Maddy Walkington

Intro to Neural Networks, PyTorch

Neural Networks...

Task:

- Classification (binary & multi-class)

- Regression

Captcha: Identifying if a hydrant is present in a set of images

Categorizing a set of music files into their respective genres

Predicting the real estate price based off size & features of a house

How it works

First, you need a large **labelled** dataset split into:

Training Data ~70%

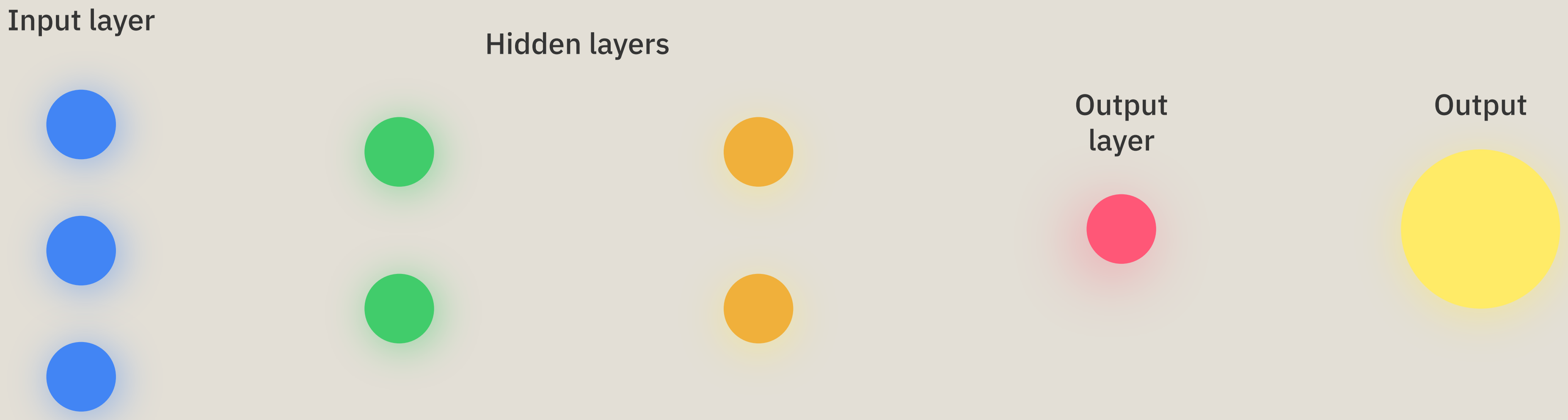
Testing Data ~30%

Labelled: Must contain both the inputs to the network, as well as it's desired output

Large datasets of interesting things can be found online!

How it works

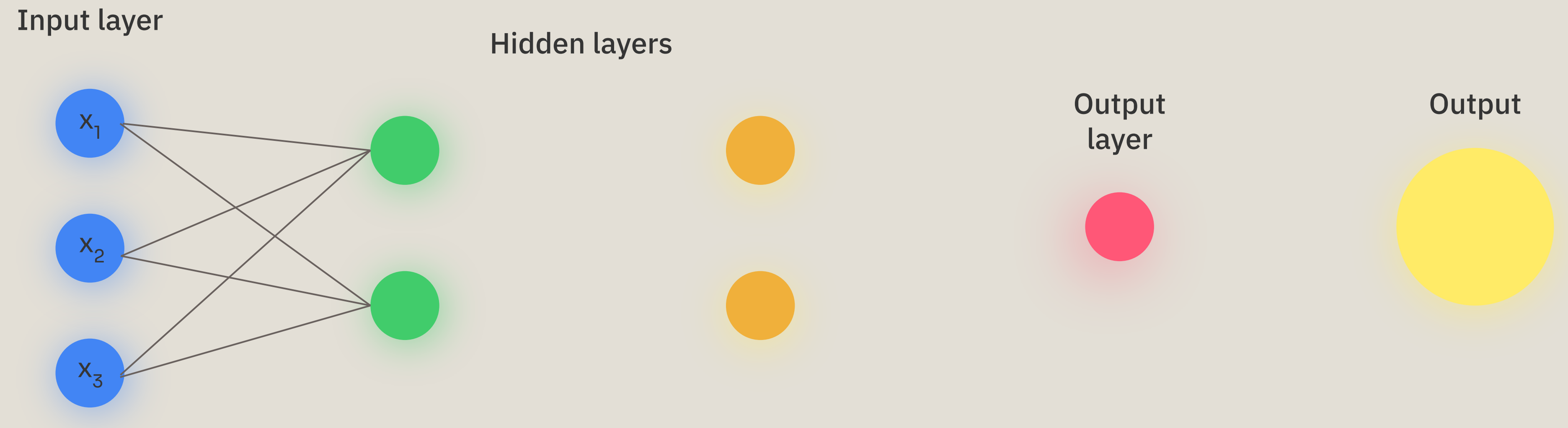
Empty NN is initialized with a certain number of hidden layers & neurons



How it works

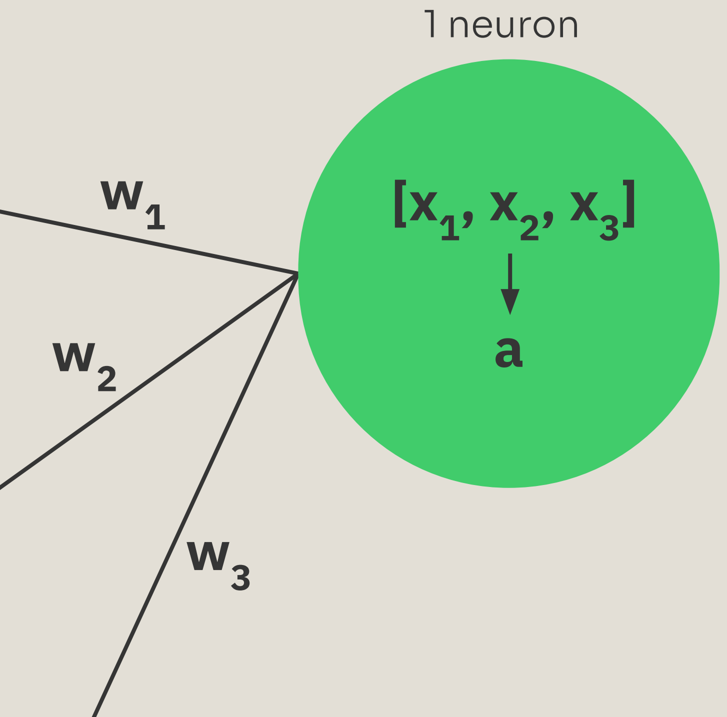
Input data is vectorized:

$x = [x_1, x_2, \ldots, x_n]$



Then, each component is fed to the first layer

Zooming in...



Makes a linear combination of the components:

$$z = [w_1x_1 + w_2x_2 + \dots + w_nx_n] + b$$

Passes the result through an “activation function”:

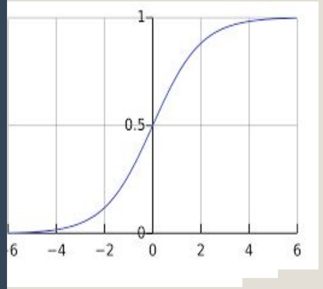
$$a = \sigma(z)$$

Zoomin

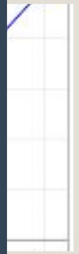
:

xamples:

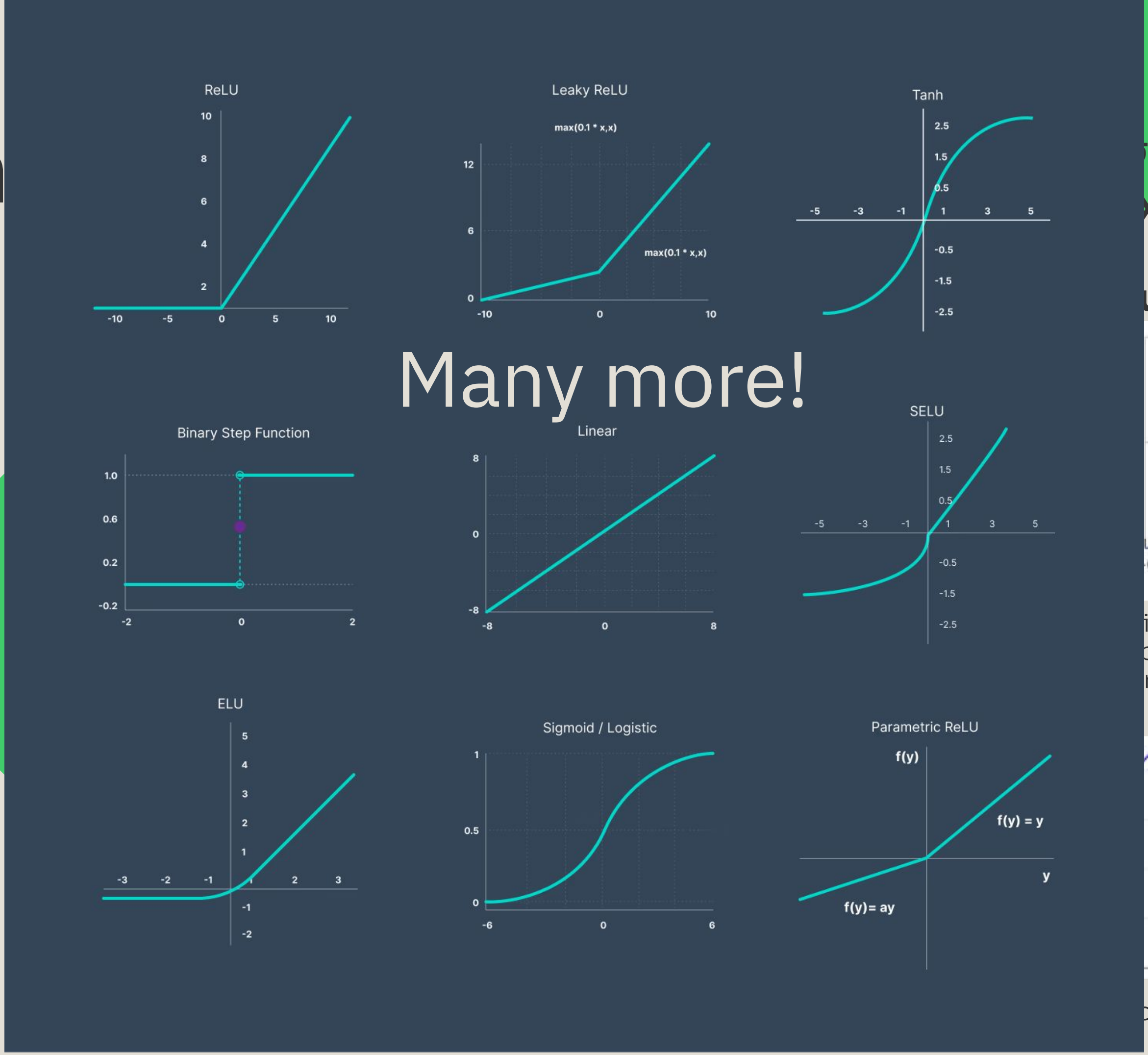
ulti-class



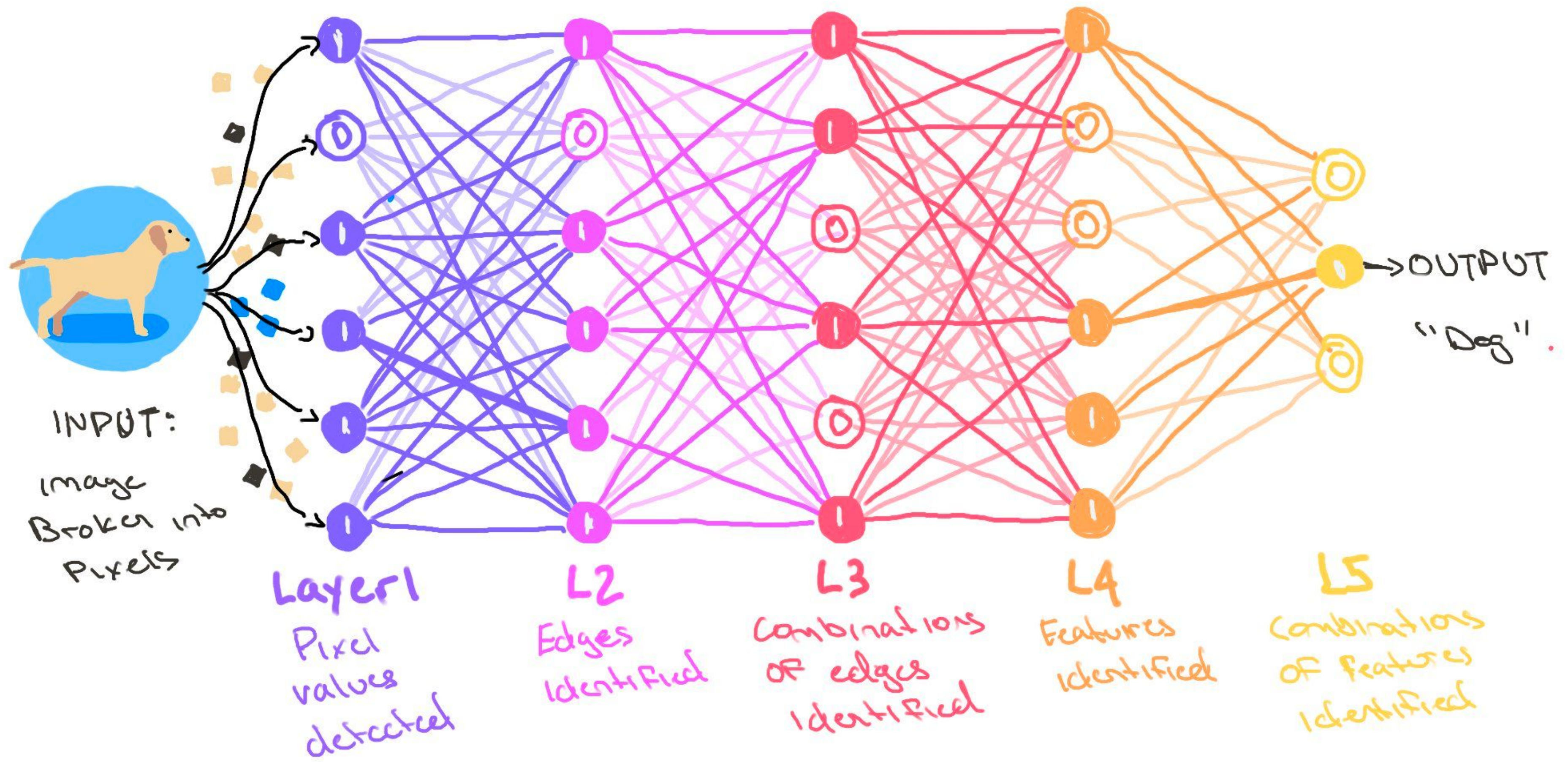
i-dimensional sigmoid,
probability for each
n, all summing to 1



omain)

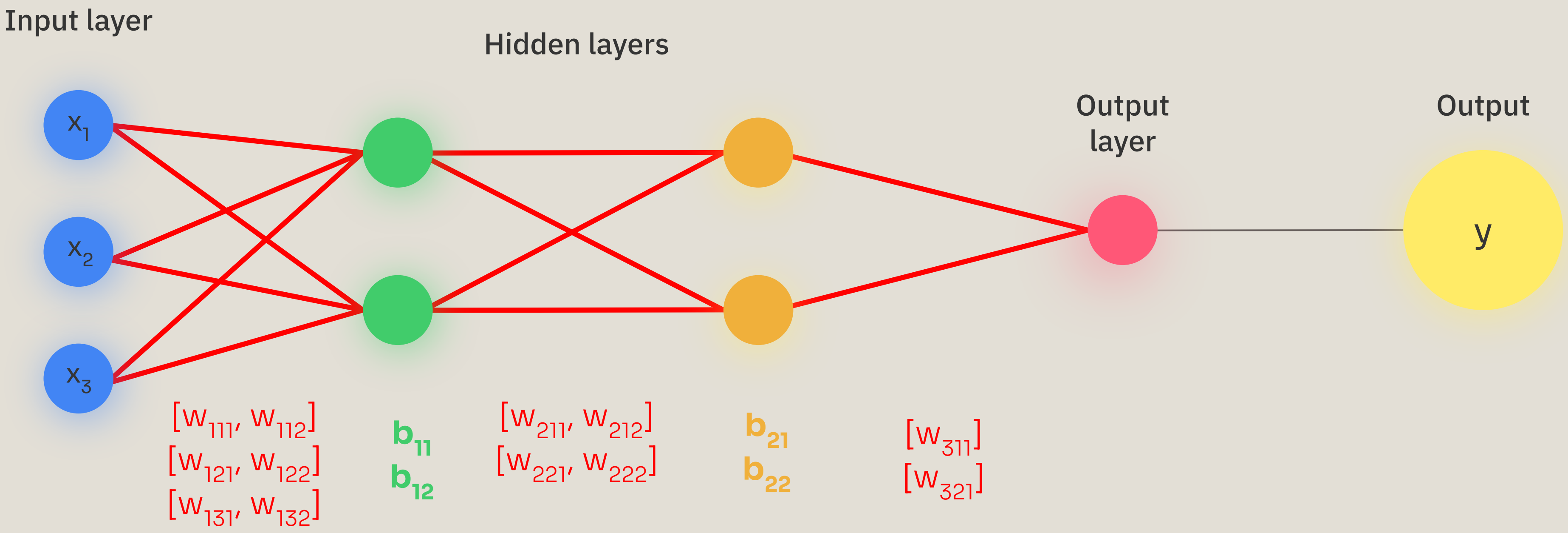


Many more!


INPUT:
Image Broken into Pixels
Layer1
Pixel values detected
L2
Edges identified
L3
Combinations of edges identified
L4
Features identified
L5
Combinations of features identified
OUTPUT
"Dog"

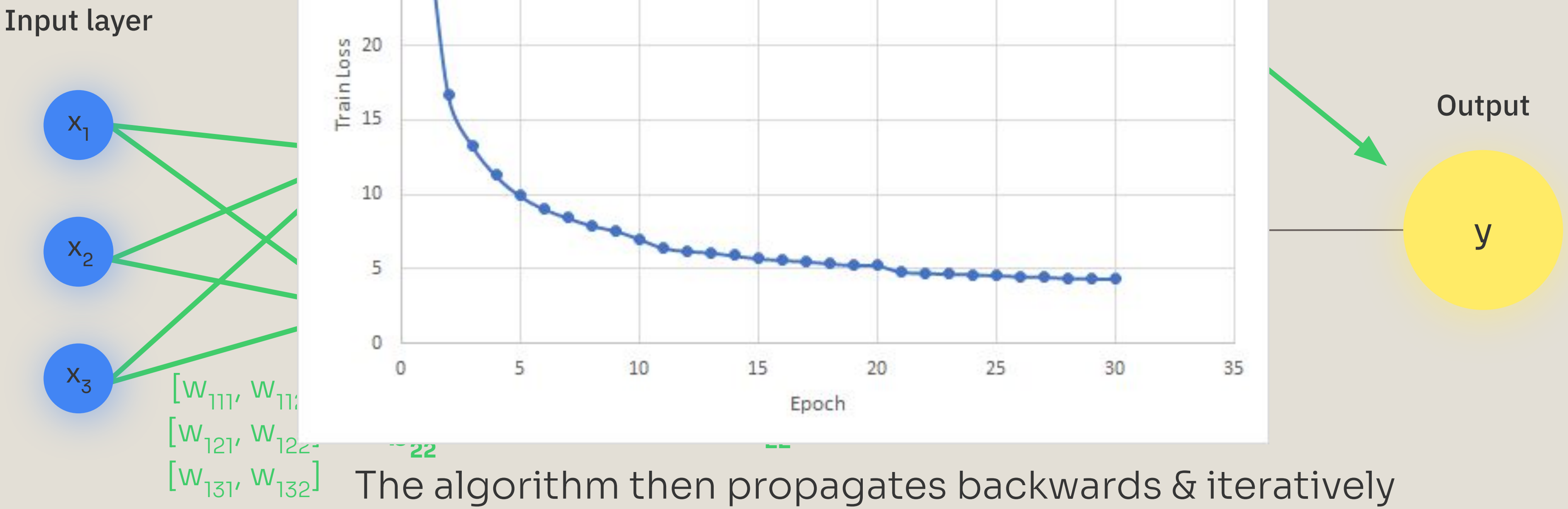
How it works

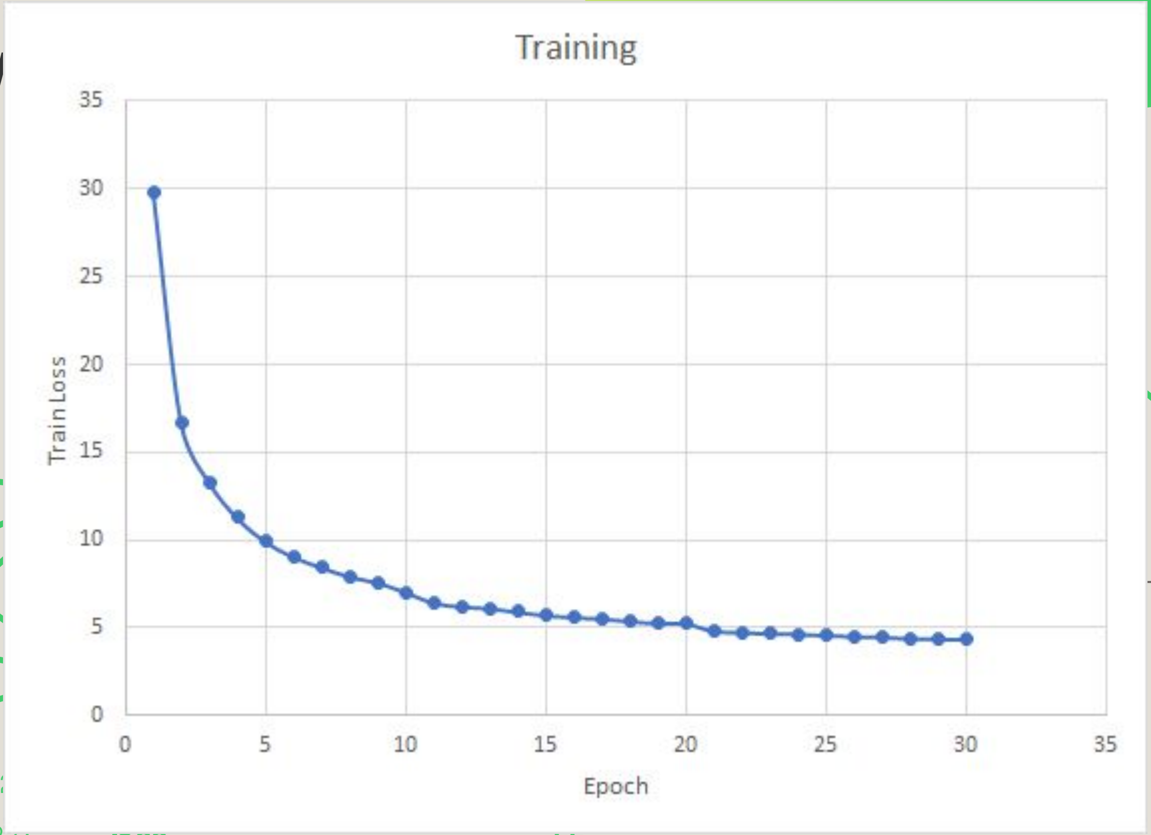
In the end...



How it w

T

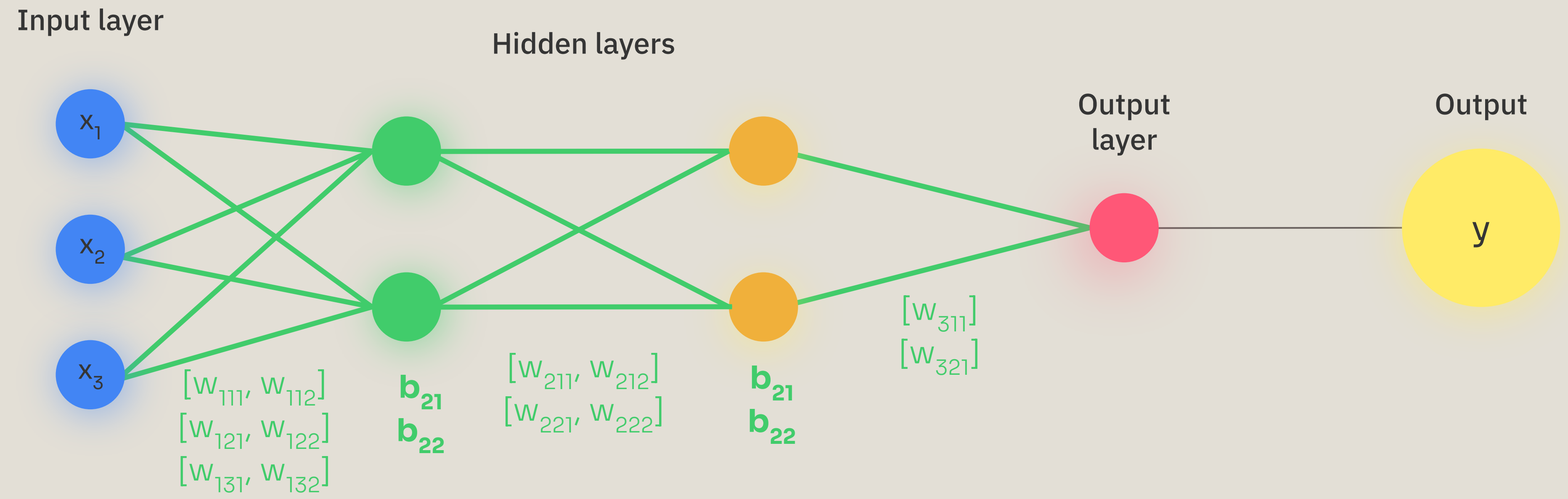




The algorithm then propagates backwards & iteratively adjusts all the weights and biases until the “loss” is minimized

How it works

Once the optimal set of weights and biases are found for the entire training set, the network is fed the testing set



Most Common Types of Neural Networks

MLP

Multi-Layer Perceptron

Most basic type, essentially functions as previously described (feed-forward + back propagation)

CNN

Convolutional Neural Network

Used for grid-like data (such as images). Utilizes convolutional layers to detect spatial hierarchies & features

RNN

Recurrent Neural Network

Used for sequential data (such as audio files, text). Uses feedback loops to allow information to persist long-term & detect large patterns in sequences

PyTorch

Fantastic DL library for Python

Installation

```
pip3 install torch torchvision
```

Python 3.10–3.14 recommended

PyTorch

Fantastic DL library for Python

Example MLP implementation

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

Create synthetic test+train dataset

-Creates 1000 samples with 20 features each, as well as 2 labels

-Split randomly into 80/20 training and testing datasets

-Convert the vectors into tensors

```
X, y = make_classification(
   n_samples=1000,
   n_features=20,
   n_informative=15,
   n_redundant=5,
   n_classes=2,
   random_state=42
)

# Train / test split
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42
)

# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)
```

Example MLP implementation

-Create an MLP class with 2 hidden layers both utilizing the ReLU activation function

-Initialize the MLP with 20 input features, 64 neurons in each hidden layer, and 2 output dimensions

```python
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, output_dim)
        )

    def forward(self, x):
        return self.net(x)

model = MLP(input_dim=20, hidden_dim=64, output_dim=2)
```

Example MLP implementation

-Setup how the loss is measured and how the weights/biases are optimized

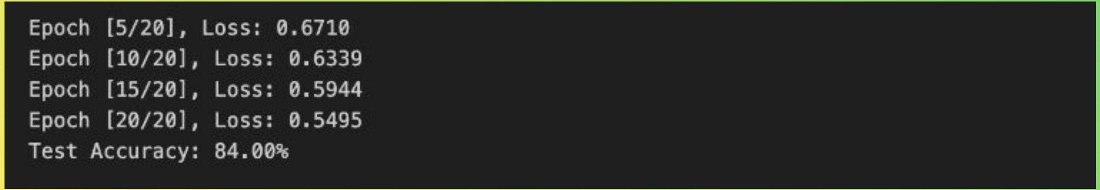
-Setup a training loop with 20 iterations, then run the loop on the training set (and print the loss)

```
# Training setup
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Training loop
num_epochs = 20

for epoch in range(num_epochs):
   model.train()

   optimizer.zero_grad()
   outputs = model(X_train)
   loss = criterion(outputs, y_train)
   loss.backward()
   optimizer.step()

   if (epoch + 1) % 5 == 0:
       print(f"Epoch [{epoch+1}/{num_epochs}], Loss:
{loss.item():.4f}")
```

Example MLP implementation

-Evaluate the model on the testing dataset and print it’s accuracy

```python
model.eval()
with torch.no_grad():
   outputs = model(X_test)
   _, predicted = torch.max(outputs, dim=1)
   accuracy = (predicted == y_test).float().mean()

print(f"Test Accuracy: {accuracy.item() * 100:.2f}%")
```



```
Epoch [5/20], Loss: 0.6710
Epoch [10/20], Loss: 0.6339
Epoch [15/20], Loss: 0.5944
Epoch [20/20], Loss: 0.5495
Test Accuracy: 84.00%
```

Thank you