2-1.演習：勾配消失問題

In [23]:
```
import numpy as np
from collections import OrderedDict
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# 講義用のスクリプト
from common import functions
from common import layers
from data.mnist import load_mnist
```

In [24]:

```
class MultiLayerNet:
    '''
    input_size: 入力層のノード数
    hidden_size_list: 隠れ層のノード数のリスト
    output_size: 出力層のノード数
    activation: 活性化関数
    weight_init_std: 重みの初期化方法
    '''
    def __init__(self, input_size, hidden_size_list, output_size, activation='relu', weight_init_
std='relu'):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size_list = hidden_size_list
        self.hidden_layer_num = len(hidden_size_list)
        self.params = {}

        # 重みの初期化
        self.__init_weight(weight_init_std)

        # レイヤの生成, sigmoidとreluのみ扱う
        activation_layer = {'sigmoid': layers.Sigmoid, 'relu': layers.Relu}
        self.layers = OrderedDict() # 追加した順番に格納
        for idx in range(1, self.hidden_layer_num+1):
            self.layers['Affine' + str(idx)] = layers.Affine(self.params['W' + str(idx)], self.param
s['b' + str(idx)])
            self.layers['Activation_function' + str(idx)] = activation_layer[activation]()

        idx = self.hidden_layer_num + 1
        self.layers['Affine' + str(idx)] = layers.Affine(self.params['W' + str(idx)], self.params[
'b' + str(idx)])

        self.last_layer = layers.SoftmaxWithLoss()

    def __init_weight(self, weight_init_std):
        """重みの初期化"""
        all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]

        for idx in range(1, len(all_size_list)):
            scale = weight_init_std

            if str(weight_init_std).lower() in ('relu', 'he'):
                scale = np.sqrt(2.0 / all_size_list[idx - 1])
            elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
                scale = np.sqrt(1.0 / all_size_list[idx - 1])

            self.params['W' + str(idx)] = scale * np.random.randn(all_size_list[idx-1], all_size_l
ist[idx])
            self.params['b' + str(idx)] = np.zeros(all_size_list[idx])

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    def loss(self, x, d):
        y = self.predict(x)

        weight_decay = 0

        for idx in range(1, self.hidden_layer_num + 2):
            W = self.params['W' + str(idx)]

        return self.last_layer.forward(y, d) + weight_decay

    def accuracy(self, x, d):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
```

```
        if d.ndim != 1 : d = np.argmax(d, axis=1)

        accuracy = np.sum(y == d) / float(x.shape[0])
        return accuracy

    def gradient(self, x, d):
        # forward
        self.loss(x, d)

        # backward
        dout = 1
        dout = self.last_layer.backward(dout)

        layers = list(self.layers.values())
        layers.reverse()
        for layer in layers:
            dout = layer.backward(dout)

        # 設定
        grad = {}

        for idx in range(1, self.hidden_layer_num+2):
            grad['W' + str(idx)] = self.layers['Affine' + str(idx)].dW
            grad['b' + str(idx)] = self.layers['Affine' + str(idx)].db

        return grad
```

活性化関数にsigmoidを選択した場合

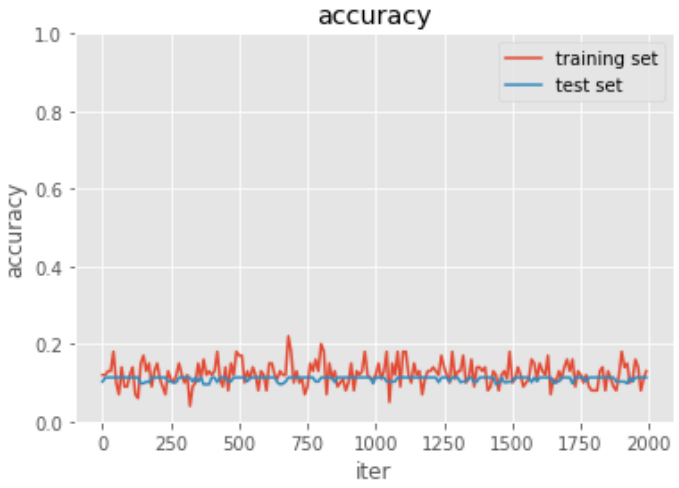
In [25]:
```
# MNISTデータの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)
```

In [26]:
```
# 入力層は28x28=784ノード、出力層は0～9までの分類なので10
network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
                        activation='sigmoid', weight_init_std=0.01)
```

In [27]:
```
iters_num = 2000  # 繰り返し回数
train_size = x_train.shape[0]
batch_size = 100  # ミニバッチ数
learning_rate = 0.1  # 学習率

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
```

In [28]:
```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    # パラメータの更新
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        if (i + 1) % 50 == 0:
            print(f'Generation: {i+1}. 正答率(トレーニング) = {accr_train}')
            print(f'                : {i+1}. 正答率(テスト) = {accr_test}')
```

In [29]:
```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend()
plt.title("accuracy")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



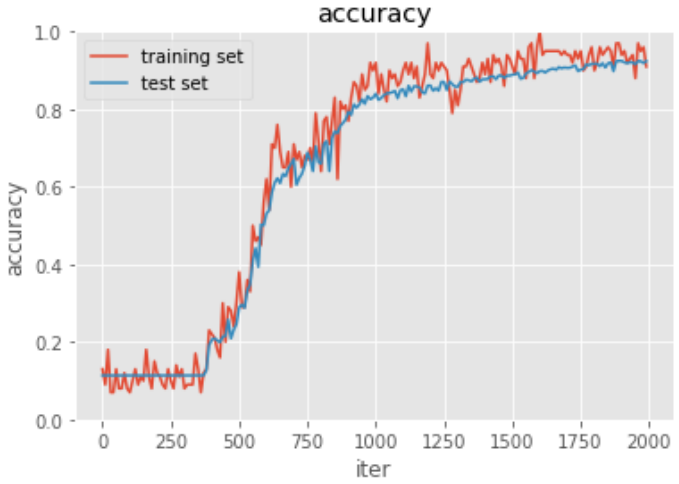
活性化関数にReLUを使用した場合

In [33]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
                        activation='relu', weight_init_std=0.01)
```

In [34]:
```
iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
```

In [35]:
```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        if (i + 1) % 50 == 0:
            print(f'Generation: {i+1}. 正答率(トレーニング) = {accr_train}')
            print(f'                : {i+1}. 正答率(テスト) = {accr_test}')
```

In [36]:
```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend()
plt.title("accuracy")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



考察

活性化関数にsigmoid関数を用いた場合、訓練セットもテストセットも学習を進めても共に精度がほぼ全く上がらなかった。
ReLUを用いた場合、350回辺りから精度が上昇していき、最終的に精度が90%あたりに収束した。

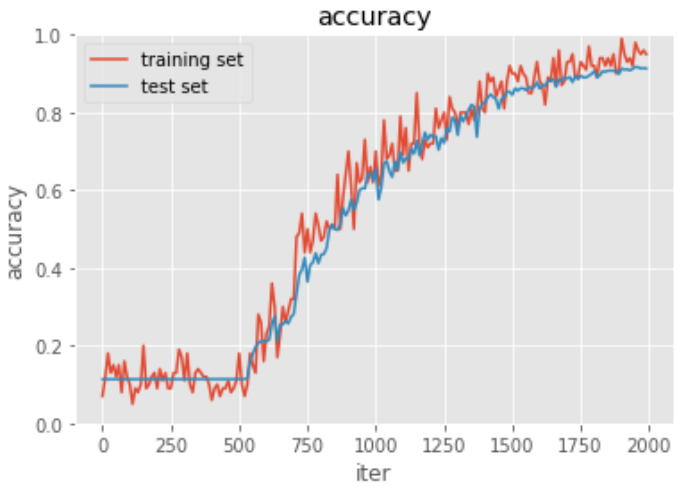
[try] hidden_size_listの数字を変更してみよう

In [37]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[30, 15], output_size=10,
                        activation='relu', weight_init_std=0.01)
```

In [38]:
```
iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
```

In [39]:
```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        if (i + 1) % 50 == 0:
            print(f'Generation: {i+1}. 正答率(トレーニング) = {accr_train}')
            print(f'                : {i+1}. 正答率(テスト) = {accr_test}')
```

In [40]:
```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend()
plt.title("accuracy")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

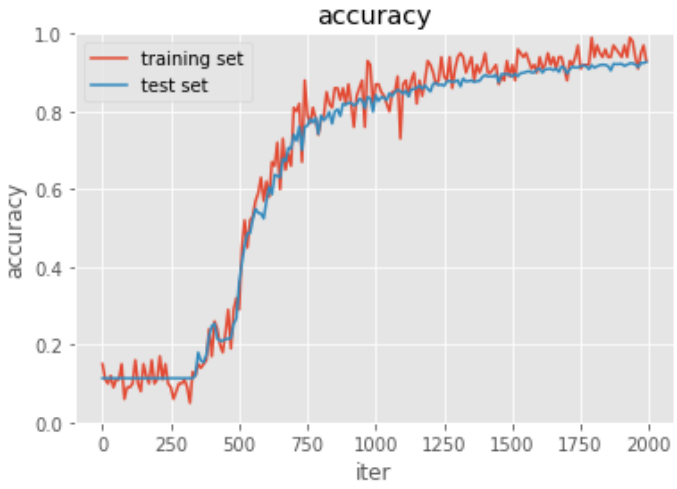


In [45]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[60, 30], output_size=10,
                        activation='relu', weight_init_std=0.01)
```

In [46]:
```
iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
```

In [47]:
```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        if (i + 1) % 50 == 0:
            print(f'Generation: {i+1}. 正答率(トレーニング) = {accr_train}')
            print(f'                : {i+1}. 正答率(テスト) = {accr_test}')
```

In [48]:
```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend()
plt.title("accuracy")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



考察

hidden_size_list=[30, 15] のとき
500回目辺りから緩やかに精度が上昇していき、2000回で訓練セット、テストセット共に90%程度の精度になった。
hidden_size_list=[60, 30] のとき
[40, 20] のときより急に精度が上昇し、2000回で訓練セットは精度がほぼ100%になり、テストセットは90%程度の精度になった。

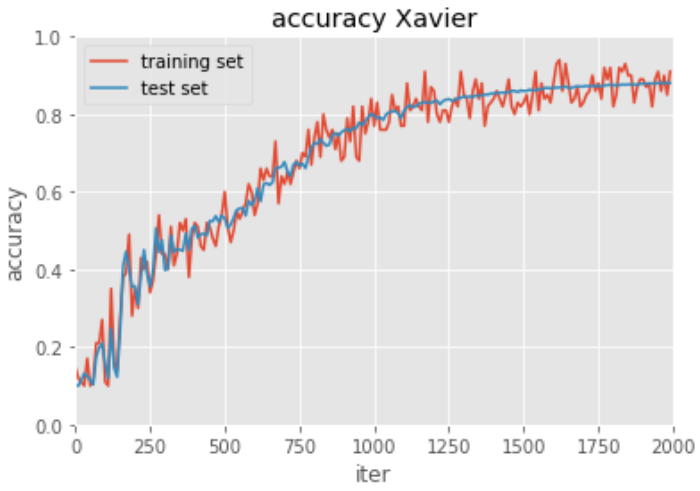
[try] sigmoid - He と relu - Xavier についても試してみよう

In [58]:
```
# 重みの初期化をXavierにする
network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
                        activation='sigmoid', weight_init_std='Xavier')
```

In [59]:
```
iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
```

In [60]:
```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        if (i + 1) % 50 == 0:
            print(f'Generation: {i+1}. 正答率(トレーニング) = {accr_train}')
            print(f'                : {i+1}. 正答率(テスト) = {accr_test}')
```

In [61]:
```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend()
plt.title("accuracy Xavier")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.xlim(0, 2000)
plt.ylim(0, 1.0)
plt.show()
```

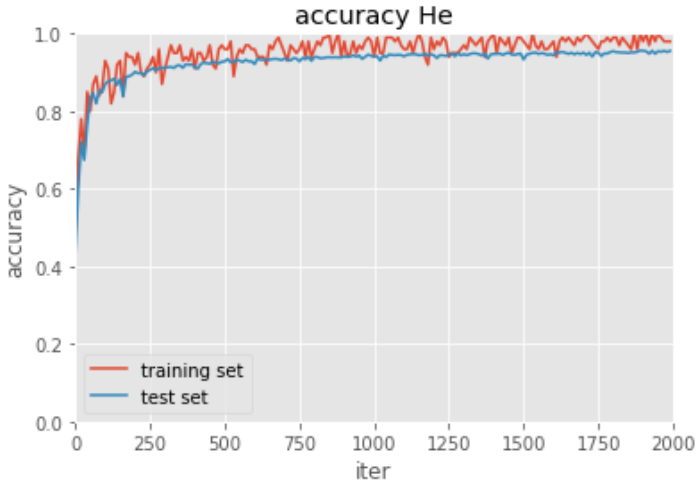


In [62]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
                        activation='relu', weight_init_std='He')
```

In [63]:
```
iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
```

In [64]:
```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        if (i + 1) % 50 == 0:
            print(f'Generation: {i+1}. 正答率(トレーニング) = {accr_train}')
            print(f'                : {i+1}. 正答率(テスト) = {accr_test}')
```

In [65]:
```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend()
plt.title("accuracy He")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.xlim(0, 2000)
plt.ylim(0, 1.0)
plt.show()
```



考察

まず、重みの初期値を設定することでsigmoidでも学習が全く進まなくなるということは無くなった。
XavierもHeもどちらも、訓練セットとテストセットでaccuracyが上昇しているが、特にHeはかなり早い段階でaccuracyが1.0に近くなる。

In [ ]:
```
```