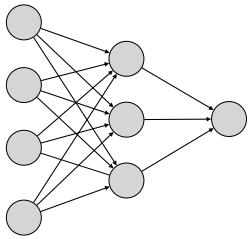
Neural Networks

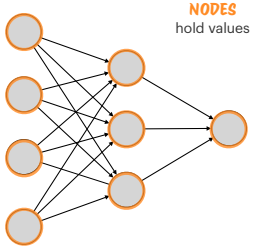
Lecture 11

Termeh Shafie

Terminology

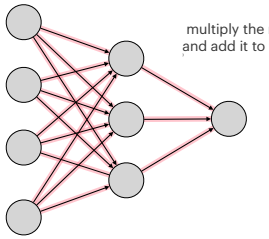


Terminology



NODES
hold values

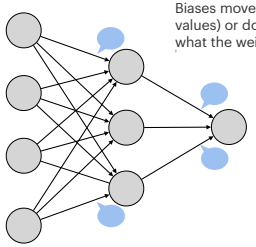
Terminology



WEIGHTS

multiply the number in a previous node and add it to the next node (parameters)

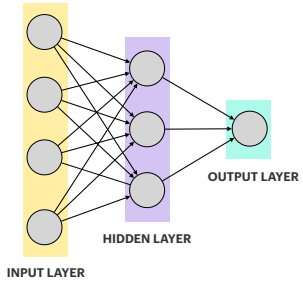
Terminology



BIAS

Biases move the value of a node up (for positive values) or down (for negative values) no matter what the weights and previous nodes' values were

Terminology

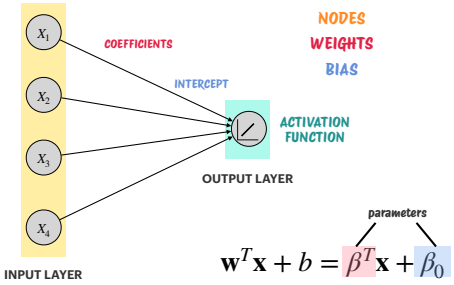


NODES

WEIGHTS

BIAS

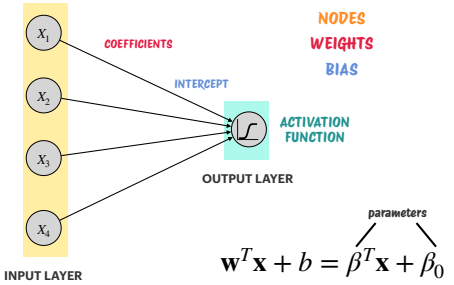
Linear Regression as a Neural Network



parameters

$$\mathbf{w}^T\mathbf{x} + b = \beta^T\mathbf{x} + \beta_0$$

Logistic Regression as a Neural Network



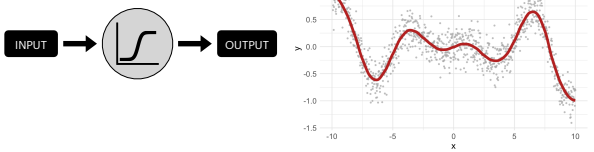
parameters

$$\mathbf{w}^T\mathbf{x} + b = \beta^T\mathbf{x} + \beta_0$$

Activation Function

the purpose of an activation function is to add non-linearity to the neural network.

they determine whether a neuron should be activated based on its input



Without activation functions, the hidden layers would only apply linear transformations (weighted sums), and the entire network is collapsed into a single-layer model because the composition of two linear functions is a linear function itself

Activation Function

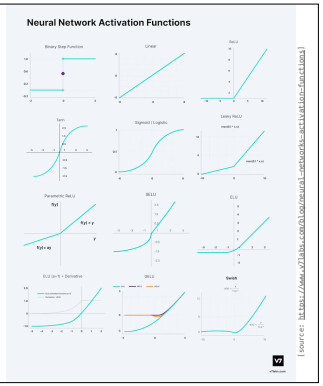
Activation Function	Equation	Range	Use Case
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	(0, 1)	Binary classification, hidden layers in small networks.
Tanh	$f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)	Hidden layers, encourages zero-centered outputs.
ReLU	$f(x) = \max(0, x)$	[0, ∞)	Default for most hidden layers, efficient computation.
Leaky ReLU	$f(x) = \max(0.01x, x)$	(-∞, ∞)	Avoids dying ReLU problem, suitable for hidden layers.
Softmax	$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	(0, 1)	Output layers in multi-class classification problems.
Swish	$f(x) = x \cdot \text{sigmoid}(x)$	(-∞, ∞)	Recent innovation, smooth activation for better gradients.
GELU	$f(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$	(-∞, ∞)	High-performing activation in transformers.

Activation Function

Examples:

- Regression — Linear Activation
- Binary Classification— Sigmoid/Logistic
- Multiclass Classification—Softmax
- Multilabel Classification—Sigmoid

⋮



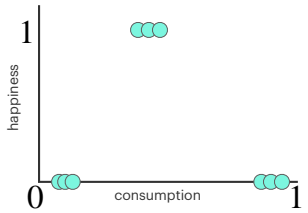
[source: https://www.v7labs.com/blog/neural-networks-activation-functions]

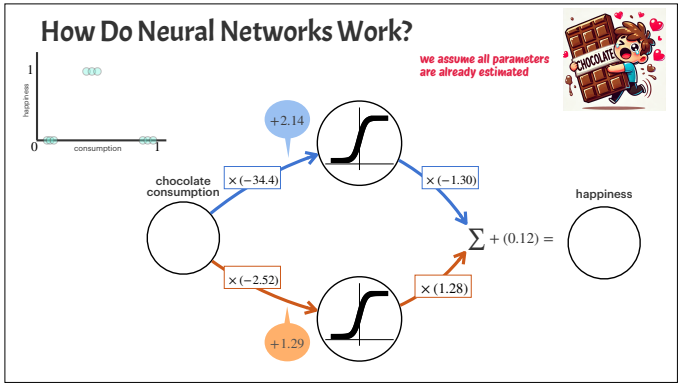
How Do Neural Networks Work?

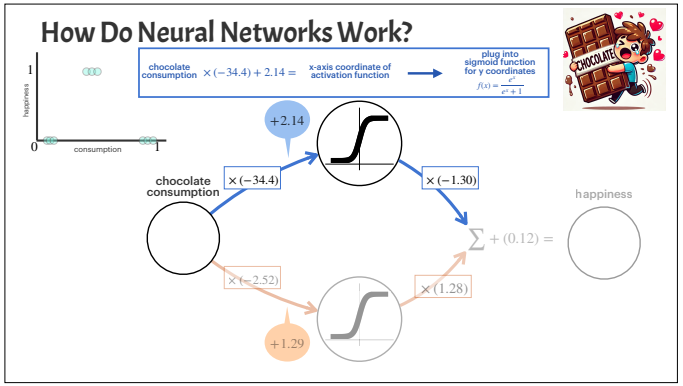
a ***VERY*** simple example:

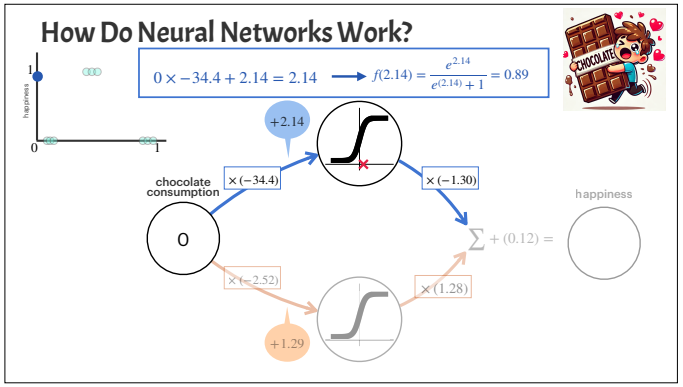
X = normalized chocolate consumption (0—1)

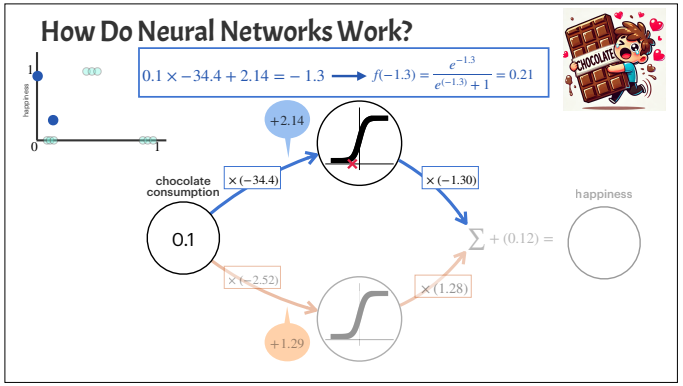
Y = happiness (0/1)

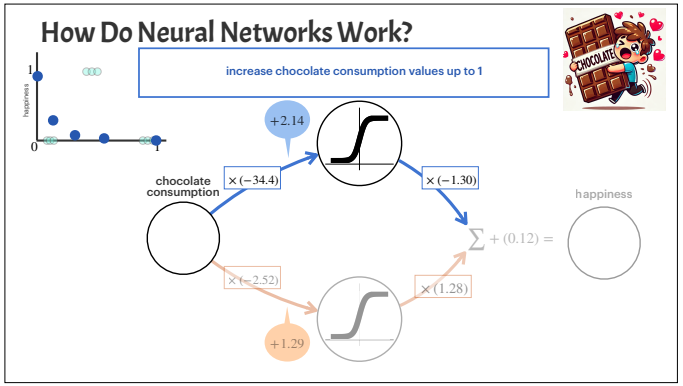


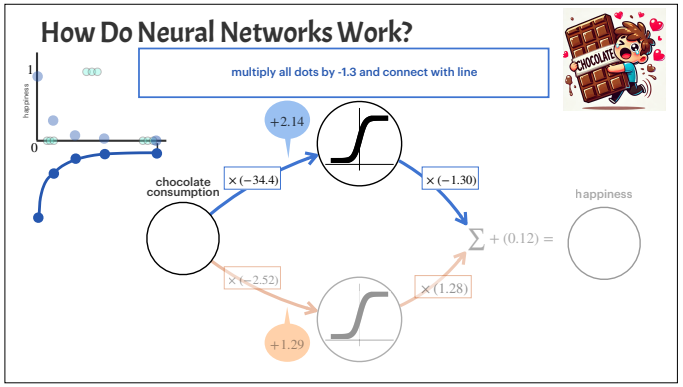

How Do Neural Networks Work?
we assume all parameters are already estimated
happiness
consumption
chocolate consumption
×(−34.4)
+2.14
×(−1.30)
happiness
∑ + (0.12) =
×(−2.52)
+1.29
×(1.28)

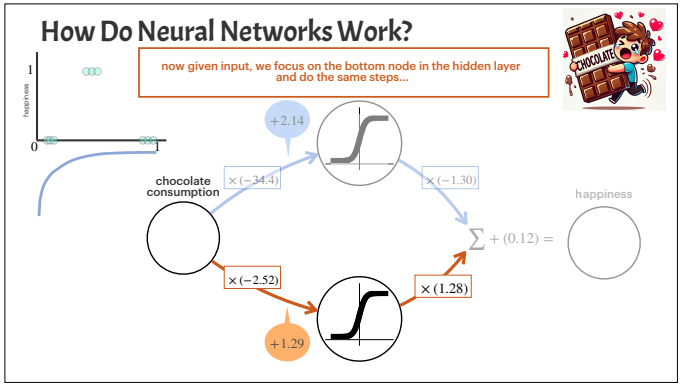

How Do Neural Networks Work?
chocolate consumption ×(−34.4) + 2.14 =
x-axis coordinate of activation function
plug into sigmoid function for y coordinates
happiness
consumption
chocolate consumption
×(−34.4)
+2.14
×(−1.30)
happiness
∑ + (0.12) =
×(−2.52)
+1.29
×(1.28)

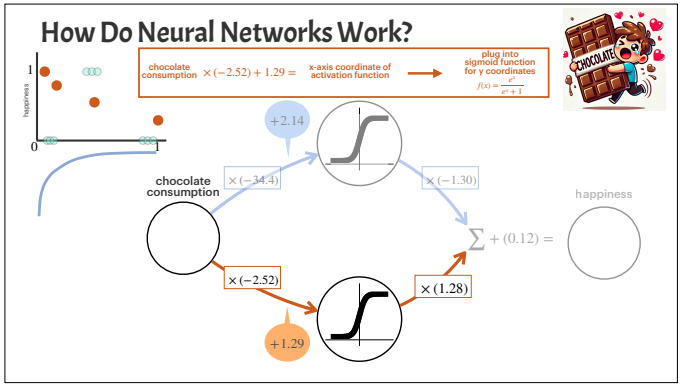

How Do Neural Networks Work?
0 × −34.4 + 2.14 = 2.14
f(2.14) = e^2.14 / (e^(2.14) + 1) = 0.89
happiness
chocolate consumption
0
×(−34.4)
+2.14
×(−1.30)
happiness
∑ + (0.12) =
×(−2.52)
+1.29
×(1.28)

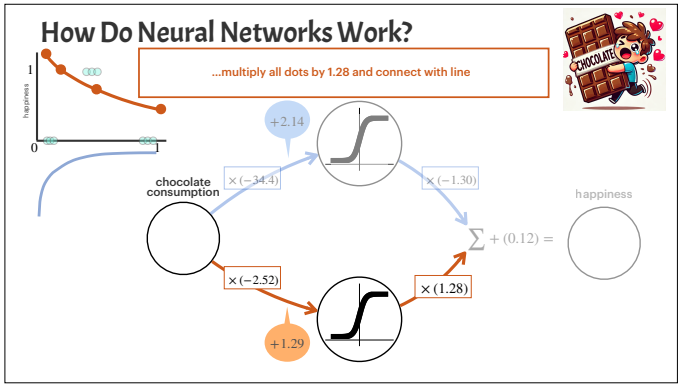

How Do Neural Networks Work?
$0.1 \times -34.4 + 2.14 = -1.3 \rightarrow f(-1.3) = \frac{e^{-1.3}}{e^{(-1.3)}+1} = 0.21$
CHOCOLATE
happiness
chocolate consumption
0.1
+2.14
×(−34.4)
×(−1.30)
∑ + (0.12) =
×(−2.52)
×(1.28)
+1.29


How Do Neural Networks Work?
increase chocolate consumption values up to 1
CHOCOLATE
happiness
chocolate consumption
+2.14
×(−34.4)
×(−1.30)
∑ + (0.12) =
×(−2.52)
×(1.28)
+1.29

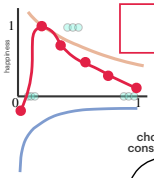

How Do Neural Networks Work?
multiply all dots by -1.3 and connect with line
CHOCOLATE
happiness
chocolate consumption
+2.14
×(−34.4)
×(−1.30)
∑ + (0.12) =
×(−2.52)
×(1.28)
+1.29


How Do Neural Networks Work?
now given input, we focus on the bottom node in the hidden layer and do the same steps...
CHOCOLATE
happiness
1
0
+2.14
chocolate consumption
×(−34.4)
×(−1.30)
happiness
∑ + (0.12) =
×(−2.52)
×(1.28)
+1.29

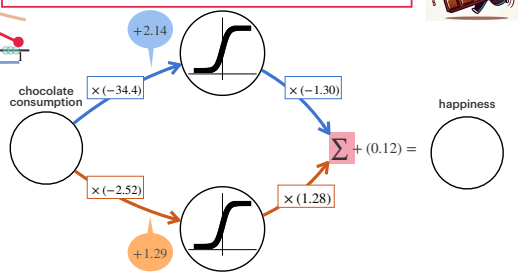

How Do Neural Networks Work?
chocolate consumption ×(−2.52) + 1.29 =
x-axis coordinate of activation function
plug into sigmoid function for y coordinates
$f(x) = \frac{e^x}{e^x + 1}$
CHOCOLATE
happiness
1
0
+2.14
chocolate consumption
×(−34.4)
×(−1.30)
happiness
∑ + (0.12) =
×(−2.52)
×(1.28)
+1.29


How Do Neural Networks Work?
...multiply all dots by 1.28 and connect with line
CHOCOLATE
happiness
1
0
+2.14
chocolate consumption
×(−34.4)
×(−1.30)
happiness
∑ + (0.12) =
×(−2.52)
×(1.28)
+1.29

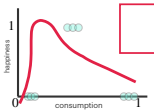
How Do Neural Networks Work?



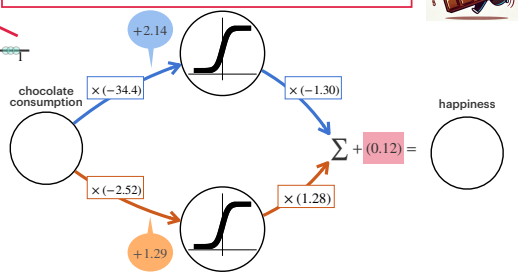
next we sum the Y coordinates the blue and orange line...



How Do Neural Networks Work?

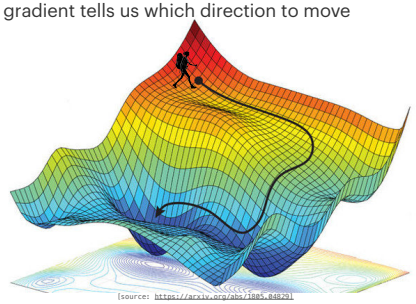


and then add 0.12 to get the final curve



Estimation: Gradient Descent

- The goal is to minimize the loss function
- The gradient tells us which direction to move



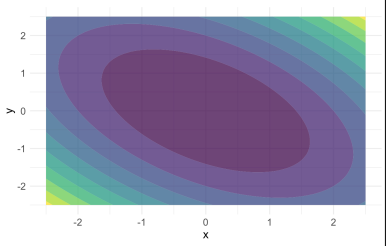
Gradient Descent

$f(x, y) = x^2 + xy + y^2$

Partial Derivatives:

$$\frac{\partial f}{\partial x} = 2x + y \quad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial x} \end{bmatrix} = \begin{bmatrix} 2x + 1 \\ x + 2y \end{bmatrix}$$



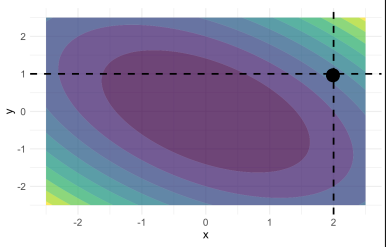
Gradient Descent

$f(x, y) = x^2 + xy + y^2$

Partial Derivatives:

$$\frac{\partial f}{\partial x} = 2x + y \quad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial x} \end{bmatrix} = \begin{bmatrix} 2x + 1 \\ x + 2y \end{bmatrix}$$



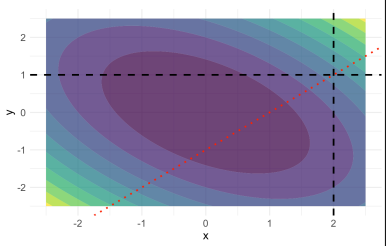
Gradient Descent

$f(x, y) = x^2 + xy + y^2$

Partial Derivatives:

$$\frac{\partial f}{\partial x} = 2x + y \quad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial x} \end{bmatrix} = \begin{bmatrix} 2x + 1 \\ x + 2y \end{bmatrix}$$

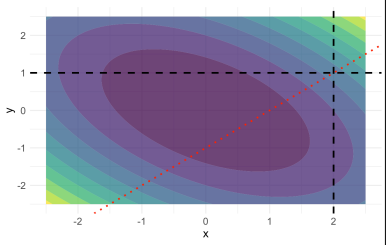


Gradient Descent

$f(x, y) = x^2 + xy + y^2$

Step size determined by Learning Rate ρ:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \rho \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

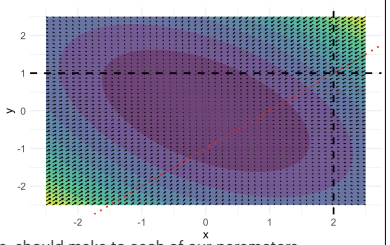


Gradient Descent

$f(x, y) = x^2 + xy + y^2$

Step size determined by Learning Rate ρ:

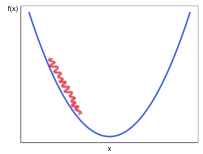
$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \rho \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

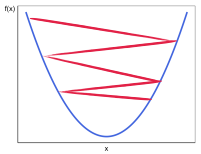


gradient tells us what adjustments we should make to each of our parameters

Estimation: Gradient Descent

- The goal is to minimize the loss function
- The gradient tells us which direction to move
- The **Learning Rate** controls how big of a step we take
  - Small steps mean slower convergence
  - Large steps mean we might step over minima





Example: Very Simple Linear Regression

$\hat{y} = b_0 + b_1 x$

Loss function:

$$RSS = \sum_i^N (\text{actual - predicted})^2 = \sum_i^N (y_i - \hat{y}_i)^2$$
$$= \sum_i^N (y_i - (b_0 + b_1 x))^2$$
$$= \sum_i^N (y_i - b_0 - b_1 x)^2$$

Assume only 2 data points: $(x_1, y_1) = (1,2), (x_2, y_2) = (2,3)$

Example: Very Simple Linear Regression

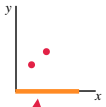
Gradient:

$$\begin{bmatrix} \frac{\partial RSS}{\partial b_0} \\ \frac{\partial RSS}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$

Initialize the gradient algorithm at **(0,0)**

$$\implies \begin{bmatrix} -2\sum_i^N (y_i - (0 + 0 x_i)) \\ -2\sum_i^N x_i (y_i - (0 + 0 x_i)) \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i) \\ -2\sum_i^N x_i (y_i) \end{bmatrix}$$

plug in our actual values



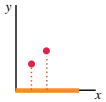
Example: Very Simple Linear Regression

Gradient:

$$\begin{bmatrix} \frac{\partial RSS}{\partial b_0} \\ \frac{\partial RSS}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$

Initialize the gradient algorithm at (0,0)

$$\implies \begin{bmatrix} -2\sum_i^N (y_i) \\ -2\sum_i^N x_i (y_i) \end{bmatrix} = \begin{bmatrix} -2(2+3) \\ -2(1 \cdot 2 + 2 \cdot 3 \end{bmatrix} = \begin{bmatrix} -10 \\ -16 \end{bmatrix}$$

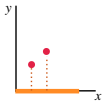


These are the changes we need to make to intercept and slope in order to reduce our loss function.

Example: Very Simple Linear Regression

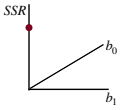
Gradient:

$$\begin{bmatrix} \frac{\partial RSS}{\partial b_0} \\ \frac{\partial RSS}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$



Initialize the gradient algorithm at (0,0)

$$\Longrightarrow \begin{bmatrix} -2\sum_i^N (y_i) \\ -2\sum_i^N x_i (y_i) \end{bmatrix} = \begin{bmatrix} -2(2+3) \\ -2(1\cdot 2 + 2\cdot 3 \end{bmatrix} = \begin{bmatrix} -10 \\ -16 \end{bmatrix}$$



Compute loss function value:

$$SSR = \sum_i^N (y_i - b_0 - b_1 x_i)^2 = (2 - 0 - 0\cdot 1)^2 + (3 - 0 - 0\cdot 3)^2 = 13$$

Example: Very Simple Linear Regression

Gradient:

$$\begin{bmatrix} \frac{\partial RSS}{\partial b_0} \\ \frac{\partial RSS}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$

Apply the changes (learning rate 0.01):

$$\begin{bmatrix} b_{0_{new}} \\ b_{1_{new}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 0.01 \begin{bmatrix} -10 \\ -16 \end{bmatrix}$$

$$\Longrightarrow \begin{bmatrix} b_{0_{new}} \\ b_{1_{new}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.01 \begin{bmatrix} 10 \\ 16 \end{bmatrix} = \begin{bmatrix} 0.10 \\ 0.16 \end{bmatrix}$$

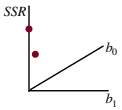
Example: Very Simple Linear Regression

Gradient:

$$\begin{bmatrix} \frac{\partial RSS}{\partial b_0} \\ \frac{\partial RSS}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$

Apply the changes:

$$\begin{bmatrix} b_{0_{new}} \\ b_{1_{new}} \end{bmatrix} = \begin{bmatrix} 0.10 \\ 0.16 \end{bmatrix}$$



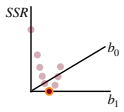
Compute loss function value:

$$SSR = \sum_i^N (y_i - b_0 - b_1 x_i)^2 = (2 - 0.10 - 0.16\cdot 1)^2 + (3 - 0.10 - 0.16\cdot 3)^2 = 8.88$$

Example: Very Simple Linear Regression

Gradient:

$$\begin{bmatrix} \frac{\partial RSS}{\partial b_0} \\ \frac{\partial RSS}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$

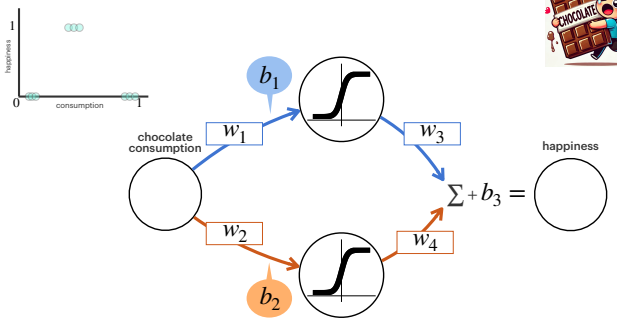


Repeat until RSS is doesn't reduce significantly anymore

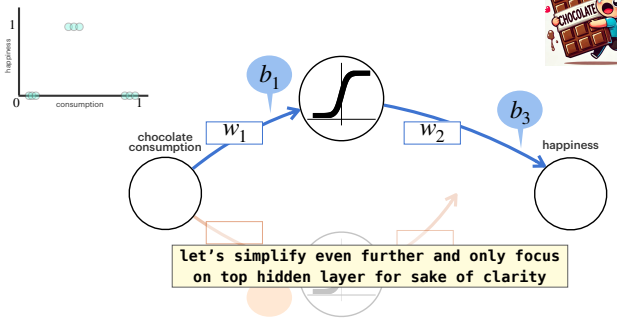
in this toy example, it happens at

$b_0 = 1,\ b_1 = 1 \implies RSS = 0$

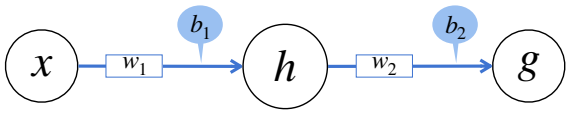
Back to Our Neural Network



Back to Our Neural Network



Backpropagation



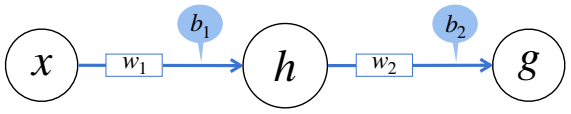
Chain rule:

If we want to know how changing x affects $f(g(x))$ we first need to think about how changing x affects $g(x)$ and then how changing $g(x)$ affects $f(g(x))$:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

the derivative of the outer function f evaluated at $g(x)$ multiplied by the derivative of the inner function $g(x)$

Backpropagation



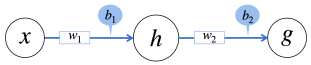
$$h = w_1 \cdot x + b_1$$
$$g = w_2 \cdot x + b_2$$

how close is g to our actual value?

Loss function (MSE):

$$\frac{1}{N}\sum_i^N (y_i - g_i)^2 \implies \frac{1}{N}\sum_i^N (y_i - (w_2 \cdot \underbrace{(w_1 \cdot x_i + b_1)}_{=h} + b_2)^2$$

Backpropagation

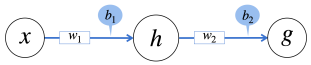


$$\frac{1}{N}\sum_i^N (y_i - g_i)^2 \implies \frac{1}{N}\sum_i^N (\underbrace{y_i}_{\text{actual}} - \underbrace{(w_2 \cdot (w_1 \cdot x_i + b_1) + b_2)^2}_{\text{predicted}}$$

this is what the gradient tells us!

- How change w_1 to reduce our loss?
- How change w_2 to reduce our loss?
- How change b_1 to reduce our loss?
- How change b_2 to reduce our loss?

Backpropagation

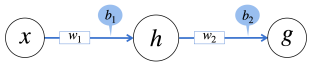


$$\frac{1}{N}\sum_i^N (y_i - g_i)^2 \implies \frac{1}{N}\sum_i^N (\underbrace{y_i}_{\text{actual}} - \underbrace{(w_2 \cdot (w_1 \cdot x_i + b_1) + b_2)}_{\text{predicted}})^2$$

$$\frac{\partial \text{Loss}}{\partial w_1} = \frac{\text{Loss}}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial w_1}$$

changing w_1 changes h, and changing h will change g, and changing g will change overall loss
$\implies$ we need the chain rule!

Backpropagation



$$\frac{1}{N}\sum_i^N (y_i - g_i)^2 \implies \frac{1}{N}\sum_i^N (\underbrace{y_i}_{\text{actual}} - \underbrace{(w_2 \cdot (w_1 \cdot x_i + b_1) + b_2)}_{\text{predicted}})^2$$

$$\frac{\partial \text{Loss}}{\partial w_1} = \underbrace{\frac{\text{Loss}}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial w_1}}_{-2(y_i - g_i)\ w_2\ x}$$

sum over all observations

this is the first part of our gradient

Backpropagation

$$\frac{1}{N}\sum_i^N (y_i - g_i)^2 \implies \frac{1}{N}\sum_i^N (\underbrace{y_i}_{\text{actual}} - \underbrace{(w_2 \cdot (w_1 \cdot x_i + b_1) + b_2)}_{\text{predicted}})^2$$

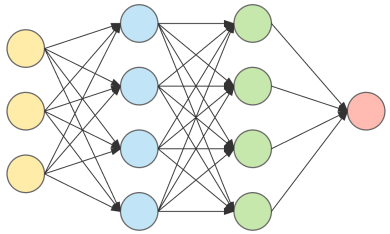
next part of gradient is with respect to our second parameter

$$\frac{\partial \text{Loss}}{\partial b_1} = \underbrace{\frac{\text{Loss}}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial b_1}}_{-2(y_i - g_i)\ w_2\ 1}$$

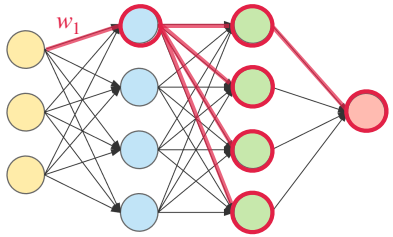
sum over all observations

and so forth for all parameters… this is backpropagation

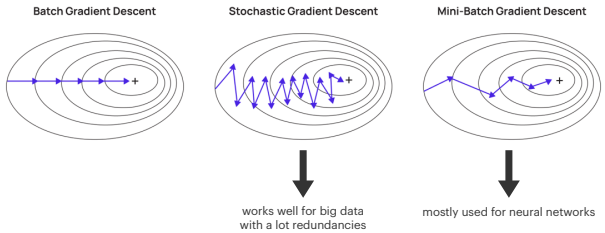
Butterfly Effect



Butterfly Effect



Versions of Gradient Descent



Problems with Gradient Descent

- Choosing a learning rate is hard
- Setting learning rate schedules ahead of time is hard
- Using the same learning rate for all parameters
- Local minima and saddle points
- Variations exist to overcome some of these problems, e.g.
  - **Momentum:**
    - Momentum allows us to "build up speed" as we go downhill
    - It uses a moving average
    - Momentum often helps us converge faster
    - Momentum allows us to escape (some) local minima

Image Classification with Convolutional Networks

neural networks with weight sharing

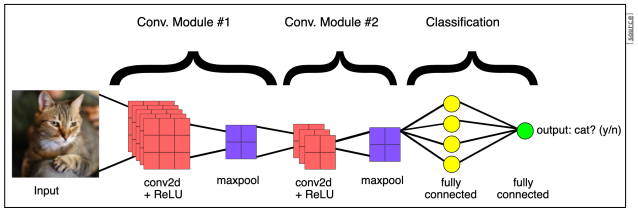


Image Classification with Convolutional Networks

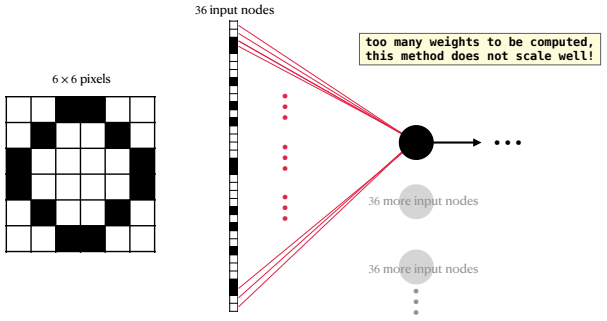
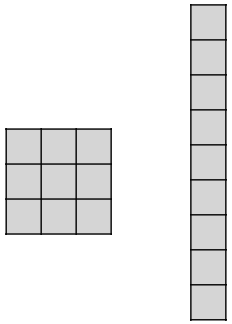


Image Classification with Convolutional Networks

Convolutional networks will

- reduces number of input nodes
- tolerate pixel shifts in image
- take advantage of pixel correlations

Image Classification with Convolutional Networks



we send the original picture through a filter

Image Classification with Convolutional Networks

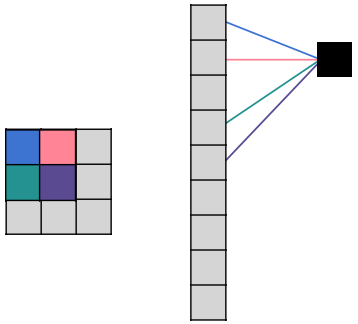


Image Classification with Convolutional Networks

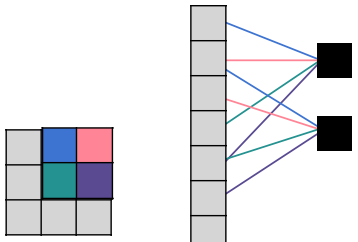


Image Classification with Convolutional Networks

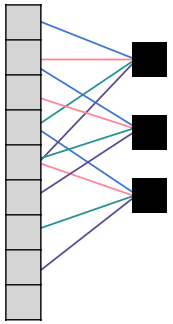
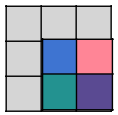


Image Classification with Convolutional Networks



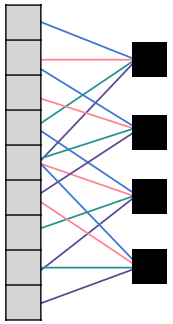
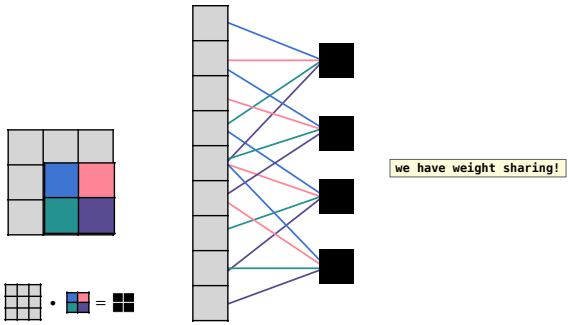
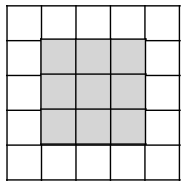


Image Classification with Convolutional Networks



we have weight sharing!

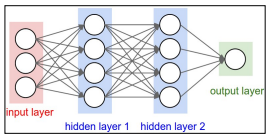
Image Classification with Convolutional Networks



note: the resulting image will be $(n-1)\times(n-1)$
to avoid that and keep original dimensions we can use *padding*

Image Classification with Convolutional Networks

neural networks with weight sharing and ***3D activations***



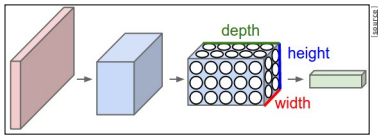


Image Classification with Convolutional Networks

strides

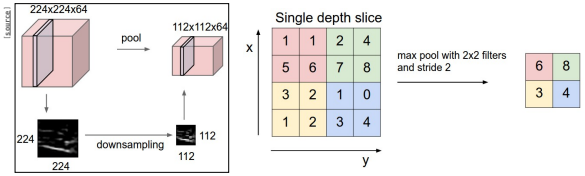
strides determine how much the convolution operation simplifies or compresses the input data

- examples
  - Stride = 1 (default): the kernel moves one pixel at a time horizontally or vertically. High degree of overlap receptive fields but usually retains more spatial details.
  - Stride > 1: The kernel skips pixels as it moves, reducing overlap between receptive fields. This leads to a smaller output size and down-sampling of the input image.

Image Classification with Convolutional Networks

pooling

- Downsample feature maps
- Max pooling works best because when looking for a feature it is better to look at the maximal presence rather than the average presence
- It's best practice to do un-strided convolutions then downsample with maxpooling rather than using strides to downsample



Almost done for today…

Aspect	Forward Pass	Backward Pass
Direction	Input → Hidden Layers → Output	Output → Hidden Layers → Input
Purpose	Calculate output and loss	Update weights and biases to reduce loss
Mathematics	Linear transformations and activations	Gradients and chain rule
Focus	Prediction	Learning (optimization)

what are Epochs?

- refers to one complete pass through the entire training dataset by the model. During this
  1. the model sees every single example in the training dataset once.
  2. It tries to learn by adjusting its weights through forward passes (making predictions) and backward passes (updating weights using the errors).
- more epochs = more practice = better learning
  (up to a point; too many epochs can lead to overfitting)

This Week's Practical

Neural Networks