Neural Networks

Lecturer: Masha Itkina

Announcements

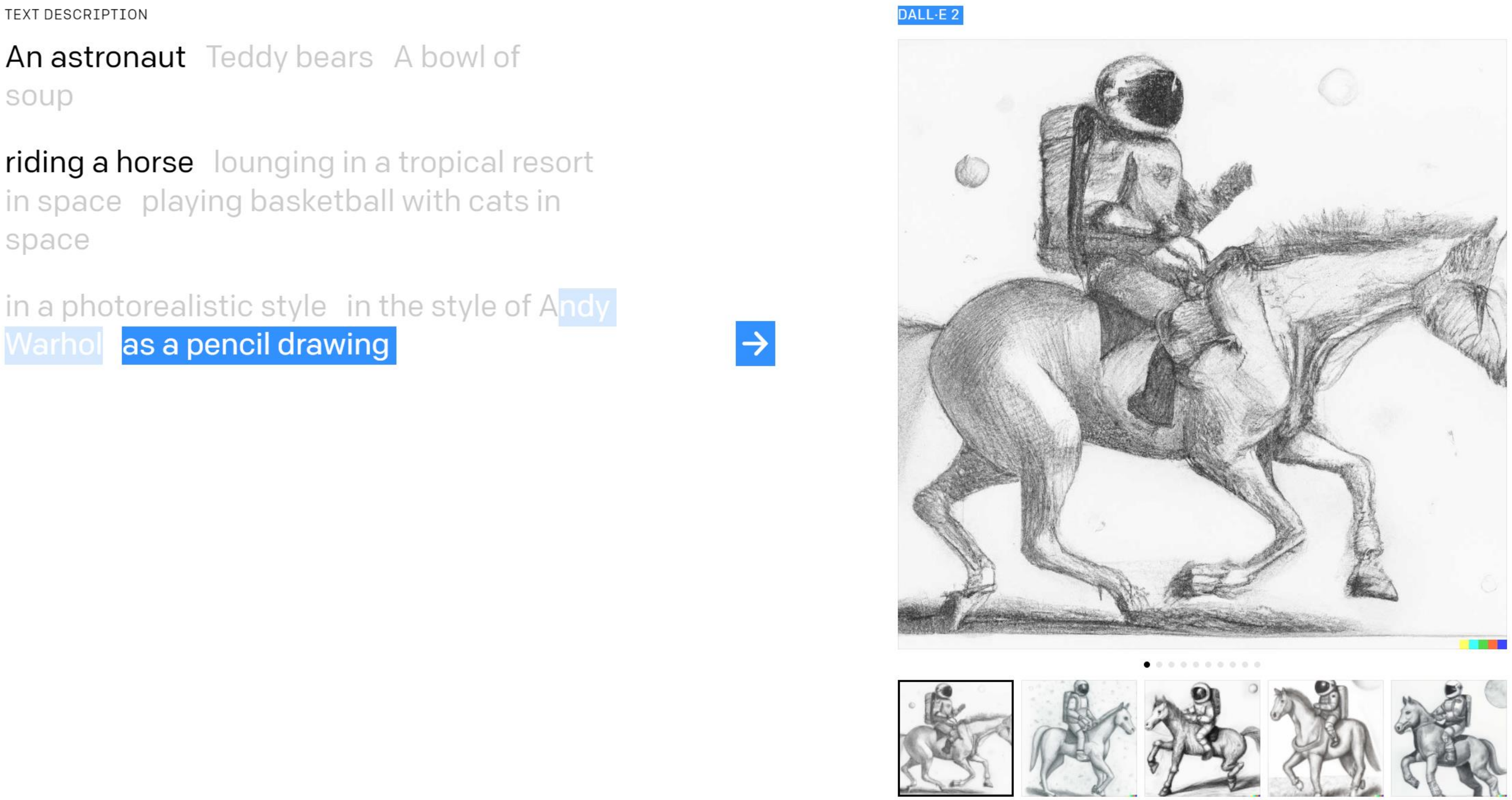
- Problem Set 1 is due today
- Problem Set 2 will be out later tonight; due May 4th
- Feedback on Project Proposals will be released within a week

Neural Networks in the Wild

```
-------- Generated Poem 1 --------

I must have shadows on the way
If I am to walk I must have
Each step taken slowly and alone
To have it ready made

And I must think in lines of grey
To have dim thoughts to be my guide
Must look on blue and green
And never let my eye forget
That color is my friend
And purple must surround me too

The yellow of the sun is no more
Intrusive than the bluish snow
That falls on all of us.  I must have
Grey thoughts and blue thoughts walk with me
If I am to go away at all.
```

GPT-3: Brown et. Al, “Language Models are Few-Shot Learners”, NeurIPS 2020.

Neural Networks in the Wild



DALLE-2: Ramesh et. Al, “Hierarchical Text-Conditional Image Generation with CLIP Latents”, ArXiv 2022.

Agenda for Today

- Supervised learning with non-linear models

- Neural networks

Linear Regression Review

$\{x^{(i)}, y^{(i)}\}_{i=1}^{n}$

$h_\theta(x) = \theta^T x + b$

$$J(\theta) = \sum_{i=1}^{n} (\underbrace{y^{(i)}}_{\text{label}} - \underbrace{h_\theta(x^{(i)})}_{\text{prediction}})^2 = \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)} - b)^2$$

Run GD or SGD to optimize.

Non-Linear Models: Kernels

$\{x^{(i)}, y^{(i)}\}_{i=1}^{n}$ lin. in θ

$h_\theta(x) = \theta^T \phi(x)$ non-linear in x

Non-linear in both θ and x:

$h_\theta(x) = \theta_1^3 x_2 + \sqrt{\theta_5 x_4} + \cdots$

Non-Linear Models

$\{x^{(i)}, y^{(i)}\}_{i=1}^{n}$

Assume $x^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \mathbb{R}$

$h_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$

Cost function for example i:

$$J^{(i)}(\theta) = \left(y^{(i)} - h_\theta(x^{(i)})\right)^2$$

Cost function for dataset:

$$J^{(i)}(\theta) = \frac{1}{n} \sum_{i=1}^{n} J^{(i)}(\theta)$$

↳ constant does not matter! Minimizer will remain the same

Non-Linear Models

We want to optimize: $\min_{\theta} J(\theta)$

Gradient Descent (GD): $\theta := \theta - \alpha \nabla_\theta J(\theta)$ $= \theta - \alpha \nabla\left(\frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta)\right)$

update value of left side w/ right side

$\alpha > 0$

consider all examples at once

Non-Linear Models

We want to optimize: $\min_{\theta} J(\theta)$

Stochastic Gradient Descent (SGD):

alternative SGD:

for $k=1:n_{epoch}$:

shuffle dataset

for $j=1:n_{iter}$:

$\theta = \theta - \alpha \nabla J^{(j)}(\theta)$

no replacement

Algorithm 1 Stochastic Gradient Descent

1: Hyperparameter: learning rate α, number of total iteration n_{iter}.

2: Initialize θ randomly.

3: **for** $i = 1$ to n_{iter} **do**

with replacement

4: Sample j uniformly from $\{1, \ldots, n\}$, and update θ by

$$\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta)$$

Non-Linear Models

We want to optimize: $\min_{\theta} J(\theta)$

Mini-batch SGD: Computing B gradients $\nabla J^{(j_1)}(\theta), \ldots, \nabla J^{(j_B)}(\theta)$ simultaneously is faster than separately.
GPU parallelism.

Algorithm 2 Mini-batch Stochastic Gradient Descent

1: Hyperparameters: learning rate α, batch size B, # iterations n_{iter}.
2: Initialize θ randomly
3: **for** $i = 1$ to n_{iter} **do** $B < n$
4: Sample B examples $j_1, \ldots, j_B$ (without replacement) uniformly from $\{1, \ldots, n\}$, and update θ by

$$\theta := \theta - \frac{\alpha}{B} \sum_{k=1}^{B} \nabla_\theta J^{(j_k)}(\theta)$$

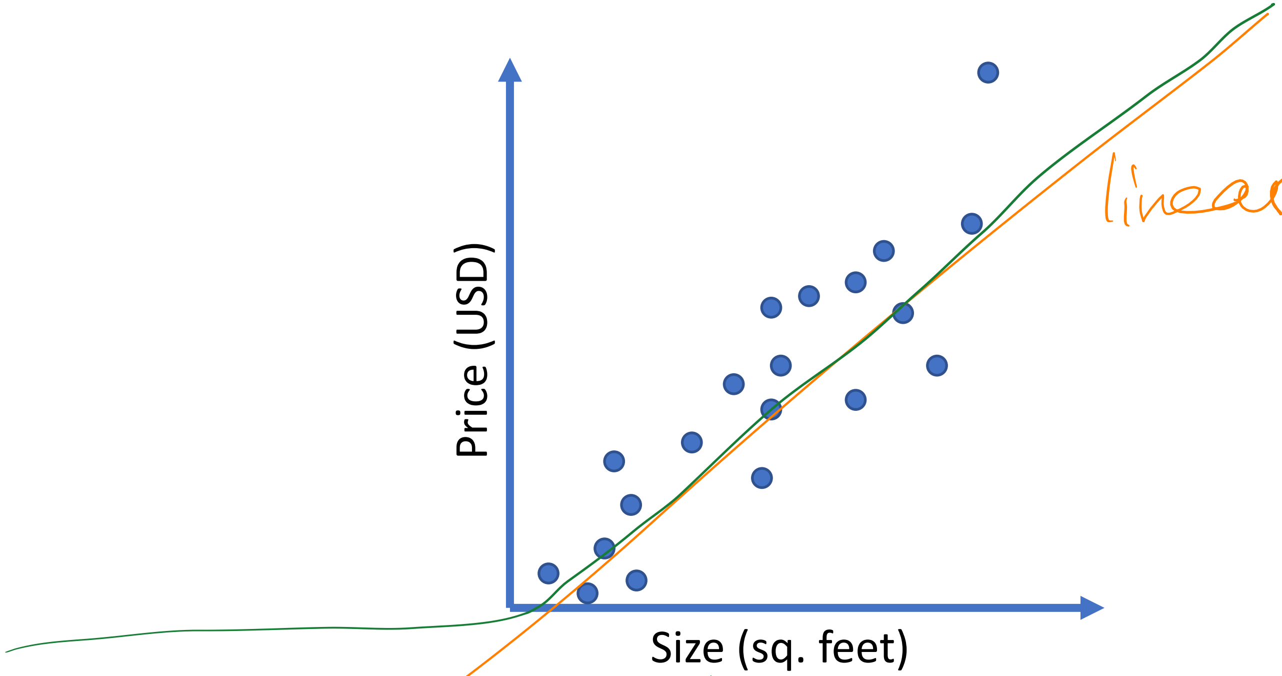
How large is B?
$\downarrow B$: better
$\uparrow B$: faster runtime
Max B that you can store in GPU memory.

Neural Networks

- How to define $h_\theta(x)$? Expressivity, how easy to compute
  - Neural network!

- How to compute $\nabla J^{(j)}(\theta)$?
  - Backpropagation (next lecture)

Housing Price Prediction



linear model

1. gives negative number
2. prediction might have nonlinear relationship with input

$$h_\theta(x) = \max\{wx+b, 0\}$$

$$\theta = (w, b)$$

$$\text{ReLU} = \max\{0, t\}$$

$$h_\theta(x) = \underset{\substack{\uparrow \\ \text{activation function}}}{\text{ReLU}}(wx+b) \leftarrow$$ NN w/ one neuron (one non-linearity)

Housing Price Prediction

High-dimensional input : $x \in \mathbb{R}^d$, $y \in \mathbb{R}$

$$h_\theta(x) = \mathrm{ReLU}(w^T x + b)$$

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d, \quad \underset{\substack{\uparrow \\ \text{weight} \\ \text{vector}}}{w \in \mathbb{R}^d}, \quad \underset{\substack{\uparrow \\ \text{bias}}}{b} \in \mathbb{R}$$

We want to stack neurons! Output of activation $\rightarrow$ input to the next.

Housing Price Prediction

$x \in \mathbb{R}^4$ $\quad x_1, \; x_2, \; x_3, \; x_4$

$x_1 \uparrow$ size, $x_2 \uparrow$ # bedrooms, $x_3 \uparrow$ zip code, $x_4 \uparrow$ wealth

intermediate variables:

max family size : a_1

walkable : a_2

school quality : a_3

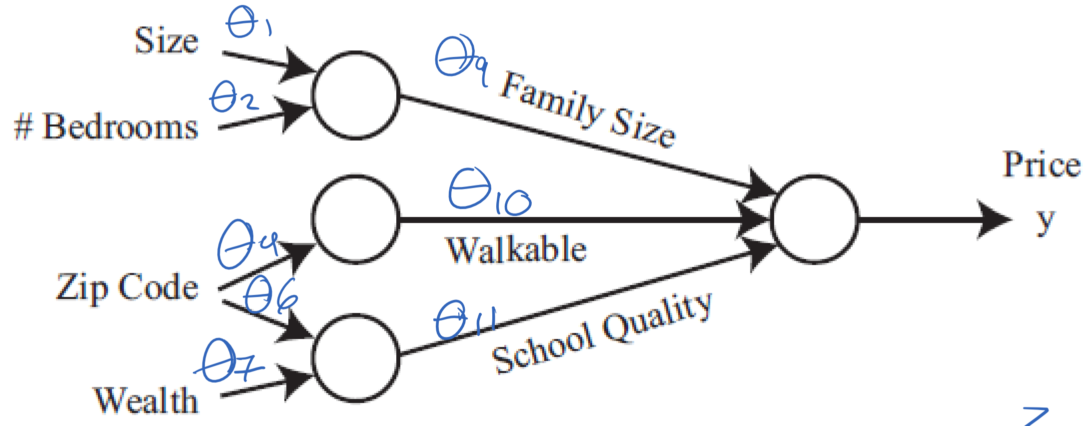
$$a_1 = \mathrm{ReLU}(\theta_1 x_1 + \theta_2 x_2 + \theta_3)$$

$$a_2 = \mathrm{ReLU}(\theta_4 x_3 + \theta_5)$$

$$a_3 = \mathrm{ReLU}(\theta_6 x_3 + \theta_7 x_4 + \theta_8)$$

$$\boxed{h_\theta(x) = \cancel{\mathrm{ReLU}}(\theta_9 a_1 + \theta_{10} a_2 + \theta_{11} a_3 + \theta_{12})}$$

usually have linear layer at the end



prior knowledge

Two-Layer Neural Network

What if we do not have prior knowledge?

- Fully connected neural network
- Intermediate variables -> hidden units

$a_1 = \mathrm{ReLU}(\_\,x_1 + \_\,x_2 + \_\,x_3 + \_\,x_4 + \_)$

$a_2 = \mathrm{ReLU}(\_\,x_1 + \_\,x_2 + \_\,x_3 + \_\,x_4 + \_)$

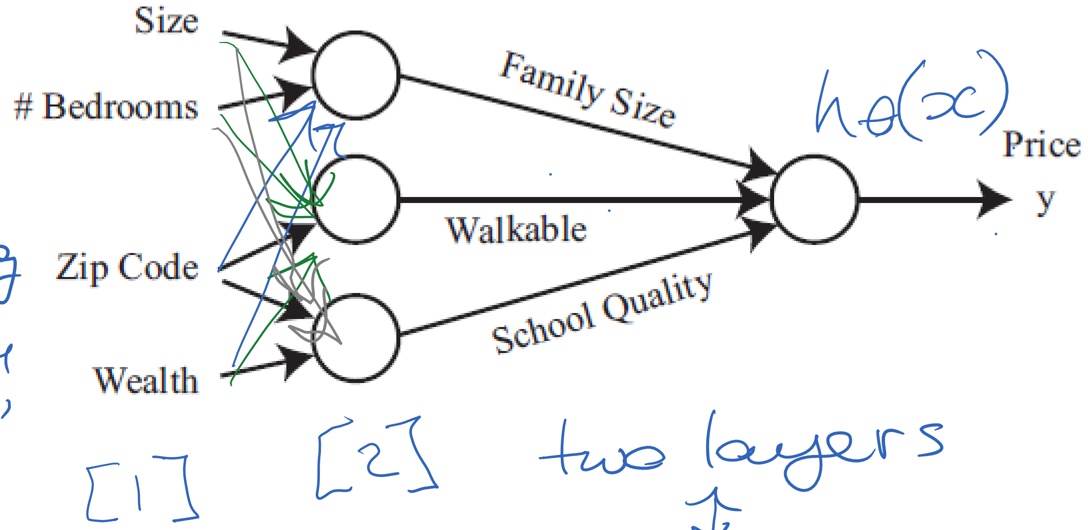
$\vdots$ ↳ more general, depends on everything

$a_1 = \mathrm{ReLU}(w_1^{[1]T} x + b_1^{[1]})$, $w^{[1]} \in \mathbb{R}^4$, $x \in \mathbb{R}^4$, $b^{[1]} \in \mathbb{R}$

$a_2 = \mathrm{ReLU}(w_2^{[1]T} x + b_2^{[1]})$

$a_3 = \mathrm{ReLU}(w_3^{[1]T} x + b_3^{[1]})$

$h_\theta(x) = w^{[2]T} a + b^{[2]}$, $w^{[2]} \in \mathbb{R}^3$, $a \in \mathbb{R}^3$, $b^{[2]} \in \mathbb{R}$



two layers ⇕ one hidden layer

Two-Layer Neural Network

$$\forall j \in [1, ..., m], \quad z_j = {w_j^{[1]}}^\top x + b_j^{[1]} \text{ where } w_j^{[1]} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R}, \; x \in \mathbb{R}^d$$

$$a_j = \mathrm{ReLU}(z_j),$$

$$a = [a_1, \ldots, a_m]^\top \in \mathbb{R}^m \quad \text{m hidden units}$$

$$h_\theta(x) = {w^{[2]}}^\top a + b^{[2]} \text{ where } w^{[2]} \in \mathbb{R}^m, b^{[2]} \in \mathbb{R},$$

Vectorization

$$W^{[1]} = \begin{bmatrix} — & {w_1^{[1]}}^\top & — \\ — & {w_2^{[1]}}^\top & — \\ & \vdots & \\ — & {w_m^{[1]}}^\top & — \end{bmatrix} \in \mathbb{R}^{m \times d}$$

Vectorization

$$\underbrace{\begin{bmatrix} z_1 \\ \vdots \\ \vdots \\ z_m \end{bmatrix}}_{z \in \mathbb{R}^{m\times 1}} = \underbrace{\begin{bmatrix} — & {w_1^{[1]}}^\top & — \\ — & {w_2^{[1]}}^\top & — \\ & \vdots & \\ — & {w_m^{[1]}}^\top & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m\times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x \in \mathbb{R}^{d\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{m\times 1}}$$

$$\boxed{z = W^{[1]}x + b^{[1]}}$$

Vectorization

Pre-activation : $z = W^{[1]}x + b^{[1]} \in \mathbb{R}^m$

$$a = \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \mathrm{ReLU}(z_1) \\ \vdots \\ \mathrm{ReLU}(z_m) \end{bmatrix} \triangleq \mathrm{ReLU}(z)$$

overloaded notation for multi-dim vector inputs

$W^{[2]} = [w^{[2]^T}] \in \mathbb{R}^{1\times m}$, $b^{[2]} \in \mathbb{R}$

$h_\theta(x) = W^{[2]}a + b^{[2]}$

Vectorization helps us parallelize on GPU!

Multi-Layer Fully-Connected Neural Networks

weight matrix

hidden units

$$
\begin{aligned}
a^{[1]} &= \mathrm{ReLU}(W^{[1]}x + b^{[1]}) \\
a^{[2]} &= \mathrm{ReLU}(W^{[2]}a^{[1]} + b^{[2]}) \\
&\cdots \\
a^{[r-1]} &= \mathrm{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]}) \\
h_\theta(x) &= W^{[r]}a^{[r-1]} + b^{[r]}
\end{aligned}
$$

bias

$\dim(a^{[k]}) = m_k$

$W^{[1]} \in \mathbb{R}^{m_1 \times d}$

$W^{[2]} \in \mathbb{R}^{m_2 \times m_1}$

$W^{[k]} \in \mathbb{R}^{m_k \times m_{k-1}}$,

$b^{[k]} \in \mathbb{R}^{m_k}$

Why do we need an activation function (e.g., ReLU)?

$$a^{[1]} = W^{[1]}x + b^{[1]}$$

$$h_\theta(x) = W^{[2]}a + b^{[2]} = W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]}$$

$$= \underbrace{W^{[2]}W^{[1]}}_{\tilde{W}}x + \underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{\tilde{b}}$$

linear in these parameters

Connection to Kernel Methods

Kernel method: $h_\theta(x) = \theta^T \phi(x)$ (linear in parameters, not x)

$$a^{[r-1]} = \phi_\beta(x), \quad \beta = (W^{[1]}, \ldots, W^{[r-1]}, b^{[1]}, \ldots, b^{[r-1]})$$

↳ fix beta

$$h_\theta(x) = W^{[r]} \phi_\beta(x) + b^{[r]}$$

Neural network: $\phi_\beta(x)$ is learned - best that works for data.

$a^{[r-1]} \to$ features/representations

Summary

- Supervised learning with non-linear models
- Neural networks
- Next time: backpropagation