1.3.1 Binary Classification

Training Our Model

Summary: In this lesson, we'll continue using PyTorch and build our very first neural network model. We'll use this model to classify if a wildlife camera image shows a hog or not.

Objectives:

- Convert images from grayscale to RGB
- Resizes images
- Create a transformation pipeline to standardize images for training
- Build and train a simple neural network model in PyTorch
- Save our trained neural network to disk

New Terms:

- Activation function
- Automatic differentiation
- Backpropagation
- Binary classification
- Cross-entropy
- Epoch
- Layers
- Logits
- Optimizer

Getting Ready

Just like in the last lesson, there are a few things we need to do before we can begin. We'll start by importing the packages we'll need for this lesson.

Refer to the notebook to import the packages and inspect the GPUs available.

Exploring Our Data

In this section our objective will be to explore and load the data we'll use. Refer to the notebook for instructions.

Activities in this section

Task 1.3.1

Assign `train_dir` the path to the training data. Follow the pattern of `data_dir`

Task 1.3.2:

Determine the number of blank images in the training data and assign the result to `blank_images`. Use the code for `hog_images` as a model.

Task 1.3.3:

Display the path of one image of the blank class. We've illustrated how this is done for the hog class.

Task 1.3.4

Print out the mode and size for the blank image sample. We've shown how this is done for the hog sample.

Preparing Our Data

In this section, we'll perform some operations to get our data ready for processing.

Activities in this section

Task 1.3.5:

Add the missing last step where we convert images to PyTorch tensors. In a previous lesson we used `transform.ToTensor` to accomplish the conversion.

Task 1.3.6:

Prove that the only distinct values of `im` are `0` and `1`. You should use a `set` data structure.

Task 1.3.7:

Print out the length of the validation dataset. We've done so for the training set.

Task 1.3.8:

Use the function and pandas to make the same plot for the validation data.

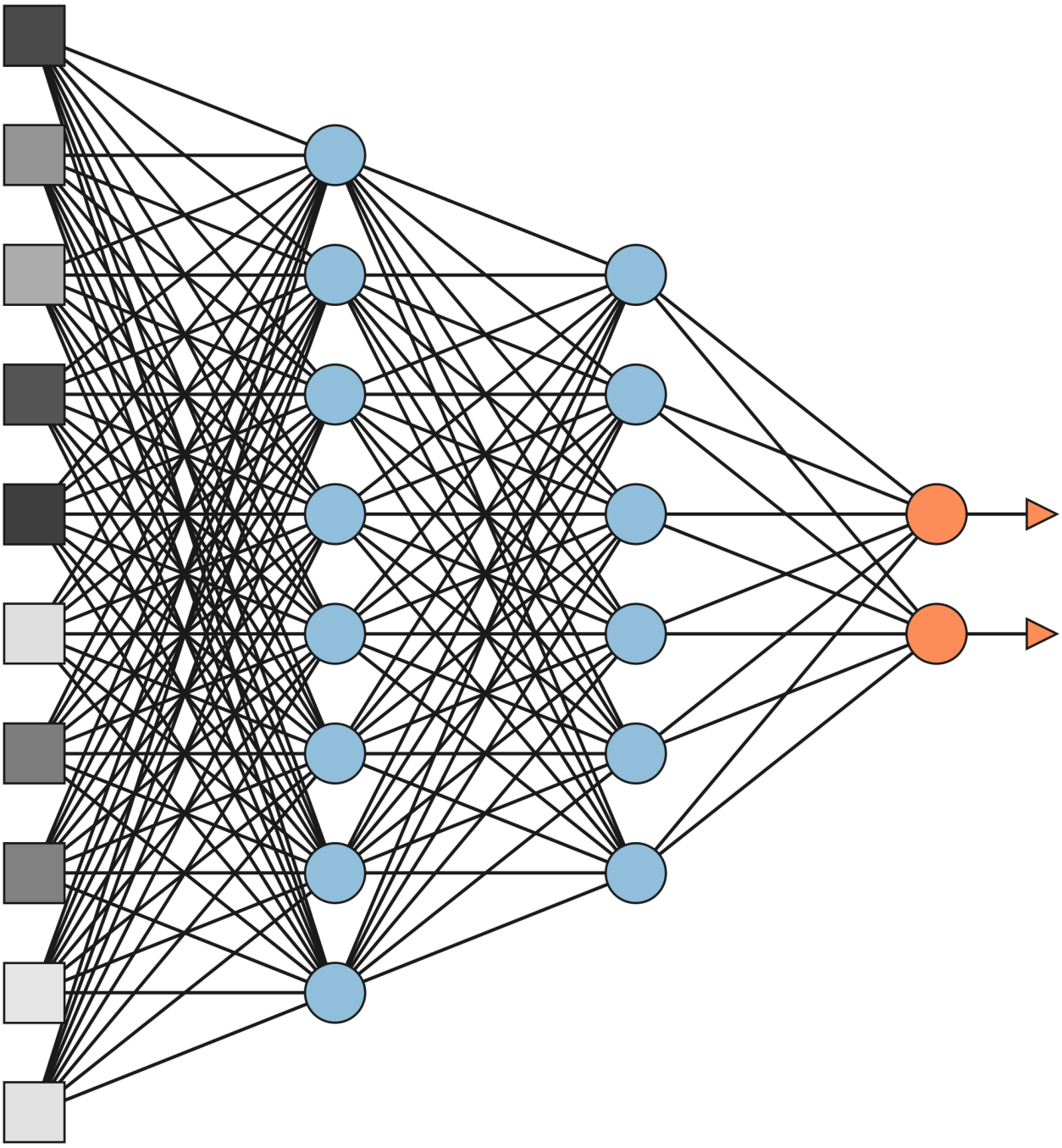
Task 1.3.9:

Create the validation loader. Make sure to set shuffling to be off.

Building a Shallow Neural Network

We're now ready to create our first neural network model. The model will be a shallow fully connected network. It'll not have too many layers, in fact, it'll have four. Here's the architecture: - The input layer - Two hidden layers - The output layer

The image below is an example of our architecture. Note, the image shows fewer neurons than what we'll build as it's hard to visualize a model with many neurons. The important part is that we have an input layer, two hidden layers, and two nodes for the output layer.



Task 1.3.10:

Print the shape of a batch of images and the shape of a batch of labels.

Task 1.3.11:

Flatten the image and print out the resulting shape of the tensor. Use the `nn.Flatten` class to flatten.

Task 1.3.12:

Create the output layer. The last line of the cell below will append it to the model.

Graded activities

Task 1.3.13

Use `model.to` function to place the model on `device`

🚨 **Expected outcome:**

Your `model` should look like this:

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=150528, out_features=512, bias=True)
  (2): ReLU()
  (3): Linear(in_features=512, out_features=128, bias=True)
  (4): ReLU()
  (5): Linear(in_features=128, out_features=2, bias=True)
)
```

Training Our Model

In this section we'll keep making progress towards training our model.

Activities in this section

Task 1.3.14:

Train the model using a single epoch with the *training data*. Print out the model's loss.

Task 1.3.15:

Make a prediction for each row of the *validation data*.

Task 1.3.16

Print out the prediction for the first row of the validation set.

Task 1.3.17

Sum the probabilities to show that they indeed sum to one.

Task 1.3.18

Predict the most likely class label using the validation set.

Task 1.3.19

Calculate the accuracy of the model on the validation set.

Task 1.3.20

Use the `score` function on the validation data, and check that we get the same accuracy as we computed above.

Task 1.3.21

Continue training the model for two more epochs.

Graded Activities

1.3.22

Make a prediction for each image in the validation set

Using the `predict` function, make predictions for each image in the validation set. Store your results in the variable `predictions_val`.

🚨 **Expected outcome:**

```
tensor([0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1
...
0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0], device='cuda:0')
```