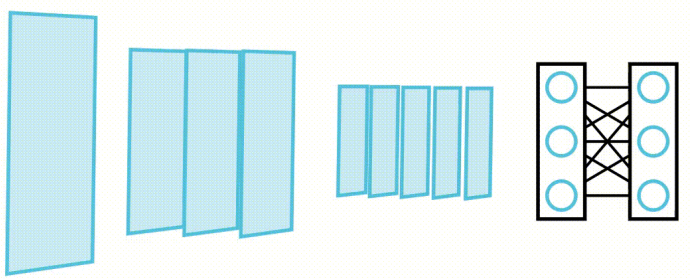
🔥 Deep Learning Workshop



Aron van de Pol

Leiden University Centre For Digital Humanities

2024-03-05

Non linearity

Today

- Classification tasks
- Focus on Model architecture
- Activation Functions

⭕ Linearity & Non linearity

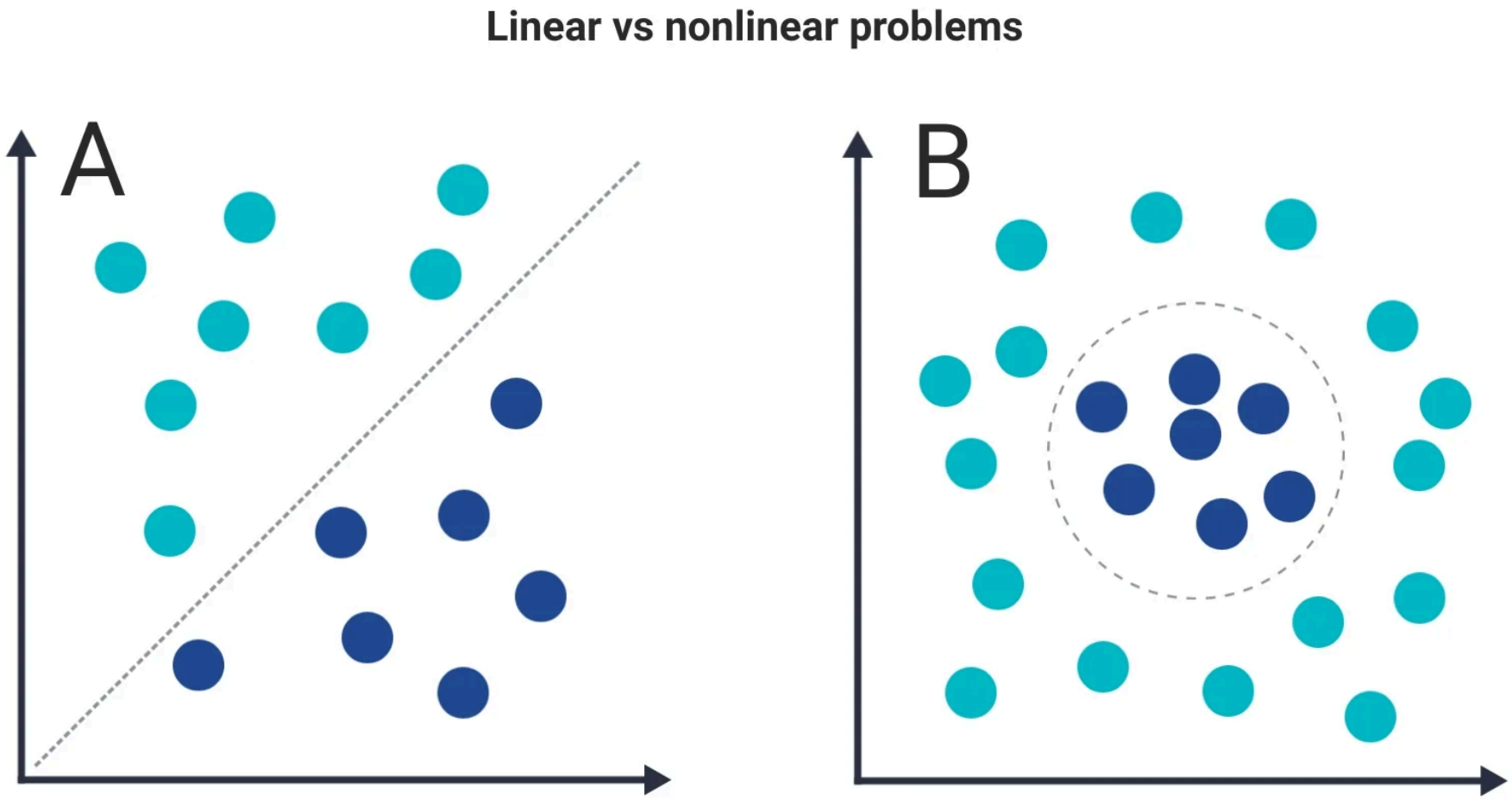
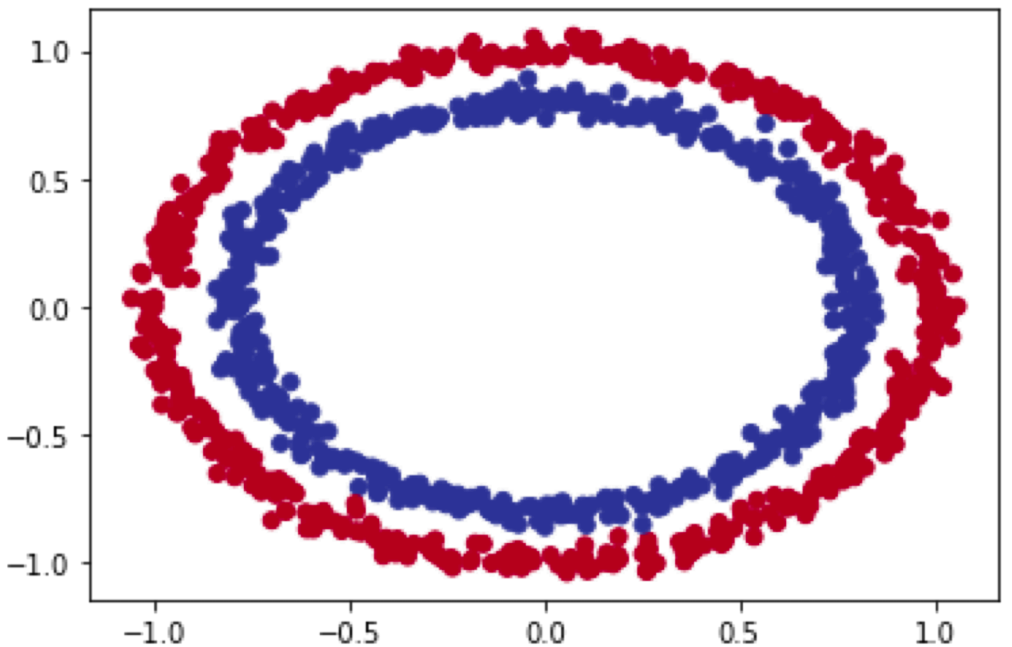


Figure 1: Linear & Non-linear dataset

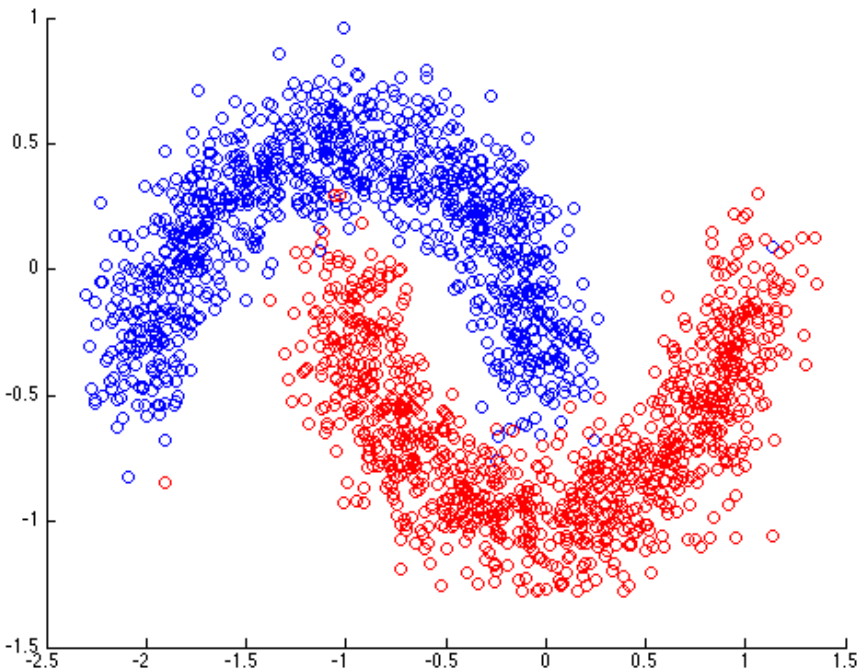
Classification

Problem type	What is it?	Example
Binary classification	Two classes	yes or no, class 0 or 1
Multi-class classification	One of multiple classes	class 0/1/2/3/4 etc. (**OR**)
Multi-label classification	One or more classes	class 0/1/2/3/4 etc. (**AND**)

Datasets



(a) `make_circles()` *dataset*



(b) `make_moons()` *dataset*

Figure 2: Examples of Non-linear datasets.

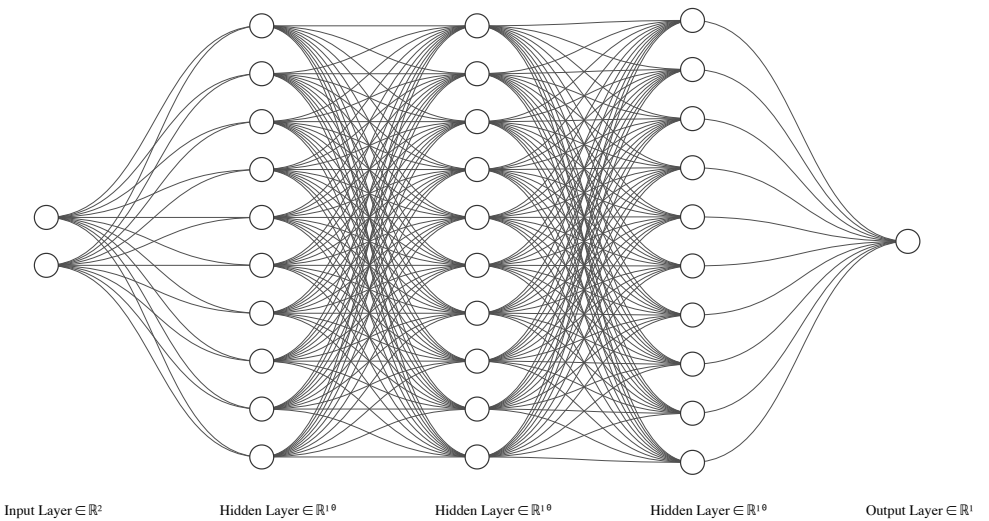
Model Construction - Base model

```
from torch import nn

class NL_ModelV0(nn.Module):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(in_features=2, out_features=10), # 2 > 10
            nn.Linear(in_features=10, out_features=10), # 10 > 10
            nn.Linear(in_features=10, out_features=1), # 10 > 1
            )

    # Define a forward method containing the forward pass computati
    def forward(self, x):
        return self.network(x)
```

Model Construction - Base model



A visualization of the produced network.

Model Construction - Base model

There are a few things to note on the correspondence of our code and the visualization:

- Initial data points align with the first layer's `in_features`.
- Hidden layers connect via matching `out_features` and `in_features`.
- Final `out_features` equals the number of prediction classes. For binary classification, a single neuron indicates class 0 (inactive) or 1 (active).

Training for Classification

🔢 Logits

- **logits** are the raw output of a model
- Hard to interpret alone
- Need to be transformed

🔢 Logits

```
tensor([[-0.4279],
        [-0.3417],
        [-0.5975],
        [-0.3801],
        [-0.5078]], device='cuda:0', grad_fn=<SliceBackward0>)
```

🔢 Logits

Sum outputs to one!

Done with the `torch.sigmoid()` function.

```
y_pred_probs = torch.sigmoid(y_logits)
y_pred_probs

# Returns:
tensor([[0.3946],
        [0.4154],
        [0.3549],
        [0.4061],
        [0.3757]], device='cuda:0', grad_fn=<SigmoidBackward0>)
```

🔢 Logits

- If `y_pred_probs` >= 0.5, y=1 (class 1)
- If `y_pred_probs` < 0.5, y=0 (class 0)

```
# In full
y_pred_labels = torch.round(torch.sigmoid(model_0(X_test.to(device))

# Get rid of extra dimension
y_preds.squeeze()

# Returns:
tensor([0., 0., 0., 0., 0.], device='cuda:0', grad_fn=<SqueezeBackw
```

🔢 Logits

1. we converted our model's raw outputs (logits) to prediction probabilities using a sigmoid activation function.
2. Then converted the prediction probabilities to prediction labels by rounding them.

It is important to be aware of that `torch.sigmoid()` is only used for binary classification. In the next sessions we will almost always work on multi-class problems which require

BCELoss

- Last session we used MAE (for regression)
- This time we are predicting classes, not a data point
- Binary Cross Entropy loss as solution

BCELoss

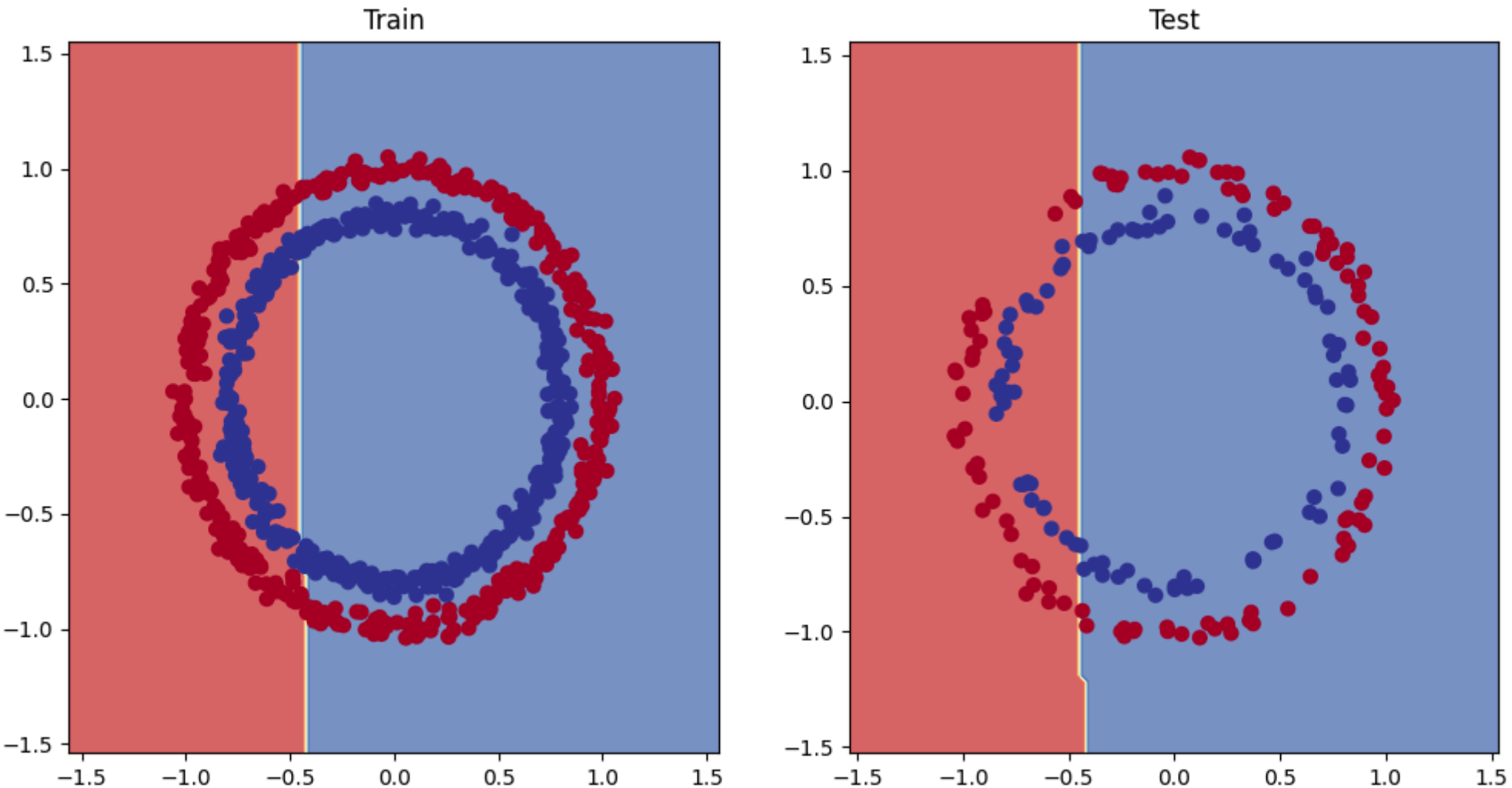
In pytorch we have two versions.

1. `torch.nn.BCELoss()` - Creates a loss function that measures the binary cross entropy between the target (label) and input (features).
2. `torch.nn.BCEWithLogitsLoss()` - This is the same as above except it has a sigmoid layer (`nn.Sigmoid`) built-in.

🤔 Linear models in a Non-linear dataset.

- Usually reaches between 40-70% accuracy
- Especially with unbalanced datasets
- why?

🤔 Linear models in a Non-linear dataset.

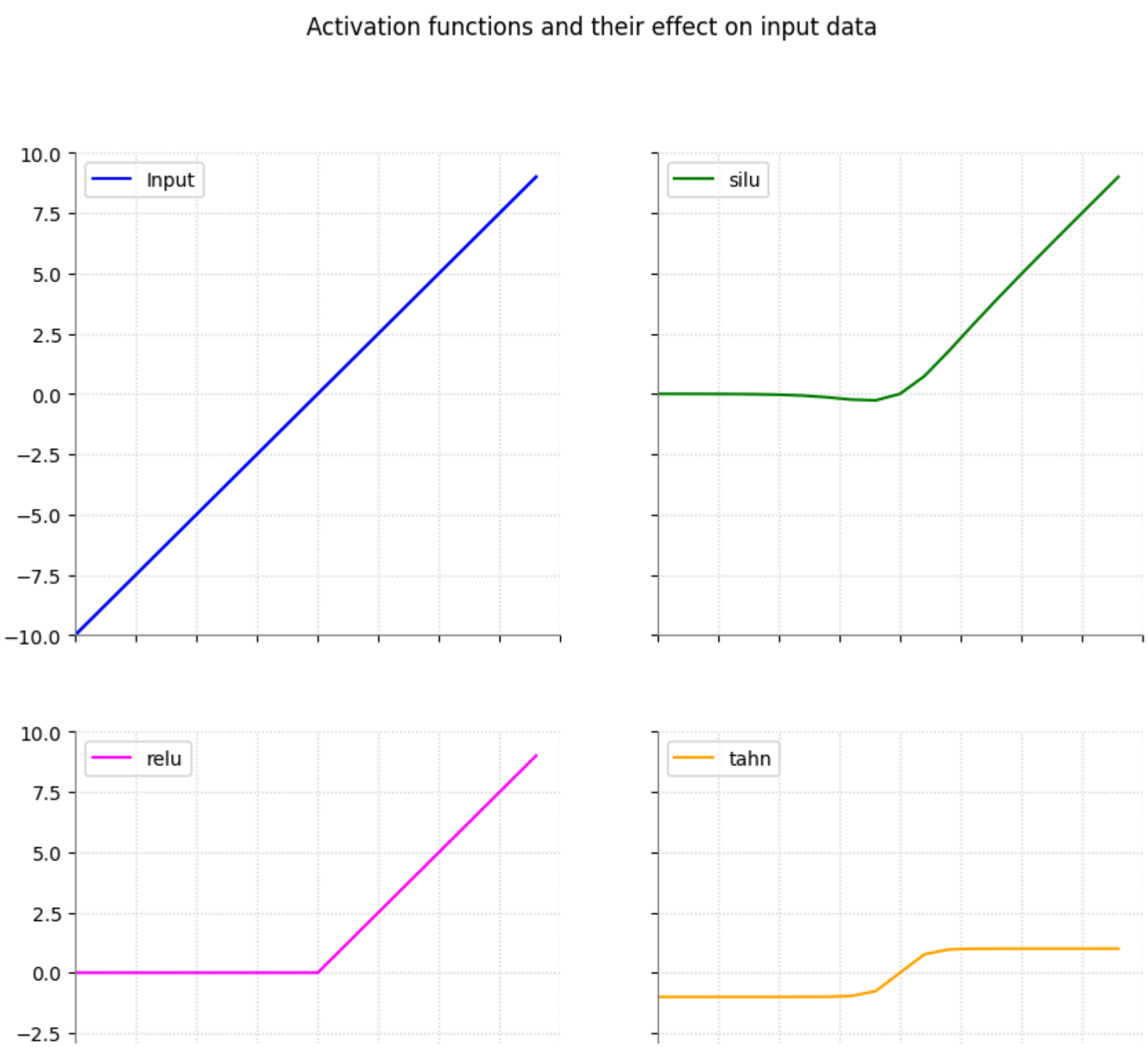


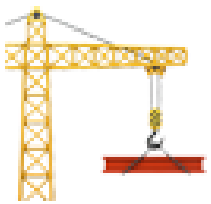
Activation Functions

Activation Functions

- Activation functions allow for ‘bends’
- Name comes from biological neurons
- Function is applied on the output of each neuron

Activation Functions


Activation functions and their effect on input data
Input
silu
relu
tahn
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0



Model Construction - Non-linear model

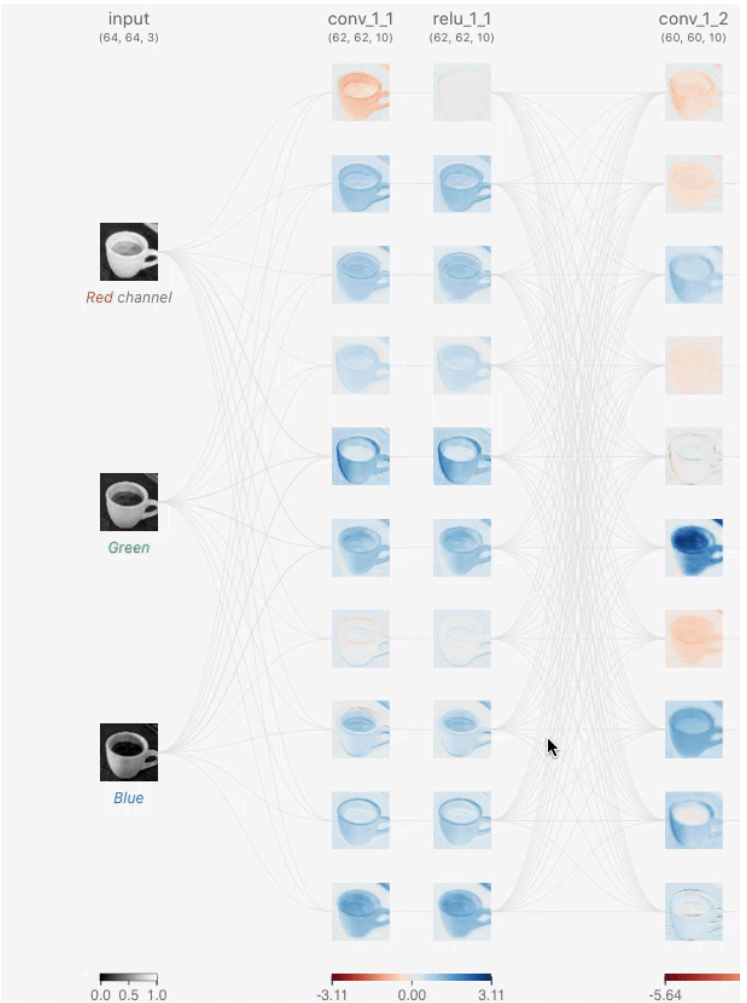
Model Construction - Non-linear model

```
class NL_ModelV1(nn.Module):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(in_features=2, out_features=10),
            nn.ReLU(), # ReLU activation added
            nn.Linear(in_features=10, out_features=10),
            nn.ReLU(), # ReLU activation added
            nn.Linear(in_features=10, out_features=1)
            )

    def forward(self, x):
        return self.network(x)
```

Next Session

Working with Images



Works Cited