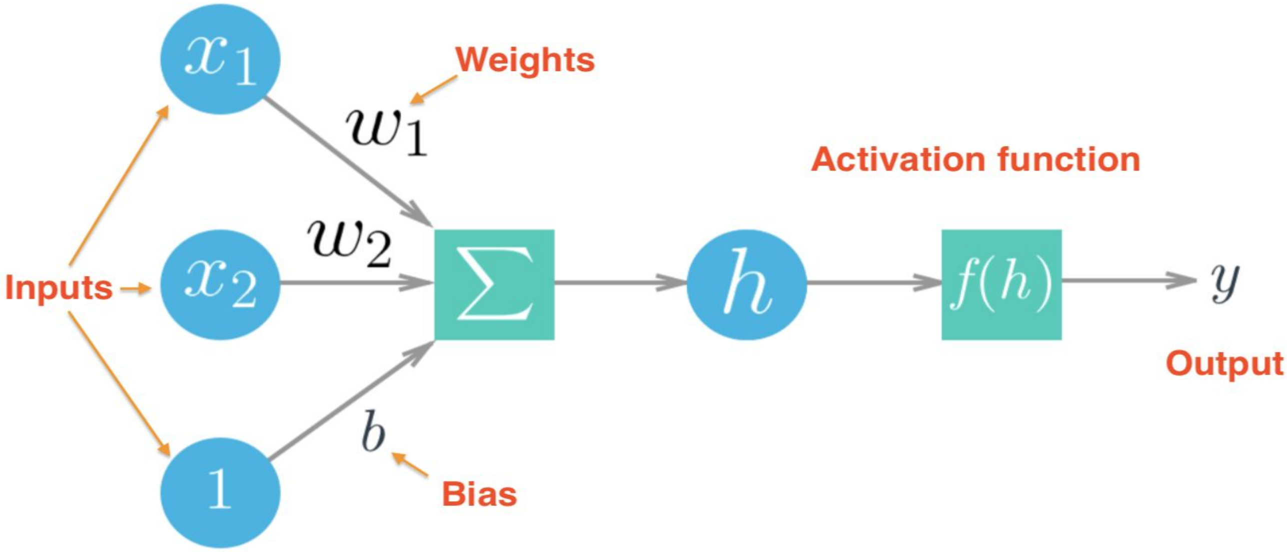
Welcome

- PyTorch was developed as an open source by Facebook AI Research team: https://ai.facebook.com/
- Important questions to watch out for throughout this lesson:
    - Tensors: main data structure of PyTorch
    - How to create tensors ?
    - How to do operations on tensors ?
    - How tensor interact with NumPy ?
    - What is PyTorch module: Autograd ? it is used to calculate gradients for training neural network
        - This autograd module also do backpropagation for us; it calculates gradient at each operation and update the network weights
    - How to build a network with PyTorch
    - How to run data through it
    - How to define a loss
    - How to define optimization method
    - How to do validation to test that your network is able to generalize
    - How to use transfer learning technique ? it use pre-trained network to improve the performance of classifier

Single layer neural networks

- **Problem**: calculate output of a single neural network with PyTorch
- These Numpy arrays are just tensors. PyTorch takes these tensors and makes it simple to move them to GPUs for faster processing needed when training neural networks



$$y = f(w_1x_1 + w_2x_2 + b)$$

$$y = f(\sum_i w_ix_i + b);\ i \textit{ is number of inputs};\ \textit{in image above } 2$$

$$\sum_i w_ix_i = [x_1 \ldots\ldots x_n] \ . \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$; “.” is dot/inner product of vectors

- **Understand Tensor**

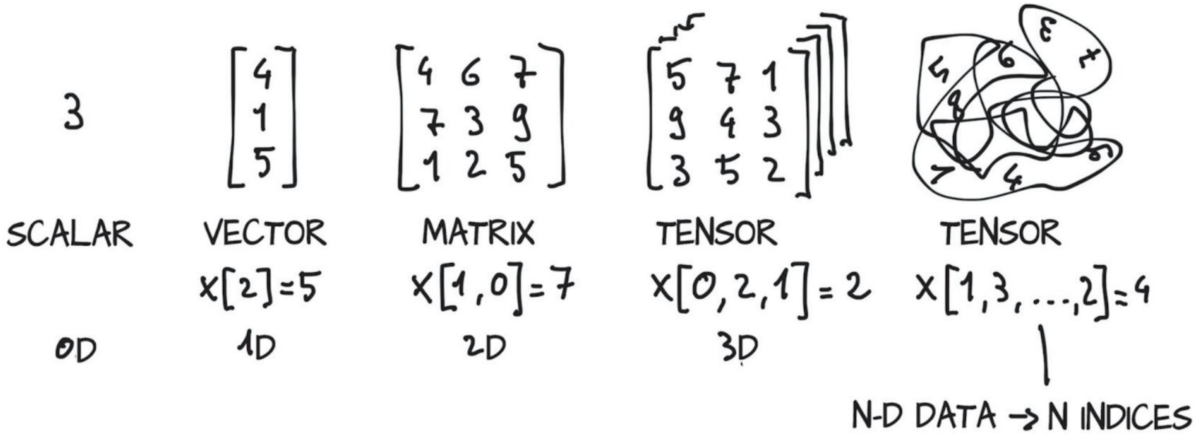


Figure 2.2 Tensors are the building blocks for representing data in PyTorch

- Resource to understand more about Tensor: https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf

- **Build a simple neural network (Single Neuron)**

- **Step 1**: In real world project, we should already have the data and just load the data; here in this lesson, we do not. Since we need data (input and weight and bias) for testing this, we will use PyTorch to generate random values

```
torch.manual_seed(7) # Set the random seed so things are
predictable

# Features are 5 random normal variables
features = torch.randn((1, 5))

# True weights for our data, random normal variables again
weights = torch.randn_like(features)
# and a true bias term
bias = torch.randn((1, 1))
```

  - `manual_seed` method is used to set the random seed from pytorch random number generators; this ensures that PyTorch will set the seed of the random number generator to a fixed value, so that when you re-execute the cell, it provide the same random numbers http://pytorch.org/docs/master/torch.html?highlight=manual_seed#torch.manual_seed

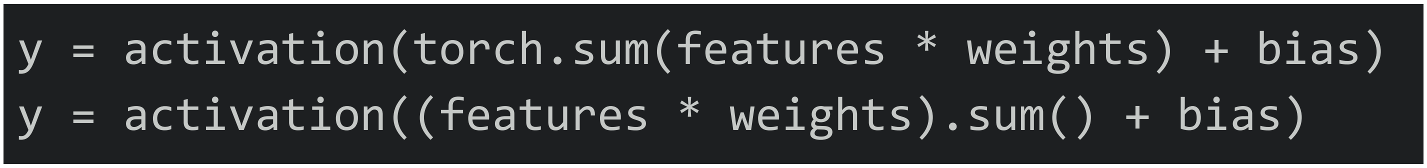
  - `randn` method is used to generate random value by Pytorch random number generator
  - `Randn_like` method is used to create a tensor with the same shape of the input

- **Step 2**: now that we have data for input (features), weights and bias, we will find the output (label) y

- Calculate the linear combination of input values

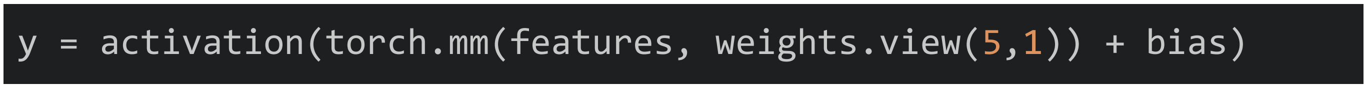
  $h = \sum_i w_i x_i + b$; i is the number of inputs

- Then apply activation (in this case sigmoid function) to h



```
y = activation(torch.sum(features * weights) + bias)
y = activation((features * weights).sum() + bias)
```

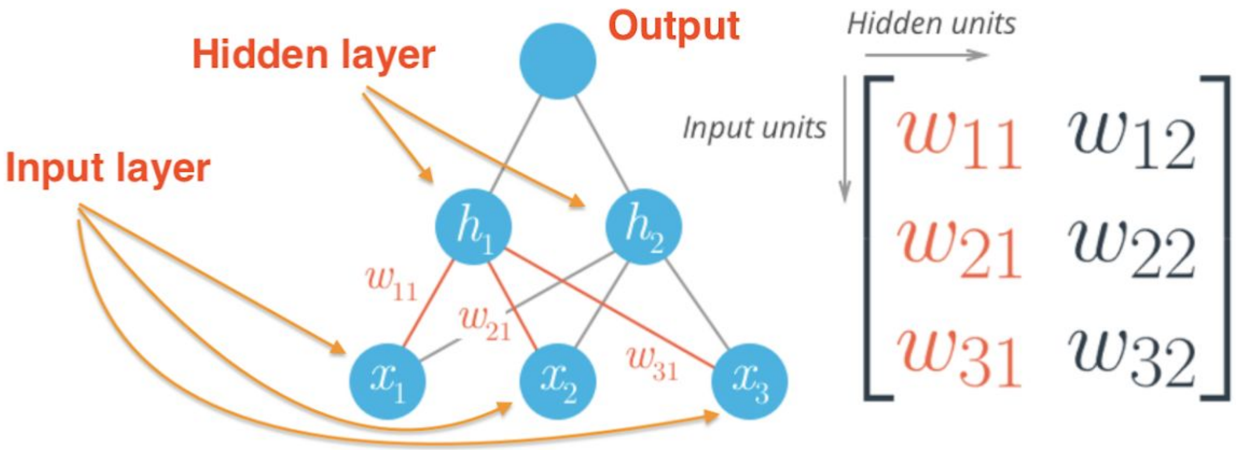
- It’s recommended to use Pytorch matrix multiplication since they are more efficient and accelerated using modern libraries and high-performance computing on GPUs. Matrix multiplication: torch.mm() or torch.matmul()
- **Issue:** to be aware when using method mm() or matmu(): size mismatch between two arguments; in other words, the two inputs don’t have correct shape.
- **Solution**
  - Use method shape() to see the shape of inputs
  - Reshape the tensor to get desire shape; in this case we want to reshape weights; three options:
    - weights.reshape(a, b)
    - weights.resize_(a, b)
    - weights.view(a, b)



```
y = activation(torch.mm(features, weights.view(5,1)) + bias)
```

- **Stack them up**

- Problem: How can we calculate this ?



- Solution:
  - Calculate linear combinations for each unit in one operation

$$\vec{h} = [h_1 \; h_2] = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ \vdots & \vdots \\ w_{n1} & w_{n2} \end{bmatrix}$$

  - Calculate output y $$y = f_2\left(f_1\left(\vec{x}\,\mathbf{W_1}\right) \mathbf{W_2}\right)$$

```
h = activation(torch.mm(features, W1) + B1)
output = activation(torch.mm(h, W2) + B2)
```

- Hyperparameter: The number of hidden units a parameter of the network, it is different from weights and bias parameters

    - the more hidden units a network has, and the more layers, the better able it is to learn from data and make accurate predictions.

- **Numpy to Torch and back**
    - Create tensor from numpy array

        torch.from_numpy()

    - Convert tensor to a numpy

        torch.numpy()

The memory is shared between the Numpy array and Torch tensor, so if you change the values in-place of one object, the other will change as well.

    - Numpy random rand
      https://docs.scipy.org/doc/numpy-1.14.1/reference/generated/numpy.random.rand.html

===

Mini summary

- We addressed these questions
    - Tensors: main data structure of PyTorch
    - How to create tensors ?
    - How to do operations on tensors ?
    - How tensor interact with NumPy ?
- How to build a simple neuron
- The power of stack up neural network