Introduction

Basics of Deep Learning

Learn a Saddle function

```
Z = 2X^2 - 3Y^2 + 1 + error
```

Load Libraries

In [2]:
```
# Dl & Numerical Library
import numpy as np
import pandas as pd

# Visualisation
import matplotlib.pyplot as plt
import altair as alt
from recoflow.vis import Vis3d
```

In [3]:
```
%matplotlib inline
```

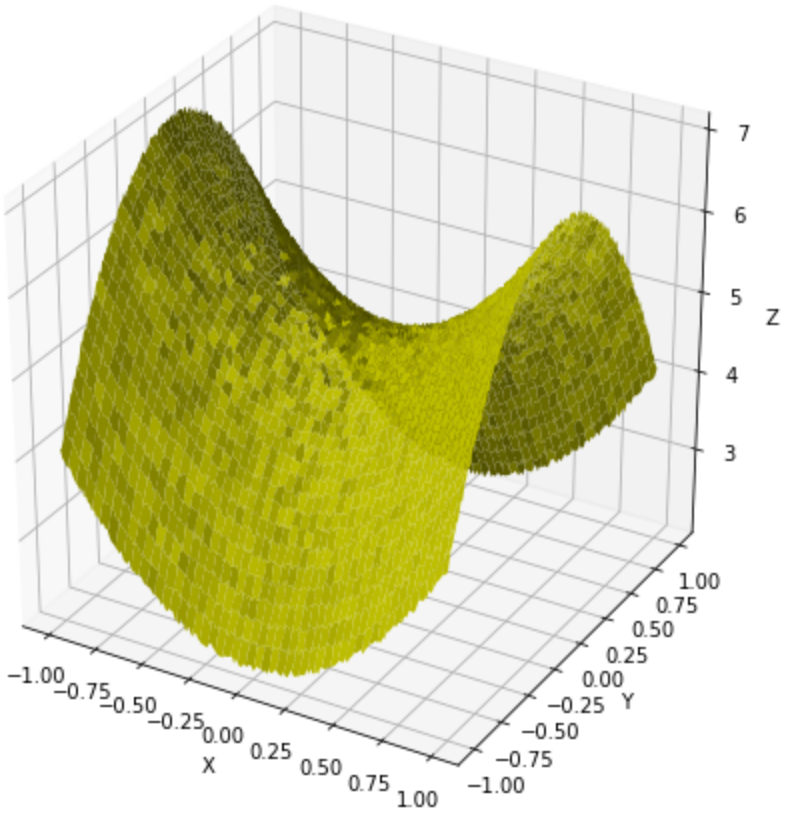
Create the Input and Output Data

In [4]:
```
x = np.arange(start = -1, stop = 1, step = 0.01)
y = np.arange(-1, 1, 0.01)
```

In [5]:
```
X, Y = np.meshgrid(x,y)
c = np.ones([200, 200])
e = np.random.rand(200, 200)*0.1
```

In [6]:
```
Z = 2*X*X - 3*Y*Y + 5*c + e
```

In [7]:
```
Vis3d(X,Y,Z)
```



Using Deep Learning

Step 0: Load Libraries

In [10]:
```
from keras.models import Model
from keras.layers import Dense, Input, Concatenate
```

Step 1: Build a Learning Architecture

In [40]:
```
def deep_learning_model():

    # Get the input
    x_input = Input(shape=[1], name="X")
    y_input = Input(shape=[1], name="Y")

    # Concatenate the input
    xy_input = Concatenate(name="Concat")([x_input, y_input])

    # Create Transform functions
    Dense_1 = Dense(32, activation="relu", name="Dense1")(xy_input)
    Dense_2 = Dense(4, activation="relu", name="Dense2")(Dense_1)

    # Create the Output
    z_output = Dense(1, name="Z")(Dense_2)

    # Create the Model
    model = Model([x_input, y_input], z_output, name="Saddle")

    # Compile the Model
    model.compile(loss="mean_squared_error", optimizer="sgd")

    return model
```

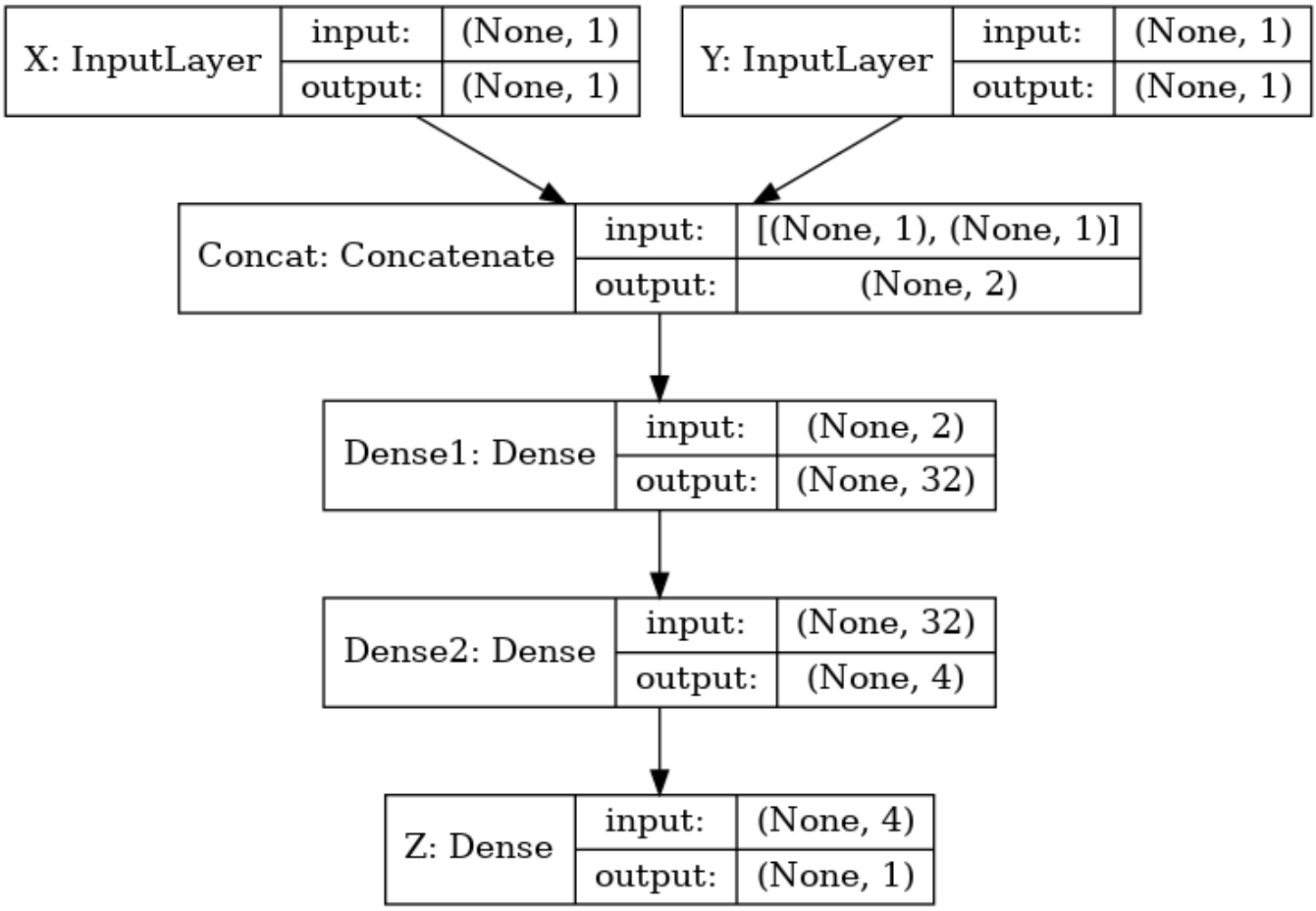
In [41]:
```
model = deep_learning_model()
model.summary()
```

```
Model: "Saddle"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
X (InputLayer)               (None, 1)                 0         
_________________________________________________________________
Y (InputLayer)               (None, 1)                 0         
_________________________________________________________________
Concat (Concatenate)         (None, 2)                 0         
_________________________________________________________________
Dense1 (Dense)               (None, 32)                96        
_________________________________________________________________
Dense2 (Dense)               (None, 4)                 132       
_________________________________________________________________
Z (Dense)                    (None, 1)                 5         
=================================================================
Total params: 233
Trainable params: 233
Non-trainable params: 0
_________________________________________________________________
```

In [42]:
```
from keras.utils import plot_model
```

In [43]:
```
plot_model(model, show_layer_names=True, show_shapes=True)
```

Out[43]:



Step 2: Learn the weights

In [44]:
```
input_x = X.reshape(-1)
input_y = Y.reshape(-1)
output_z = Z.reshape(-1)
```

In [45]:
```
X.shape, Y.shape, Z.shape
```

Out[45]:
```
((200, 200), (200, 200), (200, 200))
```

In [46]:
```
input_x.shape, input_y.shape, output_z.shape
```

Out[46]:
```
((40000,), (40000,), (40000,))
```

In [47]:
```
df = pd.DataFrame({"X": input_x, "Y": input_y, "Z": output_z})
```

In [48]:
```
df.head()
```

Out[48]:

	X	Y	Z
0	-1.00	-1.0	4.089405
1	-0.99	-1.0	3.979668
2	-0.98	-1.0	3.947342
3	-0.97	-1.0	3.897573
4	-0.96	-1.0	3.880772

In [49]:
```
%%time
output = model.fit( [input_x, input_y], output_z, epochs=10,
                    validation_split=0.2, shuffle=True, verbose=1)
```

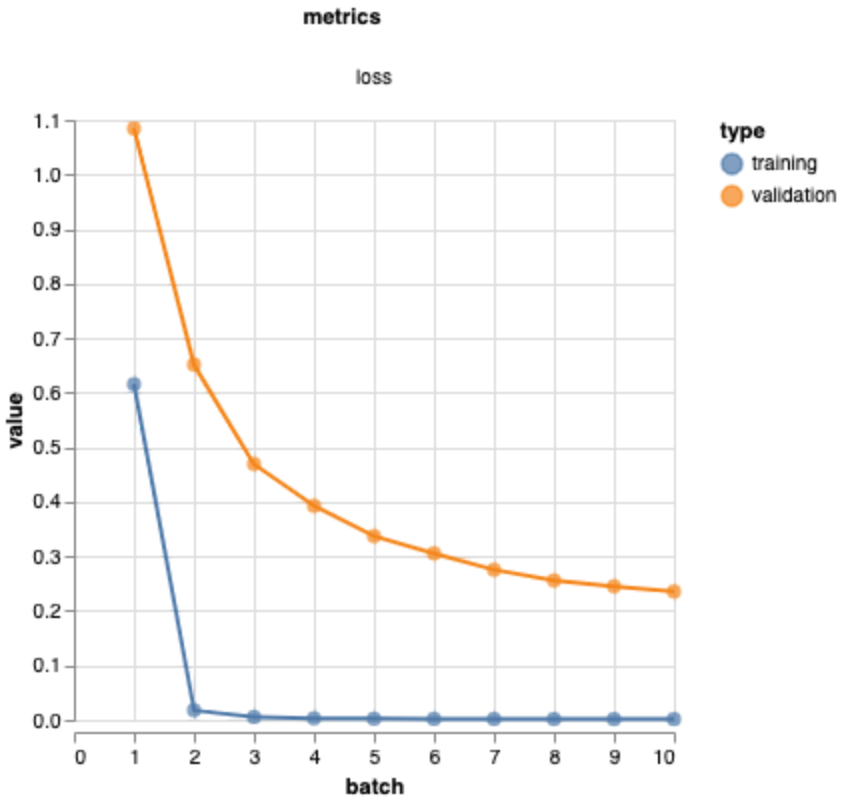
```
Train on 32000 samples, validate on 8000 samples
Epoch 1/10
32000/32000 [==============================] - 1s 30us/step - loss: 0.6155 - val
_loss: 1.0843
Epoch 2/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0178 - val
_loss: 0.6515
Epoch 3/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0055 - val
_loss: 0.4693
Epoch 4/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0030 - val
_loss: 0.3926
Epoch 5/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0023 - val
_loss: 0.3370
Epoch 6/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0020 - val
_loss: 0.3055
Epoch 7/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0018 - val
_loss: 0.2754
Epoch 8/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0017 - val
_loss: 0.2557
Epoch 9/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0016 - val
_loss: 0.2450
Epoch 10/10
32000/32000 [==============================] - 1s 27us/step - loss: 0.0016 - val
_loss: 0.2356
CPU times: user 12.2 s, sys: 1.28 s, total: 13.5 s
Wall time: 8.88 s
```

Step 4: Evaluate Model Performance

In [50]:
```
from recoflow.vis import MetricsVis
```

In [51]:
```
MetricsVis(output.history)
```

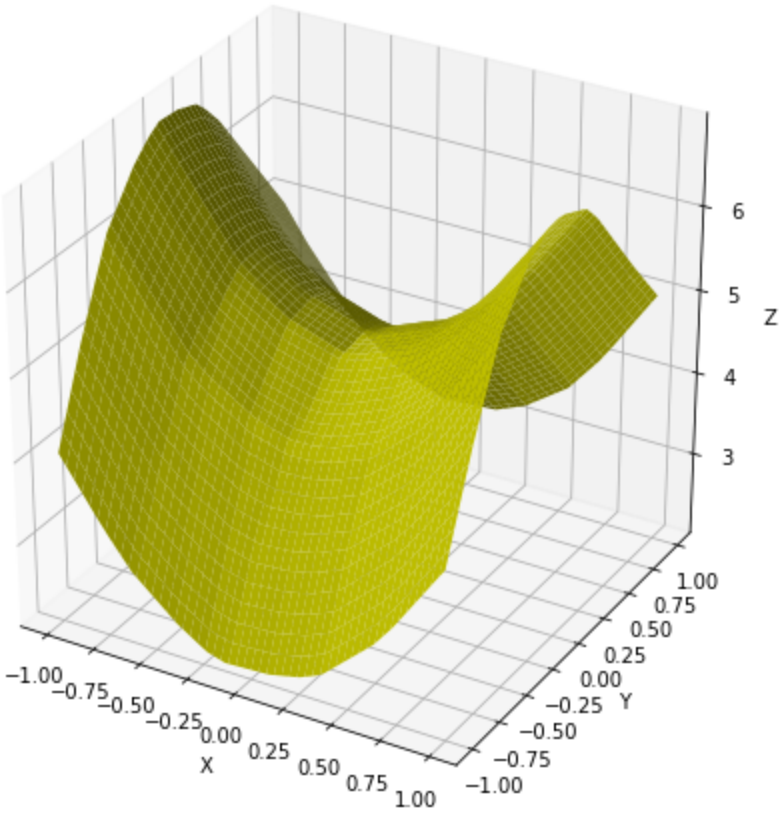
Out[51]:



Step 5: Make a Prediction

In [52]:
```
Z_pred = model.predict([input_x, input_y]).reshape(200,200)
```

In [53]:
```
Vis3d(X,Y,Z_pred)
```



Experimentation

- Change the number of layers: 2 -> 3
- Change the number of learning units in the layer: 32, 4 -> 16,2
- Change the activation function from `relu` to `linear`

In [ ]:
```
```