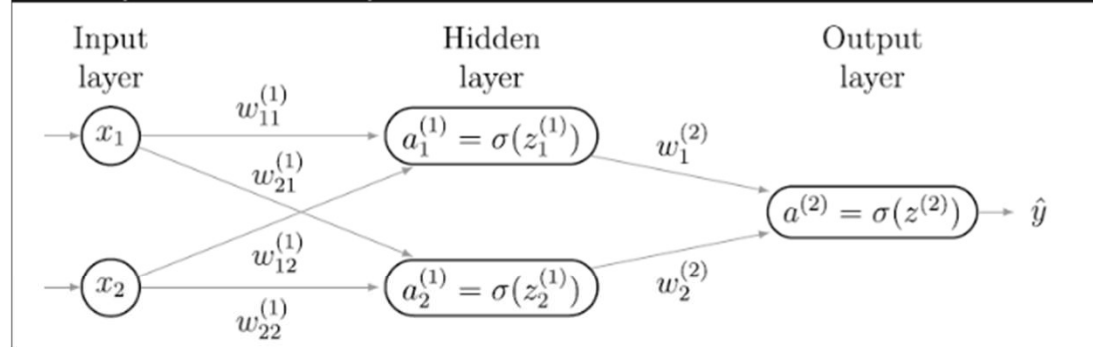


Planning of course:

- Dimensional reduction: singular value decomposition, principal component analysis, reduced order modeling
- Regression, optimization, model selection/performance
- Neural networks
- Physics informed neural networks
- Projects

Example forward propagation

For the simple neural network example



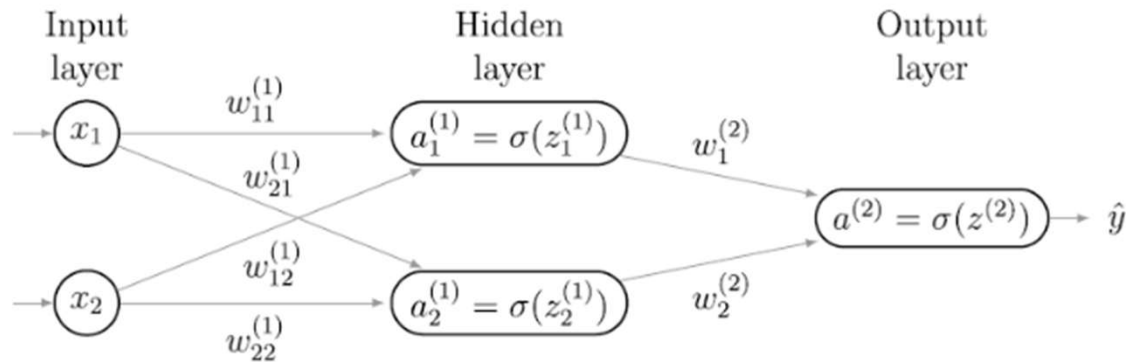
Assume $w_{11}^{(1)} = 1$, $w_{12}^{(1)} = -3$, $w_{21}^{(1)} = -2$, $w_{22}^{(1)} = 1$, $w_1^{(2)} = 2$, $w_2^{(2)} = -1$ and that the activation functions are the sigmoid. Compute the output \hat{y} for $x = [3, 2]$.

```
w1=np.array([[1.,-3.],[-2.,1.]])
w2=np.array([2.,-1.])
x=np.array([3.,2.])
z1=w1 @ x
a1=sigmoid(z1)
z2= np.dot(w2,a1)
yhat=sigmoid(z2)
print('z2=',z2)
print('output yhat= ',yhat)

#if we use function
def example1_fcn(x,w1,w2,fcn):
    z1=w1 @ x
    a1=fcn(z1)
    z2= np.dot(w2,a1)
    return fcn(z2)
print('Alternative: yhat=',example1_fcn(x,w1,w2,sigmoid))
```

```
z2= 0.076865536393042
output yhat= 0.5192069283210303
Alternative: yhat= 0.5192069283210303
```

How is the example network can be built in PyTorch



```
# Define a custom neural network class named 'MyNet'
class MyNet(nn.Module):
    # Constructor (__init__ method) for initializing the network
    def __init__(self):
        # Call the constructor of the parent class (nn.Module)
        super(MyNet, self).__init__()
        # Define the network layers and operations
        # First fully connected layer: 2 input feature, 2 output features
        self.fc1 = nn.Linear(2, 2)
        # Second fully connected layer: 2 input features, 1 output features
        self.fc2 = nn.Linear(2, 1)
        # Sigmoid activation function
        self.sigmoid = nn.Sigmoid()

    # Forward method defines how data flows through the network
    def forward(self, x):
        # Apply sigmoid activation to the first fully connected layer
        x = self.sigmoid(self.fc1(x))
        # Apply sigmoid activation to the second fully connected layer
        x = self.sigmoid(self.fc2(x))

        return x
```

```
# Create an instance of the SinNet model
model = MyNet()
```

```
custom_params = {
    'fc1.weight': torch.tensor([[1., -3.], [-2., 1.]], dtype=torch.float32),
    'fc1.bias': torch.tensor([0., 0.], dtype=torch.float32),
    'fc2.weight': torch.tensor([[2., -1.]], dtype=torch.float32),
    'fc2.bias': torch.tensor([0.], dtype=torch.float32),
}
```

```
# Set the model's state_dict to the custom parameter values
model.load_state_dict(custom_params)
```

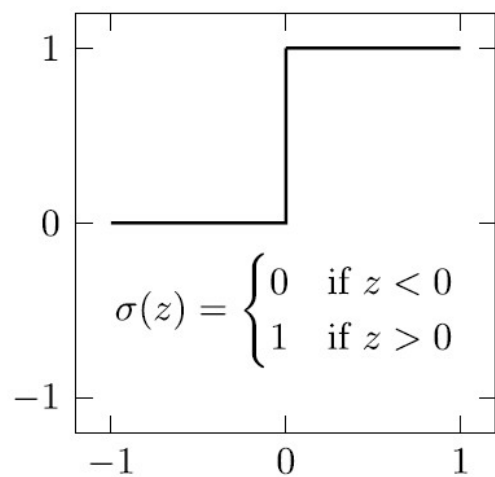
```
x = torch.tensor([3., 2.], dtype=torch.float32) # Explicitly set the data type to float32
```

```
# Ensure that the input has the correct shape
#x = x.view(1, -1)
```

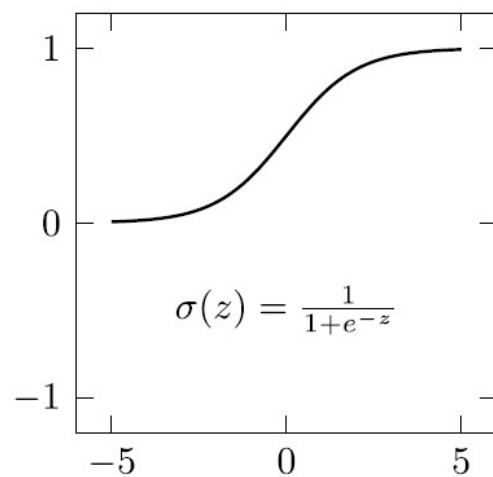
```
print(x)
output = model(x)
print(output)
```

```
tensor([3., 2.])
tensor([0.5192], grad_fn=<SigmoidBackward0>)
```

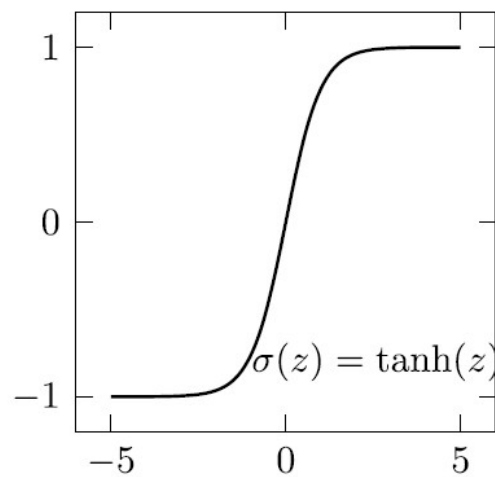
Perceptron



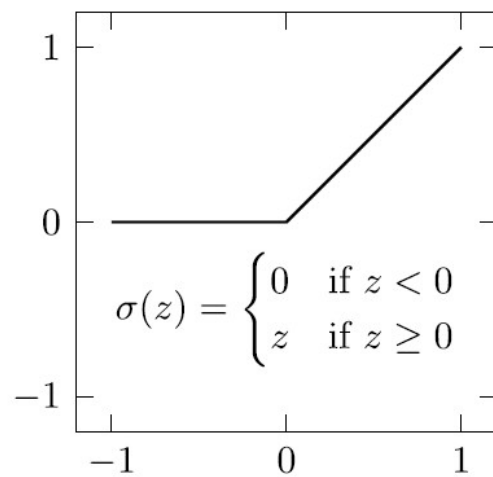
Sigmoid



Tanh



ReLU



Scaling class

change to e.g. -1, 1

```
class MinMaxScaler:
    def __init__(self, feature_range=(0, 1)):
        self.min_val = feature_range[0]
        self.max_val = feature_range[1]
        self.data_min_ = None
        self.data_max_ = None

    def fit(self, data):
        """Compute the minimum and maximum values for scaling."""
        self.data_min_ = np.min(data)
        self.data_max_ = np.max(data)

    def transform(self, data):
        """Scale the data using the computed min and max values."""
        # Avoid division by zero for constant columns
        scale = (self.max_val - self.min_val) / (self.data_max_ - self.data_min_ + 1e-8)
        return self.min_val + (data - self.data_min_) * scale

    def fit_transform(self, data):
        """Fit and then transform the data."""
        self.fit(data)
        return self.transform(data)

    def inverse_transform(self, scaled_data):
        """Revert the scaling to original values."""
        scale = (self.max_val - self.min_val) / (self.data_max_ - self.data_min_ + 1e-8)
        return (scaled_data - self.min_val) / scale + self.data_min_

# Example usage
data = np.array([[1.0, 2.0],
                 [2.0, 3.0],
                 [3.0, 4.0],
                 [4.0, 5.0]])

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

print("Original data:\n", data)
print("Scaled data:\n", scaled_data)

# Inverse transform to get the original data
original_data = scaler.inverse_transform(scaled_data)
print("Inverse transformed data:\n", original_data)
```

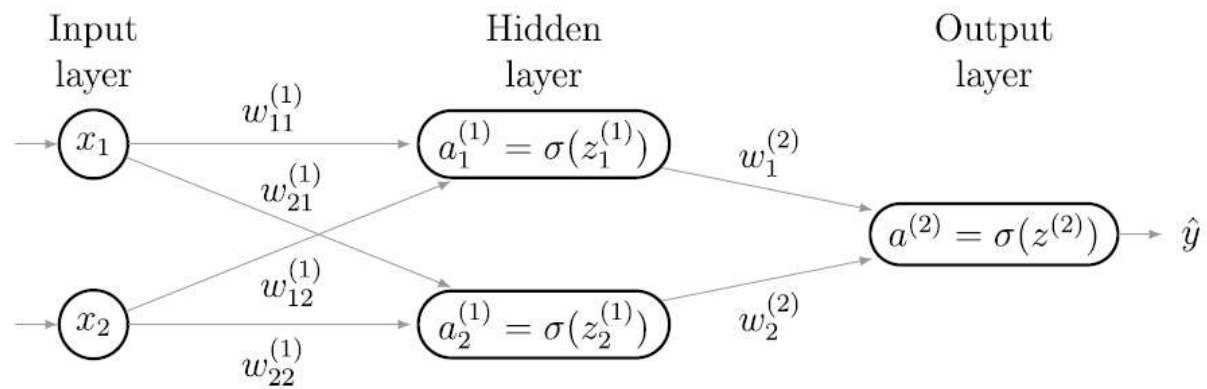


Fig. 3.5 A simple feed-forward neural network example

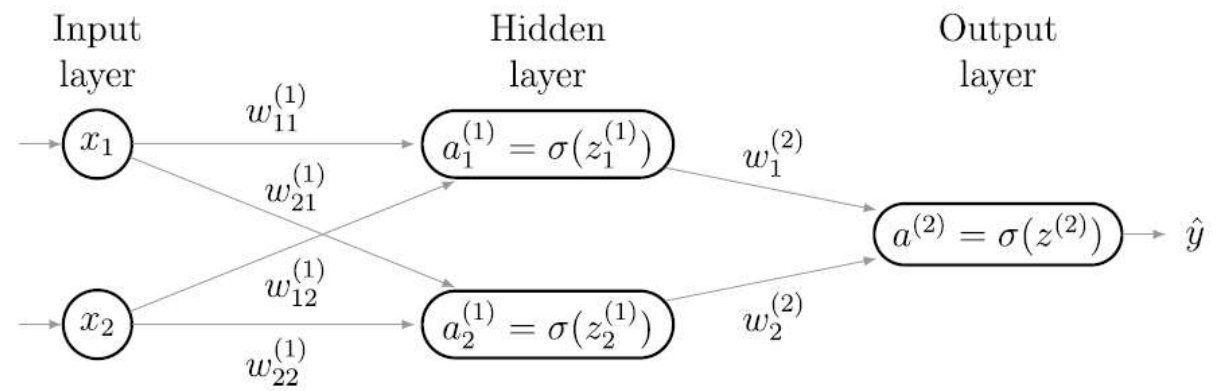


Fig. 3.5 A simple feed-forward neural network example


```

import torch
x = torch.tensor([3.,2.])
W1=torch.tensor([[1.,-3.],[-2.,1.]], requires_grad=True)    #We want to compute gradient w.r.t. W1
z1=torch.matmul(W1,x)
#Definition of activation function with torch
def sigmoid(x):
    return 1 / (1 + torch.exp(-x))
a1=sigmoid(z1)
print('a1=',a1)
w2=torch.tensor([2.,-1.],requires_grad=True) #We want to compute gradient w.r.t. w2
z2= torch.dot(w2,a1)
C=1./2.*(y-sigmoid(z2))**2. #Computation of cost function
print('C=',C)
# Perform automatic differentiation
C.backward()
# Print the gradients w.r.t. W1 and w2
print(W1.grad)
print(w2.grad)

```

```

a1= tensor([0.0474, 0.0180], grad_fn=<MulBackward0>)
C= tensor(0.0240, grad_fn=<MulBackward0>)
tensor([[ 0.0148,  0.0099],
        [-0.0029, -0.0019]])
tensor([0.0026, 0.0010])

```