

**Name: Nikhilesh**

**Roll no: 64**

## **Experiment no:1**

### **1. Creating Tensors:**

#### **# Constant Tensor**

```
tensor = tf.constant(value, dtype=None, shape=None, name=None)
```

#### **# Variable Tensor**

```
variable = tf.Variable(initial_value, dtype=None, shape=None, name=None)
```

#### **# Placeholder (Used in older versions)**

```
placeholder = tf.compat.v1.placeholder(dtype, shape, name=None)
```

### **2. Mathematical Operations:**

#### **# Addition**

```
result = tf.add(tensor_a, tensor_b)
```

#### **# Subtraction**

```
result = tf.subtract(tensor_a, tensor_b)
```

#### **# Multiplication**

```
result = tf.multiply(tensor_a, tensor_b)
```

#### **# Division**

```
result = tf.divide(tensor_a, tensor_b)
```

#### **# Matrix Multiplication**

```
result = tf.matmul(matrix_a, matrix_b)
```

### **3. Activation Functions:**

#### **# ReLU (Rectified Linear Unit)**

```
result = tf.nn.relu(tensor)
```

#### **# Sigmoid**

```
result = tf.nn.sigmoid(tensor)
```

#### **# Softmax**

```
result = tf.nn.softmax(tensor)
```

#### **# Tanh (Hyperbolic Tangent)**

```
result = tf.nn.tanh(tensor)
```

### **4. Loss Functions:**

#### **# Mean Squared Error**

```
loss = tf.losses.mean_squared_error(labels, predictions)
```

#### **# Categorical Cross-Entropy**

```
loss = tf.losses.categorical_crossentropy(onehot_labels, logits)
```

#### **# Sparse Categorical Cross-Entropy (for integer labels)**

```
loss = tf.losses.sparse_categorical_crossentropy(labels, logits)
```

## 5. Optimizers:

### # Stochastic Gradient Descent (SGD)

```
optimizer = tf.keras.optimizers.SGD(learning_rate)
```

### # Adam Optimizer

```
optimizer = tf.keras.optimizers.Adam(learning_rate)
```

## 6. Defining a Model with Keras API:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units, activation='relu', input_shape=(input_dim,)),
    tf.keras.layers.Dense(output_dim, activation='softmax')
])
```

## 7. Compiling and Training the Model:

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_val, y_val))
```

## 8. Custom Training Loop:

```
optimizer = tf.keras.optimizers.Adam()

for epoch in range(epochs):
    with tf.GradientTape() as tape:
        predictions = model(x_train)
        loss = loss_function(y_train, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

## 9. Saving and Loading Models:

### # Saving the model

```
model.save('my_model')
```

### # Loading the model

```
loaded_model = tf.keras.models.load_model('my_model')
```