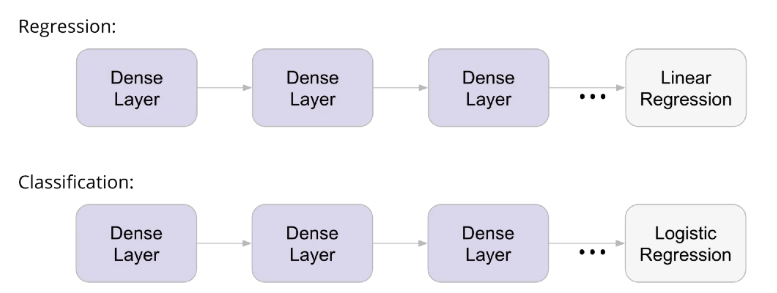
Feedforward artificial neural networks

1. Forward propagation

* Vectorize the neuron
* Consider z to be a vector of size M
* Shapes:
* Input to output for an L-layer neural network (binary classification)
* For regression:
* Each neural network layer is a ‘feature transformation’



* Feature hierarchies:
* Researchers noticed that each layer learns increasingly complex features
* E.g.: Facial recognition (lines -> parts -> face)

1. Geometrical picture

* Making the line more complicated: 2 ways we can make our problem more complicated than ‘finding a line’:
* Add more input dimensions
* Make the pattern nonlinear\*
* Why do we need something more complicated?
* Feature engineering
* Problem: too many possibilities
  + We must also consider interaction terms
* Repeating the single neuron
* Each neuron computes a different nonlinear feature of the input
* It’s nonlinear because of the sigmoid
* How does this make a nonlinear boundary?
* A linear boundary takes the form of:
* A (2-layer) NN boundary takes the form of:
* You cannot reduce the neural network equation into a linear form
* How to make a linear boundary?
* What happens if we remove ? No activation function
* Then it reduces to the linear form (using matrix arithmetic)
* ‘Automatic’ feature engineering
* W and b are randomly initialized, found iteratively using gradient descent

1. Activation functions

* Sigmoid – maps values to [0, 1] -> makes the neural network’s decision boundary nonlinear
* Problem: Sigmoid output goes between 0 and 1, center 0.5 -> output can never be centered around 0
* Output of the sigmoid is the input to the next layer -> we want the output to be centered around 0
* Hyperbolic tangent
* Hyperbolic tangent (tanh)
* Same shape as sigmoid, but goes from [-1, 1]
* Problem: vanishing gradient problem (the further we go in a neural network, the smaller the gradient become) -> weights closer to the input are not trained at all -> more layers, more problem
* ReLU (rectifier linear unit)
* ReLU doesn’t have a ‘vanishing’ gradient, the gradient in the left half (=0) is already vanished -> dead neuron problem
* Still works though
* Fixing dead neurons: Leaky ReLU, exponential linear unit]
* Leaky ReLU
* Small positive slope for negative inputs
* Still nonlinear function
* Slope is always positive (like sigmoid and tanh)
* Exponential linear unit (ELU)
* Negative values possible, mean can be 0

A graph with a line and a line

Description automatically generated

* Softplus:
* Note: both softplus and ELU have vanishing gradients on the left, but we already know its not too problematic because ReLU works
* Softplus and ReLU are in the range [0, inf]
  + Definitely can’t be centered around 0
  + Does it matter?

A graph of a line and a line

Description automatically generated

* Binodal Root Unit (BRU):
* Most people still use ReLU as a reasonable default
* Sometimes, you’ll find that LReLU and ELU offer no benefit
* You just have to try it yourself

1. Multiclass classification

* For binary classification, we use sigmoid at the output
* Replace sigmoids with reLU in the hidden layers
* Multiclass classification:
* E.g., character / handwriting recognition, speech recognition, image classification
* The final layer: Suppose we calculate the value just before applying the final activation function
* How do we turn this vector into a set of prob for each of the K classes?
* The requirements for a prob:
* We need a prob. distribution over K distinct values
* Probs must be non-negative [0, 1]
* Prob of each outcome must sum to 1
* The softmax function:
* This function meets both of our requirements
* Task summary:
* Regression: None/Identity activation function
* Binary classification: Sigmoid activation function
* Multiclass classification: Softmax activation function
* The model type doesn’t matter:
* Same pattern applies to CNN, RNN – the type of task corresponds only to the final activation function

A screenshot of a diagram

Description automatically generated

* The softmax is more general
* Sigmoid handles only binary classification
* Softmax handles multiclass classification, which includes binary classification (K=2)

1. How to represent images

* RGB scheme (most common)
* Color = (red, green, blue)
* We need 3 dimensions: height, width, color
* Color dim always has size 3 – RGB channels
* A(i, j, k) stores the value of ith row, jth column, kth color (k = (r, g, b))
* Quantization:
* Color is light, measured by intensity
* Continuous value
* An infinite number of possible values
* Unfortunately, computers don’t have infinite precision >< more precision – more space
* 8-bit is good enough
* 2^8 = 256 possible values (0, …, 255) -> 16.8 million colors
* A 500x500 image take up: 500x500x3x8 = 6 million bits = 750000 bytes = 732kB
* JPEG allows us to compress images
* Hex colors
* Each byte (8bits) can be represented by a hex number: 00, 01, 02,…, 0A, 0B,…, 0F, 10, 11, …, 1F, A0, A1, …, AF, FF
* 16x16 = 256
* Hex colors are stored using 6 hex digits
* Grayscale images
* Black = 0, white = 255
* Only requires a 2-D array (height, width)
* Plotting grayscale images in matplotlib
* If you use plt.imshow(array2d) -> heatmap
  + Not the true colors of the image, they are assigned: blue = cold, red = hot
* Thus you should use plt.imshow(array2d, cmap=’gray’) for grayscale images
* Images as input to neural networks
* More conventional to scale RGB to [0, 1] (not centered around 0)
* Can be interpreted in prob.
* Exception: VGG – famous neural network in computer network
  + Many applications
  + VGG does not scale input data, but it does subtract the mean across each color channel
  + In Tensorflow, if you are using the built-in VGG model, you have to preprocess the image using: tf.keras.applications.vgg16.preprocess\_input
* Neural network expects an input X of shape NxD
  + N=samples, D=# features
* A single image: HxWxC (height, width, color)
* A full dataset of images: NxHxWxC -> D = HxWxC
* Image to feature vector: Flattening -> Flatten()

A screenshot of a computer

Description automatically generated