Training Neural Networks

CMSC 470
Marine Carpuat
Neural Networks so far

• Powerful non-linear models for classification

• Predictions are made as a sequence of simple operations
  • matrix-vector operations
  • non-linear activation functions

• Choices in network structure
  • Width and depth
  • Choice of activation function

• Feedforward networks
  • no loop

• Next: how to train
Neural Networks as Computation Graphs
Computation Graphs Make Prediction Easy: Forward Propagation consists in traversing graph in topological order
Computation Graph

• Graph contains 3 different types of nodes
  • Parameters of the models (e.g., W1, b1, W2, b2)
  • Input x
  • operations between parameters and input (e.g., product, sum, sigmoid)

• Acyclical directed graph
  • No recursion or loops

• So far each computation node in the graph should consist of
  • A function that executes its computation operation
  • Links to input nodes
  • When processing an example, the computed value
  (we’ll add 2 more items to enable training)
How do we train a neural network?

For training, we need

• Data: (a large number of) examples paired with their correct class \((x, y)\)

• Loss/error function: quantify how bad our prediction \(y\) is compared to the truth \(t\)
  • E.g. squared error (aka L2 loss) \[ \text{error} = \frac{1}{2}(t - y)^2 \]

• An algorithm to minimize the loss: stochastic gradient descent
Extending the Computation Graph to Compute the Loss
Computing Gradients: Chain rule decomposes computation of gradient along the nodes

\[
\frac{dE}{dA} = \frac{dE}{dB} \frac{dB}{dA}
\]
Training Illustrated

\[
\begin{bmatrix}
1.0 \\
0.0
\end{bmatrix} \times
\begin{bmatrix}
3 & 4 \\
2 & 3
\end{bmatrix} - \mu
\begin{bmatrix}
0.484 & 0 \\
-0.0258 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 \\
-2
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.484 \\
-0.0258
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.731 \\
0.119
\end{bmatrix}
\]

\[
\begin{bmatrix}
5 \\
-5
\end{bmatrix} - \mu
\begin{bmatrix}
0.0360 & 0.00587
\end{bmatrix}
\]

\[
\begin{bmatrix}
3.06 \\
-2.46
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.492, 0.492
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.743
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.191 	imes 0.257 = 0.492
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.0331 \\
0.257
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.0
\end{bmatrix}
\]

\[
\begin{bmatrix}
-2
\end{bmatrix} - \mu
\begin{bmatrix}
0.0492
\end{bmatrix}
\]
Computation Graph

• Graph contains 3 different types of nodes
  • Parameters of the models (e.g., W1, b1, W2, b2)
  • Input \( x \)
  • operations between parameters and input (e.g., product, sum, sigmoid)

• Acyclical directed graph
  • No recursion or loops

• So far each computation node in the graph should consist of
  • A function that executes its computation operation
  • Links to input nodes
  • When processing an example in the forward pass, the computed value
  • A function that executes its gradient computation
  • Links to children nodes (to obtain downstream gradient values)
  • When processing an example in the backward pass, the computed gradient
Computation Graph: A Powerful Abstraction

• To build a system, we only need to:
  • Define network structure
  • Define loss
  • Provide data
  • (and set a few more hyperparameters to control training)

• Given network structure
  • Prediction is done by forward pass through graph (forward propagation)
  • Training is done by backward pass through graph (back propagation)
  • Based on simple matrix vector operations

• Forms the basis of neural network libraries
  • Tensorflow, Pytorch, mxnet, etc.
Exploiting parallel processing

• Using vector matrix operations helps
  • E.g., if a layer has 200 nodes a matrix operation $Wh$ requires $200 \times 200 = 40000$ multiplications
  • Can benefit from efficient implementations for Graphics Processing Units (GPU)
• “Minibatch” training by processing multiple examples at a time helps further
  • Compute parameter updates based on a “minibatch” of examples
    • instead of one example at a time
  • More efficient: matrix-matrix operations replace multiple matrix-vector operations
  • Can lead to better model parameters
Neural Networks

• Originally inspired by human neurons, but now simply an abstract computational device

• Can be thought of as combinations of neural units, where each unit multiplies input by a weight vector, adds a bias, and then applies a non-linear activation function

• Or alternatively as a computation graph

• Power comes from ability of early layers to learn representations (i.e. features) that can be used by later layers in the network
Neural Networks

• Choices in network structure
  • Width and depth
  • Choice of activation function

• Feedforward networks (no loop)

• Forward Propagation: predictions are made as a sequence of simple operations
  • matrix-vector operations
  • non-linear activation functions

• Training with the back-propagation algorithm
  • Requires defining a loss/error function
  • Gradient descent + chain rule
  • Easy to implement on top of computation graphs