ResearchAI

sampath kovvali

July 14, 2023

Notation

This section provides a concise reference describing the notation used throughout this document.

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector written as a column vector
$\max(\boldsymbol{a})$	max of $\boldsymbol{a}$, output is a scalar
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{\mathsf{A}}$
$\mathbf{\mathsf{A}}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudo inverse of $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$
$\boldsymbol{x}\boldsymbol{A}$	vector matrix product
$\boldsymbol{A}\boldsymbol{B}$	matrix product

Calculus

$\dfrac{dy}{dx}$	Derivative of y with respect to x
$\dfrac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{X}} y$	Tensor containing derivatives of y with respect to X
$\dfrac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
$\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E} f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parameterized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\delta_{i,j}$	Kronecker delta function, $\begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$

Datasets and Distributions

$\mathbb{X}$	A set of training examples
$x^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\hat{y}^{(i)}$	The predicted value associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

Introduction

The motivation behind writing this book is to understand the math behind machine learning algorithms.

1 Linear Algebra

Transposition of a matrix is give by $\left(\boldsymbol{A}^\top\right)_{i,j} = \boldsymbol{A}_{j,i}$

Algorithm 1: Matrix transposition

Define `Transpose(`$\boldsymbol{A}$`):`
  Require: $\boldsymbol{A}$, the matrix to transpose, size $m \times n$
  Create an empty matrix $\boldsymbol{B}$ with dimensions $m \times n$
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
      $\boldsymbol{B}_{i,j} = \boldsymbol{A}_{j,i}$
    end for
  end for
  Return $\boldsymbol{B}$

1.1 Multiplying Matrices, Vectors, Scalars and their combine operations

Matrix Addition is given by $\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,j} + \boldsymbol{B}_{i,j}$ simply written as $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$

Algorithm 2: Addition of two matrices

Define `Addition(`$\boldsymbol{A}$, $\boldsymbol{B}$`):`
  Require: $\boldsymbol{A}$, size $m \times n$
  Require: $\boldsymbol{B}$, size $m \times n$
  Create an empty matrix $\boldsymbol{C}$ with dimensions $m \times n$
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
      $\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,j} + \boldsymbol{B}_{i,j}$
    end for
  end for
  Return $\boldsymbol{C}$

Matrix multiplication is give by $\boldsymbol{C}_{i,j} = \sum_k \boldsymbol{A}_{i,k}\boldsymbol{B}_{k,j}$ simply written as $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$

Algorithm 3: Multiplication of two matrices

Define `Multiplication(`$\boldsymbol{A}$, $\boldsymbol{B}$`):`
    Require: $\boldsymbol{A}$, size $m \times n$
    Require: $\boldsymbol{B}$, size $n \times p$
    Create an empty matrix $\boldsymbol{C}$ with dimensions $m \times p$
    for $i = 1$ **to** m **do**
        for $j = 1$ **to** p **do**
            $\boldsymbol{C}_{i,j} = 0$
            for $k = 1$ **to** n **do**
                $\boldsymbol{C}_{i,j} = \boldsymbol{C}_{i,j} + \boldsymbol{A}_{i,k}\boldsymbol{B}_{k,j}$
            end for
        end for
    end for
    Return $\boldsymbol{C}$

Element-wise Matrix multiplication is give by $\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,j}\boldsymbol{B}_{i,j}$ simply written as $\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$

Algorithm 4: Element-wise multiplication of two matrices

Define `Multiplication(`$\boldsymbol{A}$, $\boldsymbol{B}$`):`
    Require: $\boldsymbol{A}$, size $m \times n$
    Require: $\boldsymbol{B}$, size $m \times m$
    Create an empty matrix $\boldsymbol{C}$ with dimensions $m \times n$
    for $i = 1$ **to** m **do**
        for $j = 1$ **to** n **do**
            $\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,j}\boldsymbol{B}_{i,j}$
        end for
    end for
    Return $\boldsymbol{C}$

Vector addition is give by $\boldsymbol{c}_i = \boldsymbol{a}_i + \boldsymbol{b}_i$ simply written as $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$

Algorithm 5: Vector addition of two vectors

Define `Addition(`$\boldsymbol{a}$, $\boldsymbol{b}$`)`**:**
- **Require:** $\boldsymbol{a}$, size m
- **Require:** $\boldsymbol{b}$, size m
- Create an empty vector $\boldsymbol{c}$ with dimensions m
- **for** $i = 1$ **to** m **do**
  - $\boldsymbol{c}_i = \boldsymbol{a}_i + \boldsymbol{b}_i$
- **end for**
- **Return** $\boldsymbol{c}$

Vector dot product is give by $c = \sum_i \boldsymbol{a}_i \boldsymbol{b}_i$ simply written as $\boldsymbol{c} = \boldsymbol{a}^\top \boldsymbol{b}$

Algorithm 6: Vector dot product

Define `Addition(`$\boldsymbol{a}$, $\boldsymbol{b}$`)`**:**
- **Require:** $\boldsymbol{a}$, size m
- **Require:** $\boldsymbol{b}$, size m
- **for** $i = 1$ **to** m **do**
  - $c = c + \boldsymbol{a}_i \boldsymbol{b}_i$
- **end for**
- **Return** $\boldsymbol{c}$

Vector matrix product is give by $\boldsymbol{c}^\top = \sum_k \boldsymbol{a}_k \boldsymbol{B}_{i,k}$ simply written as $\boldsymbol{c}^\top = \boldsymbol{a}^\top \boldsymbol{B}$

Algorithm 7: Vector matrix product

Define `Multiplication(`$\boldsymbol{a}$, $\boldsymbol{B}$`)`**:**
- **Require:** $\boldsymbol{a}$, size m
- **Require:** $\boldsymbol{B}$, size $m \times n$
- Create an empty vector $\boldsymbol{c}$ with dimensions m
- **for** $i = 1$ **to** n **do**
  - **for** $j = 1$ **to** m **do**
    - $\boldsymbol{c}_j = \boldsymbol{c}_j + \boldsymbol{a}_j \boldsymbol{B}_{j,i}$
  - **end for**
- **end for**
- **Return** $\boldsymbol{c}^\top$

Matrix vector addition is give by $\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,j} + \boldsymbol{b}_j$ simply written as $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{b}^\top$ where each row of $\boldsymbol{A}$ is added with $\boldsymbol{b}^\top$ also called as broadcasting.

Algorithm 8: Matrix vector addition

Define `Multiplication(`$\boldsymbol{a}$, $\boldsymbol{B}$`)`:

  Require: $\boldsymbol{B}$, size $m \times n$

  Require: $\boldsymbol{a}$, size n

  Create an empty matrix $\boldsymbol{C}$ with dimensions $m \times n$

  for $i = 1$ **to** m **do**

    for $j = 1$ **to** n **do**

      $\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,j} + \boldsymbol{b}_j$

    end for

  end for

  Return $\boldsymbol{C}$

Neural Networks

The working principles and architectures of various **NN** are inspired from many papers.

2 Layers

All types of layers for any type of networks.

2.1 Dense

These are the full connected, where one layer's output is input to the next layer's during **forward pass** and the reverse when using **backpropagration algorithm**.

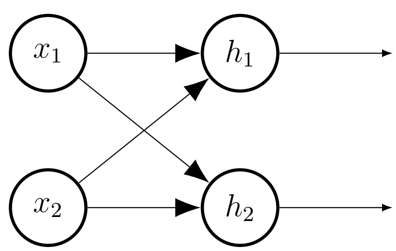

Figure 1: Simple feed forward network of one **Dense** layer with 2 inputs $x1$, $x2$ and two hidden units $h1$, $h2$ and one output y. And each **connection arrow** represents a weight and each **hidden unit** has corresponding bias.

For the hidden layer if the inputs be $\boldsymbol{x}$, weights be $\boldsymbol{W}$ where $\boldsymbol{W}_{0,:}$ the connections from $x1$ to $\boldsymbol{h}$ and so on, biases $\boldsymbol{b}$. Then output $\boldsymbol{o}$ is given by

$$\boldsymbol{o} = \boldsymbol{x}^\top \boldsymbol{W} + \boldsymbol{b}^\top \tag{1}$$

Reference Algorithms 7 and 5

Batched data

If the inputs are to be passed as a batch of data $\boldsymbol{X}$, where $\boldsymbol{X}_{0,:}$ is the first batch then the output $\boldsymbol{O}$ for one layer is computed as

$$\boldsymbol{O} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}^\top \tag{2}$$

Reference Algorithms 3 and 8

Backpropagation

let $\boldsymbol{G}$ be the incoming gradients during back-propagation using chain rule then,

- $\boldsymbol{X}$ grads are computed as $\boldsymbol{G}\boldsymbol{W}^\top$
- $\boldsymbol{W}$ grads are computed as $\boldsymbol{X}^\top\boldsymbol{G}$
- $\boldsymbol{b}$ grads are computed as $\sum_i \boldsymbol{G}_{i,j}, \forall j$, simply sum along axis=0.

Algorithm 9: bias grads

Define `Grad(`$\boldsymbol{G}$`)`**:**
  Require: $\boldsymbol{G}$, incoming gradients of size $m \times n$
  Create a vector $\boldsymbol{g}$ of size n
  for $j = 1$ **to** n **do**
    for $i = 1$ **to** n **do**
      $\boldsymbol{g}_j = \boldsymbol{g}_j + \boldsymbol{G}i, j$
    end for
  end for
  Return $\boldsymbol{g}$

3 Activation functions

All types of activation's for any layers.

3.1 ReLU

Rectified Linear Unit

After the forward pass, we can get activated output by

$$\text{ReLU}(z) = \max\{0, z\} \tag{3}$$

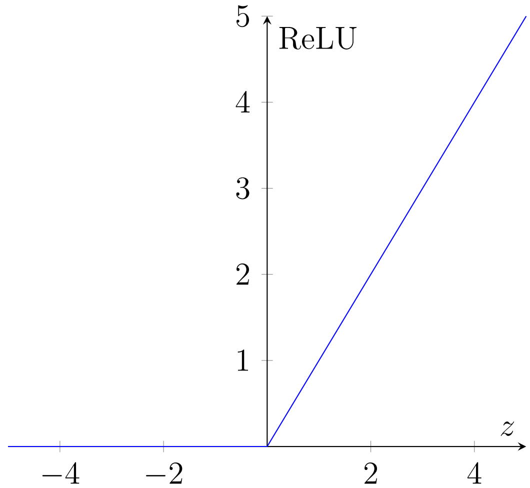

Figure 2: ReLU function

Algorithm 10: Rectified Linear Unit

Define `ReLU(`z`)`**:**
    Require: z, input to ReLU function
    Create a scalar y
    if $z \geq 0$ **then**
        $y = z$
    else
        $y = 0$
    end if
    Return y

Backpropagation

To computer gradients

$$\frac{d}{dz}\text{ReLU}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Algorithm 11: Rectified Linear Unit grad

Define `Grad(`z`)`**:**
  Require: z, input to ReLU function
  Create a scalar y
  if $z \geq 0$ **then**
    $y = 1$
  else
    $y = 0$
  end if
  Return y

3.2 Softmax

It is usually used in the output layer.

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{5}$$

so $\hat{y}_i = \text{softmax}(\boldsymbol{x})_i$

3.2.1 Numerical stability

For overflow prevention $\boldsymbol{x} = \boldsymbol{x} - \max(\boldsymbol{x})$.

- All exponential values will be between 0 and 1.
- This prevents overflow errors (but we are still prone to under-flows)
- At least one of the exponential values is 1 i.e. at least one value is guaranteed not to underflow
- Our denominator will always be $\geq$1, preventing division by zero errors.
- We have at least one non-zero numerator, so softmax can't result in a zero vector

[1]

Algorithm 12: Softmax

Define `Softmax(`$\boldsymbol{x}$`)`:
- **Require:** $\boldsymbol{x}$, logits
- Let the length of the vector be m
- Create a vector $\hat{\boldsymbol{y}}$ of size m
- Create a vector $\boldsymbol{e}$ of size m to store the exponents
- Create a scalar s to store the sum of exponents vector $\boldsymbol{e}$
- `// numerical stability`
- **for** $i = 1$ **to** m **do**
  - $x_i = x_i - \max(x_i)$
- **end for**
- `// exponents`
- **for** $i = 1$ **to** m **do**
  - $\boldsymbol{e}_i = \exp(x_i)$
- **end for**
- `// sum of exponents`
- **for** $i = 1$ **to** m **do**
  - $s = s + e_i$
- **end for**
- `// probabilities of exponents`
- **for** $i = 1$ **to** m **do**
  - $\hat{y}_i = \frac{e_i}{s}$
- **end for**
- **Return** $\hat{\boldsymbol{y}}$

Backpropagation

To computer gradients

$$\frac{\partial \hat{y}_i}{\partial x_j} = \hat{y}_i(\delta_{i,j} - \hat{y}_j) \tag{6}$$

The result is a **Jacobian Matrix**

$$J = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial x_1} & \frac{\partial \hat{y}_1}{\partial x_2} & \cdots & \frac{\partial \hat{y}_1}{\partial x_m} \\ \frac{\partial \hat{y}_2}{\partial x_1} & \frac{\partial \hat{y}_2}{\partial x_2} & \cdots & \frac{\partial \hat{y}_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_m}{\partial x_1} & \frac{\partial \hat{y}_m}{\partial x_2} & \cdots & \frac{\partial \hat{y}_m}{\partial x_m} \end{bmatrix} \tag{7}$$

4 Cost functions

4.1 Cross Entropy

Cross-entropy is a measure from the field of information theory that calculates the difference between two probability distributions. It is commonly used in machine learning as a loss function.

$$L(y, \hat{y}) = -\sum_{i} y^{(i)} \log \hat{y}^{(i)} \tag{8}$$

4.1.1 Numerical stability

Cross-entropy results in **inf** because of log(0) so, we need to clip inputs from both sides by a small positive value like 1×10^{-7}.

Algorithm 13: Cross Entropy

Define `CrossEntropy(`$\boldsymbol{y}, \hat{\boldsymbol{y}}$`):`
  Require: $\boldsymbol{y}$, target vector
  Require: $\hat{\boldsymbol{y}}$, prediction vector
  Let the length of the vector be m
  Create a scalar l
  Clip the values of $\hat{\boldsymbol{y}}$ by 1×10^{-7} from both sides
  for $i = 1$ **to** m **do**
    $l = l + y_i \log \hat{y}_i$
  end for
  Return l

Backpropagation

To computer gradients

$$\frac{\partial L(y, \hat{y})}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \tag{9}$$

Algorithm 14: Cross Entropy Grad

Define `CrossEntropyGrad`($\boldsymbol{y}, \hat{\boldsymbol{y}}$):
  Require: $\boldsymbol{y}$, target vector
  Require: $\hat{\boldsymbol{y}}$, prediction vector
  Create a vector $\boldsymbol{g}$
  Let the length of each vector be m
  for $i = 1$ **to** m **do**
    $g_i = -\frac{y_i}{\hat{y}_i}$
  end for
  Return $\boldsymbol{g}$

5 Optimizers

5.1 SGD

Stochastic Gradient Descent

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, p. 78. URL: http://www.deeplearningbook.org.