Neural Networks

Shrey Gupta

Applied Machine Learning (HOUSECS 59-01), Duke University

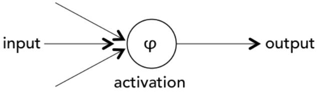
October 31, 2018

Brain View

- An original goal with the creation of neural networks was modeling the human brain.
- The human brain is a collection of neurons, connected together as part of a "network".
- Signals are "spikes" that travel through the network to produce a response.

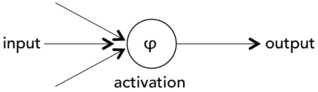
Image source: John Lieff

Neurons



- Let's try to model an individual neuron: it should be able to take in multiple inputs, and produce an output to model a spike.

Neurons



- Denote each of the inputs into a neuron as $x_1, x_2, \ldots, x_p$ (where p is known).
- Create weights $w_1, w_2, \ldots, w_p$ and a bias b. (More on how these weights and bias are determined later.)
- Determine the score $f(w, x) = w^T x + b$ that conveys information about the input.
- Apply a *nonlinear* activation function $\phi(\cdot)$, and produce an output $g(w, x) = \phi(w^T x + b)$.

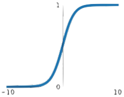
Activation Functions

- We use an activation function (ϕ) to model a "spike" in a neuron (i.e. that neuron is *activated*).
  - Ex: step function, tanh, sigmoid, ReLU, leaky ReLU.
- For the step function, if our score (dot product between weights and input) exceeds a certain threshold, our output is *one*, and the neuron is activated.
  - Else, *zero*.
- The other four more commonly used activation functions are deviations from the step function, but serve a similar purpose.

Activation Functions

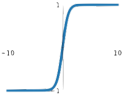
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



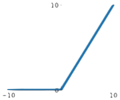
tanh

$\tanh(x)$



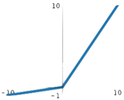
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$



Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

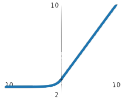
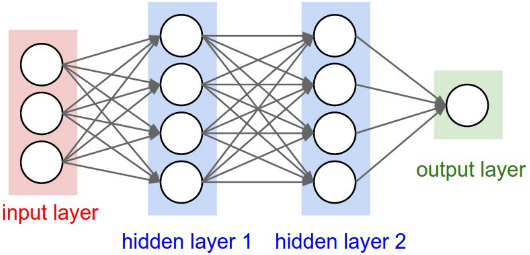


Image source: Introduction to Exponential Linear Units

Neurons

- We now have a neuron that takes some input, and produces a spike—as a function of the input and weights (and bias)—when it deems the input to be "significant".
  - We'll learn how to train the network weights and bias later.
- The next step is to connect neurons together.

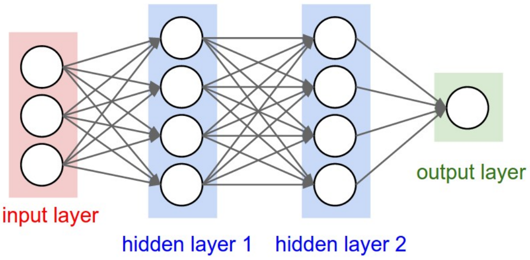
Layers



- We can construct a layer of neurons that take in the same input, and produce some output.
  - Each neuron learns something "different" about the input.
- We can stack layers to create a network of neurons (i.e. neural network).
  - Each layer will take the output of the previous layer as input.

Image source: Stanford University

Architecture Summary



- Input layer: single vector of feature inputs.
- Hidden layer(s): sets of neurons with nonlinear activation, fully connected by weights (with biases) to other layers.
- Output layer: output score or prediction ($\hat{y}_i$).

Image source: Stanford University

Backpropagation

- The backpropagation algorithm can be used to compute the weights of each neuron in the network.
  - While outside the scope of the course, the algorithm is simply an application of the chain rule, where we try to minimize the error in predictions $\sum \frac{1}{2}(y_i - \hat{y}_i)^2$.
- Backpropagation tells us the "relative update rate" (gradient) for each weight, and we can use gradient descent (with learning rate α) to actually update the weights.
- Hence, we repeat: backpropagation (to calculate gradients), gradient descent (to adjust the weights), and a forward pass (to calculate the errors).

Advantages

- Neural networks are highly expressive (nonlinear) models that can fit nearly any function of the data well (*universal approximation theorem*).
- Neural networks have performed well in areas such as computer vision and natural language processing.
- Hidden layers in the network can characterize the latent structure of the data well.

Disadvantages

- Optimization is non-convex and the algorithm can produce poor solutions (e.g. get stuck at local optima).
- Neural networks are not very interpretable (i.e. “black boxes”).
- There are several parameters, such as network structure, that must be tuned.

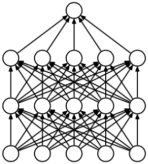
Technique: Convergence

- Convergence during gradient descent can be a problem due to vanishing and exploding gradients (think about the sigmoid and tanh activation functions).
- Fix: use the ReLU or leaky ReLU activation functions.
- Tip: common learning rates α during gradient descent are between 10^{-5} and 10^{-3}.
- Note: different initialization (of the neuron weights and biases) can affect the final model.

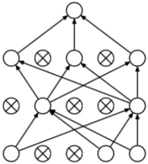
Technique: Batch Normalization

- Engineering technique to improve convergence: normalize the outputs of various neurons in the same layer (subtract the mean and divide by the standard deviation), before activation is applied.
- Include the mean and standard deviation as separate inputs into the following layer.

Technique: Dropout



(a) Standard Neural Net



(b) After applying dropout.

- Engineering technique to speed-up convergence: during each forward pass, with probability p (usually 0.5), set output weights to 0 for each neuron (during training).
- Utilize the network as normal during testing.
- Idea: "dropout" is equivalent to training an exponential number of "submodels".

Image source: Cynthia Rudin

Notebook

- Today's notebook will work through an example of neural networks.