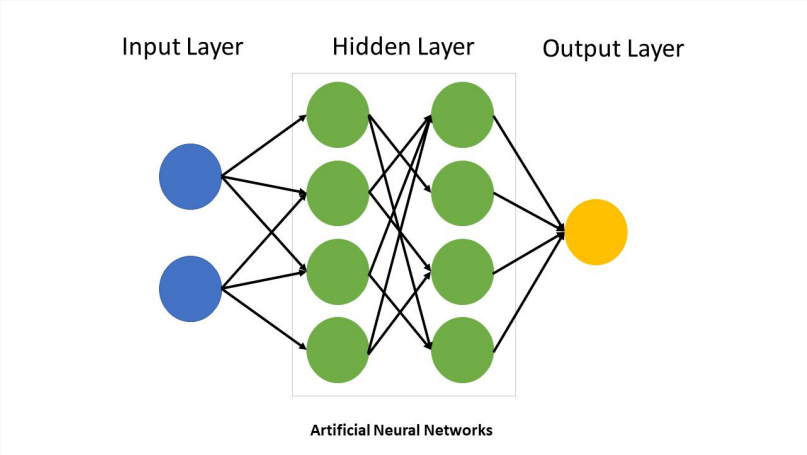
Training Neural Networks

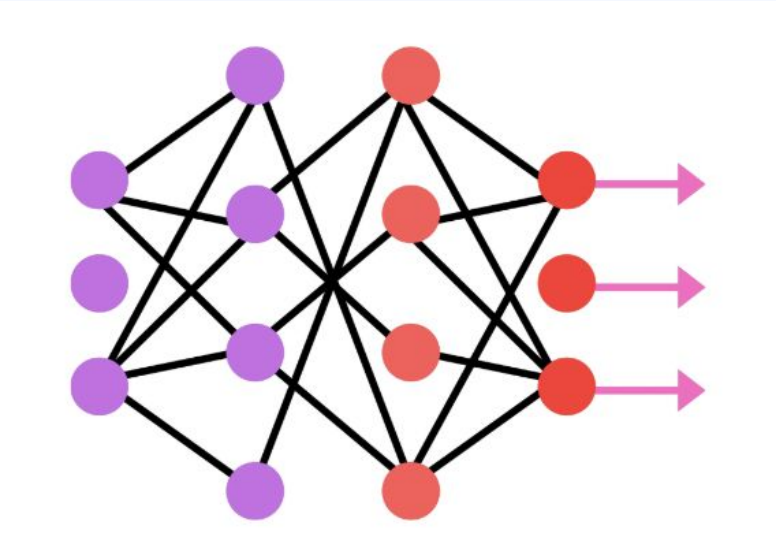
Neural Network Architecture

- Input layer (receives the data)
- Hidden layer (processes the data)
- Output layer (outputs the predictions)



Neural Network Architecture

- The activation values of each layer is connected to the next via weights
- The weights are “learnable” (i.e., are tuned in machine learning)



Neural Network Architecture

- The most basic form of neural networks, multilayer perceptrons (MLPs), are multiple linear models stacked on top of each other, with “activation functions” in between
- These activation functions help the neural network model nonlinear dependencies

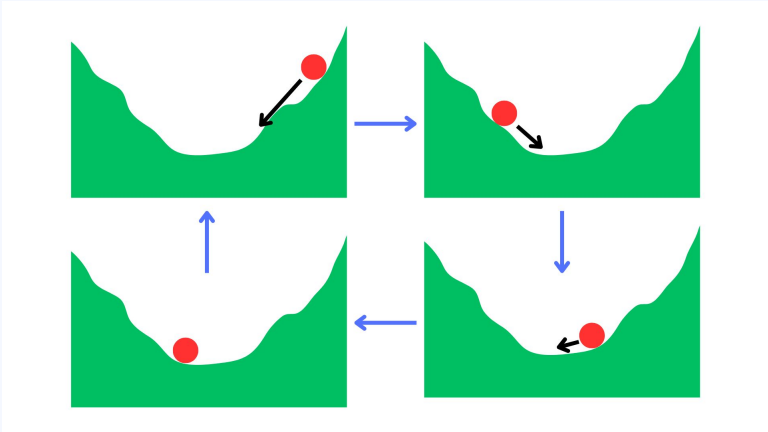
Neural Network Architecture

- Other architectures of neural networks (e.g., convolutional neural networks, transformers) will be discussed later when we get to the related topics
- But they all follow the principle of stacking multiple layers together

Neural Network Training

-The weights (or parameters) of the model is iteratively refined through

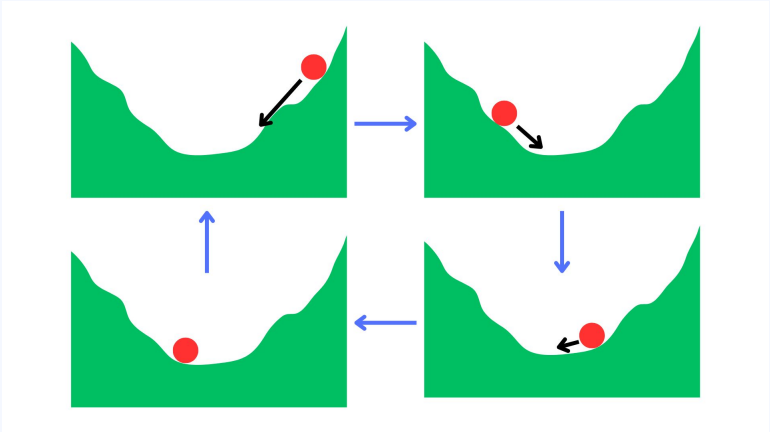
1. Receiving batches of data
2. Making predictions
3. Calculating and backpropagating losses
4. Using gradient descent to update the parameters



Neural Network Training

- Through gradient descent, the model gradually improves its performance on the dataset it has been trained on.

- The figure below shows the simplified representation of how gradient descent works.



Pros and Cons

- Advantages: can model highly complex, high-dimensional data; allows a wide range of inductive biases with its wide range of architectures that makes it suitable for modeling many types of data such as images and text.
- Deep learning methods like neural networks can automate feature extraction from unstructured data like images and text

- Disadvantages: requires a large amount of data to fit a good model, challenging to interpret and know the “why” behind its predictions

Try It Out!

- Experience the data splitting pipeline and the model training process on: https://www.kaggle.com/code/lailaicoding/spam-detection

- You can watch the series by 3blue1brown to better understand neural networks https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Hyperparameters

- Take a look at the gradient descent equation:
- w_(t+1) is the new weight, w_t is the current weight, and the sequence of symbols starting from an inverted triangle on the right is the gradient of the loss against the model parameters
- But how to obtain eta?

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t)$$

Hyperparameters

- It turns out eta is a value that is set before training!
- We call these hyperparameters: values that can adjust the training process itself

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t)$$

Learning Rate

- The eta is a hyperparameter called a “learning rate”, which denotes how fast the model parameters adjust themselves compared to the gradient of the parameters against the loss function
- With a high learning rate, the model makes large steps and trains quickly
- With a small learning rate, the model adjusts itself more slowly as it makes small weight changes at a time

Learning Rate

- But it’s not that simple!
- High learning rates can cause convergence issues – the model can be stuck in a suboptimal state because it “jumps over” local minima without actually reaching them
- Low learning rates can risk overfitting the model or getting it stuck on local minima (that are not actually the optimal solutions)

Learning Rate

- Therefore we should tune the learning rate for each training for good convergence
- That looks time-consuming, but there are actually “good orders of magnitude” for learning rate values according to the task that you would like to do
- We will talk about that later

Learning Rate

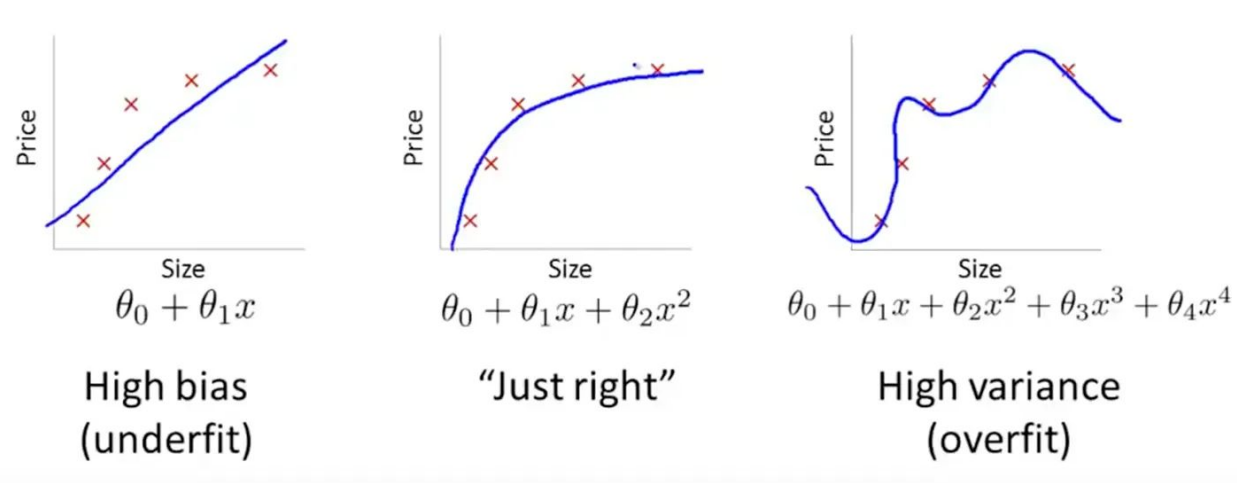
- It may also be suboptimal to use the same learning rate all through the training
- Therefore we can schedule the learning rate throughout the training, setting the learning rate depending on how many steps the training has gone through

Batch Size

- Often, neural networks do not run gradient descent on the entire dataset all at once.
- Instead, the dataset is split up into “batches” that are sampled.
- For each batch, the neural network takes one step in the gradient descent.
- Batch size (the size of each batch) can affect stability of training

Regularization

- Another question: how to prevent overfitting?



Source: https://pub.towardsai.net/how-regularization-can-help-in-overfitting-the-data-ad9ff80f9ccc

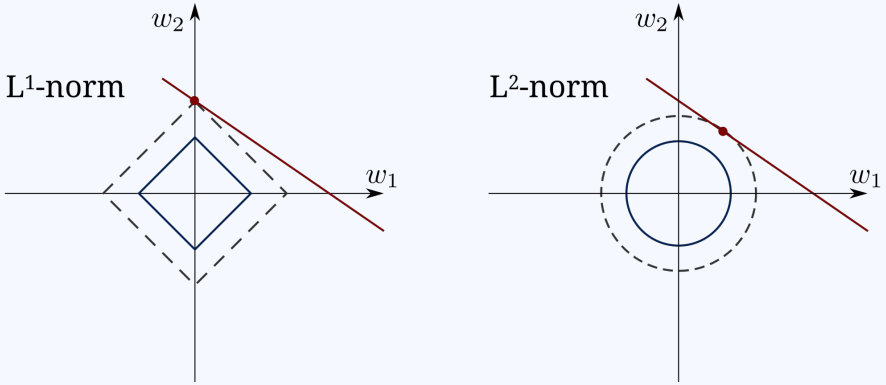
Regularization

Analogy:

- Often to force complicated function curves, the coefficients need to be quite large
- So what if we can force these coefficients (or parameters in the neural network sense) to be smaller?

Regularization

- This is where regularization comes in
- Two common forms of regularization: L1 and L2 regularization



Regularization

- L1 regularization: bound the sum of magnitudes of the weights
- L2 regularization: bound the sum of squared magnitudes of the weights

L1 regularization

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

L2 regularization

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

Source: https://pub.towardsai.net/how-regularization-can-help-in-overfitting-the-data-ad9ff80f9ccc

Regularization

- The lambda values are again hyperparameters that can be tuned to produce different training results
- L1 and L2 regularization both bound the complexity of the model, but L1 regularization can also be used for feature selection (especially for linear models)
- The weights of linear models are pushed towards exactly zero in L1 regularization, so that only the significant weights (corresponding to significant features) survive and account for the prediction