Calculus

Arthur J. Redfern
axr180074@utdallas.edu
Aug 29, 2018
Sep 05, 2018

Outline

- Motivation
- Derivatives
- Optimization
- Neural network training
- Universal approximation
- References

Motivation

Optimization And Approximation

- Neural network training is an optimization problem that we solve using calculus (math highlights listed, obviously others also present)
  - Forward propagation - linear algebra
  - Error calculation - probability*
  - Backward propagation - calculus
  - Weight update - calculus

- Neural networks are universal approximators
  - General tool for solving feature extraction and prediction problems
  - A constructive proof provides some feel for how problem complexity and network design are related

* Though in this lecture probability is not explicitly used, it will be covered later

Derivatives

Scalar Functions Of A Single Variable

- Scalar functions of a single variable: $y = f(x)$
  - $f: R \rightarrow R$

Scalar Functions Of A Single Variable

- Definition of the derivative

$$df/dx = \lim_{\Delta x \to 0} (f(x + \Delta x) - f(x)) / \Delta x$$

- Interpretation
  - The derivative is a linear operator that maps functions to function
  - Intuition of sensitivity of function f to changes in input x

Scalar Functions Of A Single Variable

- Differentiable functions
  - At values of x where the limit exists f is differentiable
- Non differentiable functions
  - At values of x where the limit does not exists f is not differentiable
  - If there are a countable number of non differentiable values then it's potentially possible to approximate the derivative of f at a value where it's not differentiable via the derivative just to the left or right (or any derivative in this range)

Scalar Functions Of A Single Variable

- Example: sigmoid
  - f(x) = $1 / (1 + e^{-x})$
  - df/dx = $f(x)(1 – f(x))$
- Example: tanh
  - f(x) = $(e^{x} – e^{-x}) / (e^{x} + e^{-x})$
  - df/dx = $1 – f^{2}(x)$
- Example: ReLU
  - f(x) = max(0, x)
  - df/dx = 0, $x \leq 0$ note that at x = 0, $df/dx \in [0, 1]$
    = 1, $x > 0$

Scalar Functions Of A Single Variable

- Product rule
  - $h(x) = f(x)\, g(x)$
  - $dh/dx = (df/dx)\, g(x) + f(x)\, (dg/dx)$
- Quotient rule
  - $h(x) = f(x) / g(x)$
  - $dh/dx = ((df/dx)\, g(x) - f(x)\, (dg/dx)) / g^2(x)$ general case
    $= -(dg/dx) / g^2(x)$ f(x) = 1 case
- Chain rule
  - $h(x) = f(g(x))$ and define $u = g(x)$
  - $dh/dx = (df/du)\, (du/dx)$

Scalar Functions Of Multiple Variables

- Scalar function of multiple variables: $y = f(\mathbf{x})$
  - $f: R^K \rightarrow R$
  - $\mathbf{x} = [x_0\ x_1\ \ldots\ x_{K-1}]^T$

Scalar Functions Of Multiple Variables

- Partial derivative
  - Let $\mathbf{e}_k$ be coordinate direction k
  - $\partial f/\partial x_k = \lim_{\Delta \to 0} (f(\mathbf{x} + \Delta \mathbf{e}_k) - f(\mathbf{x})) / \Delta$

- Interpretation
  - View all variables x_n where $n \neq k$ as constants
  - Take derivative of f at $\mathbf{x}$ with respect to x_k
  - Result is slope of the function f at point $\mathbf{x}$ in coordinate direction $\mathbf{e}_k$

Scalar Functions Of Multiple Variables

- Gradient
  - K x 1 vector of partial derivatives with respect to each variable
  - $\nabla f(\mathbf{x}) = [(\partial f/\partial x_0)\ (\partial f/\partial x_1)\ \ldots\ (\partial f/\partial x_{K-1})]^T$

- Interpretation
  - Same shape and dimension as **x** (note this is choosing denominator layout)
  - Points in direction of maximum change of f at **x**
  - When doing iterative error minimization we're typically going to figure out the gradient of the error with respect to the parameters (i.e., the sensitivity of the error to changes in the parameters) and adjust the parameters in the opposite direction of the gradient to reduce the error
  - Directional derivative in unit vector direction **u** can be found from $(\nabla f(\mathbf{x}))^T\ \mathbf{u}$

Scalar Functions Of Multiple Variables

- Examples
  - Linear algebra
    - Vector vector multiplication
      - $f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} = \mathbf{x}^T \mathbf{a}$
      - $\partial f / \partial \mathbf{x} = \mathbf{a}$
    - Vector matrix vector multiplication
      - $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$
      - $\partial f / \partial \mathbf{x} = (\mathbf{A} + \mathbf{A}^T) \mathbf{x}$ General case
        $= 2 \mathbf{A} \mathbf{x}$ Symmetric **A** case

Scalar Functions Of Multiple Variables

- Examples (continued)
  - Error calculation
    - $\mathbf{x}^*$ is ideal value
    - Mean square error
      - $f(\mathbf{x}^*, \mathbf{x}) = (1/K)\ (\mathbf{x} - \mathbf{x}^*)^T\ (\mathbf{x} - \mathbf{x}^*)$
      - $\partial f/\partial \mathbf{x} = (2/K)\ (\mathbf{x} - \mathbf{x}^*)$
    - Cross entropy
      - Assume log base e for convenience so no extra scale values
      - $f(\mathbf{x}^*, \mathbf{x}) = -\sum_k x^*(k) \log(x(k))$ we'll talk about this in the probability lecture
        $= -\log(x(k^*))$ $\mathbf{x}^*$ has a 1 at position k* and 0s elsewhere
      - $\partial f/\partial \mathbf{x} = -1/(x(k^*))$ at $k = k^*$
        $= 0$ at $k \neq k^*$
      - The $\partial f/\partial \mathbf{x}$ vector only having 1 non 0 element will allow the product of the derivative of the softmax function and cross entropy function to have a very nice form

Vector Functions Of Multiple Variables

- Vector functions of multiple variables: $\mathbf{y} = f(\mathbf{x})$
  - $f: R^K \rightarrow R^M$

Vector Functions Of Multiple Variables

- Jacobian
  - M x K matrix with entry (m, k) corresponding to the partial derivative of output f_m with respect to input variable x_k

$$J_f(\mathbf{x}) = \begin{bmatrix} \partial f_0/\partial x_0 & \cdots & \partial f_0/\partial x_{K-1} \\ \vdots & & \vdots \\ \partial f_{M-1}/\partial x_0 & \cdots & \partial f_{M-1}/\partial x_{K-1} \end{bmatrix}$$

- Interpretation
  - Above definition in numerator layout to match common calculus convention
  - All other derivatives used in this presentation not on this specific slide will be in denominator layout
  - Will use to transform error gradients through layers
  - Optimal linear approximation of a vector valued function near a point **a**

$$f(\mathbf{x}) \approx f(\mathbf{a}) + J_f(\mathbf{a})(\mathbf{x} - \mathbf{a})$$

Vector Functions Of Multiple Variables

- Examples with parameters
  - Linear transformations
    - Matrix vector multiplication
      - $f(\mathbf{H}, \mathbf{x}) = \mathbf{H}\,\mathbf{x}$
      - $\partial \mathbf{f}/\partial \mathbf{x} = \mathbf{H}^T$
      - $\partial \mathbf{f}/\partial \mathbf{H} = \mathbf{x}^T$ (note: $f^T(\mathbf{H}, \mathbf{x}) = \mathbf{x}^T\,\mathbf{H}^T$, $\partial \mathbf{f}^T/\partial \mathbf{H}^T = \mathbf{x}$, $\partial \mathbf{f}/\partial \mathbf{H} = \mathbf{x}^T$)
    - Matrix matrix multiplication
      - $f(\mathbf{H}, \mathbf{X}) = \mathbf{H}\,\mathbf{X}$
      - Via matrix vector multiplication with vectorized **X** and block diagonal **H**
    - CNN style 2D convolution
      - $f(\mathbf{H}^{No\ x\ Ni\ x\ Fr\ x\ Fc}, \mathbf{X}^{Ni\ x\ Lr\ x\ Lc}) = \mathbf{H}^{No\ x\ Ni\ x\ Fr\ x\ Fc} \circledast \mathbf{X}^{Ni\ x\ Lr\ x\ Lc}$
      - Via CNN style 2D convolution decomposed into using a sum of 1x1 filters leading to a sum of matrix matrix mults

Vector Functions Of Multiple Variables

- Examples with parameters (continued)
  - Elementwise operations
    - Elementwise addition
      - $f(\mathbf{H}, \mathbf{X}) = \mathbf{X} + \mathbf{H}$
      - $\partial \mathbf{f} / \partial \mathbf{X} = \mathbf{I}$
      - $\partial \mathbf{f} / \partial \mathbf{H} = \mathbf{I}$
    - Elementwise multiplication
      - $f(\mathbf{H}, \mathbf{X}) = \mathbf{H} \odot \mathbf{X}$
      - $\partial \mathbf{f} / \partial \mathbf{X} = \mathbf{H}^T$
      - $\partial \mathbf{f} / \partial \mathbf{H} = \mathbf{X}^T$

Vector Functions Of Multiple Variables

- Examples without parameters
  - Note: since there are no parameters only $\partial \mathbf{f}/\partial \mathbf{x}$ needs to be computed
  - Pointwise operations
    - Fixed scale
      - $f(\mathbf{X}) = \alpha \mathbf{X}$
      - $\partial \mathbf{f}/\partial \mathbf{X} = \alpha \mathbf{I}$
    - ReLU, sigmoid, tanh
      - Conceptually a pointwise application of scalar derivative rules forms a diagonal matrix
      - Though in practice for efficiency only the diagonal values are generated

Vector Functions Of Multiple Variables

To do: add specific equations; currently just lists rules that will be used in back propagation

- Examples without parameters (continued)
  - Pooling
    - Average
      - Divide the gradient at the output equally to form the gradient at all the inputs
    - Max
      - Send gradient at output to the max input that created it
      - Requires saving the max index from the forward path for each output
  - Structural operations
    - Split
      - Add all gradients at the output to form the gradient at the input

Vector Functions Of Multiple Variables

To do: get better formatting for the matrix

- Examples without parameters (continued)
  - Error calculation
    - Softmax
      - No parameters so only $\partial \mathbf{f}/\partial \mathbf{x}$ needs to be computed
      - $f(\mathbf{x}) = (1/(\Sigma_k e^{x(k)}))\ [e^{x(0)}\ e^{x(1)}\ \ldots\ e^{x(K-1)}]^T$
      - $\partial \mathbf{f}/\partial \mathbf{x} =$

$$\begin{bmatrix} f(0)(1-f(0)) & -f(1)f(0) & \ldots & -f(K-1)f(0) \\ -f(0)f(1) & f(1)(1-f(1)) & & -f(K-1)f(1) \\ \ldots & & \ldots & \ldots \\ -f(0)f(K-1) & -f(1)f(K-1) & & f(K-1)(1-f(K-1)) \end{bmatrix}$$

Vector Functions Of Multiple Variables

- Chain rule
  - $h(\mathbf{x})$ $= f(g(\mathbf{x}))$ and define $\mathbf{u} = g(\mathbf{x})$
  - $\partial h/\partial \mathbf{x}$ $= (\partial \mathbf{u}/\partial \mathbf{x})\,(\partial f/\partial \mathbf{u})$ Note denominator layout ordering

- Example: softmax cross entropy error gradient for classification
  - $\mathbf{p}^*$ has a 1 at position k* and 0s elsewhere
  - $\mathbf{p}$ $= \text{softmax}(\mathbf{x})$, Like $\mathbf{u} = g(\mathbf{x})$
  - $f(\mathbf{p})$ $= H_{CE}(\mathbf{p}^*, \mathbf{p})$, Like $f(\mathbf{u})$
  - $e(\mathbf{x})$ $= H_{CE}(\mathbf{p}^*, \text{softmax}(\mathbf{x}))$, Like $h(\mathbf{x}) = f(g(\mathbf{x}))$
  - $\partial e/\partial \mathbf{x}$ $= (\partial \mathbf{p}/\partial \mathbf{x})\,(\partial f/\partial \mathbf{p})$
    $= [p(0) \quad \ldots \quad p(k^*-1) \quad (p(k^*)-1) \quad p(k^*+1) \quad \ldots \quad p(K-1)]^T$

Optimization

Setup

- Optimization problem
  - Minimize $f_0(\mathbf{x})$ — Objective function
  - Subject to $f_c(\mathbf{x}) \leq b_c$, $c = 1, ..., C$ — Constraint functions
- Solution
  - $\mathbf{x}^*$ is the optimal solution if it has the smallest objective value out of all vectors that satisfy the constraints
- A few special classes of optimization problems
  - Convex: $f_i(\alpha\mathbf{x}_0 + \beta\mathbf{x}_1) \leq \alpha f_i(\mathbf{x}_0) + \beta f_i(\mathbf{x}_1)$, $i = 0, 1, ..., C$, $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$
  - Linear: $f_i(\alpha\mathbf{x}_0 + \beta\mathbf{x}_1) = \alpha f_i(\mathbf{x}_0) + \beta f_i(\mathbf{x}_1)$, $i = 0, 1, ..., C$

Setup

- Critical points
  - For a scalar function of multiple variables a critical point is a value **x** at which all partial derivatives of f(**x**) are 0
  - Local extrema of f(**x**) occur at critical points (Fermat's theorem)
    - Minimum (in all directions)
    - Maximum (in all directions)
    - Saddle (minimum in some directions, maximum in some directions)

- Potential notational confusion
  - In optimization literature it's common to optimize over parameters **x**
  - In xNNs we have inputs **x** and optimize over parameters **h** (or some other variable)
  - So in the optimization section our goal is to find the optimal **x**, and in the xNN training section it's to find the optimal **h** (or other parameter variable)

Closed Form Methods

- Example: least squares solution to a linear systems of equations
  - Minimize $f(\mathbf{x}) = (\mathbf{A}\,\mathbf{x} - \mathbf{y})^T(\mathbf{A}\,\mathbf{x} - \mathbf{y})$
  - Multiply out, take partial derivatives, set to 0 and solve
  - $f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}^T\mathbf{A}\,\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\mathbf{y} - \mathbf{y}^T\mathbf{A}\,\mathbf{x} + \mathbf{y}^T\mathbf{y}$
    $= \mathbf{x}^T\mathbf{A}^T\mathbf{A}\,\mathbf{x} - 2\,\mathbf{y}^T\mathbf{A}\,\mathbf{x} + \mathbf{y}^T\mathbf{y}$ Transpose of scalar = scalar
  - $\partial f/\partial \mathbf{x} = 2\,\mathbf{A}^T\mathbf{A}\,\mathbf{x} - 2\,\mathbf{A}^T\mathbf{y}$ $\mathbf{A}^T\mathbf{A}$ is symmetric
    $= \mathbf{0}$
  - $\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T y$ Over determined case with linearly indep cols of **A**
    $= \mathbf{A}^{-1}\mathbf{y}$ Uniquely determined case with invertible **A**

- For xNN training we usually don't have a simple closed form solution for finding optimal parameters (sadness)

Iterative Methods

- Gradient descent
  - Find $\mathbf{x}$ that minimizes f($\mathbf{x}$)
  - Basic strategy (many many variations exist)
    - Initialization
      - Let n = 0
      - Initial guess $\mathbf{x}_0$
    - Iteration
      - Compute the gradient $\nabla f(\mathbf{x}_n)$
      - Select a step size α_n
      - Take a step in the opposite direction of the gradient: $\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \nabla f(\mathbf{x}_n)$
      - Let n = n + 1

Iterative Methods

- Repeating the closed form example using gradient descent for ease of understanding

- Example: least squares solution to a linear systems of equations
  - Minimize $f(\mathbf{x}) = (\mathbf{A}\,\mathbf{x} - \mathbf{y})^T(\mathbf{A}\,\mathbf{x} - \mathbf{y})$
  - $f(\mathbf{x}) = \mathbf{x}^T\,\mathbf{A}^T\,\mathbf{A}\,\mathbf{x} - \mathbf{x}^T\,\mathbf{A}^T\,\mathbf{y} - \mathbf{y}^T\,\mathbf{A}\,\mathbf{x} + \mathbf{y}^T\,\mathbf{y}$
    $= \mathbf{x}^T\,\mathbf{A}^T\,\mathbf{A}\,\mathbf{x} - 2\,\mathbf{y}^T\,\mathbf{A}\,\mathbf{x} + \mathbf{y}^T\,\mathbf{y}$
  - $\nabla f(\mathbf{x}) = 2\,\mathbf{A}^T\,\mathbf{A}\,\mathbf{x} - 2\,\mathbf{A}^T\,\mathbf{y}$
    $= 2\,\mathbf{A}^T\,(\mathbf{A}\,\mathbf{x} - \mathbf{y})$
  - $\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n\,\nabla f(\mathbf{x}_n)$
    $= \mathbf{x}_n - \alpha_n\,2\,\mathbf{A}^T\,(\mathbf{A}\,\mathbf{x}_n - \mathbf{y})$

To do:
- Add how to calculate the sqrt(2) in elementary school
- Banach fixed point theorem / contraction mapping theorem

Neural Network Training

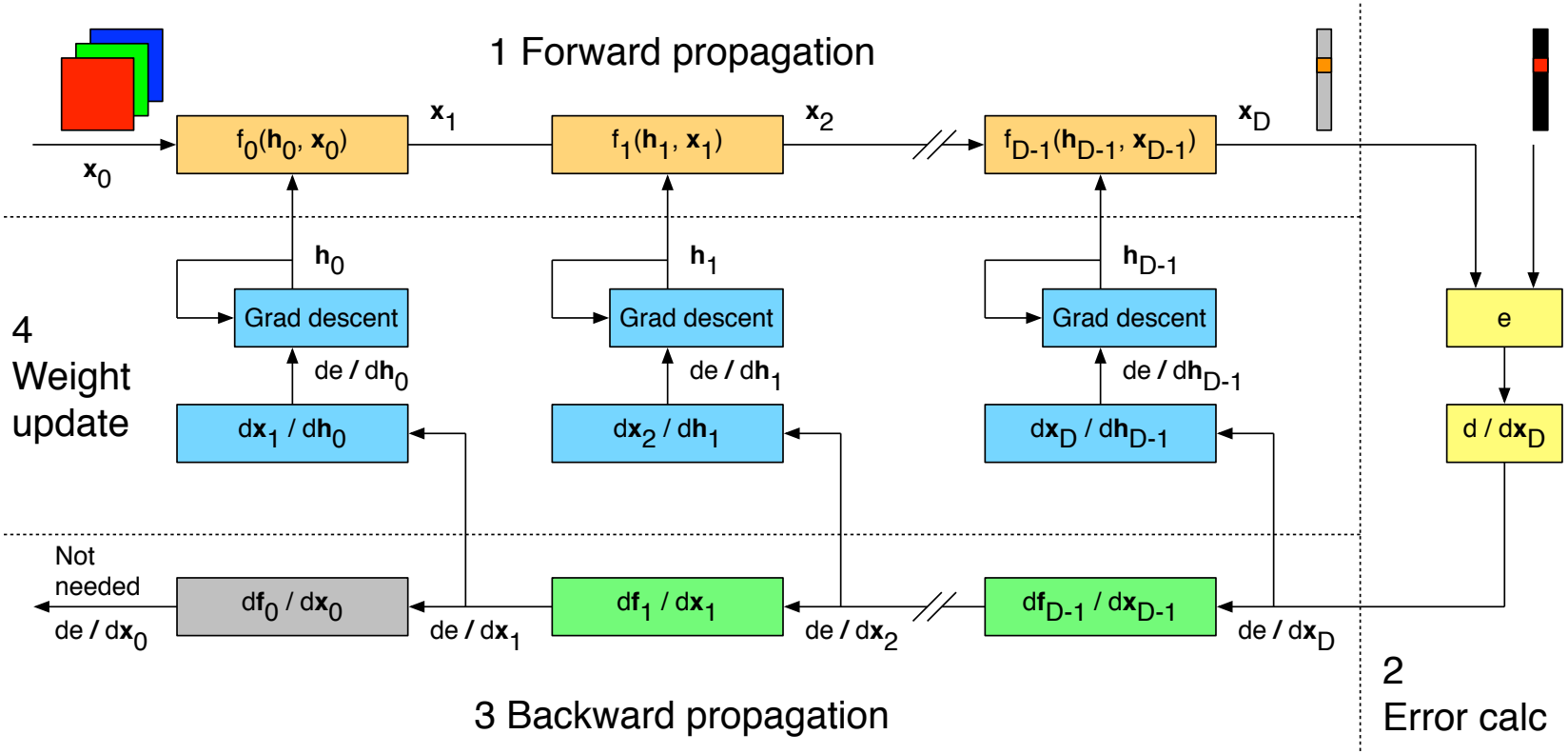
Specifically

- The basics of xNN training are described here
- Many variations to improve training will be introduced in a later lecture
  - It’s ok to start a long trip with a 1st step

Strategy

- Initialization
  - Assume for now, will discuss later
- Forward propagation
- Error calculation
- Backward propagation
  - Partial derivatives and the chain rule
- Weight update
  - Gradient descent

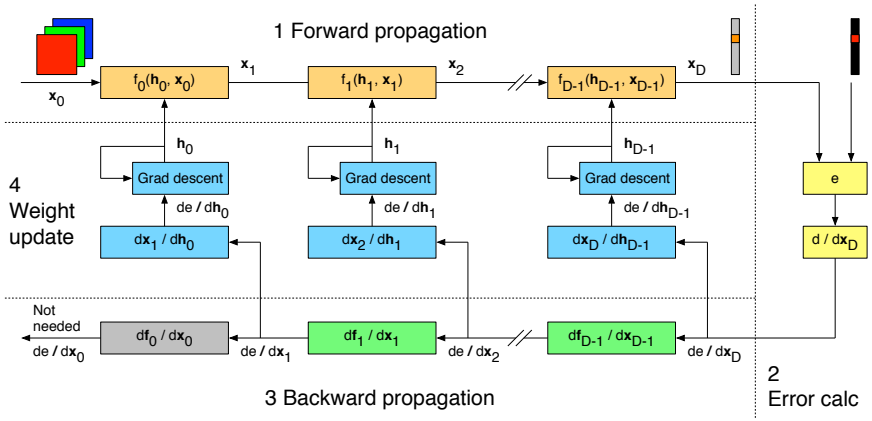
Strategy


1 Forward propagation
x_0
$f_0(h_0, x_0)$
x_1
$f_1(h_1, x_1)$
x_2
$f_{D-1}(h_{D-1}, x_{D-1})$
x_D
h_0
h_1
h_{D-1}
Grad descent
Grad descent
Grad descent
e
4
Weight
update
de / dh_0
de / dh_1
de / dh_{D-1}
dx_1 / dh_0
dx_2 / dh_1
dx_D / dh_{D-1}
d / dx_D
Not
needed
df_0 / dx_0
df_1 / dx_1
df_{D-1} / dx_{D-1}
de / dx_0
de / dx_1
de / dx_2
de / dx_D
2
Error calc
3 Backward propagation

Forward Propagation

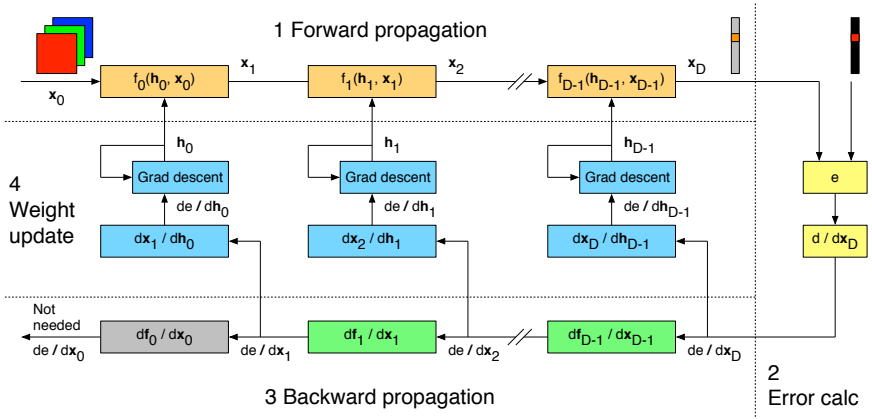
- Use the network to generate a batch of outputs from a batch of inputs

- Example
  - Image classification to 1 of K classes
  - Network input is an image
  - Network performs some calculations
    - Multiple layers
    - Some of those layers controlled are by parameters
  - Network output is a K x 1 vector
    - Each element x(k) corresponds to a class
    - k* is the correct class for the image
    - Network attempts to make x(k) large for k = k* and small for k ≠ k*



Error Calculation

- How well did the network do?
  - Error calculation quantifies an answer to this question for the training input to a single number (yes, 1 number)
  - The goal of back propagation and weight update will be to adapt all the parameters in the network to minimize this error
- For the error calculation we'll assume that the ideal output is known
  - Supervised learning
- Consider errors optimized for 2 types of predictions
  - Classification
  - Regression



Error Calculation

- For classification we'll commonly use softmax cross entropy
  - Note that this is really 2 layers but for implementation reasons we'll treat together

- Error calculation
  - Network output is K x 1 vector $\mathbf{x}_D$
  - $\mathbf{p}$ = softmax($\mathbf{x}_D$)
    - p(k) is in [0, 1] and Σ_k p(k) = 1
    - Interpretation of converting the network output to probabilities
  - e = $H_{CE}(\mathbf{p}^*, \mathbf{p})$
    - Ideal output $\mathbf{p}^*$ has a 1 at position k* and 0s elsewhere
    - e = $-\Sigma_k$ p*(k) log(p(k)) = – log(p(k*))
    - Interpretation of defining the error based on the converted probability for the correct class

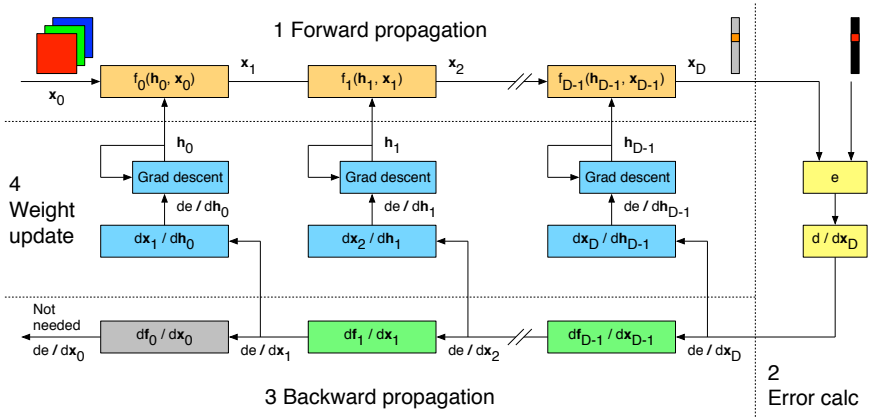
Error Calculation

- For regression we'll commonly use 0.5 MSE
  - 0.5 really isn't needed, just makes equations look more beautiful

- Error calculation
  - Network output is K x 1 vector $\mathbf{x}_D$
  - e = 0.5 mse($\mathbf{x}^*$, $\mathbf{x}_D$)
    - Ideal output $\mathbf{x}^*$
    - e = (0.5/K) ($\mathbf{x}^* - \mathbf{x}_D$)^T ($\mathbf{x}^* - \mathbf{x}_D$)
    - Interpretation of defining the error based on the 2 norm distance between the network output and the ideal value

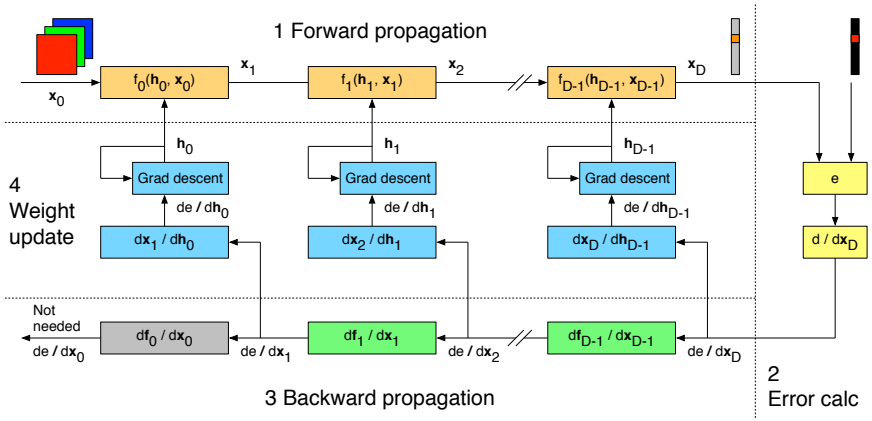
Backward Propagation

- Goal
  - Determine sensitivity (gradient) of the error with respect to all feature maps
- Automatic differentiation with reverse mode accumulation
  - Decompose forward propagation into a set of building blocks (practically layers, theoretically any primitive operation with a known derivative)
  - Build associated graph for back propagation ~ via reversing arrows and replacing forward propagation nodes representing layers with back propagation nodes representing layer derivatives
  - Back propagation nodes map from $\partial e/\partial \mathbf{x}_d$ to $\partial e/\partial \mathbf{x}_{d-1}$ via the chain rule



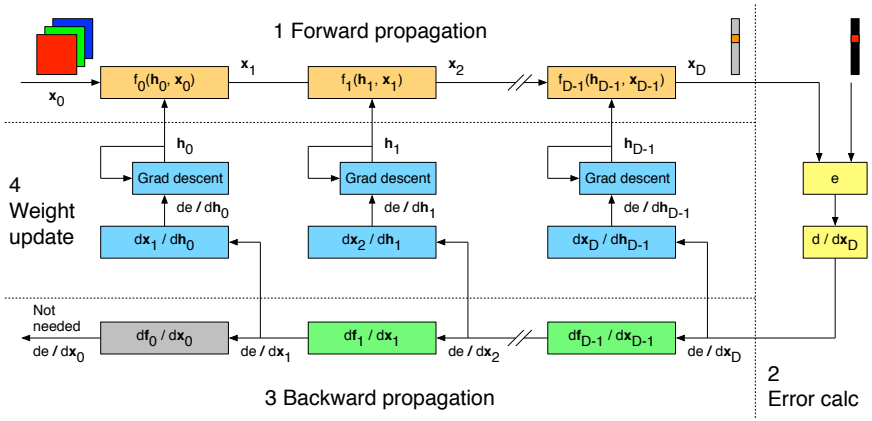
Backward Propagation

- Cookbook
  - Build backward propagation graph
  - Start with $\partial e/\partial \mathbf{x}_D$, the gradient of the error with respect to the output of the last layer
    - How to calculate this was conveniently shown on an earlier slide for softmax cross entropy and MSE errors
  - Compute the gradient of the error at the input of the last layer (which is the output of the next to last layer)
    - This is found using the chain rule
      $\partial e/\partial \mathbf{x}_{D-1} = (\partial \mathbf{x}_D/\partial \mathbf{x}_{D-1})\,(\partial e/\partial \mathbf{x}_D) = (\partial \mathbf{f}_{D-1}/\partial \mathbf{x}_{D-1})\,(\partial e/\partial \mathbf{x}_D)$
    - $\partial e/\partial \mathbf{x}_D$ is given
    - $\partial \mathbf{f}_{D-1}/\partial \mathbf{x}_{D-1}$ for many common layers was conveniently shown on an earlier slide
  - Compute the gradient of the error at the input of the next to last layer ... (and repeat)

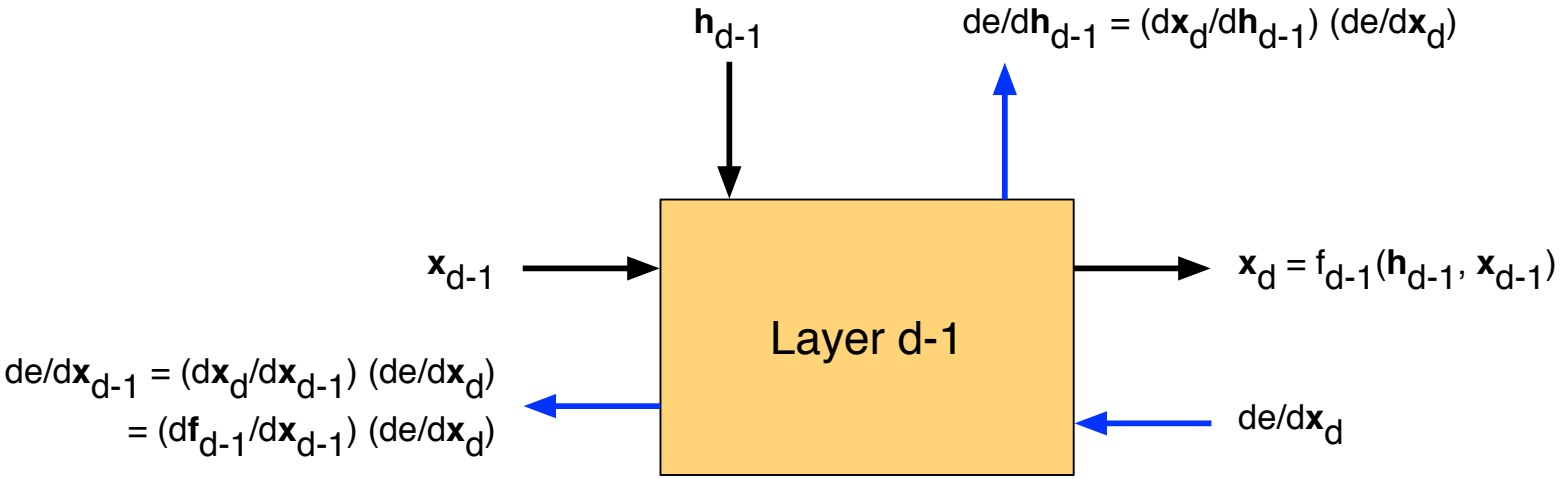


Weight Update

- Goal
  - Update weights to reduce the error
- Strategy
  - Use the error gradient with respect to the feature map output for the current layer provided via back propagation
  - With the derivative of the forward node output with respect to the parameters
  - To determine the error gradient with respect to the parameters (1)
  - Then use gradient descent to adapt the parameters to reduce the error (2)
- In math
  - d-1 is the layer, n is the current time
  - 1: $\partial e/\partial \mathbf{h}_{d-1} = (\partial \mathbf{x}_d/\partial \mathbf{h}_{d-1})\,(\partial e/\partial \mathbf{x}_d) = (\partial \mathbf{f}_{d-1}/\partial \mathbf{h}_{d-1})\,(\partial e/\partial \mathbf{x}_d)$
  - 2: $\mathbf{h}_{n+1,\,d-1} = \mathbf{h}_{n,\,d-1} - \alpha_n\, \partial e/\partial \mathbf{h}_{n,\,d-1}$



Summary For 1 Layer



Summary For 1 Layer

- Example: matrix vector multiplication
  - $\mathbf{x}_d = f_{d-1}(\mathbf{H}_{d-1}, \mathbf{x}_{d-1}) = \mathbf{H}_{d-1}\,\mathbf{x}_{d-1}$
    - Dimensions (M x 1) = (M x K) (K x 1)
- Error gradient with respect to the feature map
  - Given: $\partial e/\partial \mathbf{x}_d$
  - Compute: $\partial e/\partial \mathbf{x}_{d-1} = (\partial \mathbf{x}_d/\partial \mathbf{x}_{d-1})\,(\partial e/\partial \mathbf{x}_d) = (\partial \mathbf{f}_{d-1}/\partial \mathbf{x}_{d-1})\,(\partial e/\partial \mathbf{x}_d) = (\mathbf{H}_{d-1})^T\,(\partial e/\partial \mathbf{x}_d)$
    - Dimensions (K x 1) = (K x M) (M x 1)
- Error gradient with respect to the filter coefficients
  - Given: $\partial e/\partial \mathbf{x}_d$
  - Compute: $\partial e/\partial \mathbf{H}_{d-1} = (\partial \mathbf{x}_d/\partial \mathbf{H}_{d-1})\,(\partial e/\partial \mathbf{x}_d) = (\partial e/\partial \mathbf{x}_d)\,(\partial \mathbf{f}_{d-1}/\partial \mathbf{H}_{d-1}) = (\partial e/\partial \mathbf{x}_d)\,(\mathbf{x}_{d-1})^T$
    - Dimensions (M x K) = (M x 1) (1 x K)
    - Note the reversing of order as a consequence of the transpose for this case (to see match terms with previous case)
    - Note that this implies saving $\mathbf{x}_{d-1}$ from the forward pass
- Weight update
  - Compute: $\mathbf{H}_{n+1,\,d-1} = \mathbf{H}_{n,\,d-1} - \alpha_n\,\partial e/\partial \mathbf{H}_{n,\,d-1}$
    - Time n to time n + 1

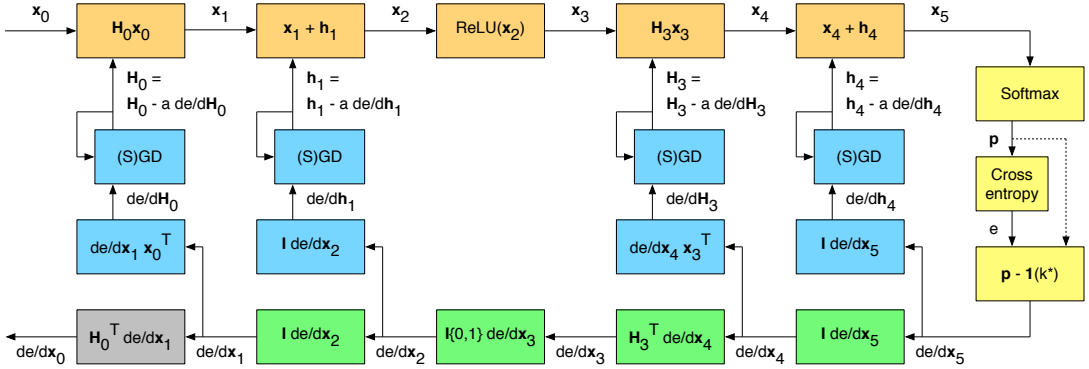
Training Vs Standard Optimization

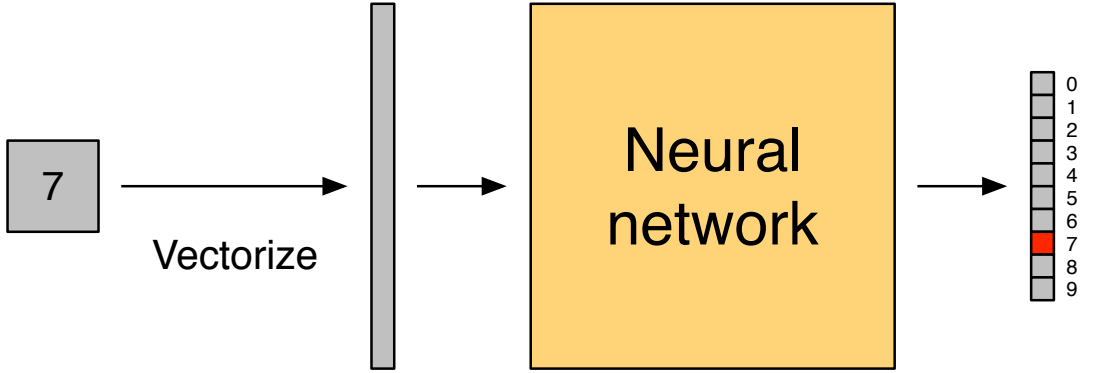
- There are a number of differences between xNN training and optimization
  - Optimize an error function with the training data to perform well on testing data
    - Typically thought of as optimizing for empirical vs true distribution
    - The concept of an input distribution is used as there's typically not a practical deterministic model for the input
  - Use a surrogate error function vs actual error function
    - E.g., softmax + cross entropy vs arg max
    - Also don't use the actual top 1 for all training data at 1x
      - Only use a fraction or batch of it at a time

- These differences highlight the need for generalization
  - Want to train xNNs on training data with a surrogate error function such that they will generalize well to new data with high levels of accuracy
  - In a later lecture will look at various regularization methods to improve generalization

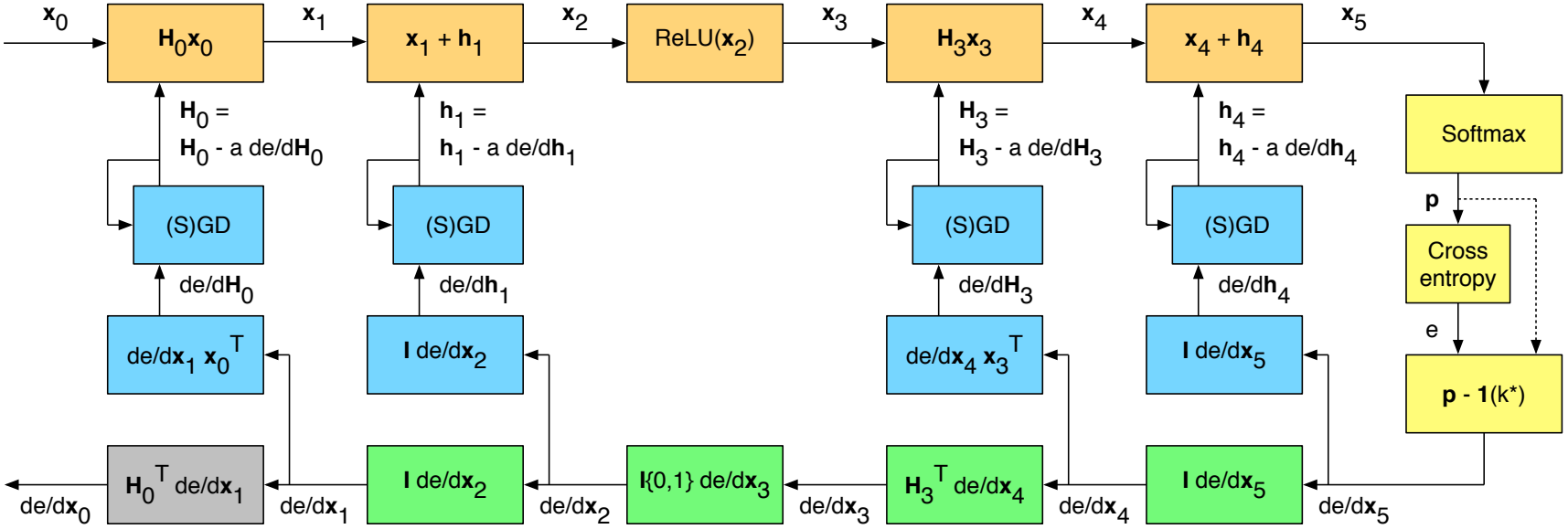
Example

- MNIST digit recognition
  - The hello, world! of machine learning
- Simple network design to illustrate training (not optimal for accuracy)
  - $\mathbf{x}_5 = \mathbf{H}_3\ \mathrm{ReLU}_2(\mathbf{H}_0\ \mathbf{x}_0 + \mathbf{h}_1) + \mathbf{h}_4$
- Goal
  - Make arg max error of testing small
  - Proxy of making training error e small
- What controls the proxy error e
  - For a fixed topology $\mathbf{H}_0$, $\mathbf{h}_1$, $\mathbf{H}_3$ and $\mathbf{h}_4$
- How to adapt $\mathbf{H}_0$, $\mathbf{h}_1$, $\mathbf{H}_3$ and $\mathbf{h}_4$ to make the proxy error e small
  - (Stochastic) gradient descent
- How to find the values (S)GD needs for the update
  - Back propagation





Example


x0
H0x0
x1
x1 + h1
x2
ReLU(x2)
x3
H3x3
x4
x4 + h4
x5
Softmax
H0 =
H0 - a de/dH0
h1 =
h1 - a de/dh1
H3 =
H3 - a de/dH3
h4 =
h4 - a de/dh4
p
Cross entropy
e
p - 1(k*)
(S)GD
(S)GD
(S)GD
(S)GD
de/dH0
de/dh1
de/dH3
de/dh4
de/dx1 x0T
I de/dx2
de/dx4 x3T
I de/dx5
H0T de/dx1
I de/dx2
I{0,1} de/dx3
H3T de/dx4
I de/dx5
de/dx0
de/dx1
de/dx2
de/dx3
de/dx4
de/dx5

Universal Approximation

Definition

- A neural network with a single hidden layer, finite number of neurons and appropriate nonlinearity can approximate arbitrarily closely any continuous function on a compact subset of R^K

- Example: classification
  - The goal is to create a function that maps input images to a 1 hot vector at the output indicating class membership
  - If a function exists then it can be approximated via a neural network
  - Lots of other problems can be cast as finding an appropriate function to map from inputs to outputs

Why Universal Approximation Matters

- Neural networks are a general tool to solve all sorts of problems
  - Theoretically we don't need to learn a 1000 different methods for information extraction and data generation
  - Theoretically we can apply neural networks and if a solution (with described constraints) exists then a neural network can approximate it
  - This is important

- Things not said
  - That a function exists that maps inputs to outputs
  - That a neural network with a single hidden layer the best way to approximate the function
  - How to train the neural network from data

Why Discuss Universal Approx Here

- It makes sense to discuss approximation within the context of calculus
  - Limits, bounds on derivatives, ...
  - And there's not a separate analysis lecture

- It's also nice to think about universal approximate before getting into network design

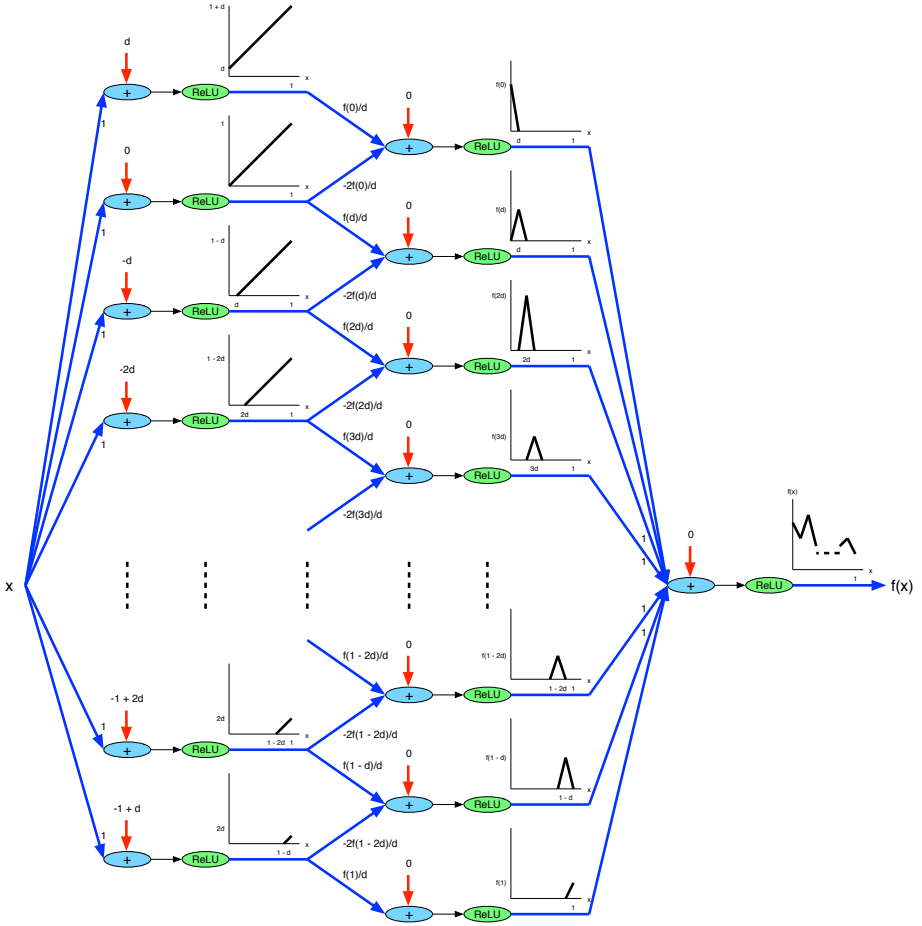
Constructive Proof

- Starting point
  - $x \in [0, 1]$
  - $f(x) \geq 0$
  - $f(x) < \infty$
  - f(x) continuous

- A constructive “proof” (exceedingly loosely speaking) that a countably infinitely wide 3 layer (input, hidden, output) neural network can implement the above f(x) is shown on the next slide

Constructive Proof

- Layer 1 identity weight matrix (blue lines), shifting bias and ReLU create 1/d + 1 building block segments
- Layer 2 creates overlapped width 2d triangles centered at multiples of d of height f(d)
- Layer 3 adds the triangles together to realize f(x) for x $\in$ [0, 1]
- Let d $\rightarrow$ 0



Constructive Proof

- Generalizing to additional cases
  - $x_{min} \leq x \leq x_{max}$ where x_{min} and x_{max} are finite and not limited to [0, 1]
    - Select initial bias values as $x_{min} + d$, x_{min}, …, $-x_{max} + d$
    - Use associated locations for hidden value weights
  - f(x) has a countably infinite number of discontinuities
    - Include a countably infinite number of additional branches each centered at the discontinuity
  - $f_{min} \leq f(x) \leq f_{max}$ where f_{min} and f_{max} are finite and can each be negative, 0 or positive
    - Approximate $f(x) + f_{min}$
    - At the last bias add $-f_{min}$
    - Do not apply the last ReLU

Constructive Proof

- Generalizing to additional cases (continued)
  - Vector input $\mathbf{x} \in R^K$
    - Replicate the input layer for each input component x(k)
    - Combine them in the hidden layer to produce K dimensional pyramids
  - Vector output $\mathbf{y} = f(\mathbf{x}) \in R^M$
    - Replicate the hidden layer and output layer for each output component y(m)

Constructive Proof

- Intuition
  - The more complex the function is with respect to changes the wider the network needs to be to approximate the function at a given level of accuracy
  - Complexity is multiplicative in input and output dimension
  - More layers would allow more complex basis shapes to be created, potentially reducing the width requirements of the network
  - The benefits of depth are observed in practice

References

List

- Convolutional neural networks: theory, implementation and application (chapter 4 calculus)
  - https://github.com/arthurredfern/UT-Dallas-CS-6301-CNNs/blob/master/References/ConvolutionalNeuralNetworks.pdf
- Calculus
  - http://mathonline.wikidot.com/calculus
- The softmax function and its derivative
  - https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
- Convex optimization
  - https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf
- Learning representations by back-propagating errors
  - https://www.nature.com/articles/323533a0
- Automatic differentiation in machine learning: a survey
  - https://arxiv.org/abs/1502.05767
- Approximation by superpositions of a sigmoidal function
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.441.7873&rep=rep1&type=pdf