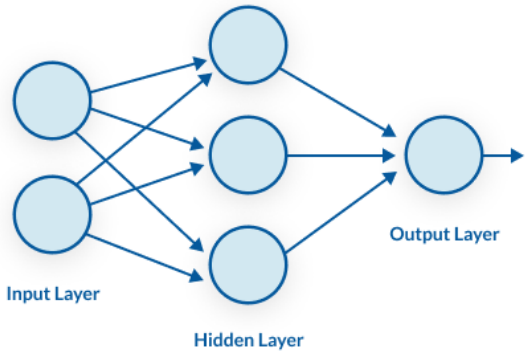
Gradient Vanishing and ResNet

Objectives and Outcomes

Use Google Colab to train and test the FFNet and ResNet models to find the maximum number of layers and how to overcome the gradient vanishing problem. By doing this, I can understand the gradient vanishing problem, the architecture of ResNet, and how to mitigate the gradient vanishing problem.

Introduction

Feed-Forward Neural Network (FFNet)



There are no cycles or loops in the network. The data moves in one direction from the input nodes through the hidden nodes to the output nodes.

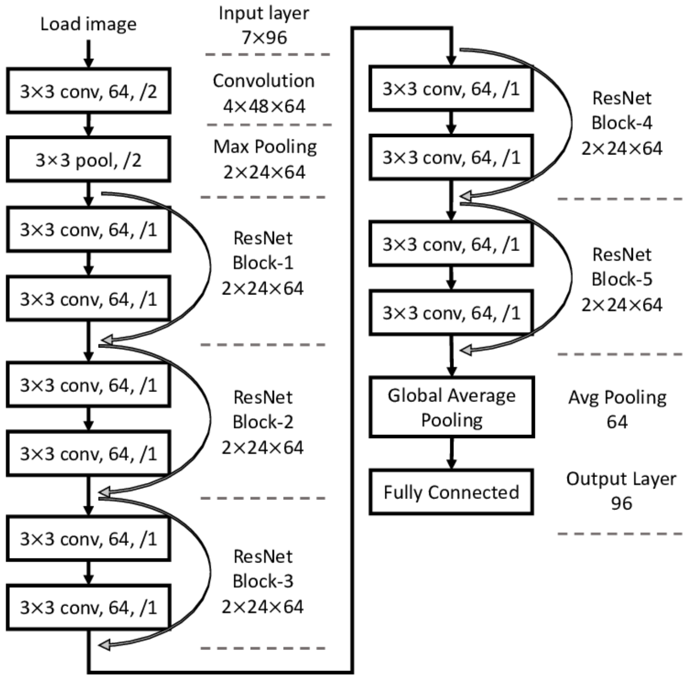
FFNet Architecture

Input Layer: There are neurons that represent a feature of the input data.

Hidden Layers: These layers are responsible for learning the complex patterns in the data. Each neuron in a hidden layer applies a weighted sum of inputs followed by a non-linear activation function.

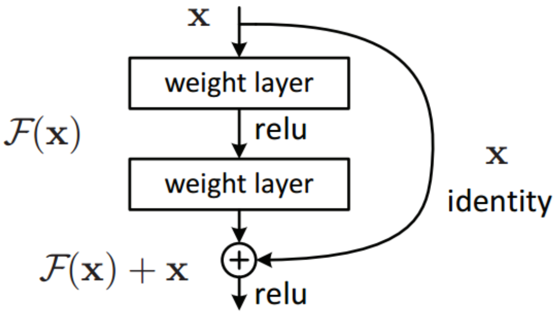
Output Layer: The number of neurons in this layer corresponds to the number of classes in classification problem or the number of outputs in regression problem.

Residual Network (ResNet)



Deeper networks do not necessarily have better performance because of the vanishing gradient problem. The main idea of ResNet is to use an identity shortcut connection that skips one or more layers. This trick can alleviate the gradient vanishing problem.

ResNet Architecture



The residual block has the shortcut connection (x identity). It can allow the output layer gradient to reach earlier layers directly. This direct pathway enables the gradient to avoid intermediate transformations that could shrink it, such as the small value with repeated multiplication, leading to gradient vanishing. Moreover, it learns residual functions instead of complete transformations.

The residual block output formula:

$$\mathcal{Y} = \mathrm{F(x)} + \mathrm{x}$$

F(x): Weight Layers Output

x: Identity Shortcut

For example:

The gradient from the loss function is $\frac{\partial L}{\partial \mathcal{Y}}$ = 0.5

The gradient of F(x) is $\frac{\partial F(x)}{\partial x}$ = 0.01

Without the Shortcut Connection:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \mathcal{Y}} \cdot \frac{\partial F(x)}{\partial x} = 0.5 \cdot 0.01 = 0.005$$

With the Shortcut Connection:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \mathcal{Y}} \cdot \left(\frac{\partial F(x)}{\partial x} + 1\right) = 0.5 \cdot (0.01+1) = 0.505$$

In view of this, the addition operation ensures that the gradient during backpropagation includes a constant component from the identity shortcut, which prevents it from vanishing entirely.

Gradient Vanishing

The input passes through the weight layer, followed by the activation function to enter the next weight layer. When there are multiple layers, the derivatives of the activation functions and weights might scale each layer's gradient, leading to small values of repeated multiplication. As a result, the vanishing gradient problem might occur.

Further investigations

Experiment 1: Change the number of layers in *ffnet* to find the maximum number of layers that a feedforward network can have without suffering from the vanishing gradient problem.

Feed-Forward Neural Network (FFNet)

```
act = 'tanh'
x_in = tf.keras.layers.Input((2,), name='Input')
x = Dense(5, input_dim=2, activation=act, name='L1')(x_in)
l = 1
for i in range(30):
  x = Dense(5, input_dim=5, activation=act, name=f'L{l+1}')(x)
  l = l + 1
x_out = Dense(1, activation='sigmoid', name='Output')(x)
ffnet = tf.keras.models.Model(inputs=x_in, outputs=x_out,
name="FFNet")
opt = SGD(learning_rate=0.01, momentum=0.9)
ffnet.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
#ffnet.summary()
ffnet_init_weights = ffnet.get_weights()
print(f"No. of layers = {len(ffnet.layers)}")
```

1. Input Layer

```
x_in = tf.keras.layers.Input((2,), name='Input')
```

The input is a 2-dimensional vector

2. Hidden Layers

```
x = Dense(5, input_dim=2, activation=act, name='L1')(x_in)
```

There are 5 neurons, 2-dimensional vector, and activation function (tanh)

3. Add the Hidden Layers

```
for i in range(30):
  x = Dense(5, input_dim=5, activation=act, name=f'L{l+1}')(x)
  l = l + 1
```

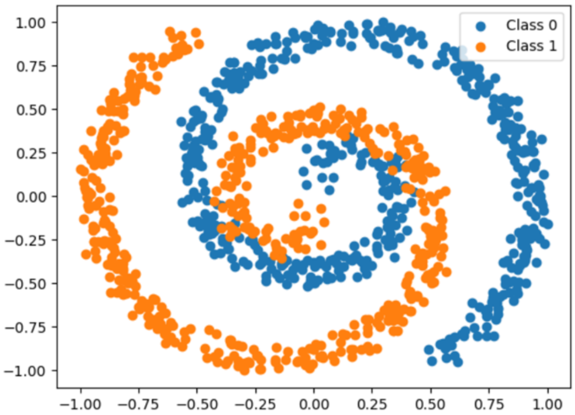
Its mains add 30 hidden layers, each layer having 5 neurons and using tanh as the activation function.

4. Output Layer

```
x_out = Dense(1, activation='sigmoid', name='Output')(x)
```

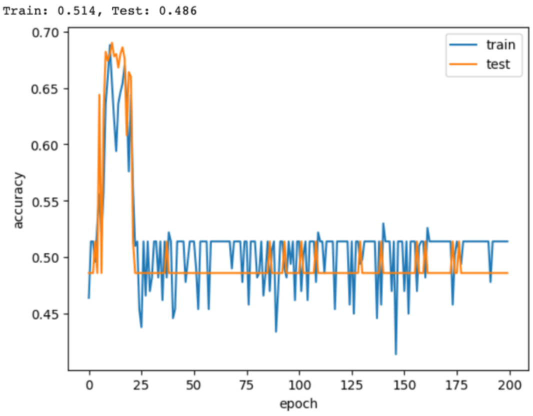
There is 1 neuron, and the sigmoid activation function is used

2d Classification Dataset



The 30 layers in FFNet (Default)

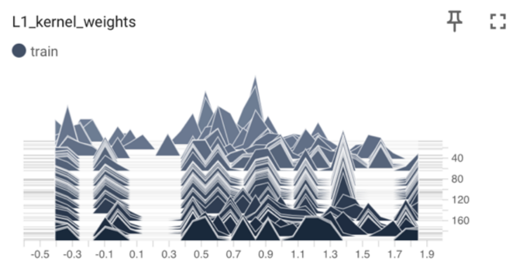
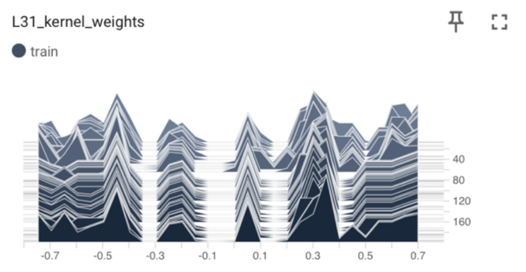
```
for i in range(30):
  x = Dense(5, input_dim=5, activation=act, name=f'L{l+1}')(x)
  l = l + 1
```

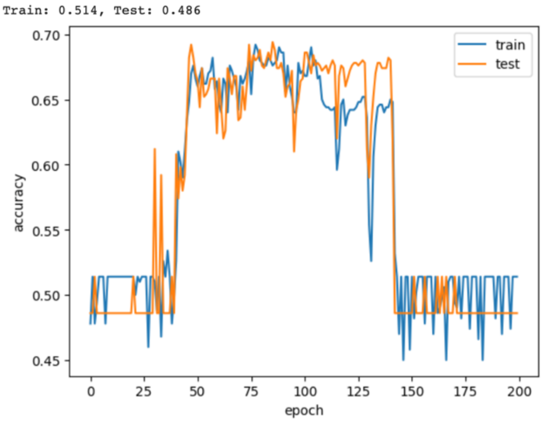


Train: 0.514	Test: 0.486	No. of layers = 33

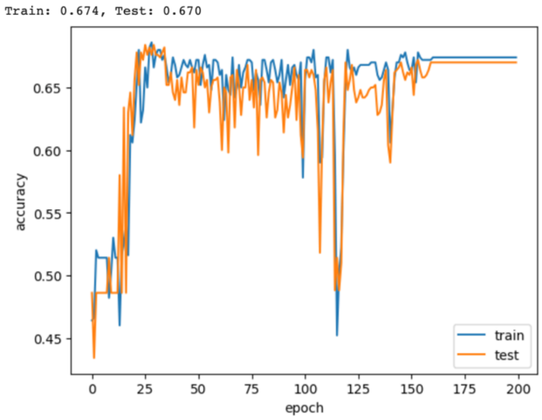
Analysis: There is gradient vanishing because the accuracy drops to around 0.514 after the 20 epochs, and it is nearly flat afterward. Therefore, the layer needs to decrease to find the maximum layer.

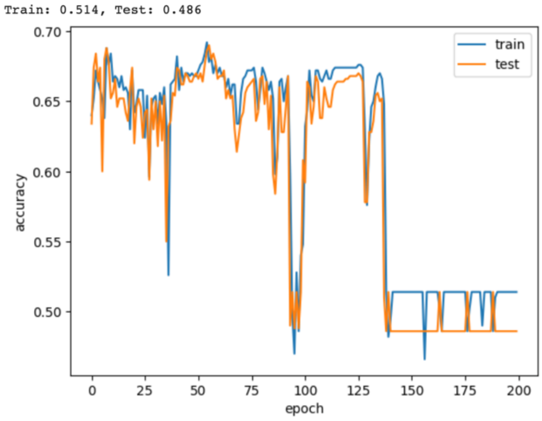
The Kernel Weights Histogram

<table>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">Analysis: About the first layer, the weights are distributed in a wide range from -0.4 to 1.8. However, the last layer weights are tightly clustered within a smaller range from -0.7 to 0.7. This means that gradients are multiplied layer by layer, and their derivatives are small, leading to the exponential shrinking of gradients in deeper layers.</td></tr>
</table>

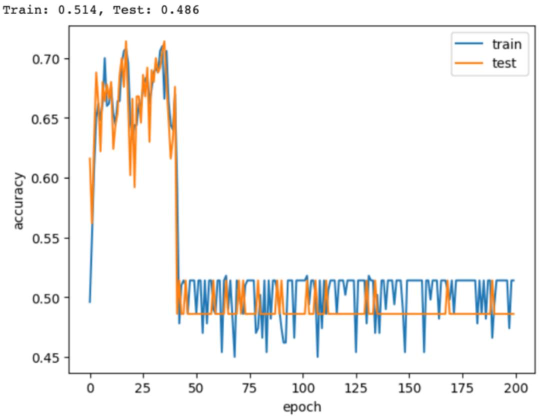
<table>
<tr><td colspan="3"><b>The 25 layers in FFNet</b></td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.514</td><td>Test: 0.486</td><td>No. of layers = 28</td></tr>
<tr><td colspan="3">Analysis: It also is gradient vanishing because the accuracy drops after 50 epochs and flattens at 0.514. Therefore, the layer needs to decrease.</td></tr>
</table>

<table>
<tr><td><b>The 20 layers in FFNet</b></td></tr>
</table>

<table>
<tr><td colspan="3"> </td></tr>
<tr><td>Train: 0.674</td><td>Test: 0.670</td><td>No. of layers = 23</td></tr>
<tr><td colspan="3">Analysis: Compared with the 25 or 30 layers, it has better stability and avoids gradient vanishing. Therefore, the layer needs to increase to find the maximum layer.</td></tr>
</table>

<table>
<tr><th colspan="3">The 21 layers in FFNet</th></tr>
<tr><td colspan="3"> </td></tr>
<tr><td>Train: 0.514</td><td>Test: 0.486</td><td>No. of layers = 24</td></tr>
<tr><td colspan="3">Analysis: There is gradient vanishing because the accuracy drops from around 0.67 to 0.48 after the 135 epochs and flattens at 0.514.</td></tr>
</table>

<table>
<tr><th>The 22 layers in FFNet</th></tr>
</table>

		
Train: 0.514	Test: 0.486	No. of layers = 25
Analysis: Compared with 21 layers, the gradient vanishing occurs in earlier layers.		

FFNet Maximum Number of Layers Summary

In view of this, the 20 layers are the maximum number of layers in FFNet. It does not occur the gradient vanishing. When over the 20 layers, the accuracy will drop and flatten.

Experiment 2: Change the number of layers in *resnet* to find the maximum number of layers that a residual network can have without suffering from vanishing gradient problem.

Residual Network (ResNet)

```
import tensorflow as tf
from keras.layers import Add
act = 'tanh'
x_in = tf.keras.layers.Input((2,), name='Input')
x = Dense(5, input_dim=2, activation=act, name='L1')(x_in)
x = Dense(5, input_dim=5, activation=act, name='L2')(x)
l = 3
for i in range(25):
  x_skip = x
  x = keras.layers.BatchNormalization()(x)
  x = Dense(5, input_dim=5, activation=act, name=f'L{l}')(x)
  x = Dense(5, input_dim=5, activation=act, name=f'L{l+1}')(x)
  x = Add()([x, x_skip])
  l = l + 2
x = Dense(5, input_dim=5, activation=act, name=f'L{l}')(x)
x_out = Dense(1, activation='sigmoid', name='Output')(x)
resnet = tf.keras.models.Model(inputs=x_in, outputs=x_out,
name="ResNet")
opt = SGD(learning_rate=0.01, momentum=0.9)
resnet.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
#resnet.summary()
resnet_init_weights = resnet.get_weights()
print(f"No. of layers = {len(resnet.layers)}")
```

1. Input Layer

```
x_in = tf.keras.layers.Input((2,), name='Input')
```

Defines the input layer with 2 features

2. Dense Layer

```
x = Dense(5, input_dim=2, activation=act, name='L1')(x_in)
x = Dense(5, input_dim=5, activation=act, name='L2')(x)
```

There are 5 neurons and using tanh activation

3. Residual Block

```
for i in range(25):
  x_skip = x
  x = keras.layers.BatchNormalization()(x)
  x = Dense(5, input_dim=5, activation=act, name=f'L{l}')(x)
  x = Dense(5, input_dim=5, activation=act, name=f'L{l+1}')(x)
  x = Add()([x, x_skip])
  l = l + 2
```

Creates 25 residual blocks to make the network deeper while avoiding gradient vanishing problems

x_skip: Skip Connection

keras.layers.BatchNormalization(): It can help for better training stability

4. Dense Layer

```
x = Dense(5, input_dim=5, activation=act, name=f'L{l}')(x)
```

Maps the 5-dimensional feature vector to another 5-dimensional vector

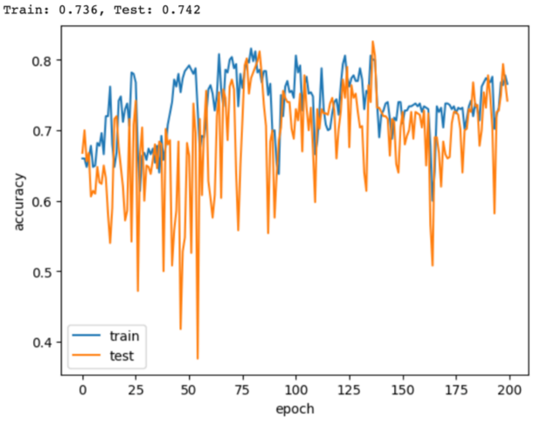
5. Output Layer

```
x_out = Dense(1, activation='sigmoid', name='Output')(x)
```

There is 1 neuron, and the sigmoid activation function is used

The 25 layers in ResNet (Default)

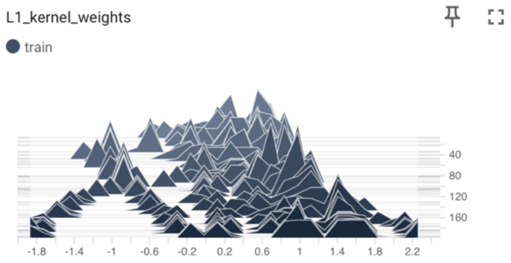
```
for i in range(25):
  x_skip = x
  x = keras.layers.BatchNormalization()(x)
```

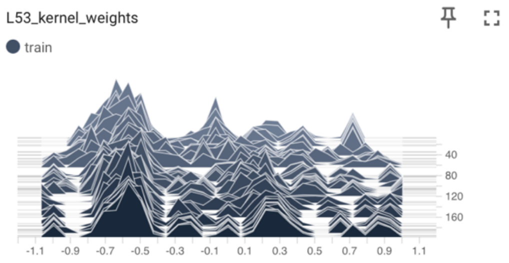


Train: 0.736	Test: 0.742	No. of layers = 105

Analysis: There is no gradient vanishing. The training and testing accuracies remain stable across 200 epochs. It has minor fluctuations.

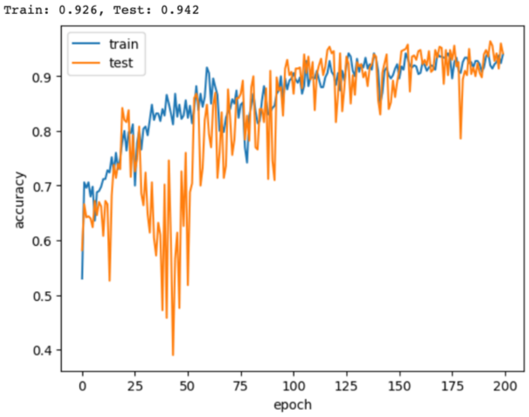
The Kernel Weights Histogram

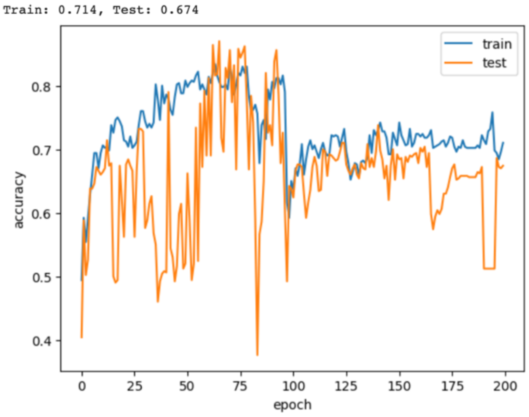


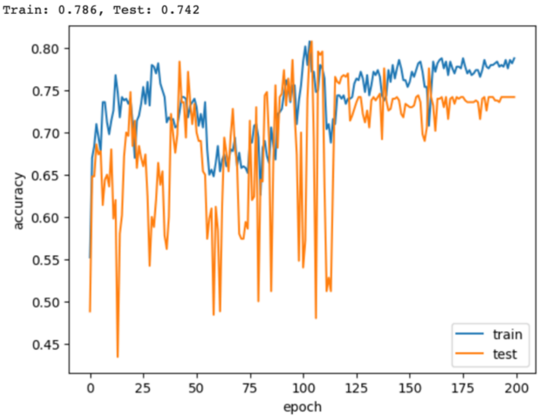


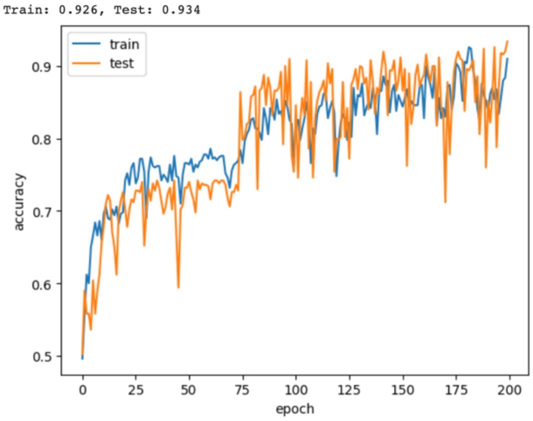
Analysis: About the first layer, the weights are distributed in a wide range from -1.8 to 2.2, while the last layer weights are tightly clustered within a smaller range from -1.1 to 1. Although gradients shrink exponentially as they propagate through the layers, the residual connections in ResNet minimize this effect,

<table>
<tr><td>resulting in only slight exponential gradient shrinking.</td></tr>
</table>

<table>
<tr><th colspan="3">The 50 layers in ResNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.926</td><td>Test: 0.942</td><td>No. of layers = 305</td></tr>
<tr><td colspan="3">Analysis: There is no gradient vanishing. The training and testing accuracies improve and stabilize after around 75 epochs. Therefore, the layer needs to increase to find the maximum layer.</td></tr>
</table>

<table>
<tr><th colspan="3">The 75 layers in ResNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.714</td><td>Test: 0.674</td><td>No. of layers = 205</td></tr>
<tr><td colspan="3">Analysis: The gradual vanishing may occur because the training accuracy peaks at 0.714 but does not improve further. Also, the testing accuracy fluctuates significantly and stabilizes lower at 0.674. The 175 to 200 epochs have a short time flatten. So, the layer needs to decrease to find the maximum layer.</td></tr>
</table>

<table>
<tr><th colspan="3">The 74 layers in ResNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.786</td><td>Test: 0.742</td><td>No. of layers = 301</td></tr>
<tr><td colspan="3">Analysis: Compared with the 75 layers, the 74 layers have fewer fluctuations, which is stability and optimization. There is no gradient vanishing.</td></tr>
</table>

<table>
<tr><th colspan="3">The 73 layers in ResNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.926</td><td>Test: 0.934</td><td>No. of layers = 297</td></tr>
<tr><td colspan="3">Analysis: The 73 layers in ResNet perform better than the 74 and 75 layers. It can achieve higher accuracy and more stable results.</td></tr>
</table>

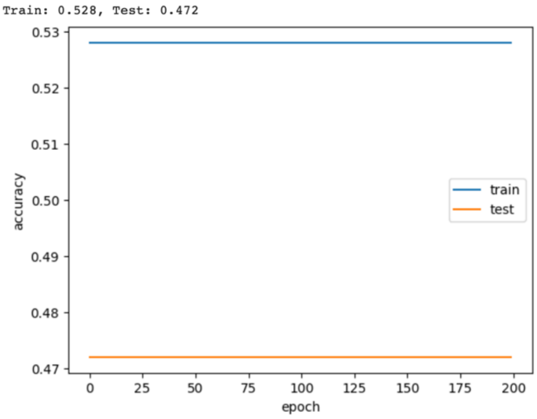
ResNet Maximum Number of Layers Summary

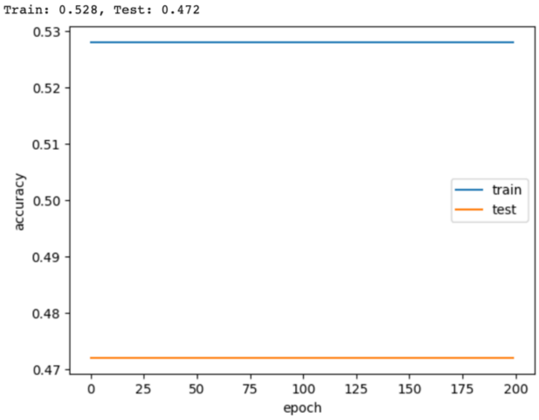
In view of this, the 74 layers are the maximum number of layers in ResNet, but the 73 layers are better than the 74 layers, which have high accuracy and small fluctuations. Both numbers of layers do not cause the gradient vanishing problem.

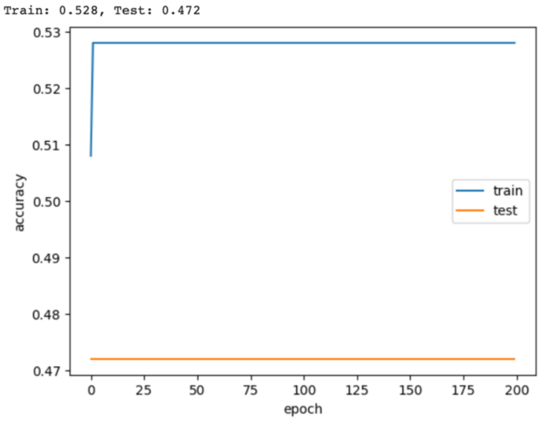
Experiment 3: Change the 'tanh' to 'relu' and repeat (1) and (2) to see if ReLU can help mitigating the vanishing gradient problem.

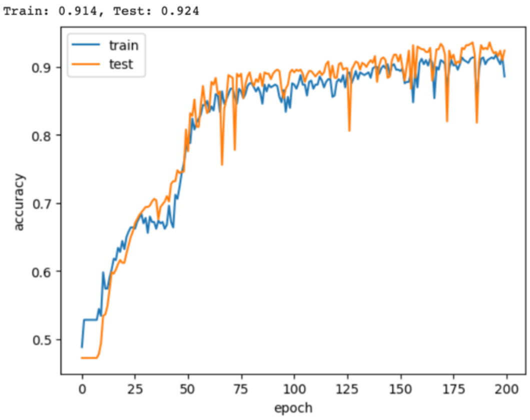
a) The FFNet part

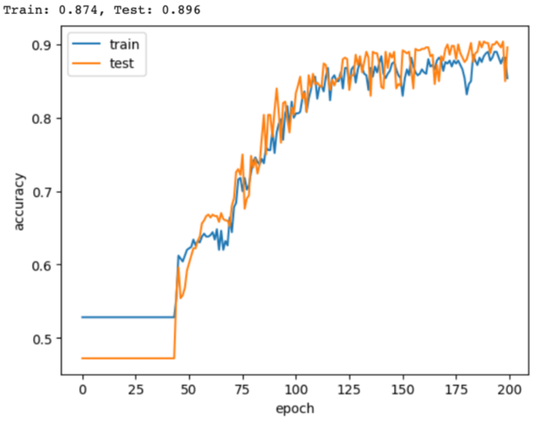
About the part (1), the FFNet maximum number of layers is 20. After the 20, there are gradient vanishing problem. Now, the 'tanh' will be changed to 'relu'.

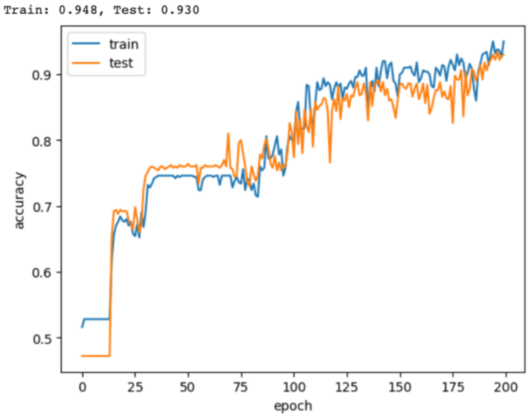
<table>
<tr><th colspan="3">The 30 layers in FFNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.528</td><td>Test: 0.472</td><td>No. of layers = 33</td></tr>
<tr><td colspan="3">Analysis: It exist gradient vanishing because the training and testing accuracies have no improvement across 200 epochs. Therefore, the layer needs to decrease to find the maximum layer.</td></tr>
</table>

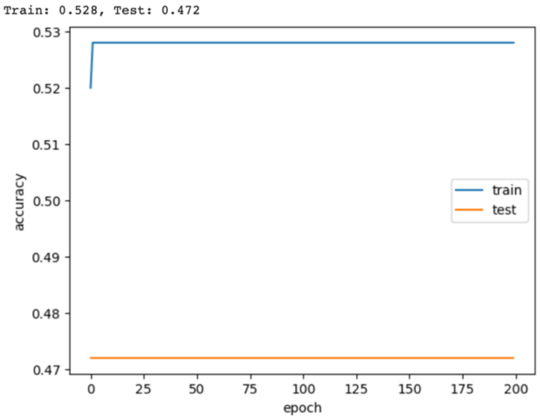
<table>
<tr><th colspan="3">The 20 layers in FFNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.528</td><td>Test: 0.472</td><td>No. of layers = 23</td></tr>
<tr><td colspan="3">Analysis: It also exists gradient vanishing like 30 layers situation.</td></tr>
</table>

<table>
<tr><th colspan="3">The 10 layers in FFNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.528</td><td>Test: 0.472</td><td>No. of layers = 13</td></tr>
<tr><td colspan="3">Analysis: It does not improve beyond the initial few epochs.</td></tr>
</table>

<table>
<tr><th colspan="3">The 5 layers in FFNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.914</td><td>Test: 0.924</td><td>No. of layers = 8</td></tr>
<tr><td colspan="3">Analysis: There is no gradient vanishing because the gradient flow effectively shows high accuracy and stable learning. Therefore, the layer needs to increase to find the maximum layer.</td></tr>
</table>

<table>
<tr><th colspan="3">The 6 layers in FFNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.874</td><td>Test: 0.896</td><td>No. of layers = 9</td></tr>
<tr><td colspan="3">Analysis: It achieves high accuracy and exhibits stable learning, with no signs of gradient vanishing. Therefore, the layer needs to increase to find the maximum layer.</td></tr>
</table>

<table>
<tr><th colspan="3">The 7 layers in FFNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.948</td><td>Test: 0.930</td><td>No. of layers = 10</td></tr>
<tr><td colspan="3">Analysis: It also does not exist the gradient vanishing problem. Therefore, the layer needs to increase to find the maximum layer.</td></tr>
</table>

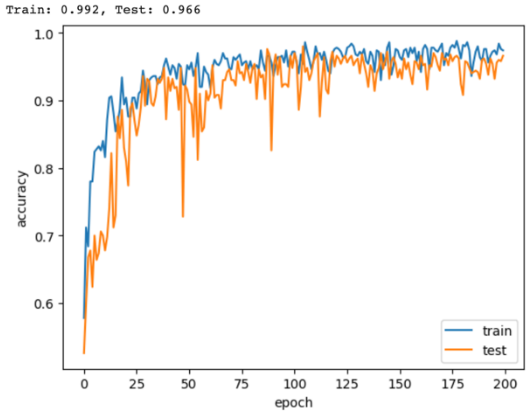
<table>
<tr><th colspan="3">The 8 layers in FFNet</th></tr>
<tr><td colspan="3"> </td></tr>
<tr><td>Train: 0.948</td><td>Test: 0.930</td><td>No. of layers = 11</td></tr>
<tr><td colspan="3">Analysis: The 8 layers occur the gradient vanishing, leading to stagnant performance and poor generalization.</td></tr>
</table>

FFNet Maximum Number of Layers Summary (Use 'relu')

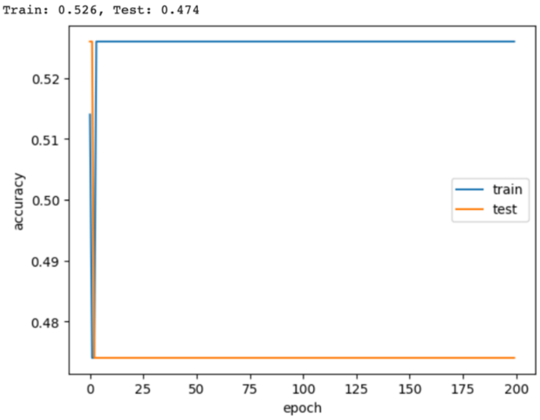
In view of this, the 7 layers are the maximum number of layers in FFNet because there is no gradient vanishing problem.

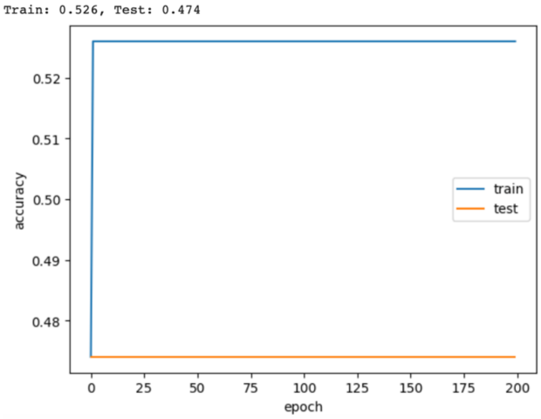
b) The ResNet part

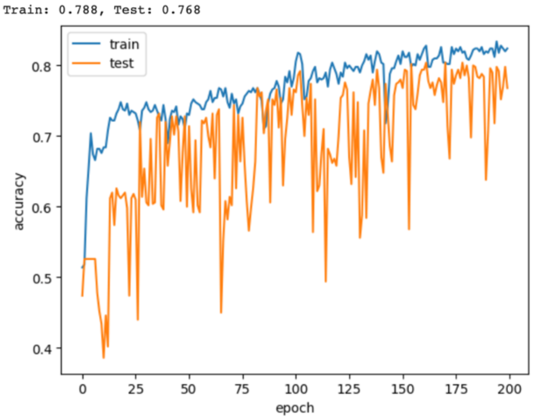
About the part (2), the ResNet maximum number of layers is 74. After the 74, the gradient vanishing may occur. Now, the 'tanh' will be changed to 'relu'.

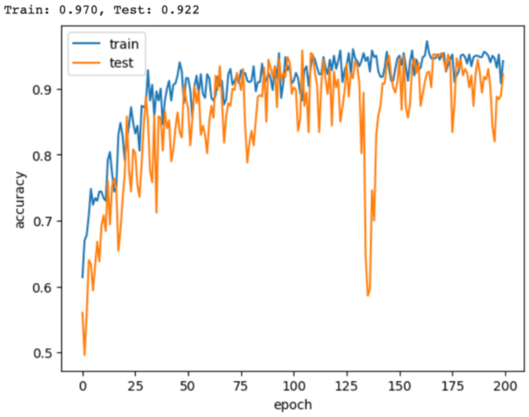
<table>
<tr><th colspan="3">The 25 layers in ResNet</th></tr>
<tr><td colspan="3"> </td></tr>
<tr><td>Train: 0.992</td><td>Test: 0.966</td><td>No. of layers = 105</td></tr>
<tr><td colspan="3">Analysis: There is no gradient vanishing. The training and testing accuracies are high and stable. It has a slight fluctuation. Therefore, the layer needs to increase</td></tr>
</table>

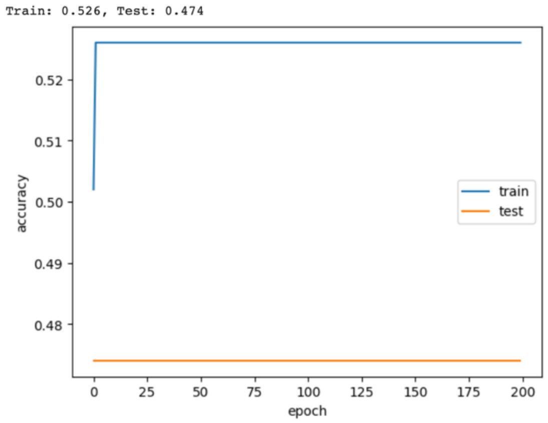
to find the maximum layer.

<table>
<tr><th colspan="3">The 75 layers in ResNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.526</td><td>Test: 0.474</td><td>No. of layers = 305</td></tr>
<tr><td colspan="3">Analysis: There is gradient vanishing because the training accuracy is remain flat at 0.526 and the testing accuracy is stagnate at 0.474. There are no improvements throughout the training process. Therefore, the layer needs to decrease to find the maximum layer.</td></tr>
</table>

<table>
<tr><th colspan="3">The 70 layers in ResNet</th></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Train: 0.526</td><td>Test: 0.474</td><td>No. of layers = 285</td></tr>
<tr><td colspan="3">Analysis: It also exists gradient vanishing like 75 layers situation.</td></tr>
</table>

The 65 layers in ResNet		
		
Train: 0.788	Test: 0.768	No. of layers = 265
Analysis: There is no gradient vanishing but the testing accuracy is fluctuate significantly. Therefore, the layer needs to increase to find the maximum layer.		

The 66 layers in ResNet		
		
Train: 0.970	Test: 0.922	No. of layers = 269
Analysis: The training and testing accuracies are better than the 65 layers. There is no gradient vanishing, so the layer needs to increase to find the maximum layer.		

The 67 layers in ResNet		
		
Train: 0.526	Test: 0.474	No. of layers = 273
Analysis: There is gradient vanishing and no improvements throughout the training process.		

ResNet Maximum Number of Layers Summary (Use 'relu')

In view of this, the 66 layers are the maximum number of layers in ResNet because there is no gradient vanishing problem.

Discussion

ReLU and Gradient Vanishing

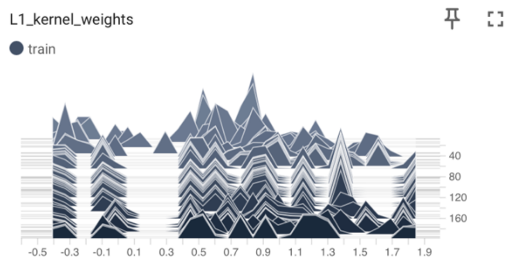
The gradient vanishing was caused by the repeated multiplication of small values. The ReLU range is 0 to infinity. This means the negative value is set to 0 while the positive value is set to 1. Therefore, the ReLU can avoid the gradient exponential shrinkage. By doing this, the ReLU can help mitigate the gradient vanishing problem. Although the ReLU can mitigate the gradient vanishing problem, the dying ReLU may occur like the experiment 3.

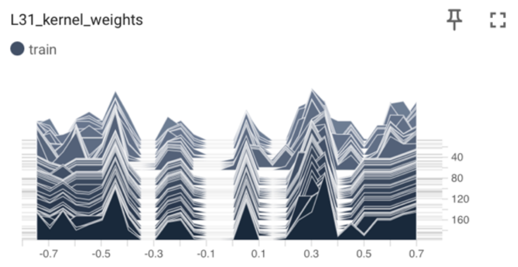
Dying ReLU

In experiment 3, the situation is that the FFNet activation function changes from Tanh to ReLU, and the maximum layers will decrease from 20 (Total layers is 23) layers to 7 (Total layers is 10) layers. Moreover, the ResNet activation function also changes from Tanh to ReLU, decreasing the maximum layers from 74 (Total layers is 301) to 66 (Total layers is 269). This means there are large negative values, leading to inactive neurons and always outputting zero for any input. Therefore, ReLU can mitigate gradient vanishing by avoiding exponential gradient shrinkage. Still, it may cause the potential for dying neurons, limiting the depth of networks like FFNet and ResNet when compared to Tanh.

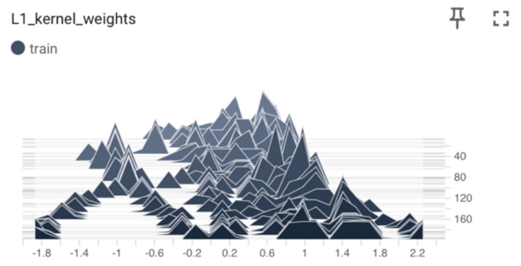
ResNet and Gradient Vanishing

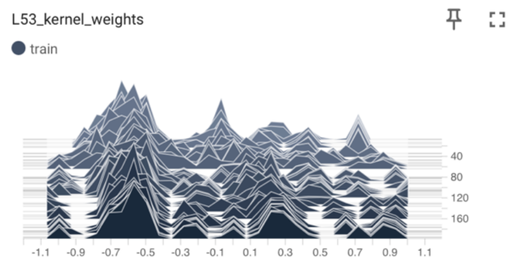
The Kernel Weights Histogram (FFNet)





The Kernel Weights Histogram (ResNet)





About the FFNet, the weight range significantly drops from -0.4 to 1.8 in the first layer to -0.7 to 0.7 in the last layer. It indicates the vanishing gradient problem, where deeper layers experience diminished weight variability and hindered training.

As for the ResNet, the weight range slightly drops from -1.8 to 2.2 in the first layer to -1.1 to 1 in the last layer. This indicates ResNet's ability to mitigate the vanishing gradient problem. It can avoid the narrow range by residual connection, enabling more stable training in deeper networks.

Conclusion

FFNet

Regarding using the Tanh activation function in FFNet, the 30 (Total layers is 33) to 21 (Total layers is 24) layers cause the gradient vanishing, while the 20 (Total layers is 23) layers do not cause the gradient vanishing. When the activation function is changed from Tanh to ReLU, the maximum number of stable layers decreases significantly. Although Tanh allows up to 20 (Total layers is 23) stable layers, ReLU limits the network to 7 (Total layers is 10) stable layers. It is due to the dying ReLU problem, where neurons with large negative activations produce zero outputs, leading to inactive neurons. The accumulation of these inactive neurons in deeper layers disrupts gradient flow, preventing effective learning and optimization.

ResNet

Concerning the residual connections, the ResNet maximum of layers is 74 (total layers is 301) when using the tanh activation function. This means that ResNet can support much deeper architectures than traditional networks to achieve higher accuracy and better generalization. The residual connection can provide a direct pathway that enables the gradient to avoid intermediate transformations that could shrink it. By doing this, it can avoid the gradient vanishing. Moreover, the ReLU activation function will also affect the ResNet depth, with the maximum layers from 74 (total layers is 301) dropping to 66 layers (total layers is 269). It is also due to the dying ReLU problem, leading to gradient flow instability and diminishing performance gains, ultimately limiting the network’s depth.