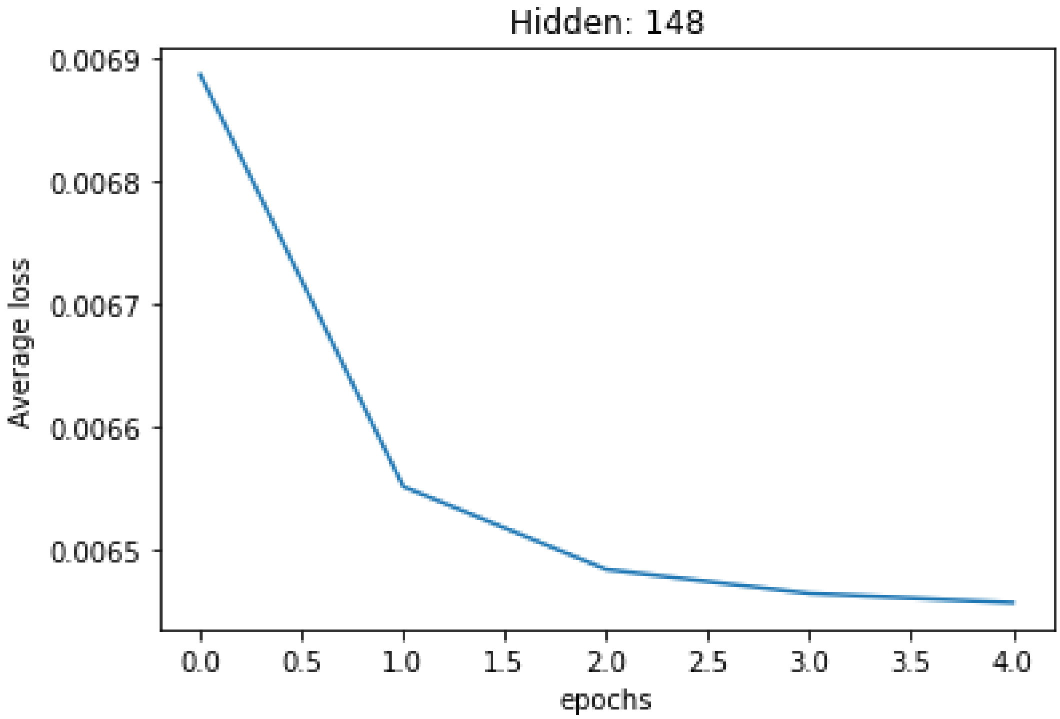

```
min: 0.006458282441813119
converged
1.2166927663646507e-05
```



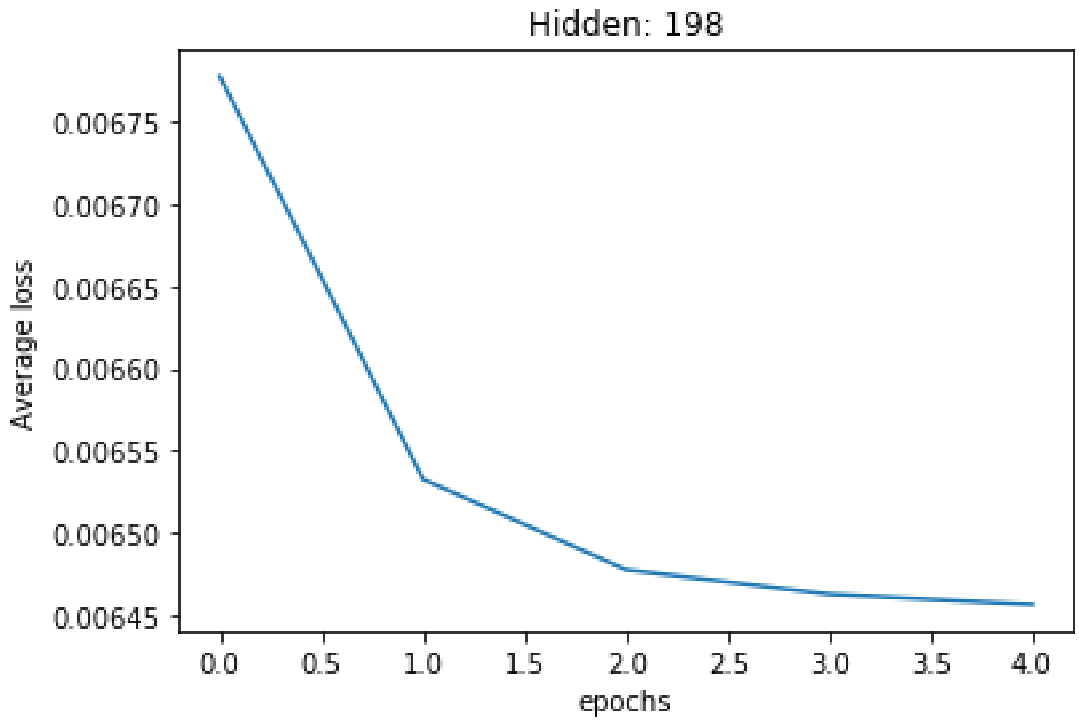

```
min: 0.006457146179311131
converged
1.111758393900538e-05
```



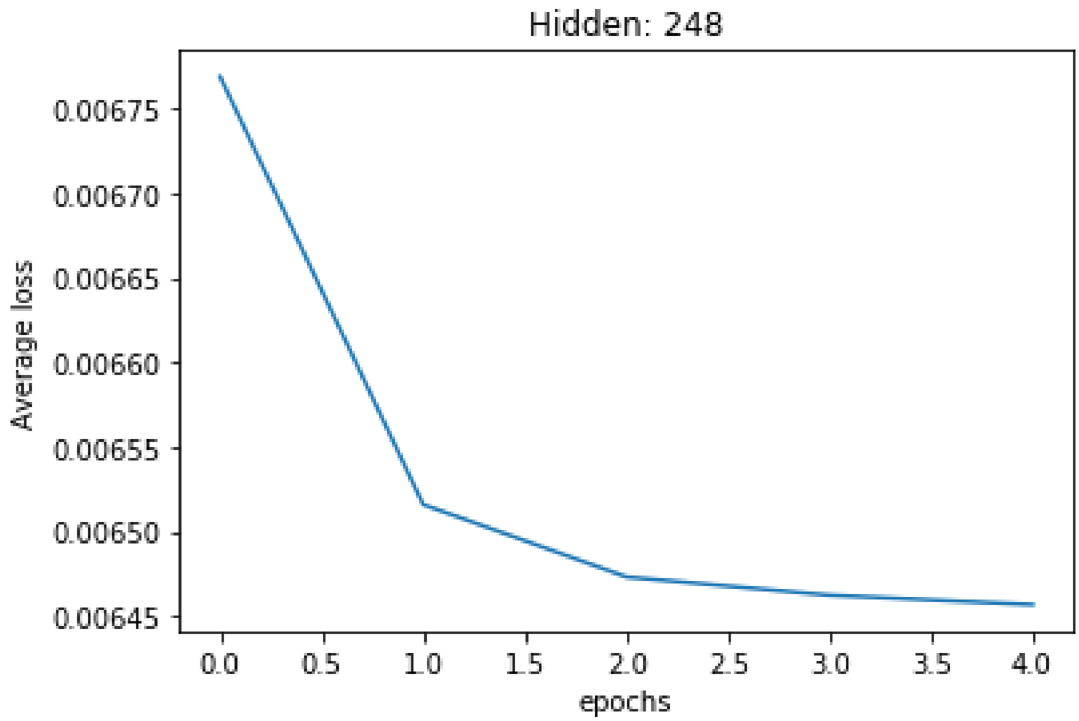

```
min: 0.006456136096801077
converged
1.0472842929314108e-05
```



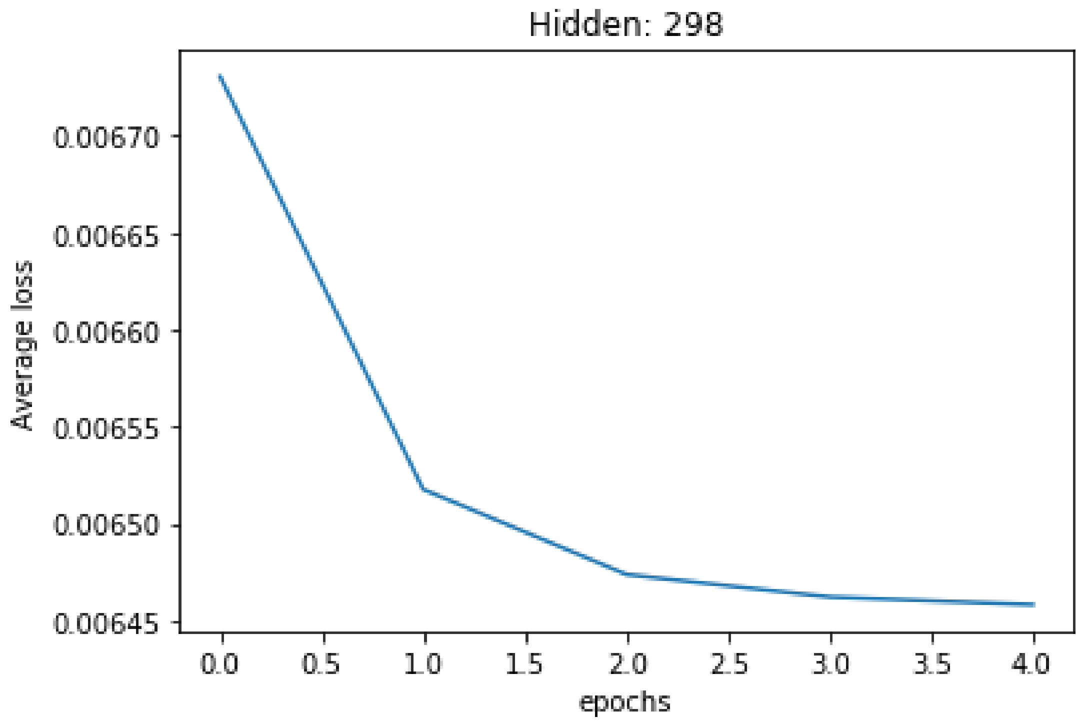

```
min: 0.006456979713573748
converged
7.456641416160072e-06
```



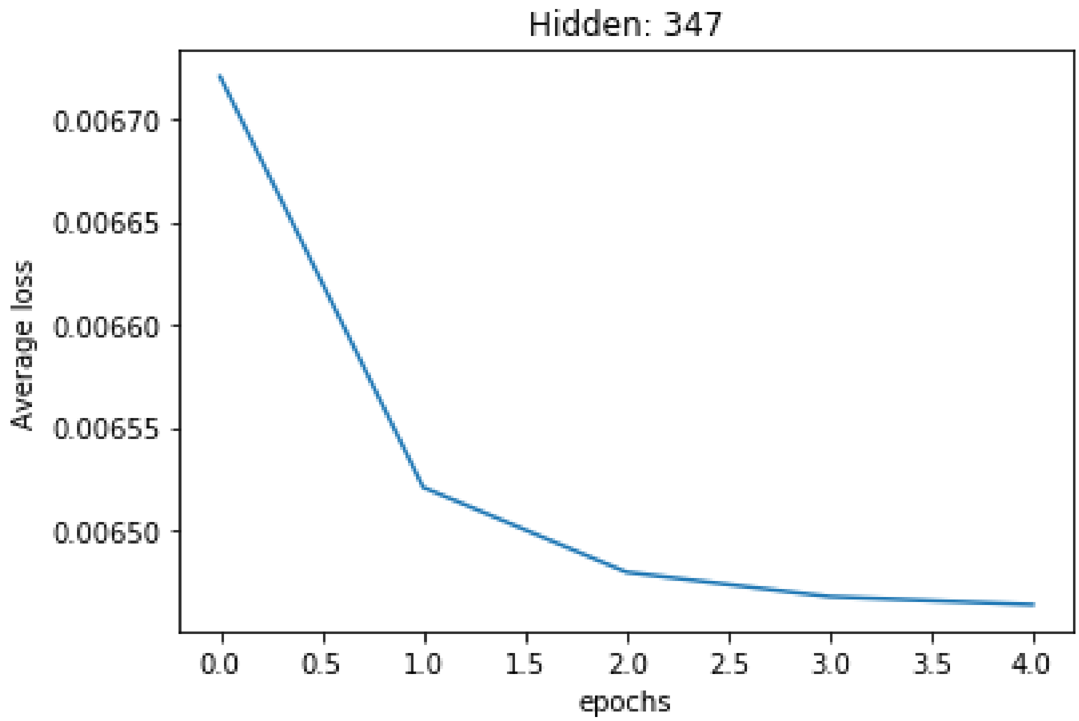

```
min: 0.006456978880325142
converged
6.24786895148631e-06
```



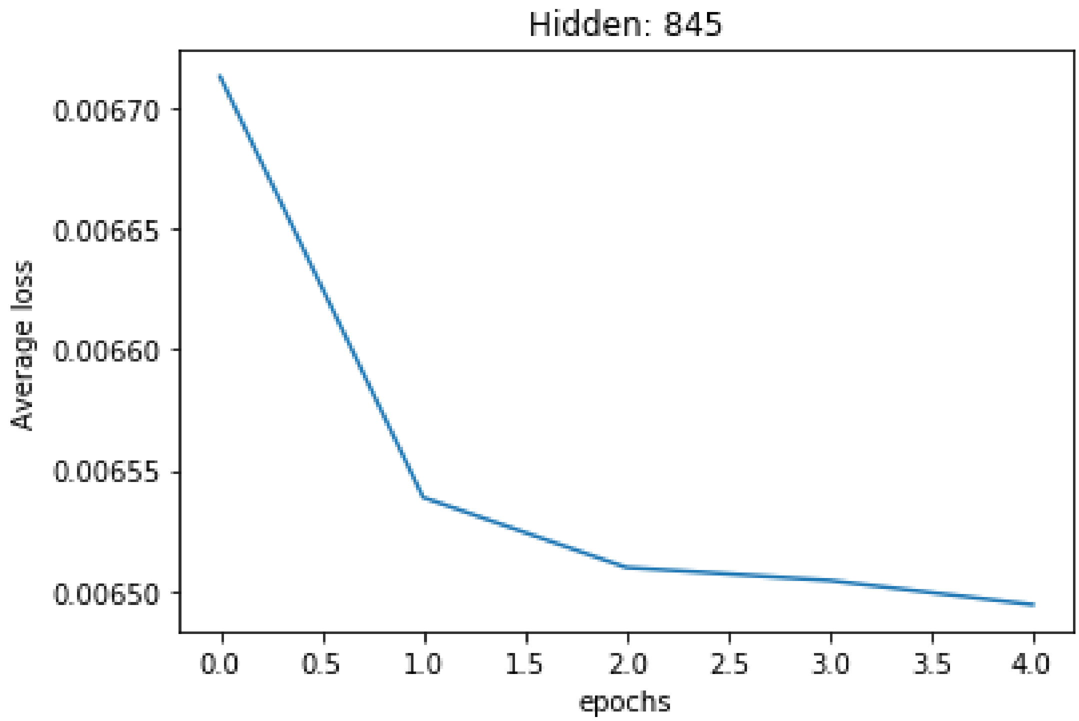

```
min: 0.006456849292832978
converged
5.5800813193226245e-06
```



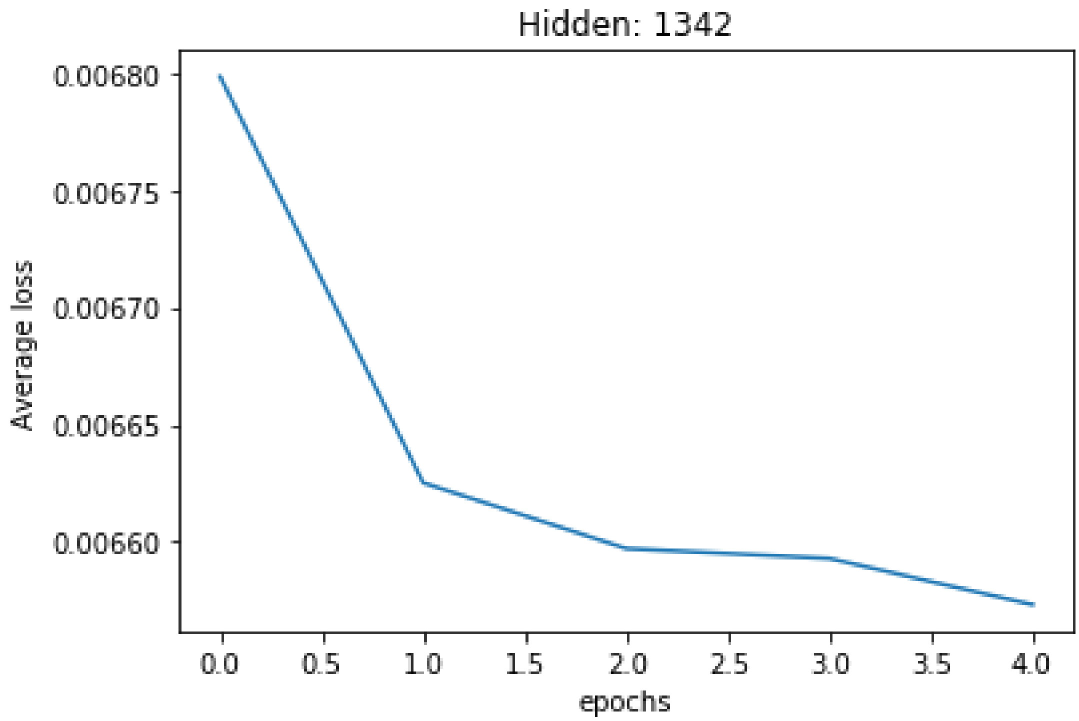

```
min: 0.006458304793250805
converged
3.9624231202252044e-06
```



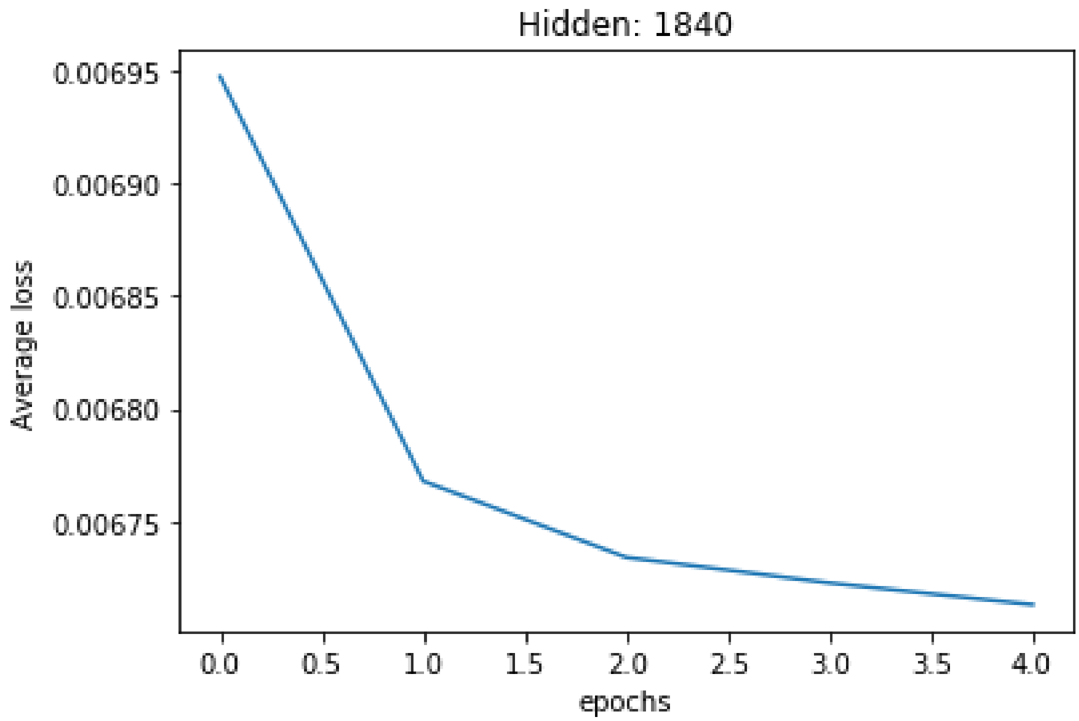

```
min: 0.006464056734831966
converged
3.880749551616464e-06
```



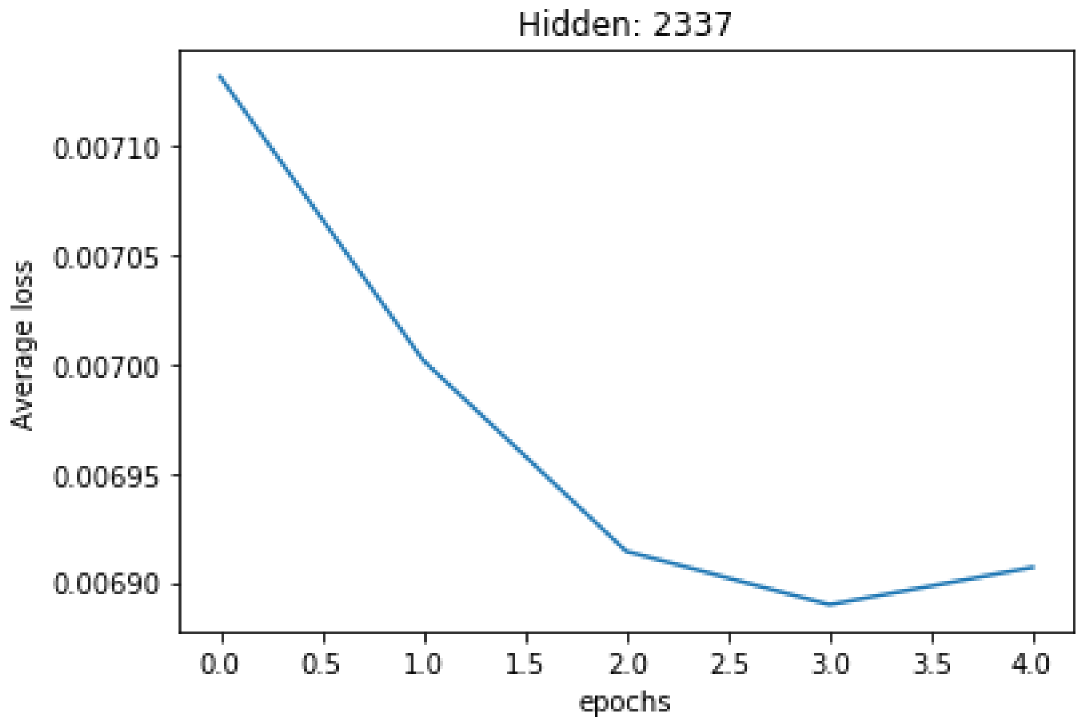

```
min: 0.006494224295628314
converged
1.0023239619877389e-05
```



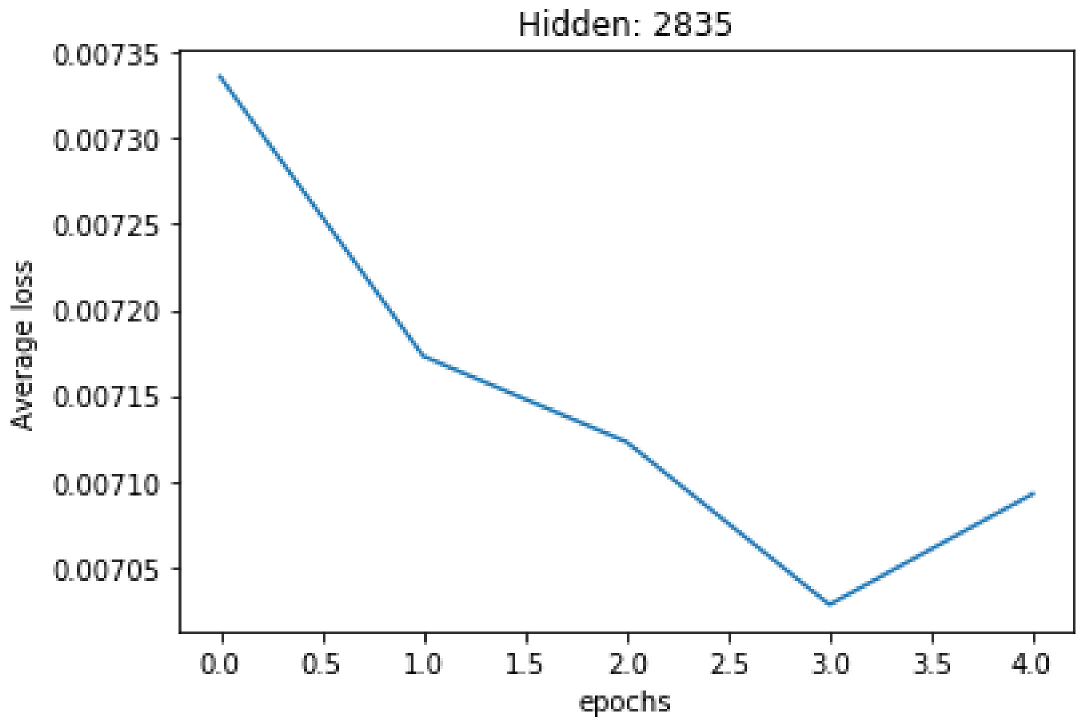

```
min: 0.006573242538133446
converged
1.965574038271991e-05
```



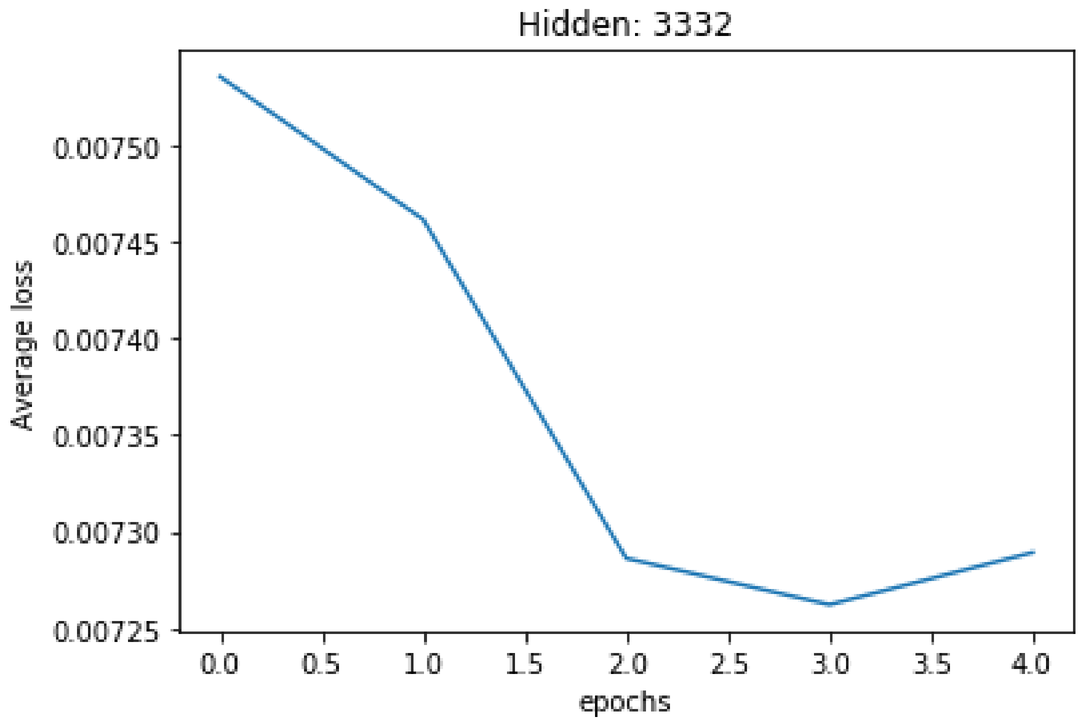

```
min: 0.006713036989375036
converged
9.63672387356649e-06
```



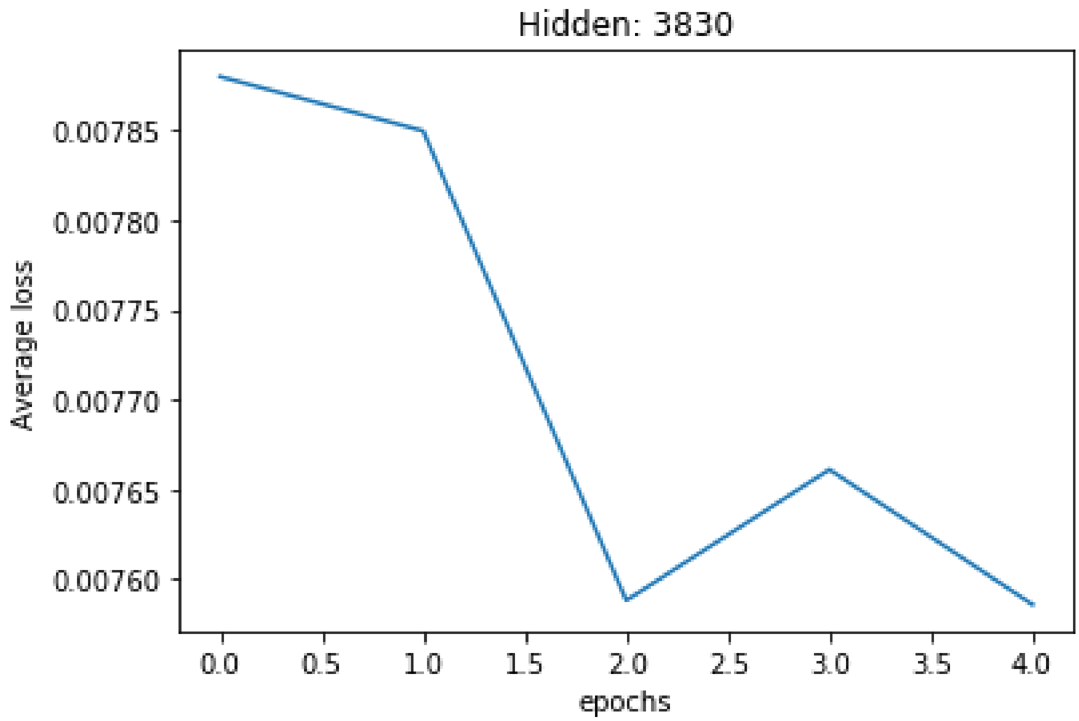

```
min: 0.006889887560995257
not converged
-1.7004675402933385e-05
```



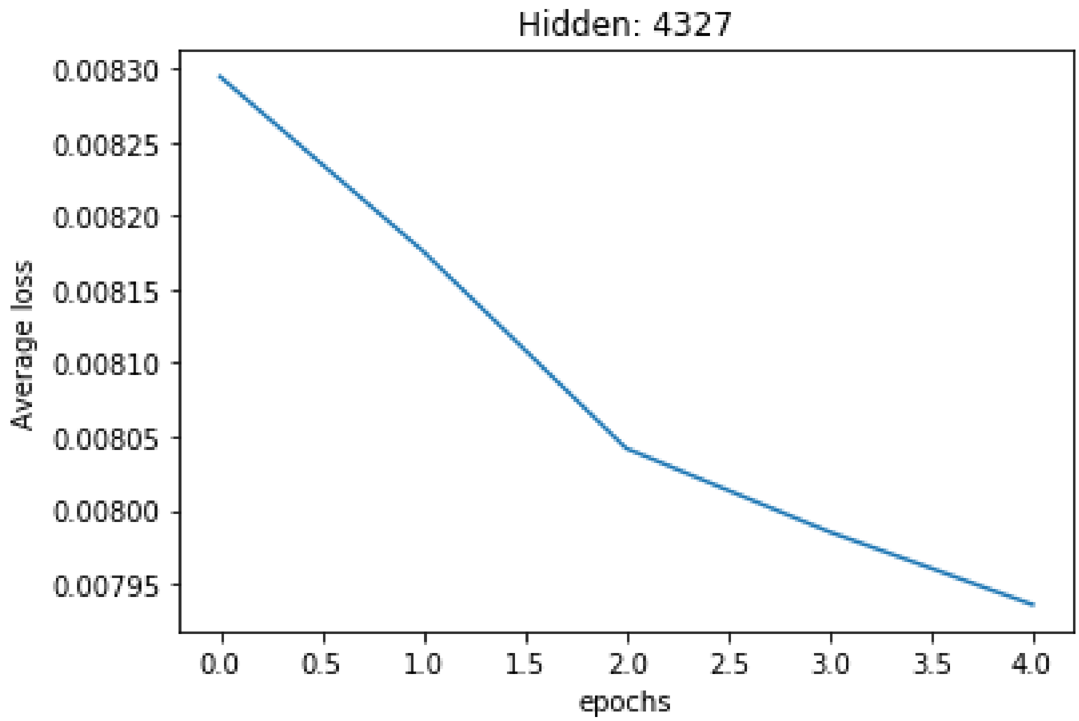

```
min: 0.007027996572912956
not converged
-6.452212680359282e-05
```



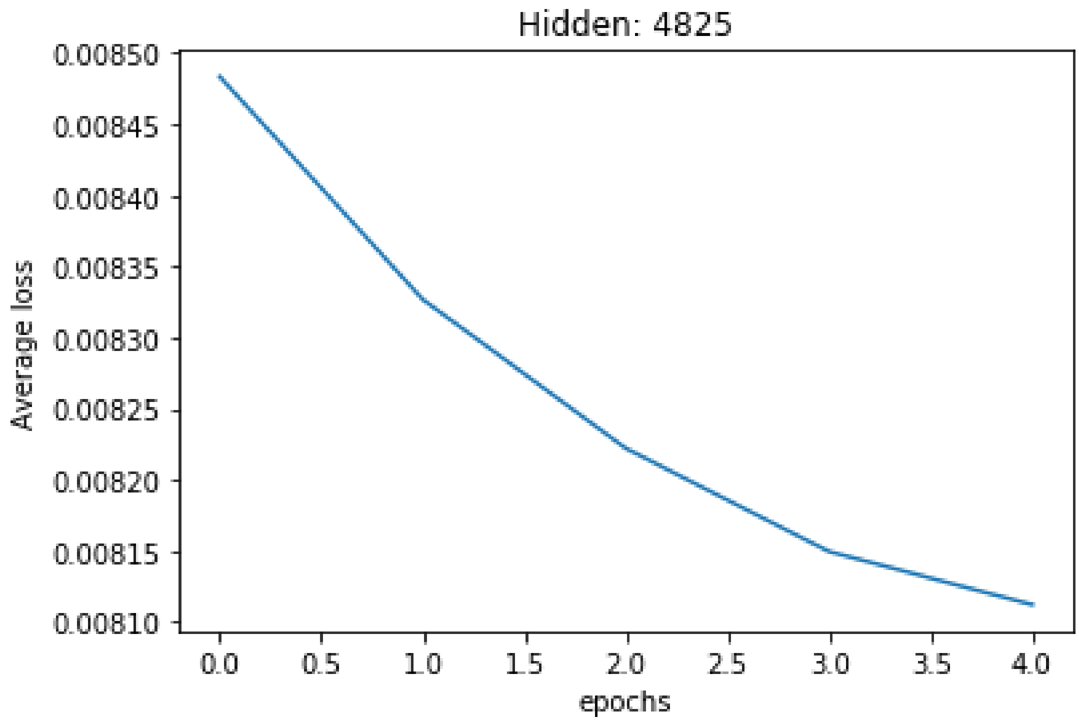

```
min: 0.0072623013324883525
not converged
-2.6917204869036232e-05
```




```
min: 0.007585708427490002
not converged
7.532115189396761e-05
```




```
min: 0.00793586202482788
converged
4.93777318268402e-05
```

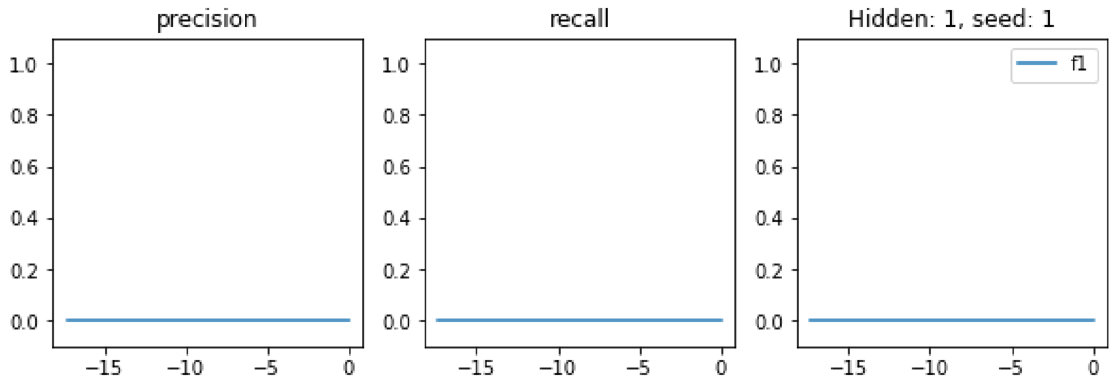



```
min: 0.008112132767329411
converged
3.725490886337911e-05
```

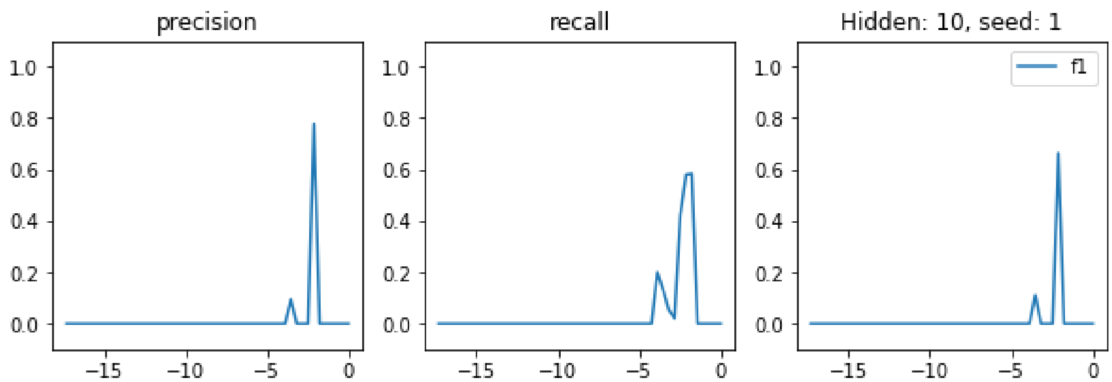
0.1 FCN

[16]:
```
hidden_sizes = hidden_sizes_fcn

loss_main["fcn"] = get_loss_dict1(trace, "/media/arjun/Shared/chaos/
↪output_files_v2/fcn",
                                  hidden_sizes=hidden_sizes, plot_graphs=True)
```

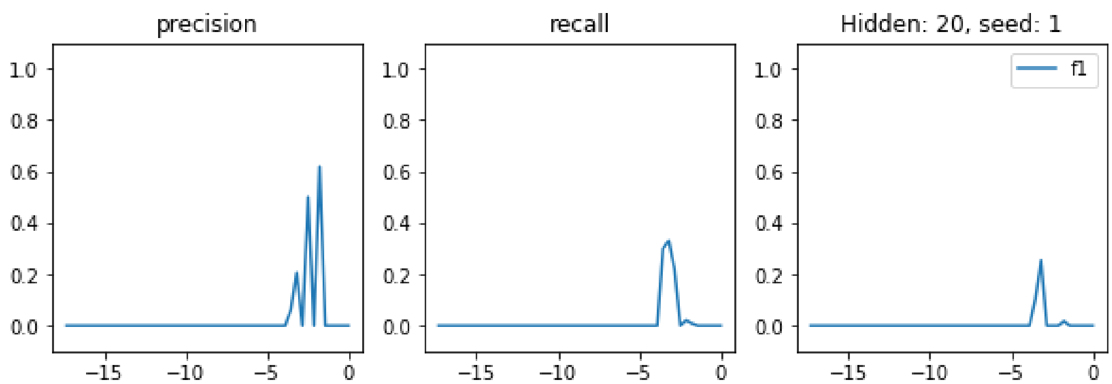
```
relevance threshold: 0.3969525474770118
hidden_size: 1
```



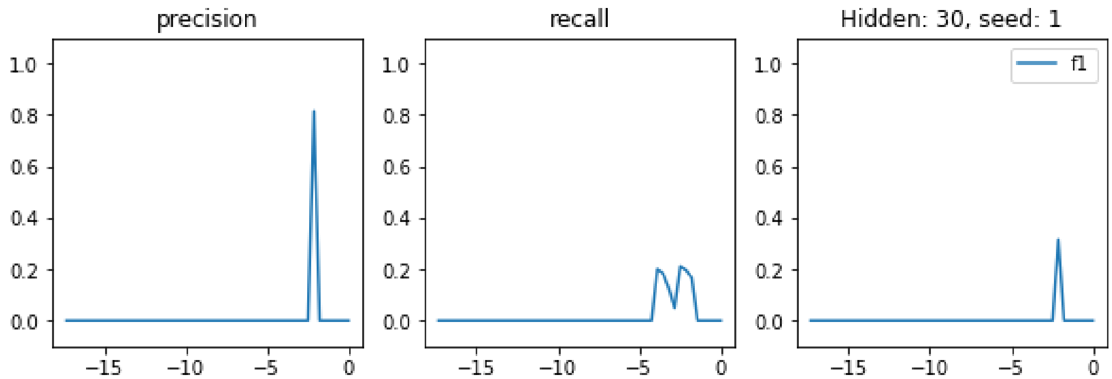

```
hidden_size: 10
```



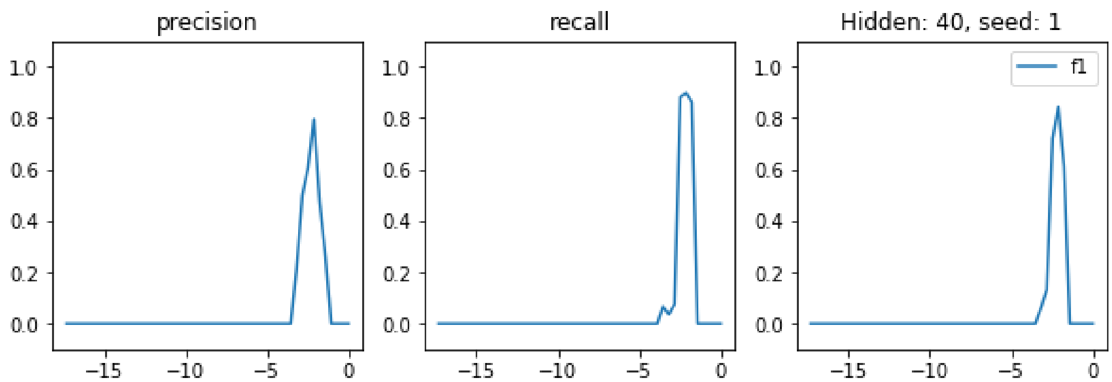

```
hidden_size: 20
```



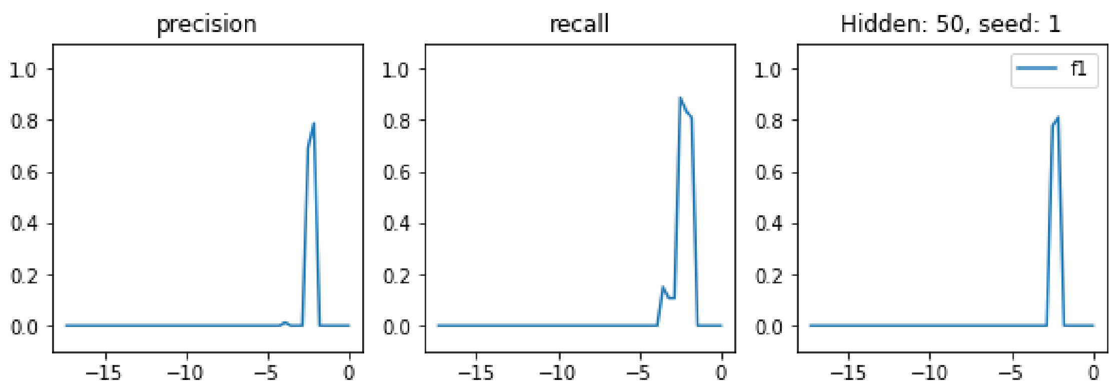

```
hidden_size: 30
```



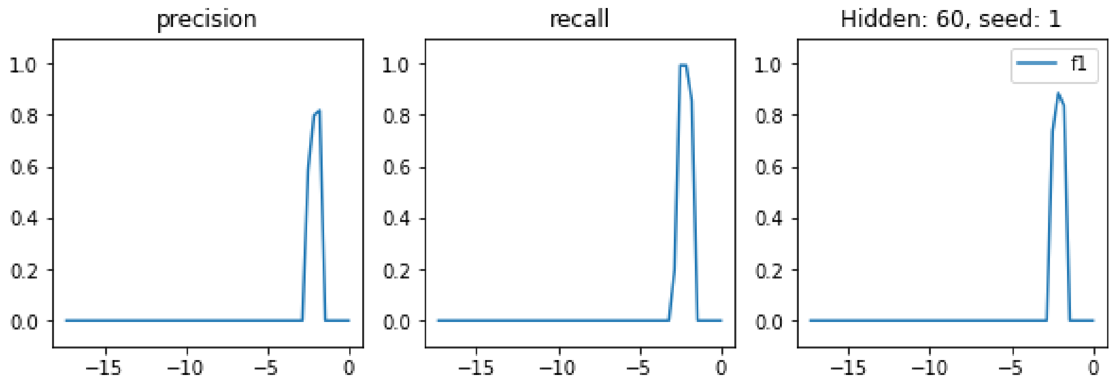
hidden_size: 40



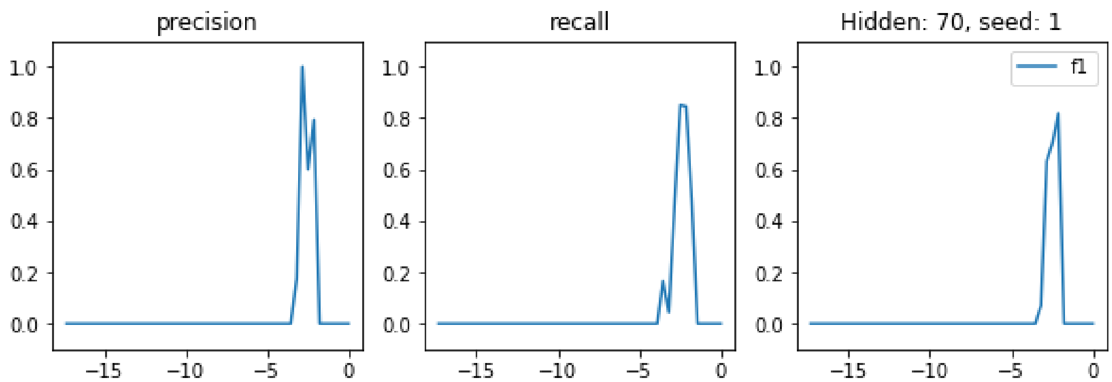
hidden_size: 50



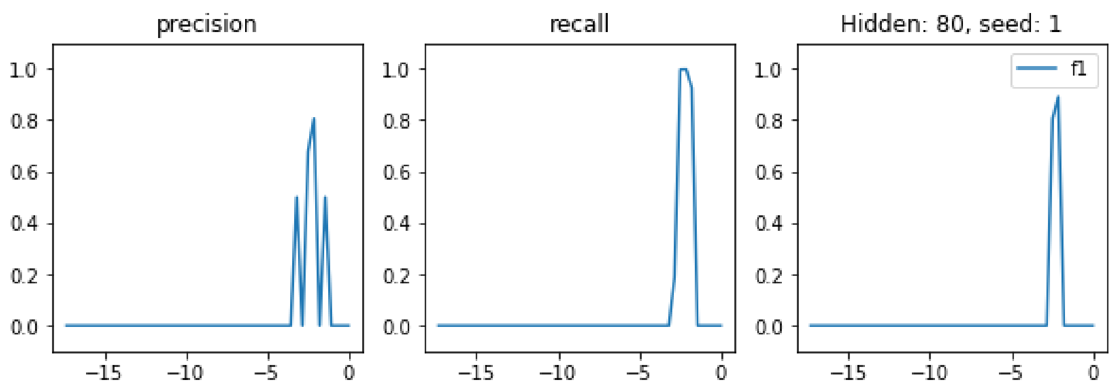
hidden_size: 60



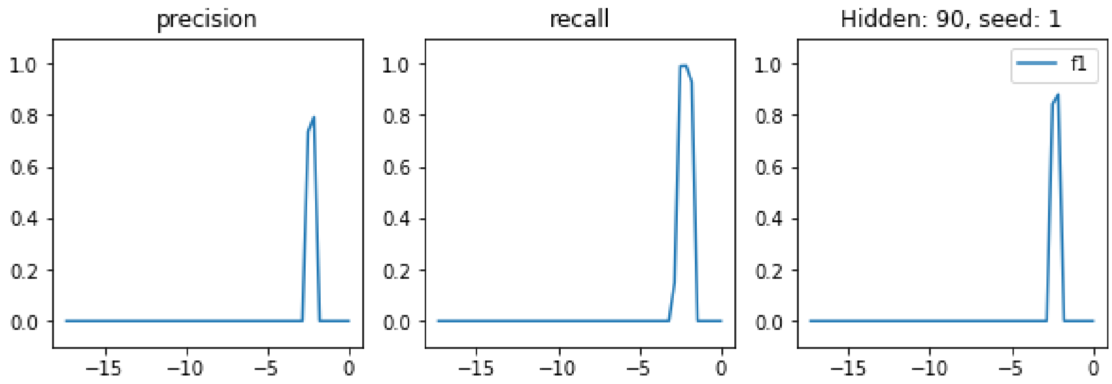
hidden_size: 70



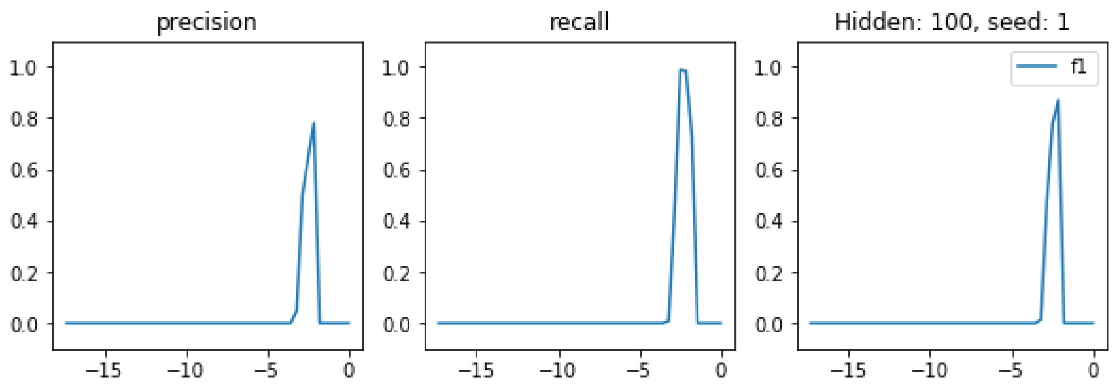
hidden_size: 80



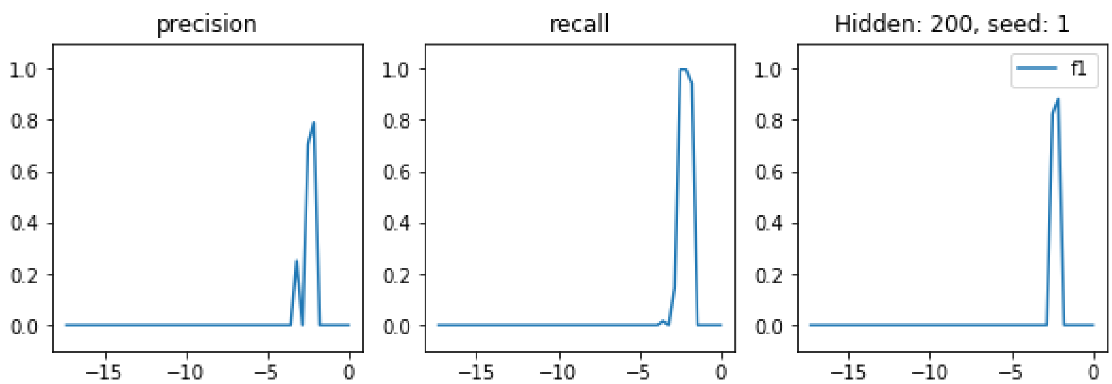
hidden_size: 90



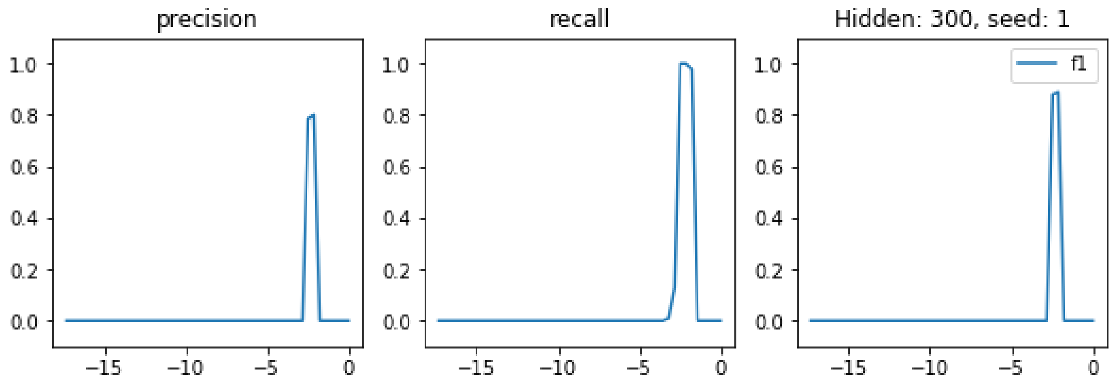
hidden_size: 100



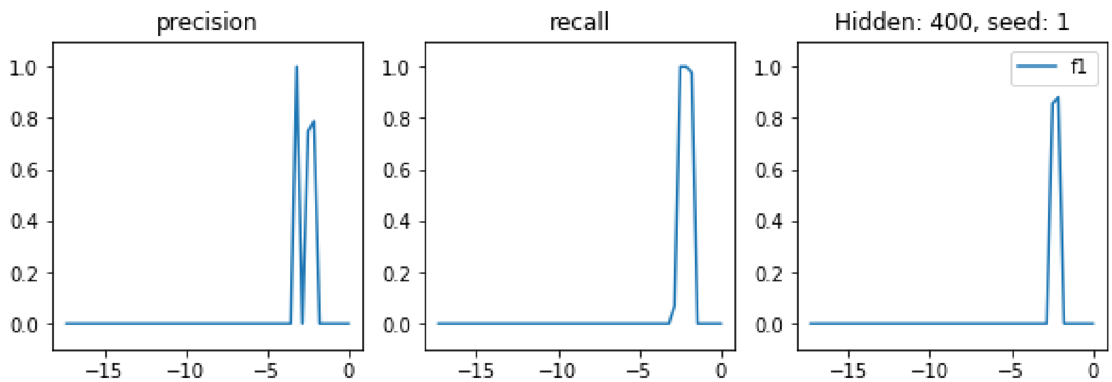
hidden_size: 200



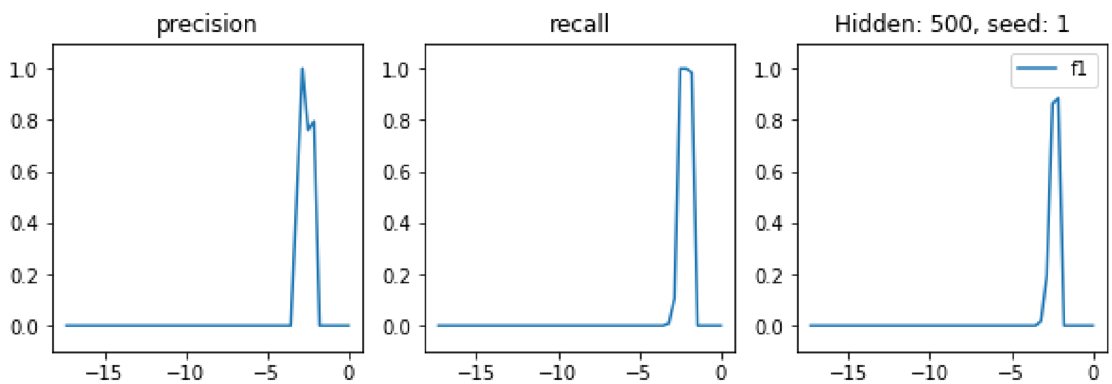
hidden_size: 300



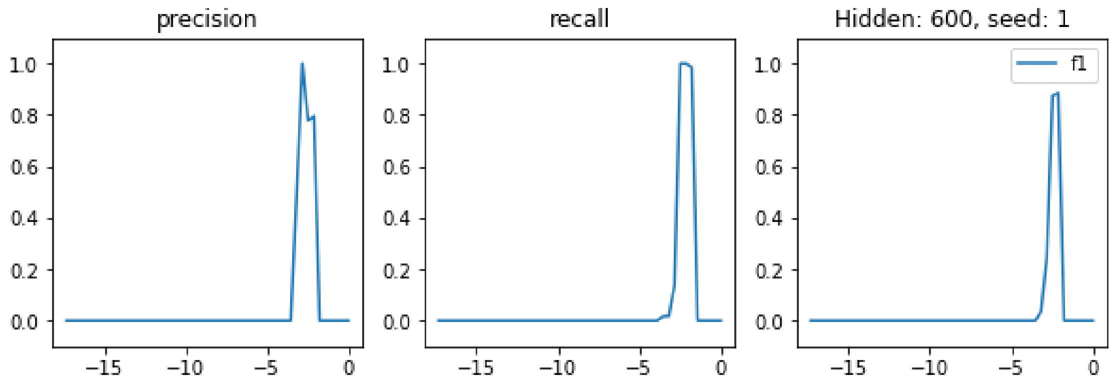
hidden_size: 400



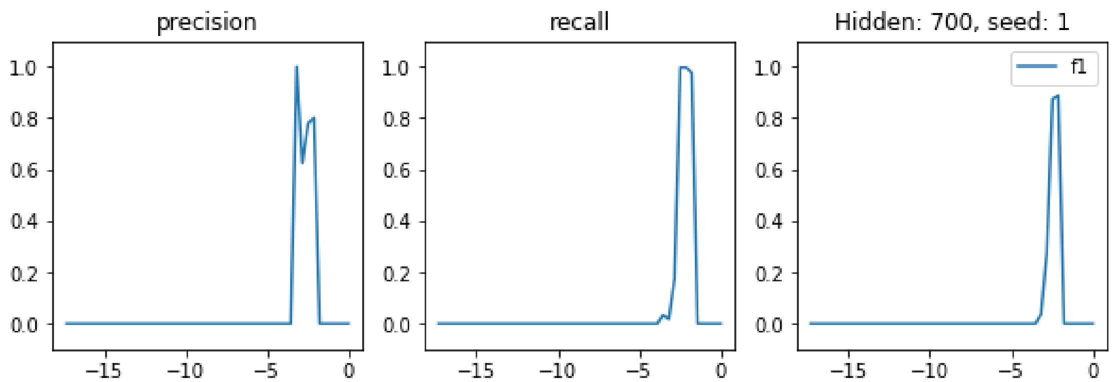
hidden_size: 500



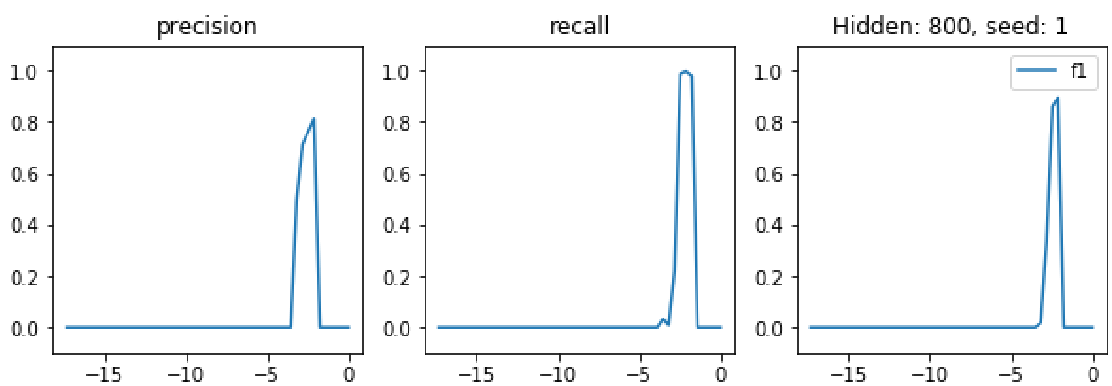
hidden_size: 600



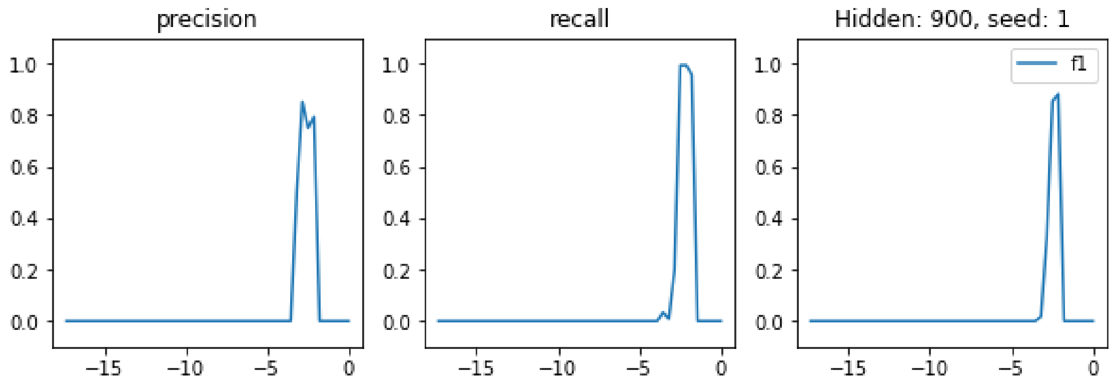
hidden_size: 700



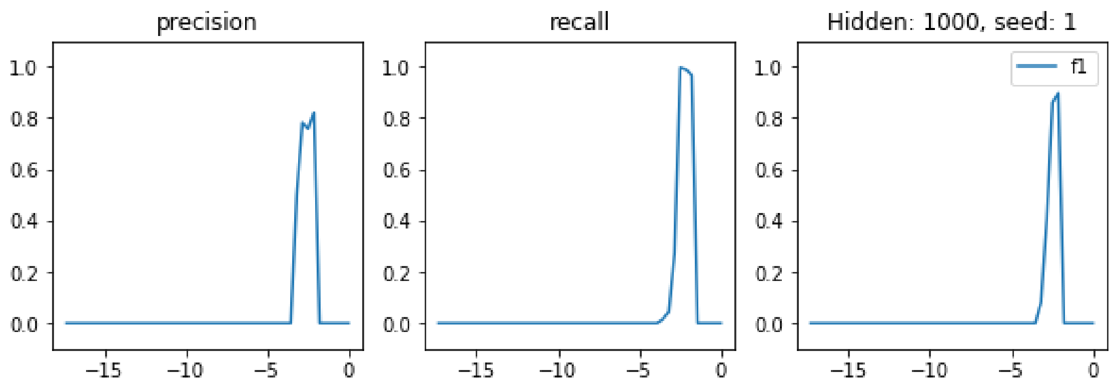
hidden_size: 800



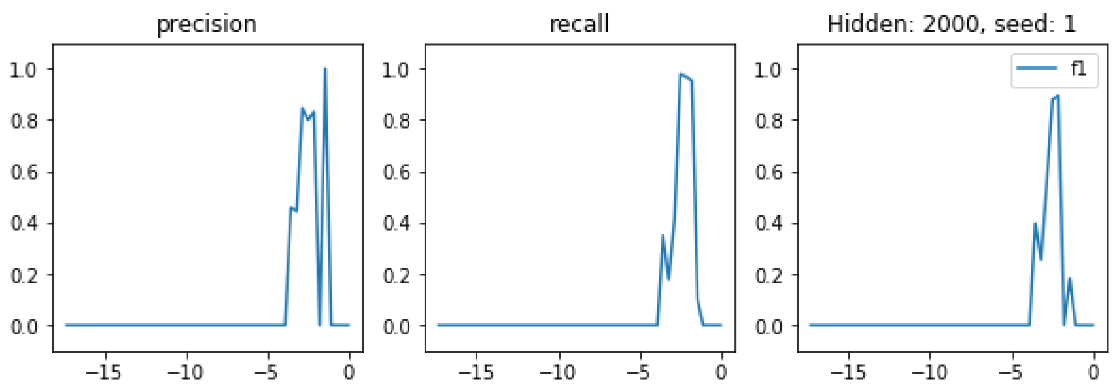
hidden_size: 900



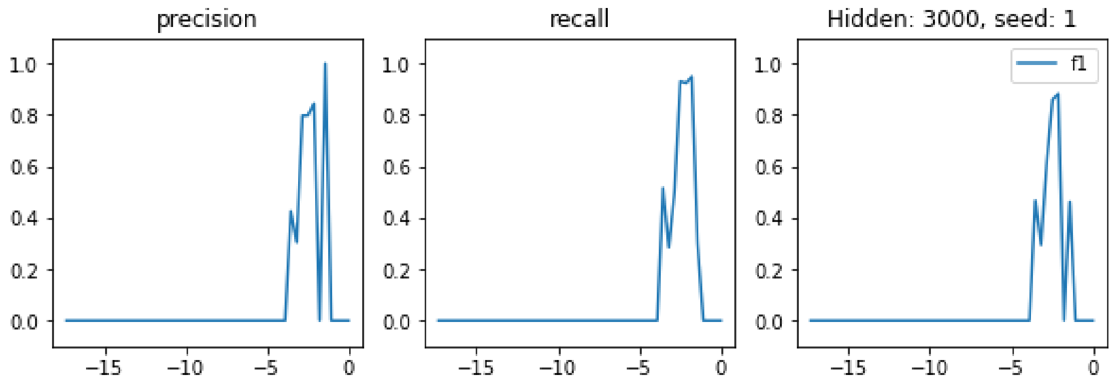

```
hidden_size: 1000
```



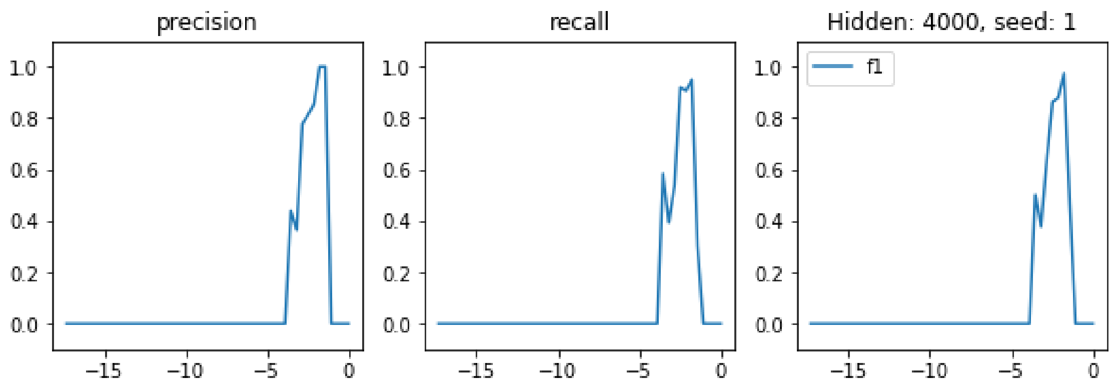

```
hidden_size: 2000
```



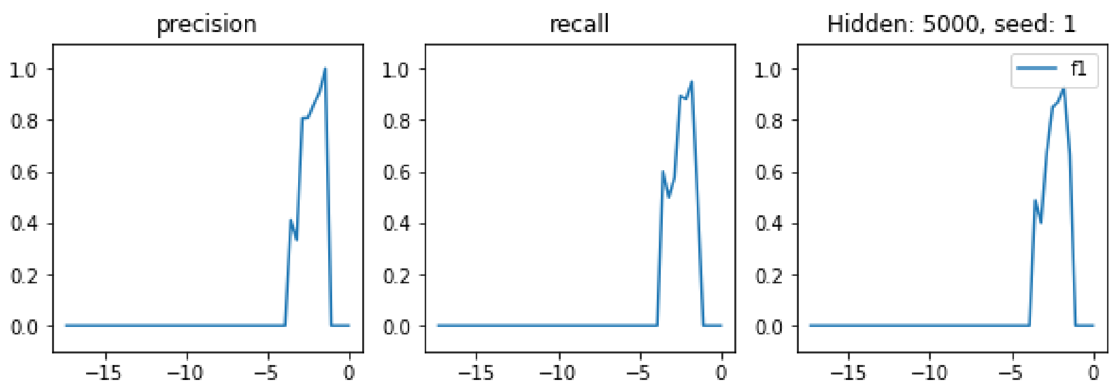

```
hidden_size: 3000
```



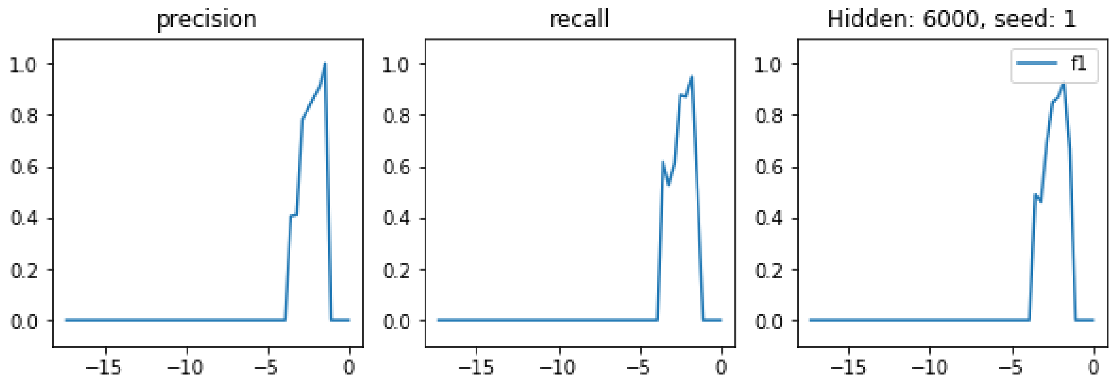
hidden_size: 4000



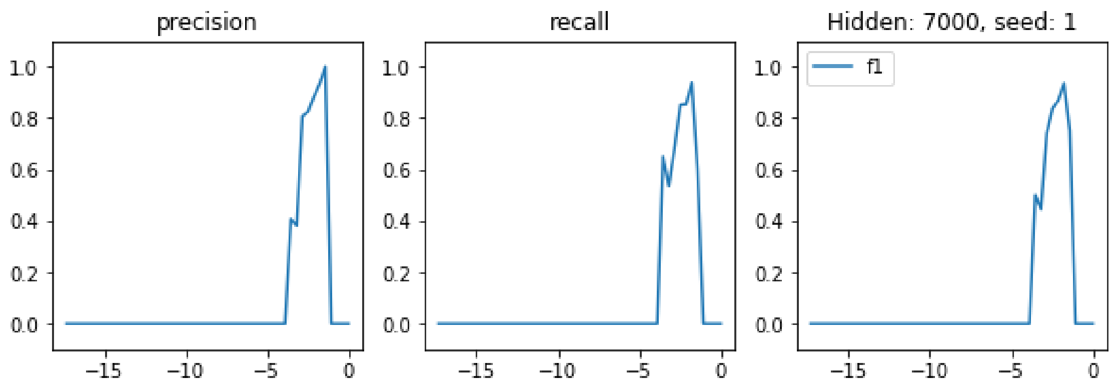
hidden_size: 5000



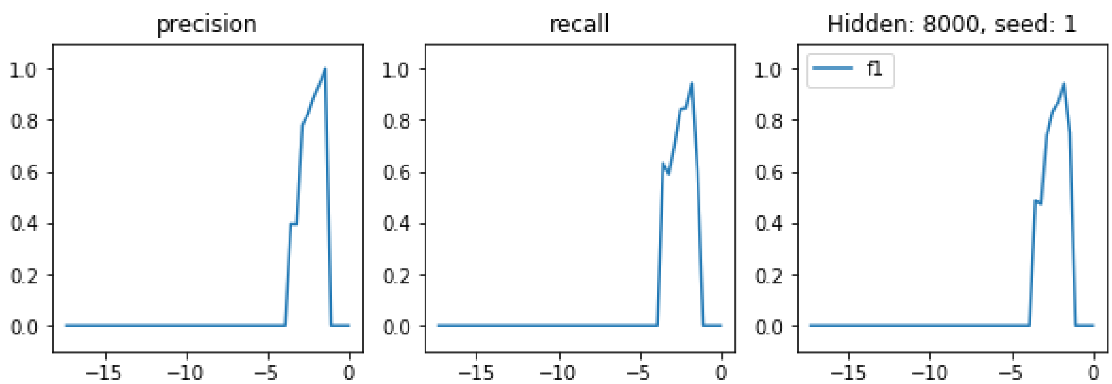
hidden_size: 6000




```
hidden_size: 7000
```




```
hidden_size: 8000
```



[17]:
```
actual = []
predicted = []

for hidden_size in hidden_sizes_lstm:
    actual = [ val[0] for val in data ]
    predicted = [ val[1] for val in data ]

    data = get_actual_predicted(trace, trace_name, jvm,␣
 ↪hidden_size=hidden_sizes_fcn[2], history_size=40,
                              model_type="lstm", output_file_location="/media/
 ↪arjun/Shared/chaos/output_files_v2/lstm")

    plt.plot(actual, label="actual")
    plt.plot(predicted, label="predicted", alpha=0.8)
    plt.title("hidden_size: {}".format(hidden_size))
#     plt.xlim((1000, 1020))
    plt.legend()
    plt.show()
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-17-069ea6e7b83c> in <module>
         3
         4 for hidden_size in hidden_sizes_lstm:
   ----> 5     actual = [ val[0] for val in data ]
         6     predicted = [ val[1] for val in data ]
         7


       NameError: name 'data' is not defined
```

[ ]:
```
rademacher_list_fcn = []

for hidden_size in hidden_sizes_fcn:
    loss_array = loss_main["fcn"][hidden_size]
    rademacher_list_fcn.append(get_rademacher(loss_array))
```

[ ]:
```
plt.plot(hidden_sizes_fcn, rademacher_list_fcn, label="fcn")
plt.legend()
```

[ ]:
```
loss_list = []

for hidden_size in hidden_sizes_fcn:
    epochTrace = np.load('{}/{}/{}/epochTrace_{}_{}_{}_{}.npy'.format(
                                                    "/media/arjun/Shared/chaos/
 ↪output_files_v2/fcn",
                                                                    trace_name,␣
 ↪jvm, 40,
                                                         1, hidden_size, 1))

    plt.title("Hidden: {}".format(hidden_size))
    plt.plot(epochTrace)
    plt.show()

    if epochTrace[-2] - epochTrace[-1] < 1e-3:
        print("converged")
```

[ ]:
```
actual = []
predicted = []


for hidden_size in hidden_sizes_fcn:
    data = get_actual_predicted(trace, trace_name, jvm,␣
 ↪hidden_size=hidden_sizes_fcn[2], history_size=40,
                              model_type="fcn", output_file_location="/media/
 ↪arjun/Shared/chaos/output_files_v2/fcn")

    actual = [ val[0] for val in data ]
    predicted = [ val[1] for val in data ]

    plt.plot(actual, label="actual")
    plt.plot(predicted, label="predicted", alpha=0.8)
    plt.title("hidden_size: {}".format(hidden_size))
#     plt.xlim((1000, 1020))
    plt.legend()
    plt.show()
```