Back to Backprop

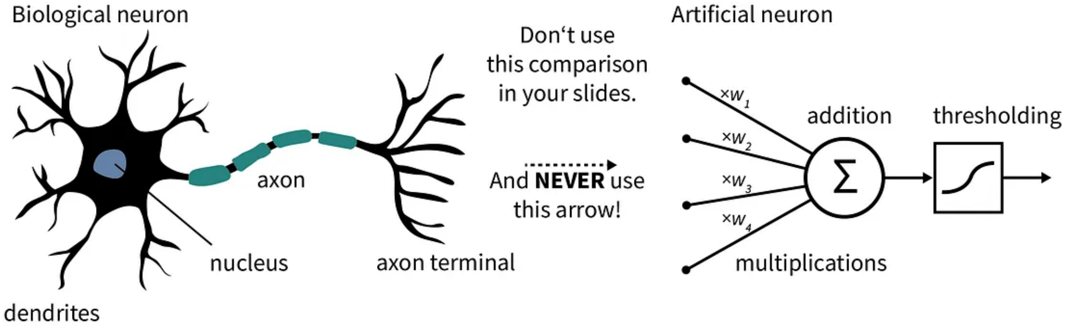
Neural networks from scratch

Jovan Krajevski

May 2025

Table of Contents

What even is a neural network?



img source: Stop using biological analogies to describe AI. It's 99.999% wrong.

Why the biological analogy?

- It is supposed to be useful... as useful as:
  - the ”car is an artificial horse” analogy
  - the ”plane is an artificial bird” analogy
- But it is compelling...
  - the road to artificial ”intelligence” is paved with artificial ”neurons”
- ...and clouds the judgment when doing research

Neurons as calculators

- Neurons can multiply numbers
- Neurons can add numbers
- Neurons can choose the larger number
- But they usually can't do a lot more
- Neurons are functions
  - Multiple inputs can be related to the same output
  - Only one output can be related to a given input

The purpose of this lecture

Boring reasons

- Know what's under the hood as an intellectual curiosity
- Improve on the core algorithm

Practical reasons

- Backprop is a leaky abstraction
- Develop a mathematical intuition useful for research/debugging

- Vanishing gradients on sigmoids (or tanh)
- Dead ReLUs
- Karpathy: Yes you should understand backprop

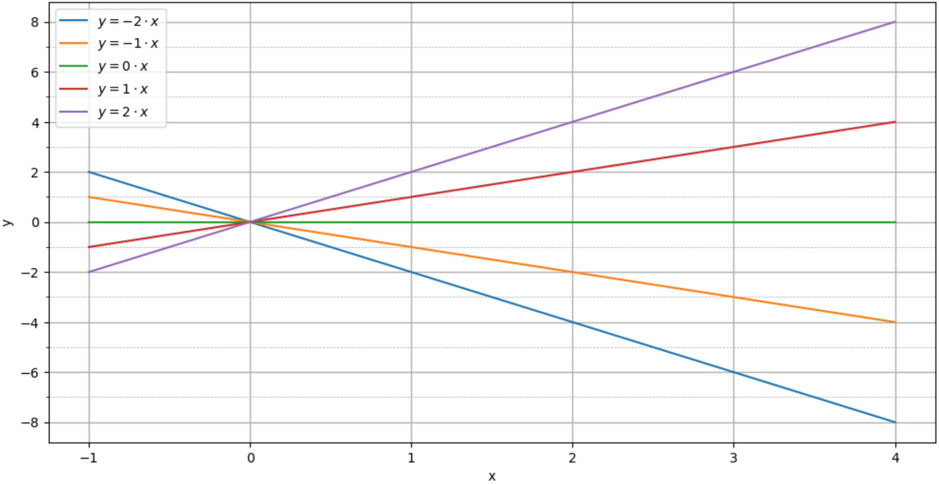
Table of Contents

A linear function

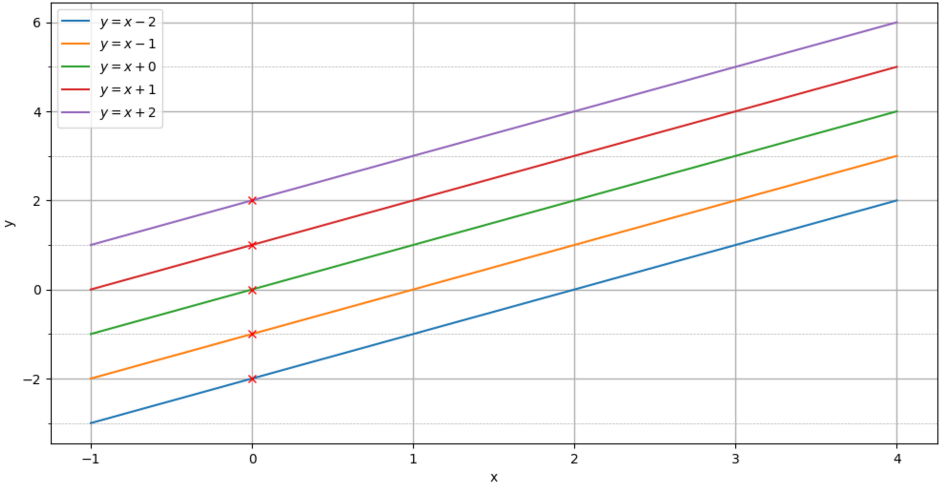
$$y = xk + m$$

- If the plane was a grid
  - m is where you start
  - if you move one block to the right, you move k blocks up
- k - the slope - the weight - the rate of change
- m - the intercept - the bias - the intersection with y-axis
  - The intersection occurs when $x = 0$
  - $x = 0 \implies y = 0 \cdot k + m = m$

Tweaking the slope



Tweaking the intercept

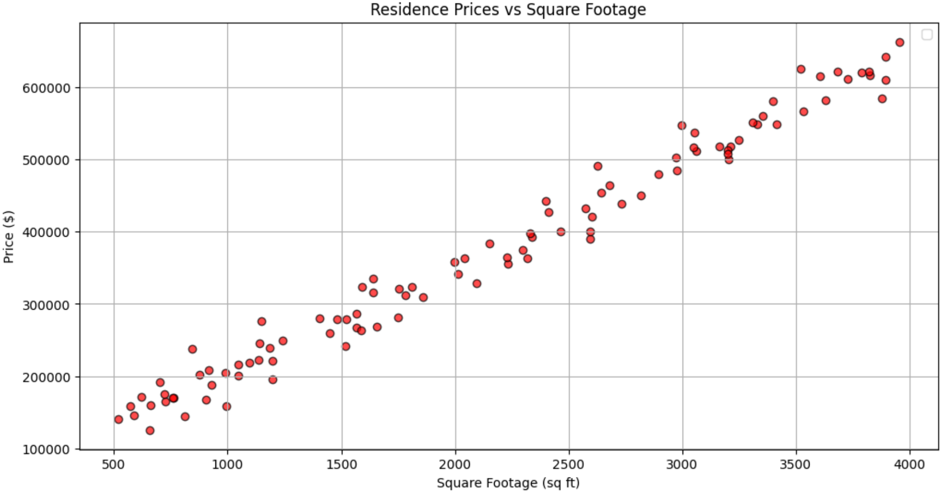


Parameters

- The slope and the intercept - parameters
- Every straight line can be expressed by tweaking k and m
  - Except the vertical line; why?
- So why is this useful?

Let's look at some data

2 Linear regression



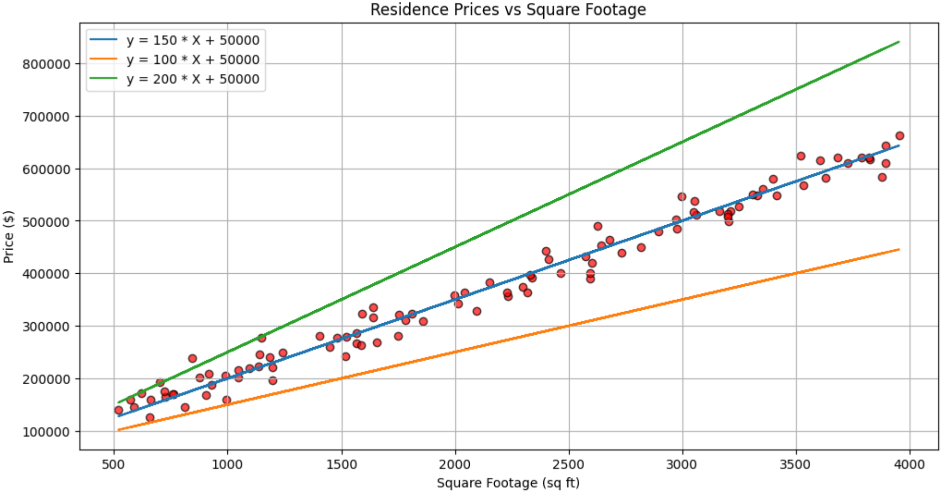
The task of linear regression

- Fit a linear function to the data
  - Find values for k and m that approximate the data
  - Use k and m to make out-of-sample predictions
- What is a good fit?
  - For each sample $(x_i, y_i); i \in \mathbb{N}, i < N$ and fixed values for $k = \widehat{k}$ and $m = \widehat{m}$ calculate the distance between $\widehat{y} = x\widehat{k} + \widehat{m}$ and y_i
  - $err_i = |y_i - \widehat{y}_i|$ or $error_i = (y_i - \widehat{y}_i)^2$
  - $err_{avg} = \sum_{i=1}^{N} err_i / N$
- We need to optimize:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \widehat{y}_i)^2$$

Some ”fits”

2 Linear regression



The errors

2 Linear regression

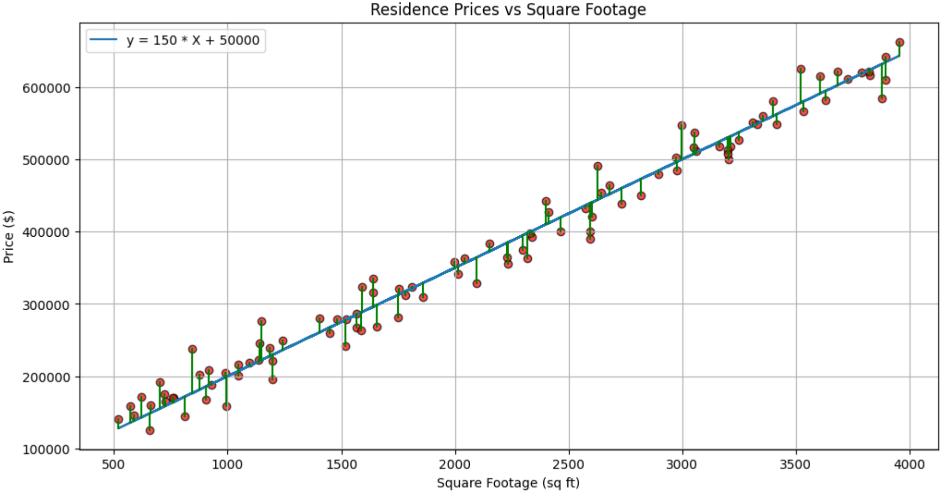


Table of Contents

3 More predictors

Multiple predictors

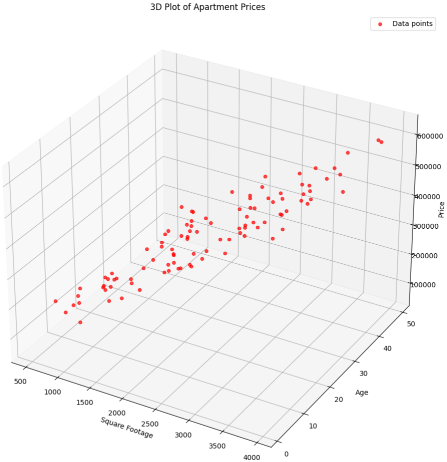
3 More predictors

- x_i - predictor - regressor - attribute - independent variable - feature?
- y_i - target - dependent variable
- We assumed that only the sq footage is available to us
  - But what if we have multiple predictors, like apartment age, floor number, city, location?

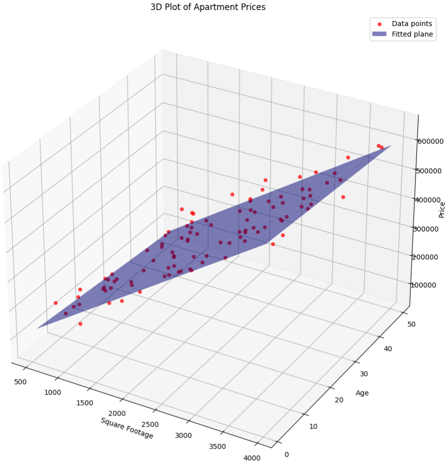
$$\mathbf{x}_i = \begin{pmatrix} x_i^{(1)} & x_i^{(2)} & \dots & x_i^{(D)} \end{pmatrix}$$

Let's look at some 3D data

3 More predictors



Let’s ”fit” that 3D data



Linear regression with multiple predictors

- Let's change the notation a little bit
  - let the slope be w - ***weight***
  - let the intercept be b - ***bias***

$$\widehat{y}_i = x_i w + b$$

- For multiple predictors:

$$\widehat{y}_i = x_i^{(1)} w_1 + x_i^{(2)} w_2 + ... + x_i^{(D)} w_D + b$$

Table of Contents

4 The matrix form

And now let's introduce vectors...

4 The matrix form

- Vectors - quantities that have magnitude and direction
- If we look at a vector as a point (we can't really...)
  - magnitude is the distance from the origin
  - direction is always origin $\rightarrow$ vector
- Vectors are finite sequences of a fixed length
  - so we can represent the sample $\mathbf{x}_i$ as a row vector
  - we can also represent the weights $\mathbf{w}$ as a column vector

$$\mathbf{x}_i = \begin{pmatrix} x_i^{(1)} & x_i^{(2)} & \dots & x_i^{(D)} \end{pmatrix}, \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$$

Why bother with vectors?

- Because of the vector operations (they are faster)
- Because of the benefits of linear algebra
- The dot (inner) product

$$\mathbf{x}_i\mathbf{w} = x_i^{(1)}w_1 + x_i^{(2)}w_2 + ... + x_i^{(D)}w_D$$

$$\implies \widehat{y}_i = \mathbf{x}_i\mathbf{w} + b$$

Representing data in matrix form

- A matrix is a rectangular array; you can think of it as:
  - a row vector consisting of column vectors
  - a column vector of row vectors

$$\mathbf{X} = \begin{pmatrix} — \mathbf{x}_1 — \\ — \mathbf{x}_2 — \\ \vdots \\ — \mathbf{x}_N — \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \ldots & \mathbf{x}^{(D)} \\ | & | & & | \end{pmatrix}$$

- Rows are samples, columns are predictors!
- What if we multiplied $\mathbf{X}\mathbf{w}$?

The matrix-vector product

4 The matrix form

$$\mathbf{X}\mathbf{w} = \begin{pmatrix} \text{---}\mathbf{x}_1\text{---} \\ \text{---}\mathbf{x}_2\text{---} \\ \vdots \\ \text{---}\mathbf{x}_N\text{---} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1\mathbf{w} \\ \mathbf{x}_2\mathbf{w} \\ \vdots \\ \mathbf{x}_N\mathbf{w} \end{pmatrix} = \begin{pmatrix} x_1^{(1)}w_1 + x_1^{(2)}w_2 + \ldots + x_1^{(D)}w_D \\ x_2^{(1)}w_1 + x_2^{(2)}w_2 + \ldots + x_2^{(D)}w_D \\ \cdots \\ x_N^{(1)}w_1 + x_N^{(2)}w_2 + \ldots + x_N^{(D)}w_D \end{pmatrix}$$

$$\widehat{\mathbf{y}} = \begin{pmatrix} \widehat{y}_1 \\ \widehat{y}_2 \\ \vdots \\ \widehat{y}_N \end{pmatrix} = \mathbf{X}\mathbf{w} + b$$

Table of Contents

5 Linear projections

Let us look at a different problem now

- Forget about ”fitting” for a second...
- Let $\mathbf{X}$ be $N \times 3$ matrix representing real estate data
  - column 1 - sq footage
  - column 2 - number of bedrooms
  - column 3 - age
- We are interested in estimating:
  - $\mathbf{h}^{(1)}$ - space and comfort
  - $\mathbf{h}^{(2)}$ - property condition

$$\mathbf{w}_1 = \begin{pmatrix} 0.8 \\ 0.6 \\ -0.2 \end{pmatrix} \quad b^{(1)} = 0.3 \quad \mathbf{w}_2 = \begin{pmatrix} 0.2 \\ 0.1 \\ -1.2 \end{pmatrix} \quad b^{(2)} = -0.7$$

$$\mathbf{h}^{(1)} = \mathbf{X}\mathbf{w}_1 + b^{(1)} \quad \mathbf{h}^{(2)} = \mathbf{X}\mathbf{w}_2 + b^{(2)}$$

Representing weights in matrix form

- Let M be the number of linear regressions

$$\mathbf{W} = \begin{pmatrix} \text{—}\mathbf{w}_1\text{—} \\ \text{—}\mathbf{w}_2\text{—} \\ \vdots \\ \text{—}\mathbf{w}_D\text{—} \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \mathbf{w}^{(1)} & \mathbf{w}^{(2)} & \dots & \mathbf{w}^{(M)} \\ | & | & & | \end{pmatrix}$$

- What if we multiplied $\mathbf{XW}$?
- And maybe created a row vector $\mathbf{b} = \begin{pmatrix} b^{(1)} & b^{(2)} & \dots & b^{(M)} \end{pmatrix}$?

The matrix product
5 Linear projections

$$
\mathbf{X}\mathbf{W} = \begin{pmatrix} —\mathbf{x}_1— \\ —\mathbf{x}_2— \\ \vdots \\ —\mathbf{x}_N— \end{pmatrix} \begin{pmatrix} | & | & & | \\ \mathbf{w}^{(1)} & \mathbf{w}^{(2)} & \dots & \mathbf{w}^{(M)} \\ | & | & & | \end{pmatrix}
$$

$$
= \begin{pmatrix} \mathbf{x}_1\mathbf{w}^{(1)} & \mathbf{x}_1\mathbf{w}^{(2)} & \dots & \mathbf{x}_1\mathbf{w}^{(M)} \\ \mathbf{x}_2\mathbf{w}^{1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{x}_{N-1}\mathbf{w}^{(M)} \\ \mathbf{x}_N\mathbf{w}^{(1)} & \dots & \mathbf{x}_N\mathbf{w}^{(M-1)} & \mathbf{x}_N\mathbf{w}^{(M)} \end{pmatrix}
$$

Broadcasting

5 Linear projections

$$\mathbf{X}\mathbf{W} + \mathbf{b} = \begin{pmatrix} \mathbf{x}_1\mathbf{w}^{(1)} & \mathbf{x}_1\mathbf{w}^{(2)} & \dots & \mathbf{x}_1\mathbf{w}^{(M)} \\ \mathbf{x}_2\mathbf{w}^{1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{x}_{N-1}\mathbf{w}^{(M)} \\ \mathbf{x}_N\mathbf{w}^{(1)} & \dots & \mathbf{x}_N\mathbf{w}^{(M-1)} & \mathbf{x}_N\mathbf{w}^{(M)} \end{pmatrix} + \begin{pmatrix} b^{(1)} & b^{(2)} & \dots & b^{(M)} \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{x}_1\mathbf{w}^{(1)} + b^{(1)} & \mathbf{x}_1\mathbf{w}^{(2)} + b^{(2)} & \dots & \mathbf{x}_1\mathbf{w}^{(M)} + b^{(M)} \\ \mathbf{x}_2\mathbf{w}^{(1)} + b^{(1)} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{x}_{N-1}\mathbf{w}^{(M)} + b^{(M)} \\ \mathbf{x}_N\mathbf{w}^{(1)} + b^{(1)} & \dots & \mathbf{x}_N\mathbf{w}^{(M-1)} + b^{(M-1)} & \mathbf{x}_N\mathbf{w}^{(M)} + b^{(M)} \end{pmatrix}$$

Linear projection

- $\mathbf{XW} + \mathbf{b}$ - multiple linear regressions - linear projection
  - a.k.a. a linear layer
- It projects data in a new space
- Useful for:
  - Feature extraction
  - Linear separability
  - Data compression (if $M < D$) or expansion (if $M > D$)
- We can "stack" multiple linear projections one after another

"Stacking" linear layers

- Let L be the number of linear layers
  - space dims: $M_0 = D, M_1, M_2, ..., M_L = 1$
  - weights: $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L$; $\mathbf{W}_i$ is a $M_{i-1} \times M_i$ matrix
  - biases: $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_L$; $\mathbf{b}_i$ is a M_i dimensional row vector
    - $N \times M_i$ matrix after broadcasting!!!
  - outputs: $\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_L = \widehat{\mathbf{y}}$; $\mathbf{H}_i$ is a $N \times M_i$ matrix

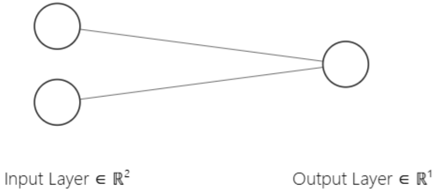
$$\mathbf{H}_1 = \mathbf{X}\mathbf{W}_1 + \mathbf{b}_1$$

$$\mathbf{H}_2 = \mathbf{H}_1\mathbf{W}_2 + \mathbf{b}_2$$

$$\vdots$$

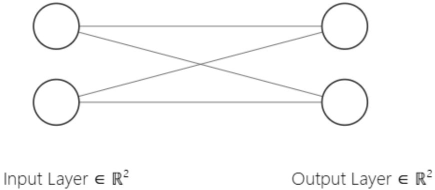
$$\widehat{\boldsymbol{y}} = \mathbf{H}_L = \mathbf{H}_{L-1}\mathbf{W}_L + \mathbf{b}_L$$

Let’s visualize this



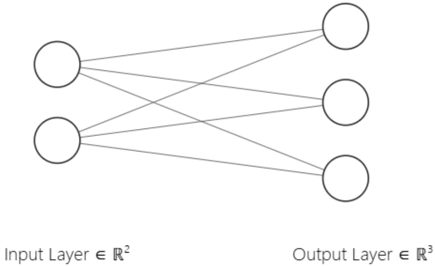
created with: https://alexlenail.me/NN-SVG/

Let’s visualize this



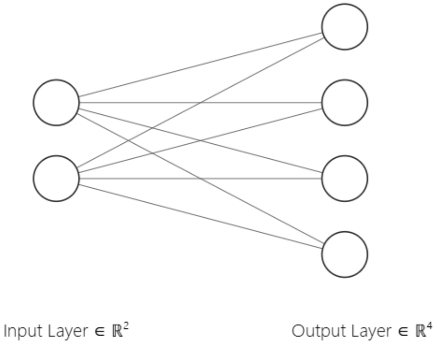
created with: https://alexlenail.me/NN-SVG/

Let’s visualize this



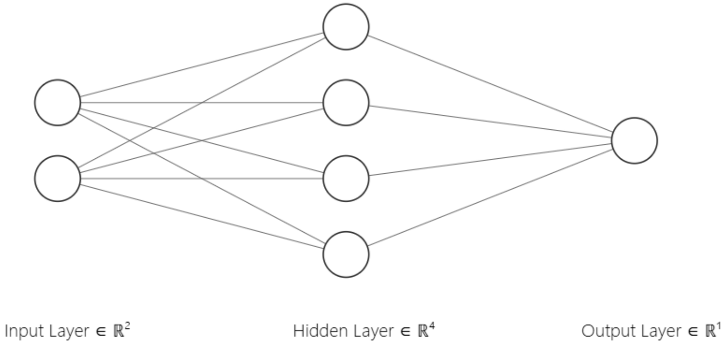
created with: https://alexlenail.me/NN-SVG/

Let's visualize this



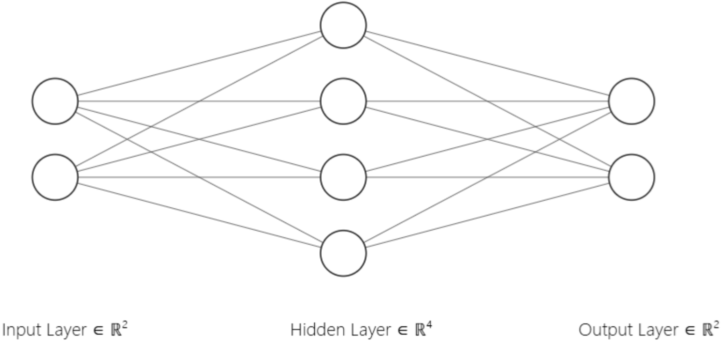
created with: https://alexlenail.me/NN-SVG/

Let's visualize this



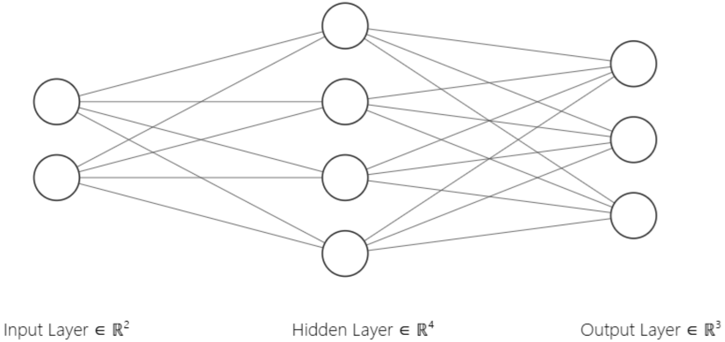
created with: https://alexlenail.me/NN-SVG/

Let's visualize this



created with: https://alexlenail.me/NN-SVG/

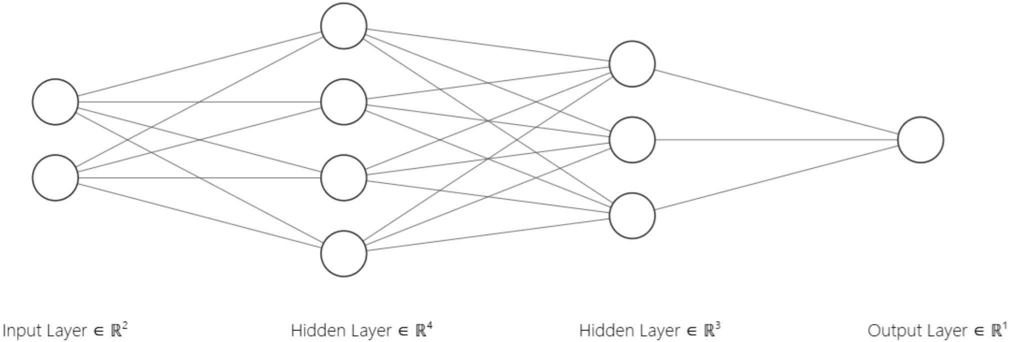
Let's visualize this



created with: https://alexlenail.me/NN-SVG/

Let’s visualize this

5 Linear projections



created with: https://alexlenail.me/NN-SVG/

Table of Contents

6 Linear layers are not enough

Some properties of the matrix product

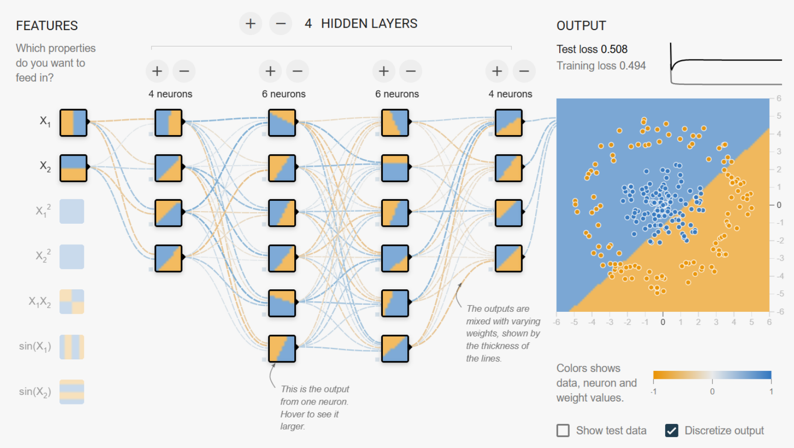
- Non-commutative: $\mathbf{AB} \neq \mathbf{BA}$
  - if $\mathbf{A}$ is $n \times p$ and $\mathbf{B}$ is $p \times m$, then $\mathbf{AB}$ is $n \times m$ and $\mathbf{BA}$ does not exist
- Associative: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
  - but $(\mathbf{AB})\mathbf{C} \neq (\mathbf{BC})\mathbf{A}$ (non-commutative)
- Distributive: $(\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{AC} + \mathbf{BC}$
  - but $(\mathbf{A} + \mathbf{B})\mathbf{C} \neq \mathbf{CA} + \mathbf{CB}$ (non-commutative)

So what if we stack multiple linear layers?

$$
\begin{aligned}
\mathbf{H}_1 &= \mathbf{X}\mathbf{W}_1^{[D\times M_1]} + \mathbf{b}_1^{[N\times M_1]} \\
\mathbf{H}_2 &= \mathbf{H}_1\mathbf{W}_2^{[M_1\times M_2]} + \mathbf{b}_2^{[N\times M_2]} \\
&= (\mathbf{X}\mathbf{W}_1^{[D\times M_1]} + \mathbf{b}_1^{[N\times M_1]})\mathbf{W}_2^{[M_1\times M_2]} + \mathbf{b}_2^{[N\times M_2]} \leftarrow \textit{distributive rule} \\
&= \mathbf{X}\mathbf{W}_1^{[D\times M_1]}\mathbf{W}_2^{[M_1\times M_2]} + \mathbf{b}_1^{[N\times M_1]}\mathbf{W}_2^{[M_1\times M_2]} + \mathbf{b}_2^{[N\times M_2]} \leftarrow \textit{associative rule} \\
&= \mathbf{X}\mathbf{Q}_2^{[D\times M_2]} + \mathbf{U}_2^{[N\times M_2]} \leftarrow \textbf{linear projection!!!} \\
\mathbf{H}_i &= \mathbf{H}_{i-1}\mathbf{W}_i^{[M_{i-1}\times M_i]} + \mathbf{b}_i^{[N\times M_i]} \\
&= (\mathbf{X}\mathbf{Q}_{i-1}^{[D\times M_{i-1}]} + \mathbf{U}_{i-1}^{[N\times M_{i-1}]})\mathbf{W}_i^{[M_{i-1}\times M_i]} + \mathbf{b}_i^{[N\times M_i]} \\
&= \mathbf{X}\mathbf{Q}_{i-1}^{[D\times M_{i-1}]}\mathbf{W}_i^{[M_{i-1}\times M_i]} + \mathbf{U}_{i-1}^{[N\times M_{i-1}]}\mathbf{W}_i^{[M_{i-1}\times M_i]} + \mathbf{b}_i^{[N\times M_i]} \\
&= \mathbf{X}\mathbf{Q}_i^{[D\times M_i]} + \mathbf{U}_i^{[N\times M_i]} \leftarrow \textbf{linear projection!!!}
\end{aligned}
$$

Stacking multiple linear layers is useless

6 Linear layers are not enough



created with: TensorFlow Playground

Non-linearities

- When you stack multiple linear layers, you end up having a linear projection
  - L layers with dims $M_1, ..., M_L \iff$ 1 layer with dim M_L
  - proof by mathematical induction
- Solution: introduce a non-linear function f between layers - activation
  - can vary depending on after which layer it is introduced

$$\mathbf{H}_1 = f(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)$$
$$\mathbf{H}_2 = f(\mathbf{H}_1\mathbf{W}_2 + \mathbf{b}_2)$$
$$\vdots$$
$$\mathbf{H}_{L-1} = f(\mathbf{H}_{L-2}\mathbf{W}_{L-1} + \mathbf{b}_{L-1})$$
$$\widehat{\boldsymbol{y}} = \mathbf{H}_L = \mathbf{H}_{L-1}\mathbf{W}_L + \mathbf{b}_L$$

Elementwise operations

6 Linear layers are not enough

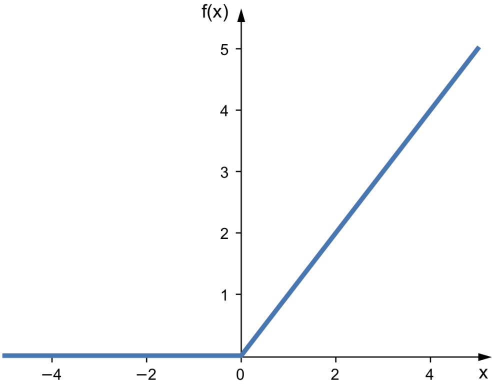
$$f(\begin{pmatrix} \mathbf{x}_1\mathbf{w}^{(1)} + b^{(1)} & \mathbf{x}_1\mathbf{w}^{(2)} + b^{(2)} & \dots & \mathbf{x}_1\mathbf{w}^{(M)} + b^{(M)} \\ \mathbf{x}_2\mathbf{w}^{(1)} + b^{(1)} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{x}_{N-1}\mathbf{w}^{(M)} + b^{(M)} \\ \mathbf{x}_N\mathbf{w}^{(1)} + b^{(1)} & \dots & \mathbf{x}_N\mathbf{w}^{(M-1)} + b^{(M-1)} & \mathbf{x}_N\mathbf{w}^{(M)} + b^{(M)} \end{pmatrix}) =$$

$$\begin{pmatrix} f(\mathbf{x}_1\mathbf{w}^{(1)} + b^{(1)}) & f(\mathbf{x}_1\mathbf{w}^{(2)} + b^{(2)}) & \dots & f(\mathbf{x}_1\mathbf{w}^{(M)} + b^{(M)}) \\ f(\mathbf{x}_2\mathbf{w}^{(1)} + b^{(1)}) & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & f(\mathbf{x}_{N-1}\mathbf{w}^{(M)} + b^{(M)}) \\ f(\mathbf{x}_N\mathbf{w}^{(1)} + b^{(1)}) & \dots & f(\mathbf{x}_N\mathbf{w}^{(M-1)} + b^{(M-1)}) & f(\mathbf{x}_N\mathbf{w}^{(M)} + b^{(M)}) \end{pmatrix}$$

ReLU

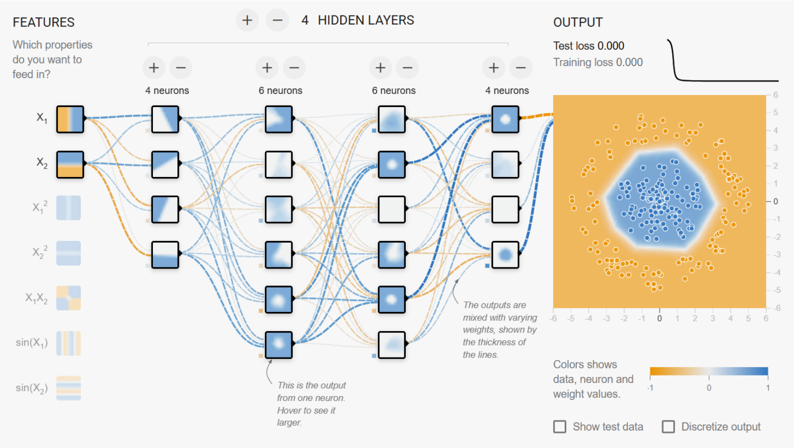
- Popular activations:
  - ReLU
  - tanh
  - sigmoid

$$ReLU(x) = max(0, x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$



Linear layers + ReLU is useful

6 Linear layers are not enough



created with: TensorFlow Playground

Table of Contents

7 Finding the best fit

The loss function

$$\begin{aligned}
\widehat{\mathbf{y}} &= \mathcal{N}\mathcal{N}(\mathbf{X}; \mathbf{W}_1, ..., \mathbf{W}_L, \mathbf{b}_1, ..., \mathbf{b}_L) \\
&= ReLU(\dots ReLU(ReLU(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2) \dots)\mathbf{W}_L + \mathbf{b}_L
\end{aligned}$$

- Parameters: $\theta = \{\mathbf{W}_1, ..., \mathbf{W}_L, \mathbf{b}_1, ..., \mathbf{b}_L\}$
- True values: $\mathbf{y}$; Predicted values: $\widehat{\mathbf{y}}$
- Find θ such that $\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2$ is minimized

Gradient Descent

Algorithm

1. Choose a random value for $\theta = \widehat{\theta}$
2. Calculate $\mathcal{L}(\widehat{\theta})$
3. Nudge $\widehat{\theta}$ a little bit
4. Repeat from 2

Intuition

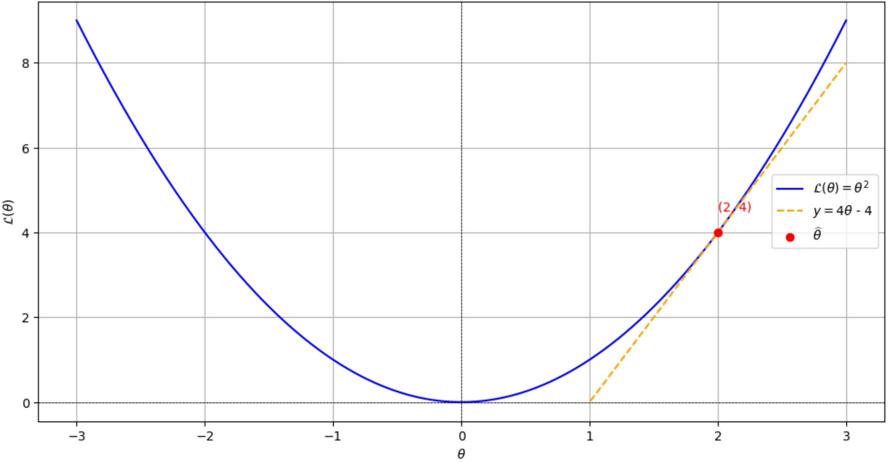
1. You are on a field
2. Estimate how low you are
3. Move in a downward direction
4. Repeat from 2

But which way is "downward"?

- Introducing the derivative - $\frac{d\mathcal{L}}{d\theta}$
  - the rate of change of $\mathcal{L}$ with respect to θ
  - if I change θ a little bit, how much does $\mathcal{L}$ changes?
  - the slope of the tangent of $\mathcal{L}$ in point θ
- Introducing the partial derivatives - $\frac{\partial\mathcal{L}}{\partial\mathbf{W}_1}, \ldots, \frac{\partial\mathcal{L}}{\partial\mathbf{W}_L}, \frac{\partial\mathcal{L}}{\partial\mathbf{b}_1}, \ldots, \frac{\partial\mathcal{L}}{\partial\mathbf{b}_L}$
  - if all other parameters are kept the same, what is the rate of change of $\mathcal{L}$ with respect to a single parameter?

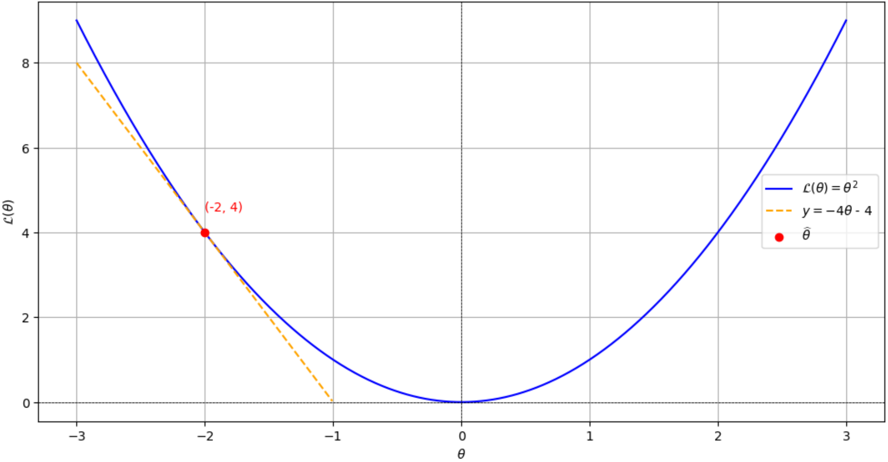
Let's visualize the derivative

7 Finding the best fit



Let's visualize the derivative

7 Finding the best fit



How much do we "nudge" $\widehat{\theta}$?

7 Finding the best fit

- Learning rate - η
  - experimentally determined
  - if η is too large - we skip over the optimum
  - if η is too small - we "fit" too slow

Gradient descent

1. Choose a random value for $\theta = \widehat{\theta} = \{\widehat{\mathbf{W}}_1, \ldots, \widehat{\mathbf{W}}_L, \widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_L\}$
2. Calculate loss $\mathcal{L}(\{\widehat{\mathbf{W}}_1, \ldots, \widehat{\mathbf{W}}_L, \widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_L\})$ on **complete** dataset $\leftarrow$ ***forward*** pass
3. Calculate partial derivatives $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_L}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \mathbf{b}_L}$ $\leftarrow$ ***backward*** pass
4. Update parameters:
$\widehat{\mathbf{W}}_1 \leftarrow \widehat{\mathbf{W}}_1 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \ldots, \widehat{\mathbf{W}}_L \leftarrow \widehat{\mathbf{W}}_L - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_L}, \widehat{\mathbf{b}}_1 \leftarrow \widehat{\mathbf{b}}_1 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}, \ldots, \widehat{\mathbf{b}}_L \leftarrow \widehat{\mathbf{b}}_L - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_L}$
5. Repeat from 2

But nobody really uses gradient descent...

- Gradient descent is slow - it calculates the loss on the complete dataset before doing an update

Stochastic gradient descent

1. Choose a random value for $\theta = \widehat{\theta} = \{\widehat{\mathbf{W}}_1, \ldots, \widehat{\mathbf{W}}_L, \widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_L\}$
2. Choose a random subset of the dataset $\leftarrow$ ***batch***
3. Calculate loss $\mathcal{L}(\{\widehat{\mathbf{W}}_1, \ldots, \widehat{\mathbf{W}}_L, \widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_L\})$ on **batch** $\leftarrow$ ***forward*** pass
4. Calculate partial derivatives $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_L}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \mathbf{b}_L}$ $\leftarrow$ ***backward*** pass
5. Update parameters:
$\widehat{\mathbf{W}}_1 \leftarrow \widehat{\mathbf{W}}_1 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \ldots, \widehat{\mathbf{W}}_L \leftarrow \widehat{\mathbf{W}}_L - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_L}, \widehat{\mathbf{b}}_1 \leftarrow \widehat{\mathbf{b}}_1 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}, \ldots, \widehat{\mathbf{b}}_L \leftarrow \widehat{\mathbf{b}}_L - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_L}$
6. Repeat from 2

Table of Contents

8 The computational graph

Backpropagation in a nutshell

- Backpropagation is an efficient way to do a ***backward*** pass
- Backpropagation = computational graph + ***the chain rule***

Transposed vector

$$\mathbf{err} = \mathbf{y} - \widehat{\mathbf{y}} = \begin{pmatrix} y_1 - \widehat{y}_1 \\ y_2 - \widehat{y}_2 \\ \vdots \\ y_N - \widehat{y}_N \end{pmatrix}$$

- Transposing converts a column vector into a row vector and vice versa
- Transposing "rotates"/switches indices in matrix $\mathbf{A}$ - $a_i^{(j)} \leftarrow a_j^{(i)}$

$$\mathbf{err}^T = \begin{pmatrix} y_1 - \widehat{y}_1 & y_2 - \widehat{y}_2 & \dots & y_N - \widehat{y}_N \end{pmatrix}$$

Vectorized loss function

$$\mathbf{err}^T\mathbf{err} = \begin{pmatrix} y_1 - \widehat{y}_1 & y_2 - \widehat{y}_2 & \dots & y_N - \widehat{y}_N \end{pmatrix} \begin{pmatrix} y_1 - \widehat{y}_1 \\ y_2 - \widehat{y}_2 \\ \vdots \\ y_N - \widehat{y}_N \end{pmatrix} = \sum_{i=1}^{N} (y_i - \widehat{y}_i)^2$$

$$\mathcal{L}(\theta) = \frac{\mathbf{err}^T\mathbf{err}}{N} = \frac{(\mathbf{y} - \widehat{\mathbf{y}})^T(\mathbf{y} - \widehat{\mathbf{y}})}{N}$$

Let's write operations as functions

- Let's use a 2 layer NN as an example

$$\widehat{\mathbf{y}} = \mathcal{NN}(\mathbf{X}; \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = ReLU(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

- Introduce functions $Add(x, y), Sub(x, y), Mul(x, y), ReLU(x), T(x)$

$$\widehat{\mathbf{y}} = Add(Mul(ReLU(Add(Mul(\mathbf{X}, \mathbf{W}_1), \mathbf{b}_1))\mathbf{W}_2), \mathbf{b}_2)$$

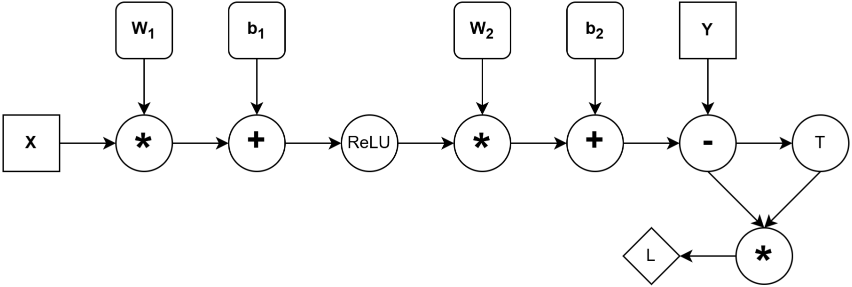
$$\mathcal{L}(\theta) = \frac{Mul(T(Sub(\mathbf{y}, \widehat{\mathbf{y}})), Sub(\mathbf{y}, \widehat{\mathbf{y}}))}{N}$$

So what is a computational graph?

- Each node represents a function call
- Each directed edge connects an output of a function/variable to an input of a different function/result

Let's visualize this

8 The computational graph



So what is the chain rule?

$$u = g(x) \qquad y = f(u) = f(g(x))$$

- Used for compositions of functions
- If we change $u \to \Delta u$, y changes $\Delta y \approx \frac{dy}{du}\Delta u$
- If we change $x \to \Delta x$, then u changes $\Delta u \approx \frac{du}{dx}\Delta x$

$$\Delta y \approx \frac{dy}{du}\Delta u \approx \frac{dy}{du}\frac{du}{dx} \times \Delta x$$

$$\Delta x \to 0 \implies \frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$$

$$\mathcal{L} = Mul(u, \dots) \implies \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u}$$

$$u = T(h) \implies \frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial h}$$

$$h = Sub(\dots, \widehat{\mathbf{y}}) \implies \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{y}}} = \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial \widehat{\mathbf{y}}}$$

- And so on...

But that is not the complete chain rule...

What if a node is an input to multiple nodes?

$$x \rightarrow \text{scalar}$$

$$u^{(1)} = g_1(x) \qquad u^{(2)} = g_2(x) \qquad \dots \qquad u^{(n)} = g_n(x)$$

$$u^{(i)} = g_i(x) \qquad \mathbf{u} = \begin{pmatrix} u^{(1)} & \dots & u^{(n)} \end{pmatrix}$$

$$y = f(g_1(x), \dots, g_n(x)) = f(\mathbf{u}) \rightarrow \text{not elem-wise!}$$

$$y \rightarrow \text{scalar}$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial x}$$

What are the shapes of the derivatives?

- y and x are scalars $\implies \frac{\partial y}{\partial x}$ is scalar
- $\mathbf{u}$ is a column vector $\implies \frac{\partial y}{\partial \mathbf{u}}$ is a row vector - $\begin{pmatrix} \frac{\partial y}{\partial u_1} & \dots & \frac{\partial y}{\partial u_n} \end{pmatrix}$
- $\mathbf{u}$ is a column vector $\implies \frac{\partial \mathbf{u}}{\partial x}$ is a column vector - $\begin{pmatrix} \frac{\partial u_1}{\partial x} & \dots & \frac{\partial u_n}{\partial x} \end{pmatrix}^T$

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y}{\partial u_1} & \dots & \frac{\partial y}{\partial u_n} \end{pmatrix} \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \vdots \\ \frac{\partial u_n}{\partial x} \end{pmatrix} = \sum_{i=1}^{n} \frac{\partial y}{\partial u_i} \frac{\partial u_i}{\partial x}$$

- **Chain rule: multiply compositions, sum up arguments!**

Chain rule in the computational graph

- Start at the final node - the gradient is 1
- Pass that gradient to the input nodes - ***upstream gradient***
- For each input node:
  - **Chain rule (sum)**: add upstream gradient to ***node gradient***
    - Because the node can be an input to multiple nodes!
  - Calculate the gradient of the output with respect to node - ***local gradient***
  - **Chain rule (multiply)**: new upstream = upstream × local
  - Pass new upstream to the input nodes
  - Repeat recursively

Shapes of the gradients

- ***node gradient*** - shape is equal to the node *output* value shape!
- ***upstream gradient*** - shape is equal to the node *output* value shape!
- ***new upstream gradient*** - shape is equal to the node *input* value shape!
- Always check the gradient shapes!

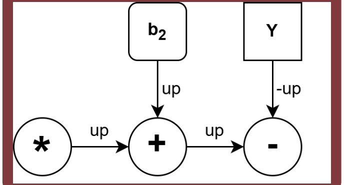
The adding & subtracting derivative
8 The computational graph

- Scalar Add first (same for Sub, with one minus sign):

$$c = Add(a, b)$$
$$\Delta a \to 0 \implies \frac{\partial c}{\partial a} = 1$$
$$\Delta b \to 0 \implies \frac{\partial c}{\partial b} = 1$$

$$e = Sub(c, d) = Add(c, -d)$$
$$\Delta c \to 0 \implies \frac{\partial e}{\partial c} = 1$$
$$\Delta d \to 0 \implies \frac{\partial e}{\partial d} = -1$$

Ins: $\mathbf{y}, Add$
Outs: $Sub(\mathbf{y}, Add)$
Ins: $\mathbf{b}_2, Mul$
Outs: $Add(\mathbf{b}_2, Mul)$



$$\mathbf{C} = Add(\mathbf{A}, \mathbf{B}); \mathbf{E} = Sub(\mathbf{C}, \mathbf{D})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{C}} = \frac{\partial \mathcal{L}}{\partial \mathbf{E}}; \frac{\partial \mathcal{L}}{\partial \mathbf{D}} = -\frac{\partial \mathcal{L}}{\partial \mathbf{E}} \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{C}}; \frac{\partial \mathcal{L}}{\partial \mathbf{B}} = \frac{\partial \mathcal{L}}{\partial \mathbf{C}}$$

The broadcasting issue

- But when we add the biases, the M_i dimensional row vector is broadcasted to a $N \times M_i$ matrix
- The same row vector is added to multiple rows in the $\mathbf{H}_{i-1}\mathbf{W}_i$ product
  - Same vector is an input to multiple "nodes"!
- **Chain rule (sum)**: tweaking the biases thus has N times the effect on the loss function
  - node gradient = sum of upstream gradients ($\times 1$ for the local *Add* gradient)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}_i} = \sum_{j=1}^{N} \mathbf{up}_j$$

Handling multiple outputs

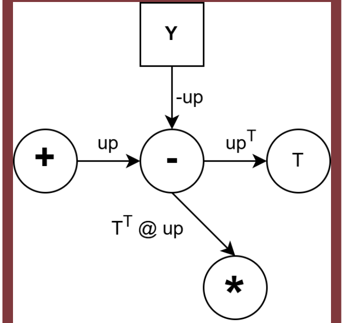
8 The computational graph

- Sub has two identical outputs
  — side effect of graph optimization

$$\mathbf{C} = Sub(\mathbf{A}, \mathbf{B}); \mathbf{D} = Sub(\mathbf{A}, \mathbf{B})$$

- Both $\frac{\partial \mathcal{L}}{\partial \mathbf{C}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{D}}$ are backpropagated
- **Chain rule (sum)**: the node gradient is $\frac{\partial \mathcal{L}}{\partial \mathbf{C}} + \frac{\partial \mathcal{L}}{\partial \mathbf{D}}$
- This is equivalent to the broadcasting issue!

Ins: Y, Add
Outs:
$Sub(Y, Add), Sub(Y, Add)$



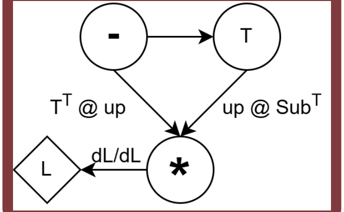
The matrix product derivative

- Scalars first:

$$c = Mul(a, b)$$
$$\Delta c = (a + \Delta a)b - ab = b\Delta a$$
$$\Delta a \to 0 \implies \frac{\partial c}{\partial a} = b$$
$$\Delta b \to 0 \implies \frac{\partial c}{\partial b} = a$$

Ins: T, Sub

Outs: $Mul(T, Sub)$



- For matrices, we could calculate the full Jacobian
  - but that is expensive
  - and not really needed (we've got the chain rule!)

The chain rule trick

8 The computational graph

$\mathcal{L}$ is a scalar; $\quad \mathbf{C}^{[n\times m]} = Mul(\mathbf{A}^{[n\times p]}, \mathbf{B}^{[p\times m]})$; $\quad \mathbf{UP}^{[n\times m]}$

$$\mathbf{c}_i = \begin{pmatrix} a_i^{(1)}b_1^{(1)} & & a_i^{(1)}b_1^{(i)} & & a_i^{(1)}b_1^{(m)} \\ + & & + & & + \\ \vdots & & \vdots & & \vdots \\ + & & + & & + \\ a_i^{(j)}\color{red}{b_j^{(1)}} & \dots & a_i^{(j)}\color{red}{b_j^{(i)}} & \dots & a_i^{(j)}\color{red}{b_j^{(m)}} \\ + & & + & & + \\ \vdots & & \vdots & & \vdots \\ + & & + & & + \\ a_i^{(p)}b_p^{(1)} & \dots & a_i^{(p)}b_p^{(i)} & \dots & a_i^{(p)}b_p^{(m)} \end{pmatrix}$$

Changing $a_i^{(j)}$ affects row i in $\mathbf{C}$

$$c_i^{(j)} = \sum_{k=1}^{p} a_i^{(k)} b_k^{(i)} = \dots + a_i^{(j)} b_j^{(i)} + \dots$$

$\implies b_j^{(i)}$ is the derivative!

The chain rule trick cont'd.

- $a_i^{(j)}$ affect all columns in $\mathbf{c}_i$ - input to multiple "nodes" - **chain rule (sum)**!
- Each column in $\mathbf{c}_i$ has upstream gradient - **chain rule (multiply)**!

$$\frac{\partial \mathcal{L}}{\partial a_i^{(j)}} = \sum_{k=1}^{m} \frac{\partial \mathcal{L}}{\partial c_i^{(k)}} \frac{\partial c_i^{(k)}}{\partial a_i^{(j)}} = \sum_{k=1}^{m} \frac{\partial \mathcal{L}}{\partial c_i^{(k)}} b_j^{(k)} = \frac{\partial \mathcal{L}}{\partial \mathbf{c}_i} \mathbf{b}_j^T$$

$$\implies \frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{C}} \mathbf{B}^T$$

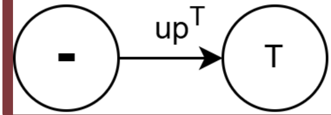
$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}} = \mathbf{A}^T \frac{\partial \mathcal{L}}{\partial \mathbf{C}} \rightarrow \text{excercise for the reader!}$$

The transpose derivative

- Transposing does not "change" the input - it rearranges it
  - $\mathbf{B} = T(\mathbf{A})$
  - $b_i^{(j)} = a_j^{(i)}$
- The derivative shows how the input was rearranged

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = (\frac{\partial \mathcal{L}}{\partial \mathbf{B}})^T$$

Ins: Sub
Outs: $T(Sub)$



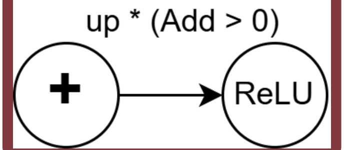
The *ReLU* derivative

$$\mathbf{B} = ReLU(\mathbf{A})$$

- When $\mathbf{A} \leq 0$, the rate of change is 0
  - *ReLU* is flat
- When $\mathbf{A} > 0$, the rate of change is 1
  - *ReLU* is a linear function with slope 1
- $\odot$ is Hadamard product - element-wise product of two matrices

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{B}} \odot \mathbf{R} \text{ where } r_i^{(j)} = \begin{cases} 0 & \text{if } a_i^{(j)} \leq 0 \\ 1 & \text{if } a_i^{(j)} > 0 \end{cases}$$

Ins: *Add*
Outs: $ReLU(Add)$



The complete backprop graph

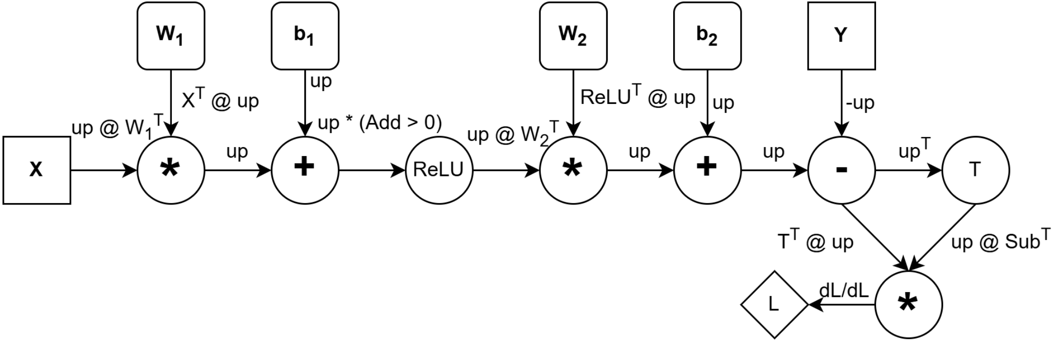


Table of Contents

9 Live coding experience

GitHub repo

https://github.com/jovan-krajevski/backprop

Q&A

Thank you for listening!

Your feedback will be highly appreciated!