Rademacher_F1

June 12, 2020

[596]:
```
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
```

[597]:
```
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"

    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):
#         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-1+idx, -1])

        data.append((actual, predicted))

    return data
```

[598]:
```
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=1e-2):
    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)
```

[599]:
```
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)
```

[600]:
```
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")
```

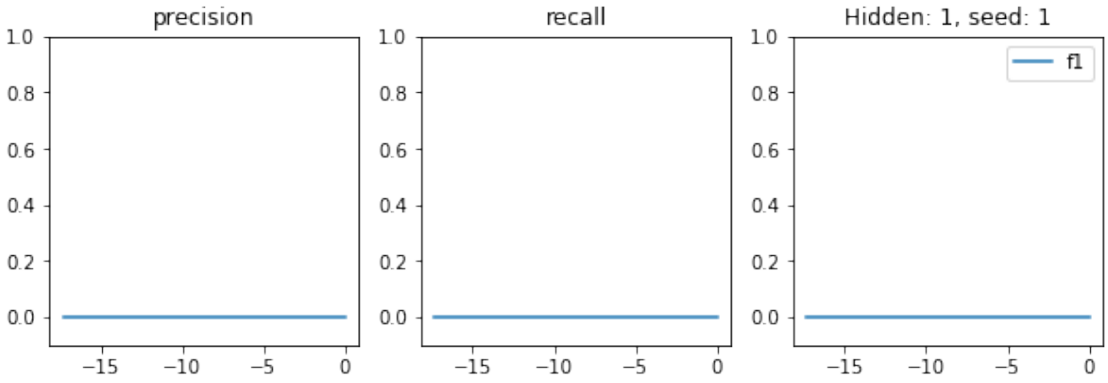
```
                    ax1.set_ylim((-0.1, 1))
                    ax2.set_ylim((-0.1, 1))
                    ax3.set_ylim((-0.1, 1))

                    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
```

[601]:
```
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]
```

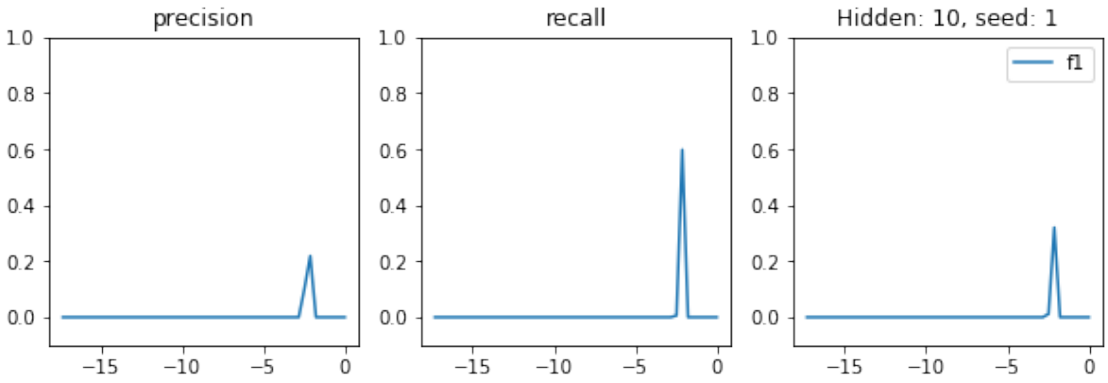
[602]:
```
loss_main = {}
hidden_sizes=[ 1, 10, 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000 ]
```

[603]:
```
loss_main["fcn"] = get_loss_dict1(trace, "/media/arjun/Shared/chaos/
 ↪output_files",
                                  hidden_sizes=hidden_sizes, plot_graphs=True)
```

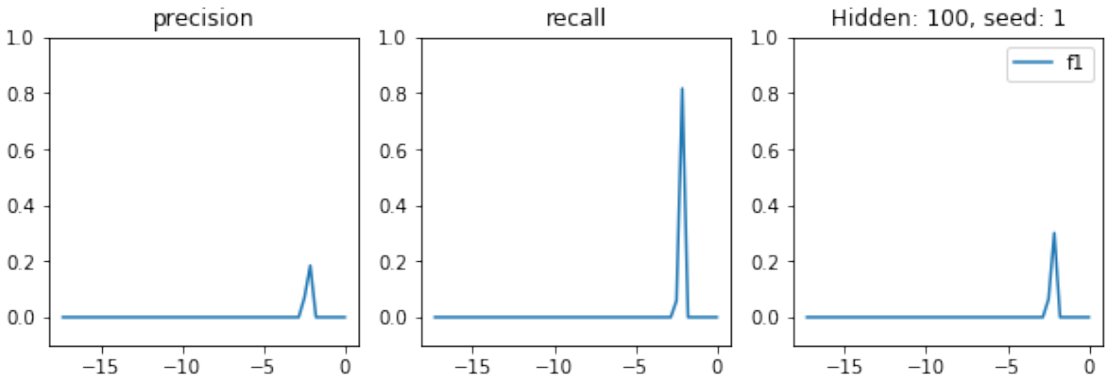
```
relevance threshold: 0.3969525474770118
hidden_size: 1
```



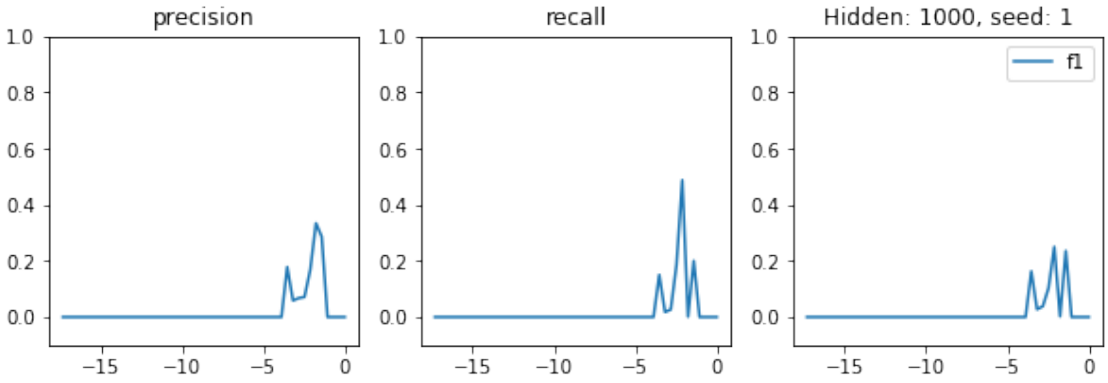
`hidden_size: 10`



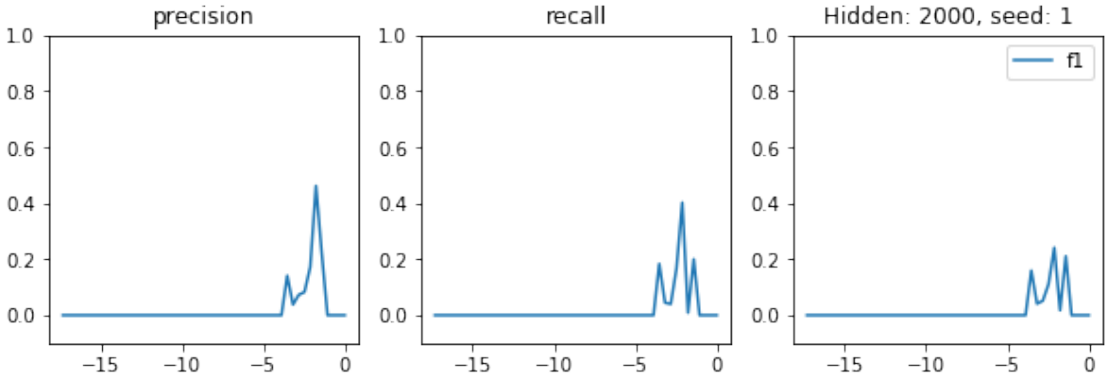
`hidden_size: 100`



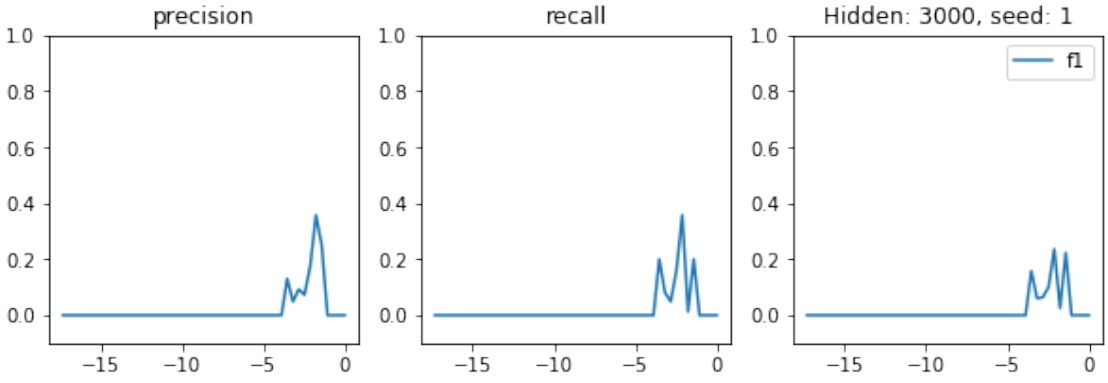
`hidden_size: 1000`



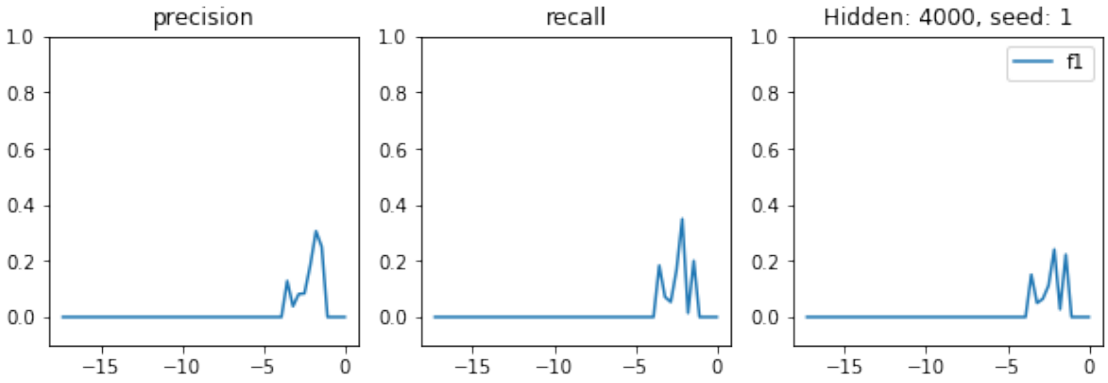
`hidden_size: 2000`



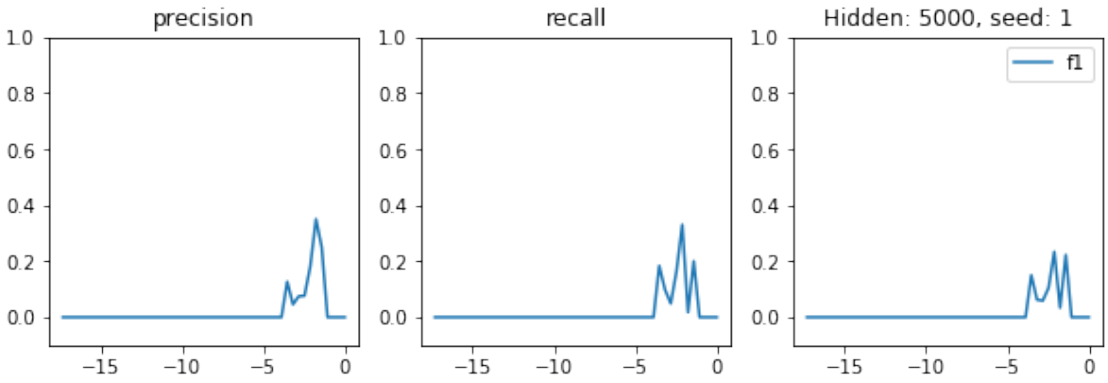
`hidden_size: 3000`



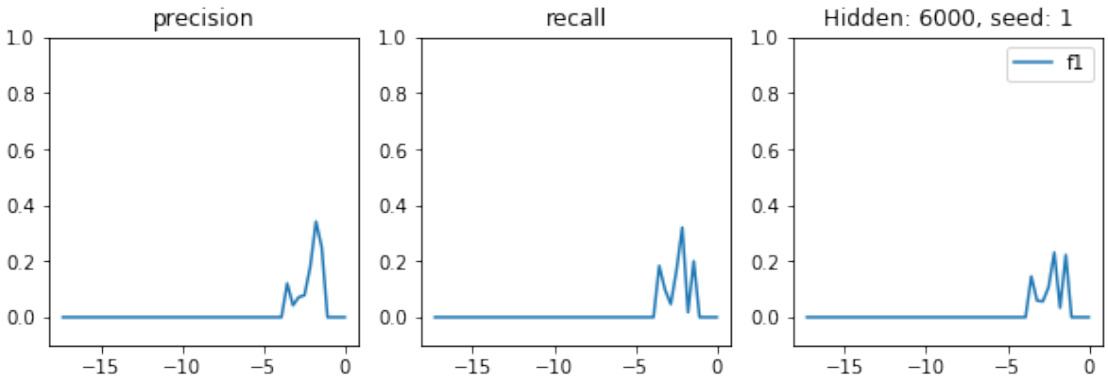
`hidden_size: 4000`



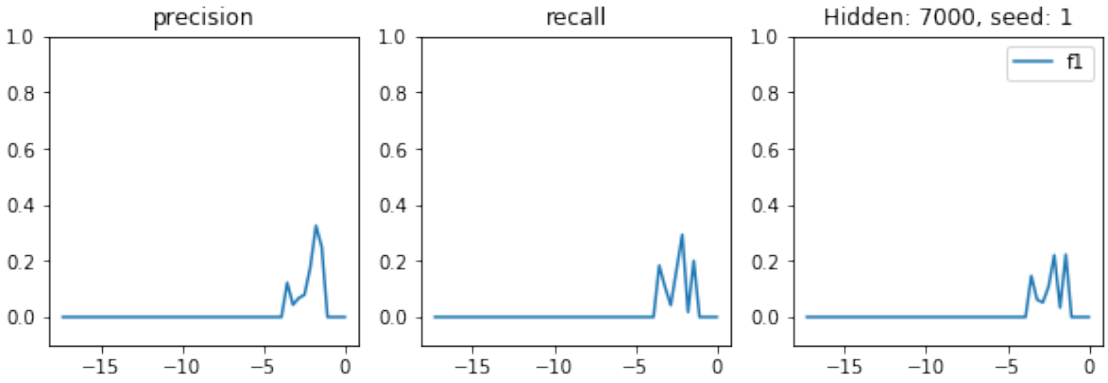
`hidden_size: 5000`



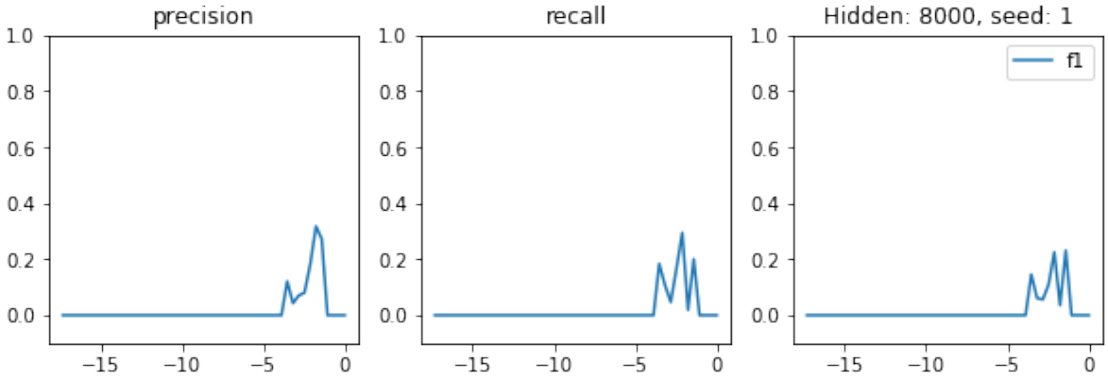
`hidden_size: 6000`



`hidden_size: 7000`

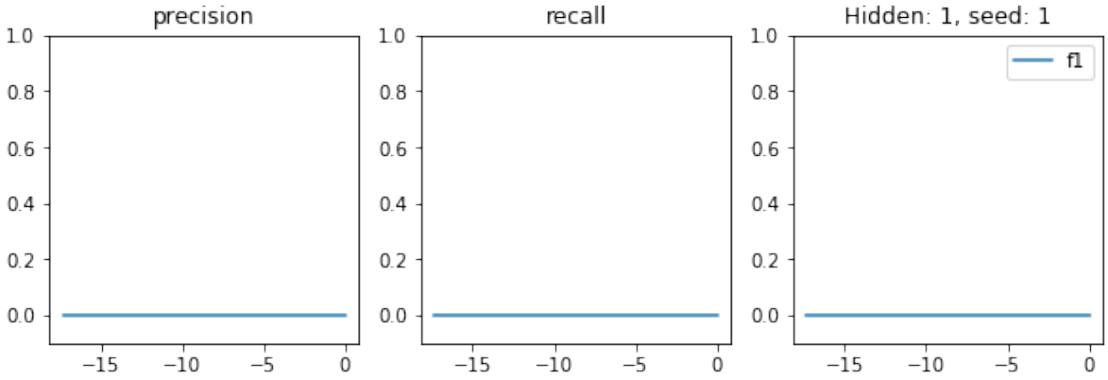



```
hidden_size: 8000
```

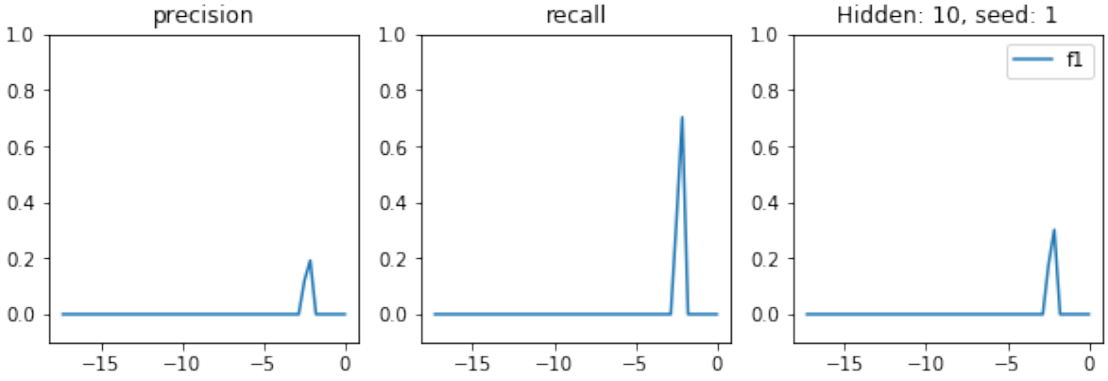


[604]:
```
loss_main["lstm"] = get_loss_dict1 (trace, "/media/arjun/Shared/chaos/
↪output_files/lstm",
                                    hidden_sizes=hidden_sizes, plot_graphs=True)
```

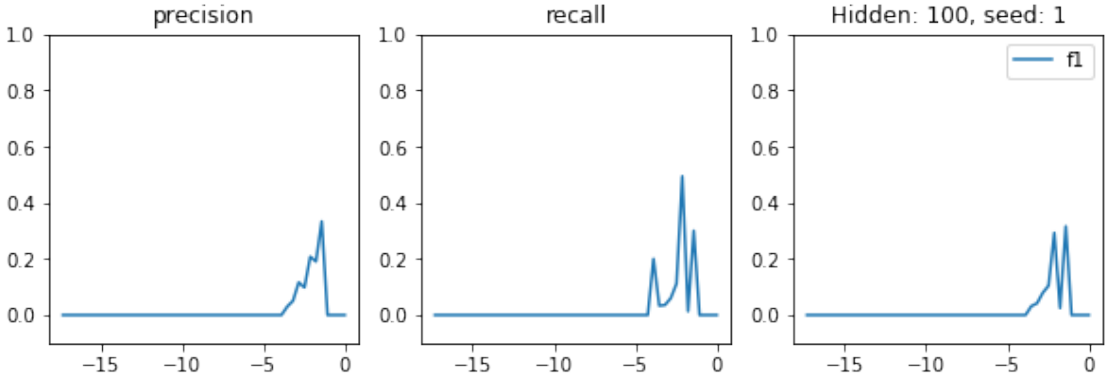
```
relevance threshold: 0.3969525474770118
hidden_size: 1
```



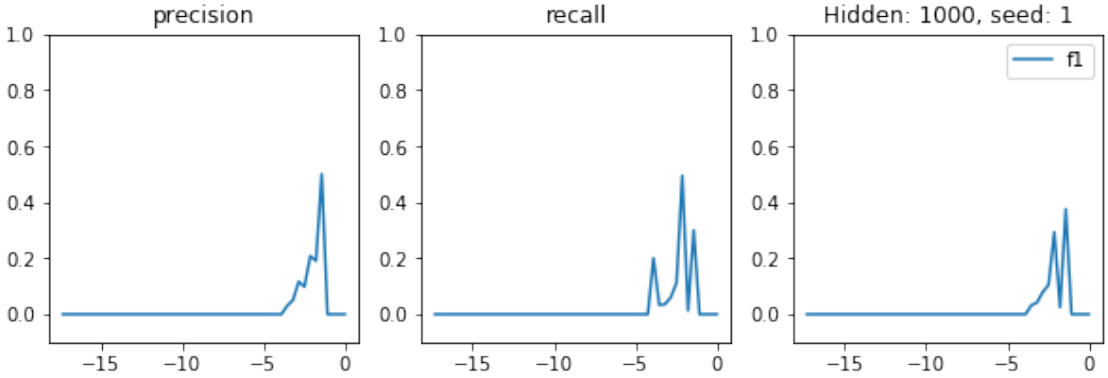
`hidden_size: 10`



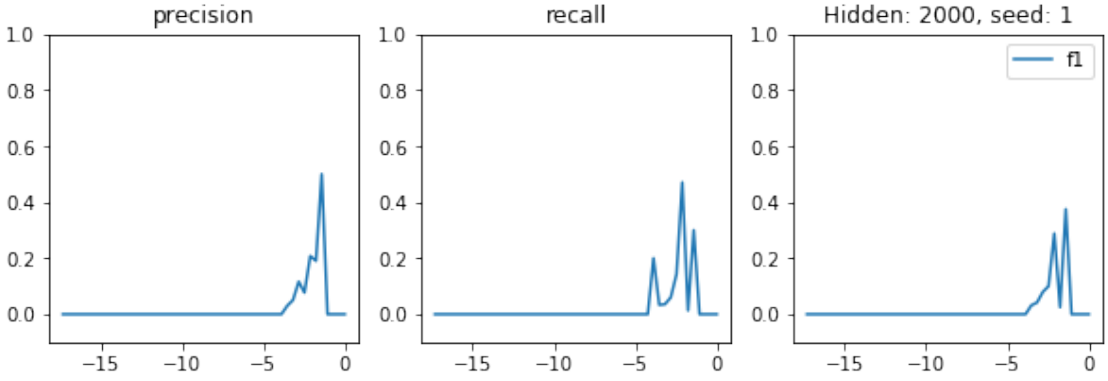
`hidden_size: 100`



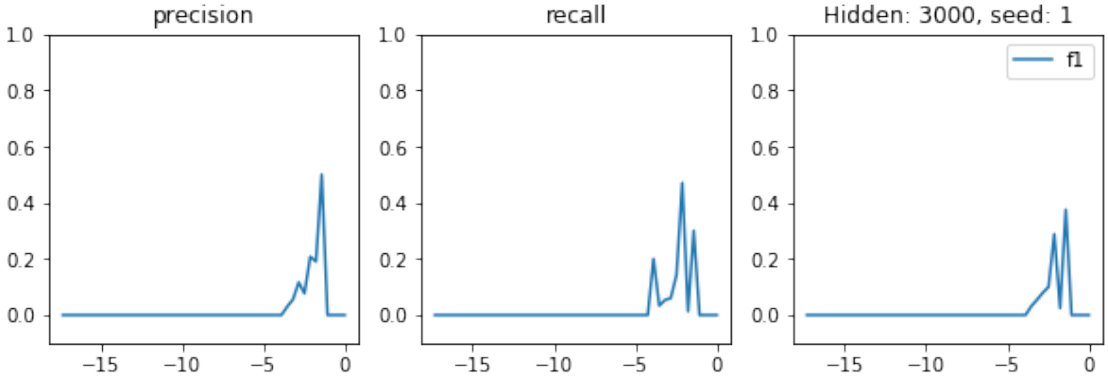
`hidden_size: 1000`



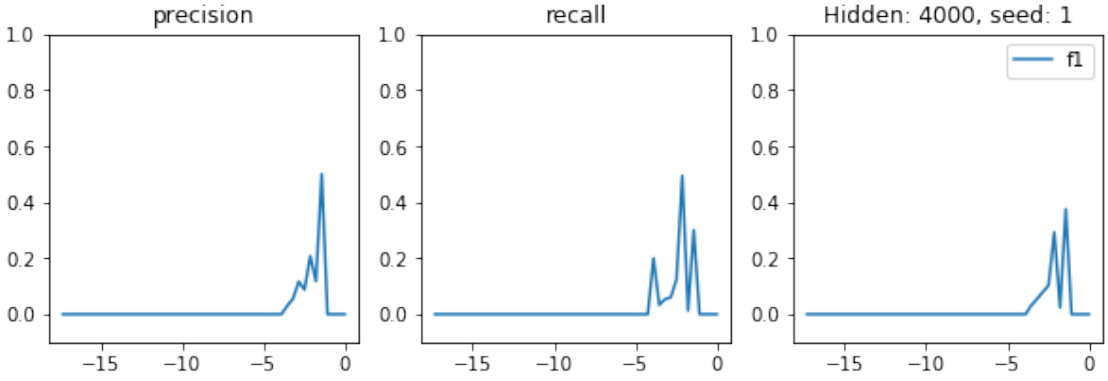
`hidden_size: 2000`



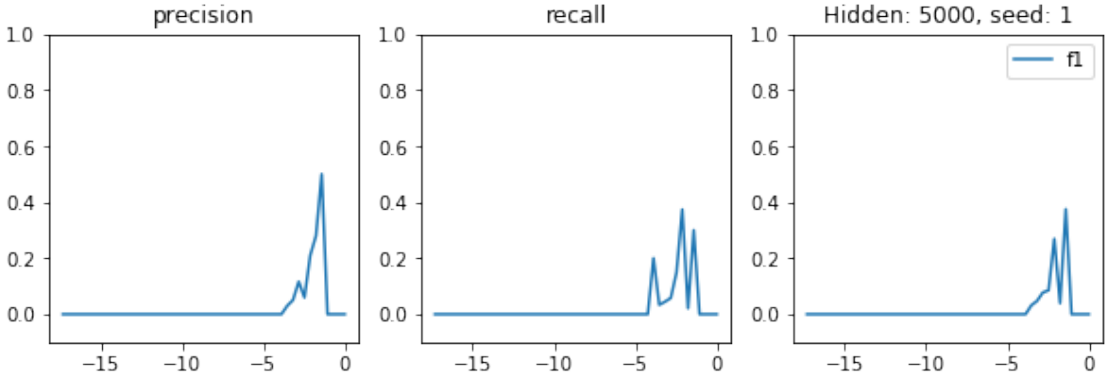
`hidden_size: 3000`



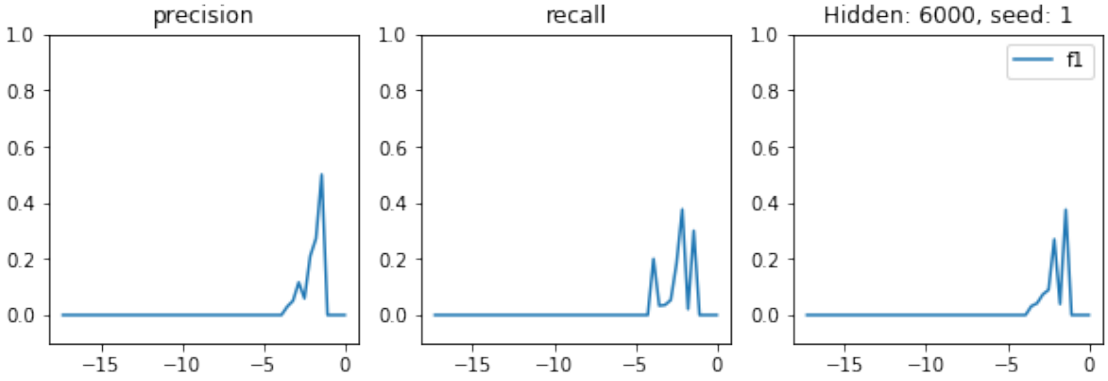
`hidden_size: 4000`



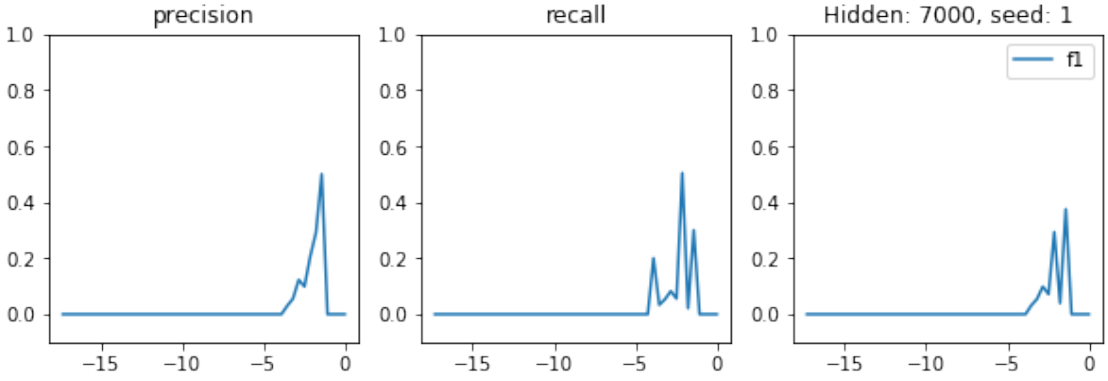
`hidden_size: 5000`



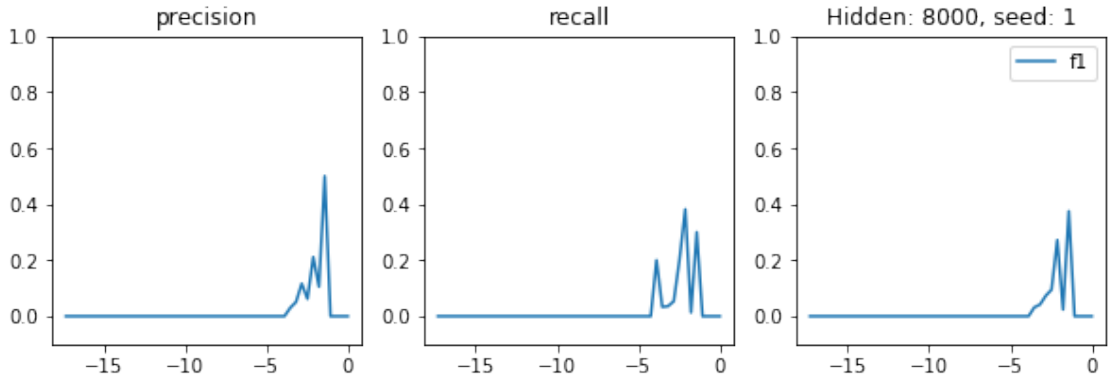
`hidden_size: 6000`



`hidden_size: 7000`



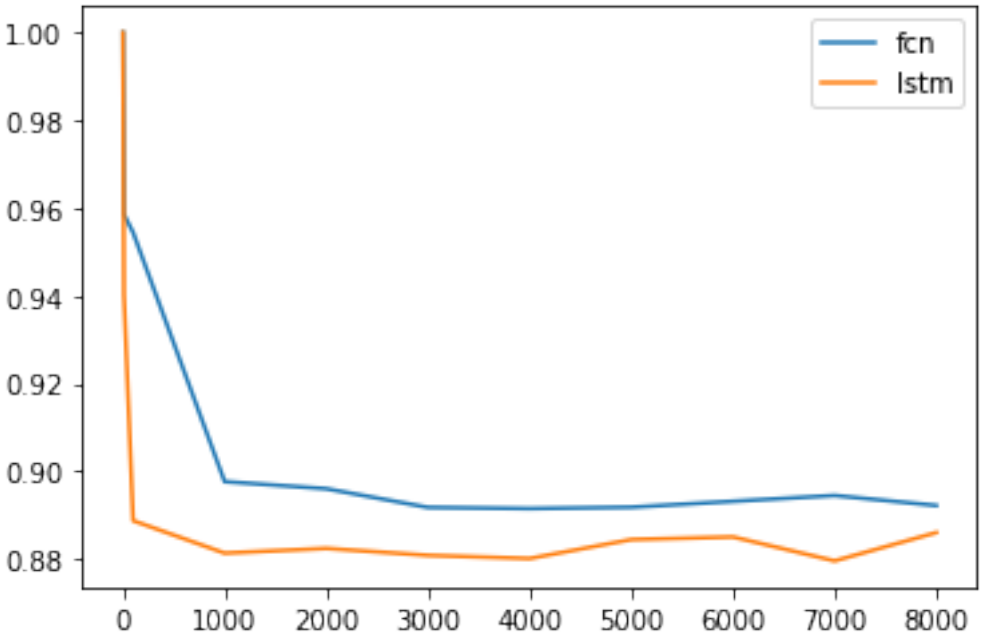
```
hidden_size: 8000
```



[605]:
```
rademacher_list_fcn = []
rademacher_list_lstm = []

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

    loss_array = loss_main["lstm"][hidden_size]
    rademacher_list_lstm.append(get_rademacher(loss_array))
```

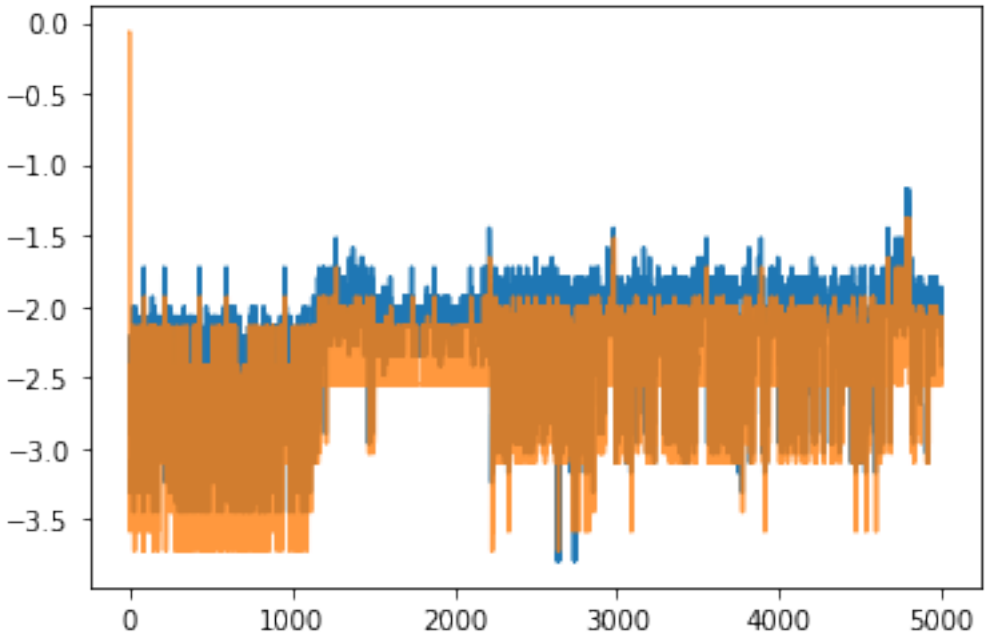
[606]:
```
# compute number of parameters
fcn_param_list = []
lstm_param_list = []

for hidden_size in hidden_sizes:
    fcn_param_list.append(compute_parameters.get_count(4000, 100, [ hidden_size␣
 ↪], [ 'fcn' ]))
    lstm_param_list.append(compute_parameters.get_count(100, 100, [1000,␣
 ↪hidden_size], ["lstm", "fcn"]))
```

[607]:
```
plt.plot(hidden_sizes, rademacher_list_fcn, label="fcn")
plt.plot(hidden_sizes, rademacher_list_lstm, label="lstm")
plt.legend()
# plt.ylim((0.36, 0.42))
# plt.xlim((-0.1e7, 1e7))
plt.show()
```



[608]:
```
actual = []
predicted = []
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_size,␣
↪history_size=40,
                            model_type="lstm",␣
↪output_file_location=output_file_location)

plt.plot(actual, label="actual")
plt.plot(predicted, label="predicted", alpha=0.8)
```

[608]:
```
[<matplotlib.lines.Line2D at 0x7f9b5ebd62e8>]
```


0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
0
1000
2000
3000
4000
5000