ECE408 Lecture 9b

Feed-Forward Networks and Gradient-Based Training

ECE408 / CS483 / CSE 408

Spring 2019

(by Carl Pearson)

Machine Learning

- An important way of building applications whose logic is not fully understood.
  - Use labeled data – data that come with the input values and their desired output values – to learn what the logic should be
  - Capture each labeled data item by adjusting the program logic
  - Learn by example!

Machine Learning Tasks (1)

- Classification
  - Which of *k* categories an input belongs to
  - Ex: object recognition
- Regression
  - Predict a numerical value given some input
  - Ex: predict tomorrow's temperature
- Transcription
  - Unstructured data into textual form
  - Ex: optical character recognition

Machine Learning Tasks (2)

- Translation
  - Convert a sequence of symbols in one language to a sequence of symbols in another
- Structured Output
  - Convert an input to a vector with important relationships between elements
  - Ex: natural language sentence into a tree of grammatical structure
- Others
  - Anomaly detection, synthesis, sampling, imputation, denoising, density estimation

Why Machine Learning Now?

- GPU computing hardware and programming interfaces such as CUDA has enabled very fast research cycle of deep neural net training
- Computer Vision, Speech Recognition, Document Translation, Self Driving Cars, Data Science…
- Most involve logic that were previously not effectively constructed with imperative programming
- Using big labeled data to train and specialize DNN based classifiers
  - Deriving a large quantity of quality labeled data is a challenge

Classification

- Formally: a function that maps an input to k categories

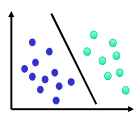
  $f: R^n \rightarrow \{1, \ldots, k\}$,
- Our formulation: a function f parameterized by Θ that maps input vector $\boldsymbol{x}$ to numeric code y

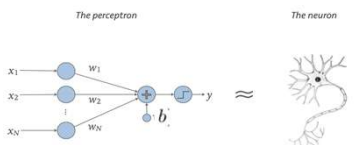
  $y = f(\boldsymbol{x}, \Theta)$
- Θ encapsulates the structure and weights in our network

Linear Classifier (Perceptron)

- Recall our formulation: $y = f(\boldsymbol{x}, \Theta)$

$y = sign(W \bullet \boldsymbol{x} + b)$ $\Theta = \{W, b\}$



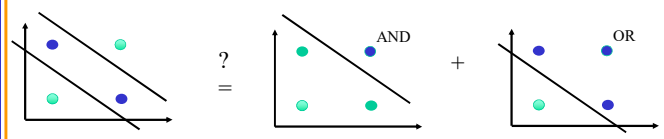


- Dot product: $y = W \bullet \boldsymbol{x}$
- Scalar addition: $+ b$

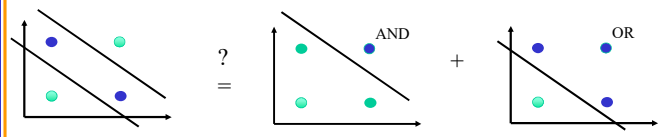
output
input
weight
bias

Multi-Layer Perceptron

- What if a linear classifier can't learn a function?



Multi-Layer Perceptron



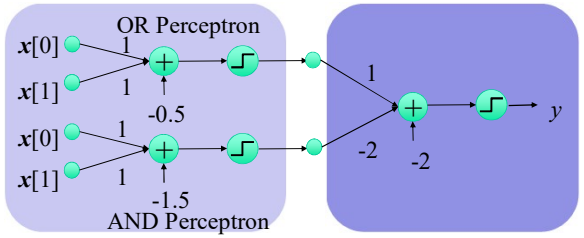
A	B	OR	AND	XOR
0	0	-1	-1	-1
0	1	1	-1	1
1	0	1	-1	1
1	1	1	1	-1

A	B	OR	AND	XOR
0	0	-1	-1	-1
0	1	1	-1	1
1	0	1	-1	1
1	1	1	1	-1

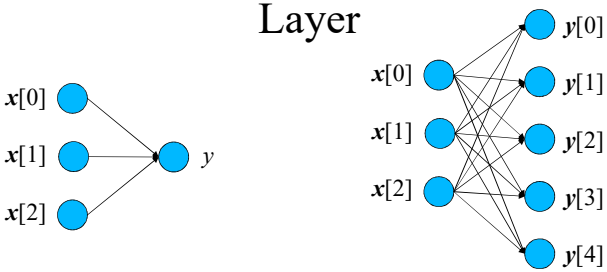
AND = sign($\boldsymbol{x}[0] + \boldsymbol{x}[1]$ - 1.5)

OR = sign($\boldsymbol{x}[0] + \boldsymbol{x}[1]$ - 0.5)

XOR = sign(2 * OR - AND - 2)



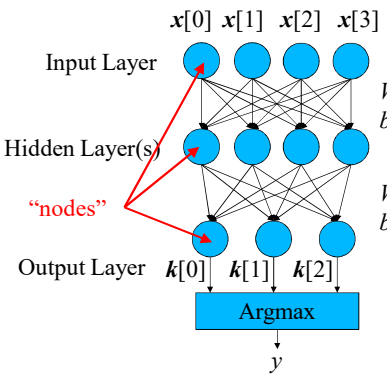
Generalize to Fully-Connected Layer



Linear Classifier:
Input vector $\boldsymbol{x}$ × weight vector $\boldsymbol{w}$ to produce scalar output y

Fully-connected:
Input vector $\boldsymbol{x}$ × weight <u>matrix</u> $\boldsymbol{w}$ to produce <u>vector</u> output y

Multilayer Terminology



Weight matrices: Entry i,j is weight between i^{th} input and j^{th} output

W_1[i, j] is [4x4]
b_1[j] is [4x1]

W_2[i, j] is [4x3]
b_2[j] is [3x1]

Probability that input is class $\boldsymbol{k}$[i]

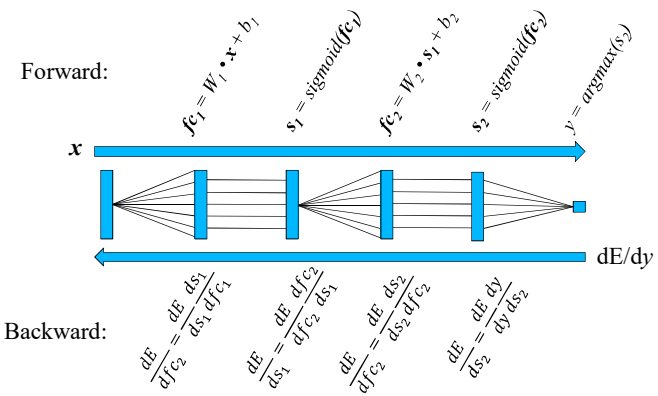
How to determine the weights?

- Fully connected layer with 784 inputs, 1024 outputs:
  - [784 x 1024] weight matrix
  - [1024 x 1] bias vector
- Look at observational data to determine the weights.
- With enough input data and corresponding desired outputs, we can model the relationship between inputs and outputs.

Forward and Backward Propagation

- Forward (inference)
  - Given parameters Θ and input $\boldsymbol{x}$, produce label y
- Backward (training)
  - Given input data and target label t, determine Θ

Forward and Backward Propagation



Ingredients for Gradient-Based Supervised Training

- One labeled dataset (large)
  - Example: 60,000 28x28 grayscale images of handwritten numbers, each labeled
- One network architecture
  - Example: perceptron
  - $y = W \bullet \boldsymbol{x} + b$ Recall: $\Theta = \{W, b\}$
- One error function
  - For target label t, network output y,
  - Example: $E = ½ (y - t)^2$

Stochastic Gradient Descent

- For each labeled image:
  - Read data to initialize input layer
  - Evaluate network to get y (forward)
  - Compare with target label t to get error E
  - Backpropagate error derivative to get parameter updates
  - Adjust parameters Θ in a direction that reduces total E over entire training set.

    $\Theta_{i+1} = \Theta_i - \varepsilon\Delta\Theta$

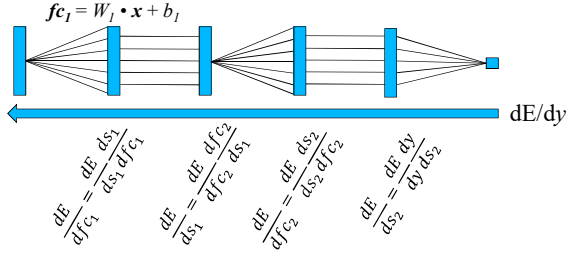
Each gradient update happens from most accurate minima estimation

Computing the Gradient Update

$\Theta_{i+1} = \Theta_i - \varepsilon\Delta\Theta \quad W_{i+1} = W_i - \varepsilon\Delta W$	Parameter Update
$y = W \bullet \boldsymbol{x} + b$	Network function
$\frac{dy}{dW} = x$	Network weight gradient
$E = ½ (y - t)^2$	Error function
$\frac{dE}{dy} = y - t = Wx + b - t$	Error function gradient
$\Delta W = \frac{dE}{dW} = \frac{dE}{dy}\frac{dy}{dW}$	Full weight update expression
$W_{i+1} = W_i - \varepsilon(W\boldsymbol{x}+b-t)x$	Full weight update term

Parameter Updates and Propagation

$\boldsymbol{fc_1} = W_1 \bullet \boldsymbol{x} + b_1$



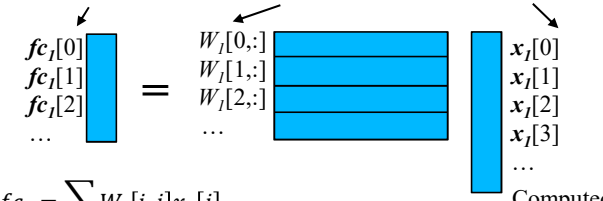
Need propagated error gradient (from backward pass)

Weight update $\frac{dE}{dW_1} = \frac{dE}{dfc_1}\frac{dfc_1}{dW_1} = \frac{dE}{df_1}x$

Need input (from forward pass)

Fully-Connected Gradient Detail

i^{th} entry in $\boldsymbol{fc_1}$ — i^{th} row in W_1 — j^{th} entry in $\boldsymbol{x_1}$



$$fc_1 = \sum_j W_1[i,j]x_1[j]$$

Computed from previous layer

$$\frac{dE}{dW_1[i,j]} = \frac{dE}{dfc_1[i]}\frac{dfc_1[i]}{dW_1[i,j]} = \frac{dE}{dfc_1[i]}x_1[j]$$

Need input to this layer

Batched Stochastic Gradient Descent

- A training *epoch* (a pass through whole training set)
  - *Set* $\Delta\Theta = 0$
  - For each labeled image:
    - Read data to initialize input layer
    - Evaluate network to get y (forward)
    - Compare with target label t to get error E
    - Backpropagate error derivative to get parameter updates
    - Accumulate parameter updates into $\Delta\Theta$
  - $\Theta_{i+1} = \Theta_i - \varepsilon\Delta\Theta$

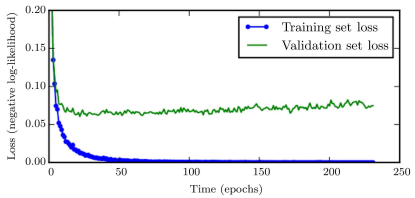
Aggregate gradient update most accurately reflects true gradient

Mini-batch Stochastic Gradient

- For each batch in training set
  - For each labeled image in batch:
    - Read data to initialize input layer
    - Evaluate network to get y (forward)
    - Compare with target label t to get error E
    - Backpropagate error derivative to get parameter updates
    - Accumulate parameter updates into $\Delta\Theta$
  - $\Theta_{i+1} = \Theta_i - \varepsilon\Delta\Theta$

Balance between accuracy of gradient estimation and parallelism

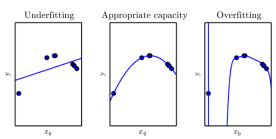
When is training done?

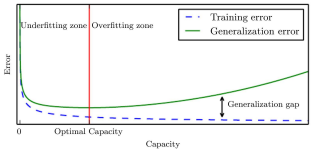


Split labeled data into *training* and *test* sets.

- Training data to compute parameter updates.
- Test data to check how model generalizes to new inputs (the ultimate goal!)
- The network can become *too good* at classifying training inputs!

How Complicated Should a Network Be?





Intuition: like a polynomial fit. More higher-order terms make a better fit, but add huge unpredictable swings as input changes

If network is too good at training data, new inputs cause big unpredictable output changes.

No Free Lunch Theorem

- Every classification algorithm has the same error rate when classifying previously unobserved inputs when averaged over all possible input-generating distributions.
- Even neural networks must be tuned for specific tasks

Summary (1)

- Classification:
  - $f: R^n \rightarrow \{1, \ldots, k\}$,
  - $y = f(\boldsymbol{x}, \Theta)$
- Current ML work driven by cheap compute, lots of available data
- Perceptron as a trivial deep network
  - $y = sign(W \bullet \boldsymbol{x} + b)$
- Forward for inference, backward for training

Summary (2)

- Chain rule to compute parameter updates
- Stochastic gradient descent for training