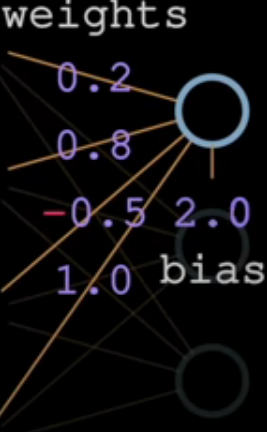
Neural Network

* 3 Components: Input 🡪 Hidden layers 🡪 Output layers
  + Lets say you want the neural network to decide between two different outputs or predicts cat or dog.
    - Input nodes would be pixels of the image which would be passed through multiple hidden layers or just one, up to you.
    - Then output layer would have two nodes, which represent cat or dog
* To train a neural network, you need to alter the weights and biases
  + Every line is a unique weight
  + Every neuron is a unique bias
  + A screenshot of a video game

    Description automatically generated
  + A screen shot of a computer

    Description automatically generatedParameters: number of things we can adjust
* Code practice:
  + **Hidden layer node**
  + Inputs = [1, 2, 3]
  + Weights = [0.2, 0.8, -0.5]
  + Bias = 2
  + **Output layer node**
  + A screenshot of a cell phone

    Description automatically generated
    - 1 output layer
  + 4 input nodes to 3 output nodes
    - , A screenshot of a cell phone

      Description automatically generated,A screenshot of a cell phone

      Description automatically generated
  + 
* **Bias** usually provide an offset to the value
* **Activation function** are functions that kind of determine the final output before it becomes an input to another layer or maybe even the final output to your neural network
* **Shape:**
  + **A screenshot of a black background

    Description automatically generated**
  + **List of list (lol):**
  + **A screenshot of a black background

    Description automatically generated**
  + **List of list of list (lolol):**
  + **A screenshot of a computer

    Description automatically generated**
  + **Tensor:** an object that can be represented as an array
    - A tensor is not just an array, but in the context of doing deep learning and programing a tensor is a representation of an array
  + **Vectors:**
    - Bias and inputs are vectors, while weights are a matrix of vectors
  + **Dot product:**
    - Basically we use dot product to produce a scalar value
    - **A screenshot of a computer

      Description automatically generated**
  + **Batch size**
    - **Computational Efficiency**: Larger batch sizes can be more computationally efficient because they allow the model to process more data in parallel, potentially leading to faster training times. This is particularly beneficial on hardware like GPUs, which are designed to handle large amounts of data simultaneously.
    - **Generalization and Robustness**: Training with smaller batch sizes tends to converge to “flat minimizers” that vary only slightly within a small neighborhood of the minimizer, whereas large batch sizes converge to “sharp minimizers,” which vary sharply. Flat minimizers tend to generalize better because they are more robust to changes between the training and test sets. Additionally, small batch sizes often find minimizers farther away from the initial weights, which can help the training process escape the loss basins of sharp minimizers and instead find flat minimizers that may be farther away.
    - **Memory Requirements and Scalability**: A common concern with larger batch sizes is the increased memory requirement, which might seem like a bad trade-off for simply avoiding to decrease a value. However, more memory is not necessarily required for larger batch sizes. One common method to mitigate this is to accumulate the gradients over several normal-sized batches and then perform one single weight update using the average/sum of the gradients. This approach virtually uses a larger batch size without the need for a larger memory footprint, making the algorithm more scalable.
    - **Learning Rate and Batch Size Interaction**: The relationship between learning rate and batch size is crucial. Small batch sizes perform best with smaller learning rates, while large batch sizes do best on larger learning rates. This interaction suggests that if large batch training is outperforming small batch training at the same learning rate, it may indicate that the learning rate is larger than optimal for the small batch training.
    - **Summary**: While larger batch sizes can offer computational efficiency and potentially faster training, they may not always lead to better generalization and performance. The choice between small and large batch sizes should be made considering the specific requirements and constraints of your training process, including memory limitations, computational resources, and the desired level of model generalization.
  + Matrix dot Matrix
    - A screenshot of a computer

      Description automatically generated
* **Note:**
  + What happens when you save a model is that you’re actually just saving the weights and the biases.
  + So when you load the model, that’s all you’re doing, you’re setting the weights and biases as whatever they were in that saved model

**Normalizing DataSet:**

* Normalizing a dataset means transforming variables to a common scale or range, which helps in eliminating differences in units or distributions. This process is crucial for several reasons
  + **Consistency**: Normalization ensures that all data points are on the same scale, making it easier to compare and analyze them. This consistency is essential for many machine learning algorithms, which often perform better when input features are on a similar scale.
  + **Efficiency**: By reducing the range of values, normalization can make computations more efficient, as algorithms can process smaller numbers more quickly. This efficiency is particularly important in deep learning models, where processing speed can significantly impact performance
  + **Avoiding Bias**: Some algorithms, especially those based on gradient descent, can be sensitive to the scale of the input features. Features on larger scales can dominate the learning process, leading to longer training times or suboptimal results. Normalization helps mitigate this issue by bringing all features to a comparable scale
  + **Improved Model Performance**: By ensuring that all input features are on a similar scale, normalization can lead to improved model performance. This is because it helps the model to learn more effectively from the data, as it reduces the impact of outlier values and minimizes the influence of features with a wider range of values.
* There are several methods to normalize a dataset, including Min-Max scaling and Z-score normalization. Min-Max scaling transforms the data to fit within a specified range, typically between 0 and 1, while Z-score normalization standardizes the data by subtracting the mean and dividing by the standard deviation, resulting in a dataset with a mean of ) and a standard deviation of 1.
* In summary, normalizing a dataset is a crucial setup in the data preprocessing pipeline, aimed at ensuring that all input features are on a similar scale, which can lead to improved model performance, more efficient computations and the elimination of bias in learning algorithms.