CS131 Deep Learning Lecture Notes

Michael Du - mdu7

December 2019

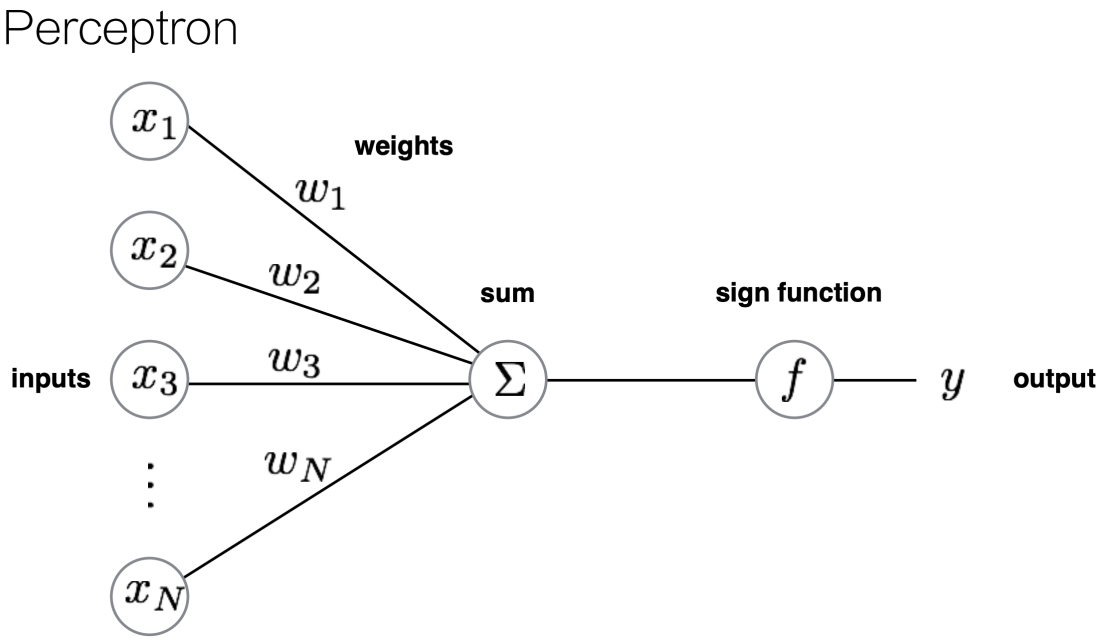
1 Perceptron

1.1 History of Perceptron

-1950s Age of the Perceptron
-1980s Age of the Neural Network
-2010s Age of the Deep Network

Compute power (GPUS) and surplus of data allowed perceptrons to evolve to neural networks and Deep Learning

1.2 Structure of Perceptron



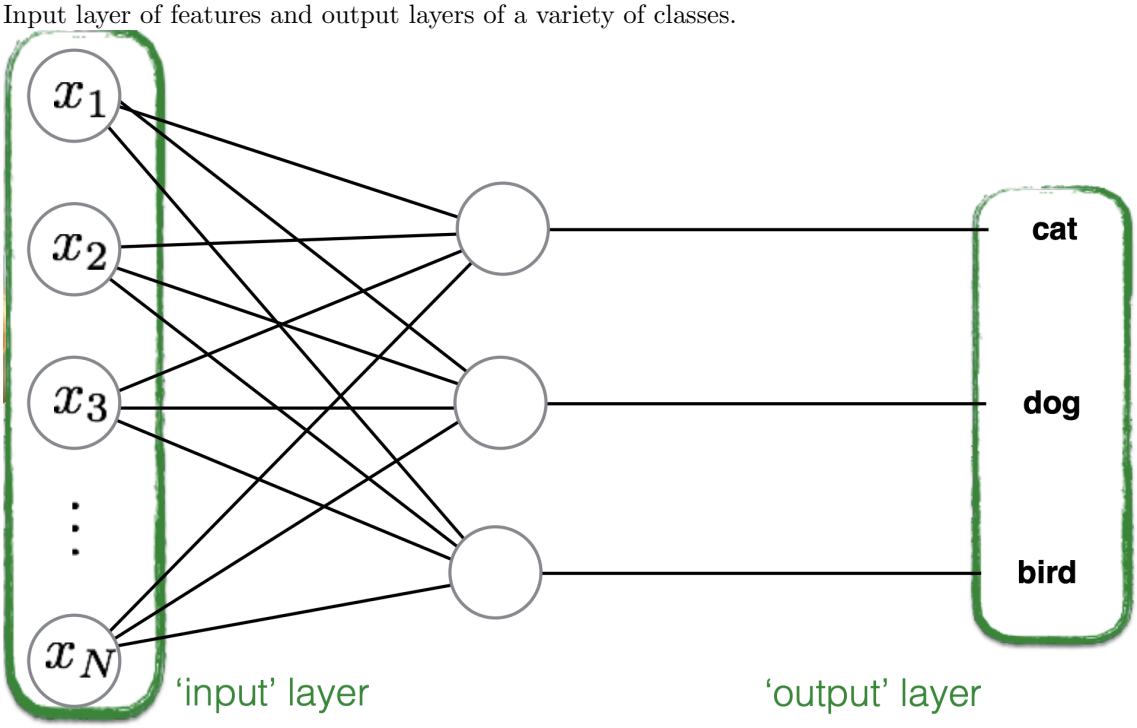
Inspired by biology with dendrites and neurons (input/output structure). In computer vision applications, the inputs (x1, x2, ... , xN) will be pixel features of an image and the output is a classification for that image.

Images can be featurized into a vector through a variety of representations:
-Raw pixels
-Raw pixels + (x,y)

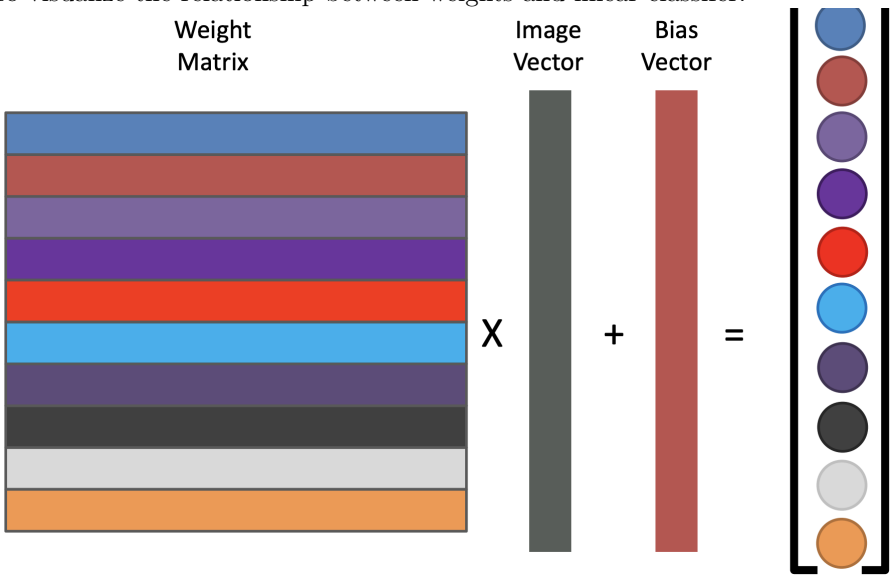
-PCA
-LDA
-Bag of Words
-Bag of Words + spatial pyrameds]]

1.3 Linear Classifier

Input layer of features and output layers of a variety of classes.



To visualize the relationship between weights and linear classifer:



Each row of the weight matrix can be reshaped into the dimensions of the original image. If trained well, the reshaped weights feature will resemble the class it is trying to detect.

2 Loss Function

Task: we must train our linear classifiers. How do we pick the weights for our classifier?

We use a Loss function to tune a weights vector w. More formally, we need to find a w such that:

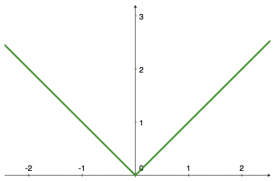
$$min_w Loss(y, \hat{y})$$

where y is the true label and $\hat{y}$ is the predicted model.

Thus, to find the optimized weights vector, we need to choose a loss function. Essentially, this loss function tells us how good our current classifier is. Choosing a loss function is up to the user; here are a few examples of commonly used loss functions.

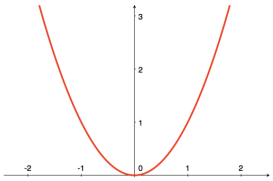
L1 Loss

$$L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$



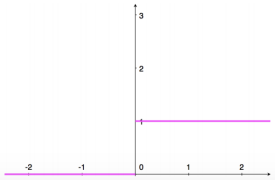
L2 Loss

$$L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$



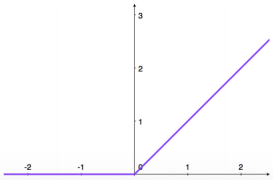
Zero-One Loss

$$L_i(y_i, \hat{y}_i) = 1||y_i \neq \hat{y}_i||$$



Hinge Loss

$$L_i(y_i, \hat{y}_i) = \max(0,\ 1 - y_i\, \hat{y}_i)$$



Each loss function will have different trade-offs and benefits, depending on its use-case. Finding the best loss function will come from trial and error with the corresponding model and data.

2.1 Softmax Classifier

Also known as Multinomial Logistic Regression, the Softmax Classifier is the most widely used loss function in modern day Computer Vision.

Using a softmax classifier allows us to treat outputs of a model as probabilities for each class.

-common way of measuring distance between probability distributions is the Kullback-Leibler (KL) divergence.

$$D_{KL} = \sum_y P(y) \frac{log(P(y))}{log(Q(y)}$$

- where P(y) is the ground truth distribution

- and Q(Y) is the model's output score distribution

To normalize the outputs in a probability range from zero to one, we arrive at the softmax classifer:

$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

3 Gradient Descent

Gradient Descent Pseudocode

```
for _ in {0,…,num_epochs}:
    L = 0
    for x_i, y_i in data:
        ŷ_i = f(x_i, W)
        L += L_i(y_i, ŷ_i)
    dL/dW = ???
    W := W − α dL/dW
```

You may notice that calculating $\frac{dL}{dW}$ is a bit hard. However, using parital derivatives makes this process much easier.

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}$$

Partial derivative of loss to update weights

$$L = -\hat{y}_k + \log \sum_j e^{\hat{y}_j}$$

To calculate $\frac{dL}{d\hat{y}}$, we need to consider two cases:

Case 1:

$$\frac{dL}{d\hat{y}_k} = -1 + \frac{e^{\hat{y}_k}}{\log \sum_j e^{\hat{y}_j}}$$

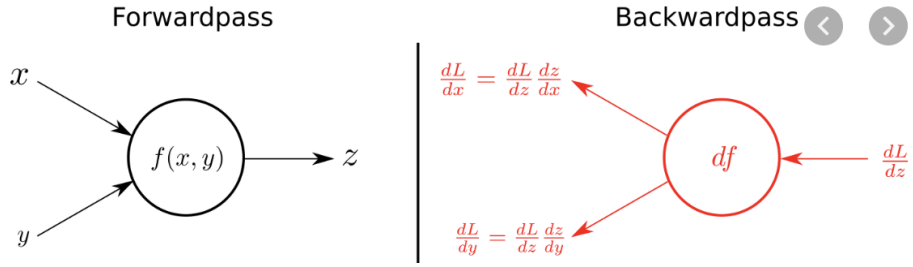
Case 2:

$$\frac{dL}{d\hat{y}_{l \neq k}} = \frac{e^{\hat{y}_l}}{\log \sum_j e^{\hat{y}_j}}$$

Gradient Descent can be tuned for a model by tweaking hyperparameters (ie. alpha).

4 Backpropogation

Another way to compute the gradient is through a method called backpropgation.



As seen in this 1D example, in the forward pass, we use a set of weights (x) and a set of features (y) and compute a $\hat{y}$ that is passed to some loss function z($\hat{y}, y$).

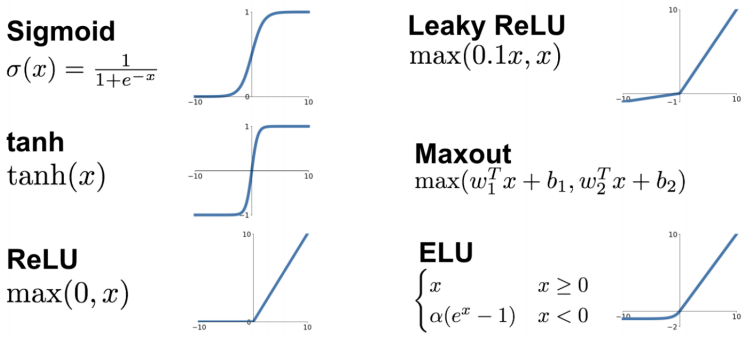
Then, in the backwards pass, the gradients with respect to the weights and features are passed back and used to make updates.

5 Activation Functions

Activation function

The non−linear max function allows models to learn more complex transformations for features.

Choosing the right activation function is another new hyperparameter!



Different activation functions are used as a hyperparameter in neural networks. Currently, ReLU (Rectified Linear Units) is widely used in research and industry.