Neural Networks

Lecturer: Masha Itkina

Announcements

- Problem Set 1 is due today

- Problem Set 2 will be out later tonight; due May 4^{th}

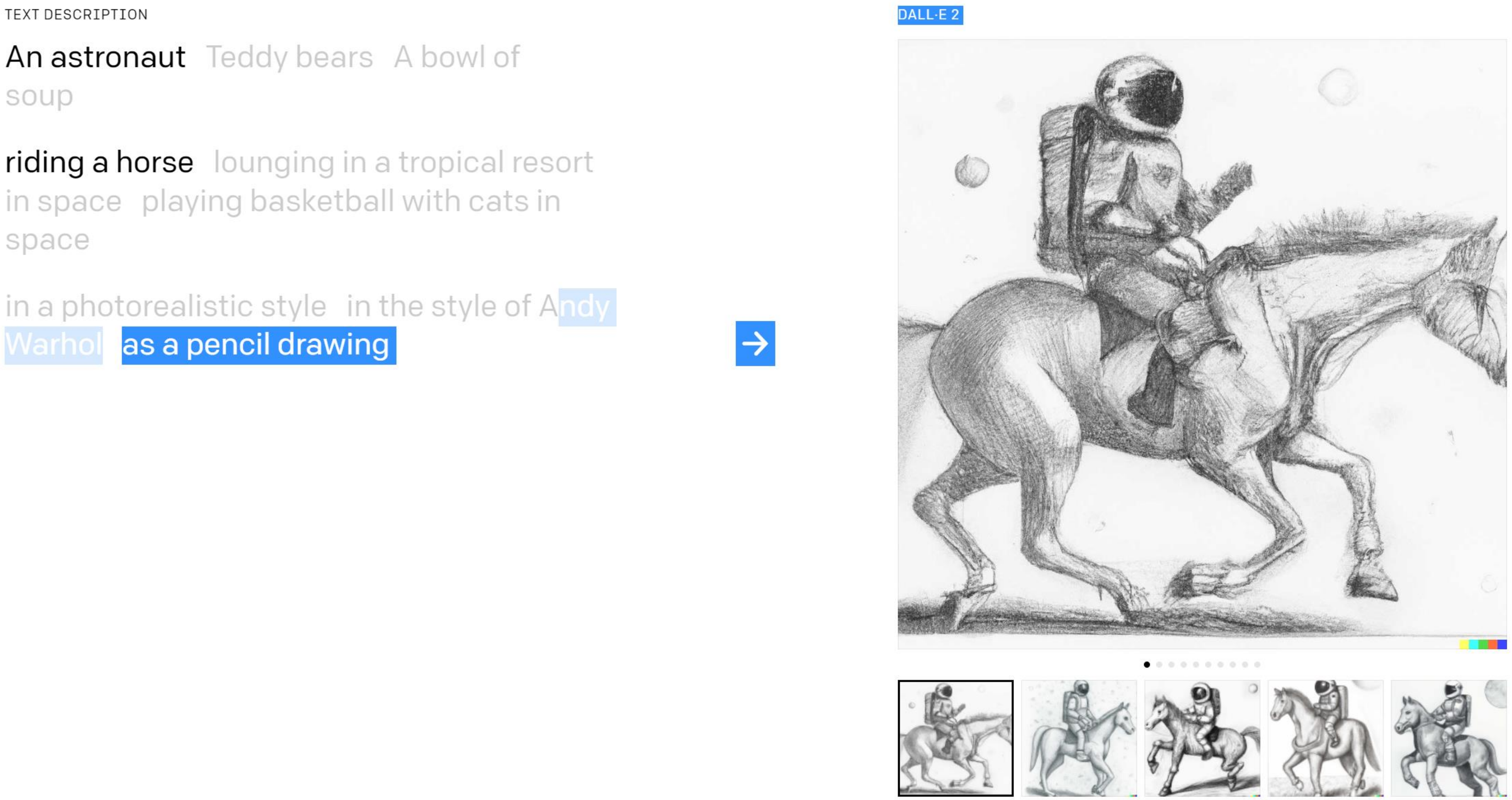
- Feedback on Project Proposals will be released within a week

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

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

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

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{2}\sum_{i=1}^{n}\left(x^{(i)T}\theta + b - y^{(i)}\right)^2$$

prediction label

Run GD or SGD to optimize

Non-Linear Models: Kernels

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

$$h_\theta(x) = \theta^T \phi(x)$$

non-linear in x

lin param.

Non-linear in x and θ:

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

Non-Linear Models

$\{x^{(i)}, y^{(i)}\}$

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

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

Cost function for example i:

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

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J^{(i)}(\theta) \to \text{for entire dataset!}$$

$\hookrightarrow$ const. doesn't matter!

Non-Linear Models

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

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

↳ assigning right side to left side

↑ whole dataset

Non-Linear Models

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

Stochastic Gradient Descent (SGD):

alternative SGD:
for $k = 1 : n_{epoch}$:
shuffle the 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** → w/ replacement
4:	Sample j uniformly from $\{1, \ldots, n\}$, and update θ by $$\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta)$$ $\alpha > 0$

Non-Linear Models

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

Compute B gradients $\nabla J^{(j_1)}(\theta), \ldots, \nabla J^{(j_B)}(\theta)$ simultaneously (faster than iteratively) → GPU parallelism

Mini-batch SGD:

How large is B?

↑B: faster

↓B: better

Pick max B that fits in GPU memory

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**
4: Sample B examples $j_1, \ldots, j_B$ (without replacement) uniformly from $\{1, \ldots, n\}$, and update θ by

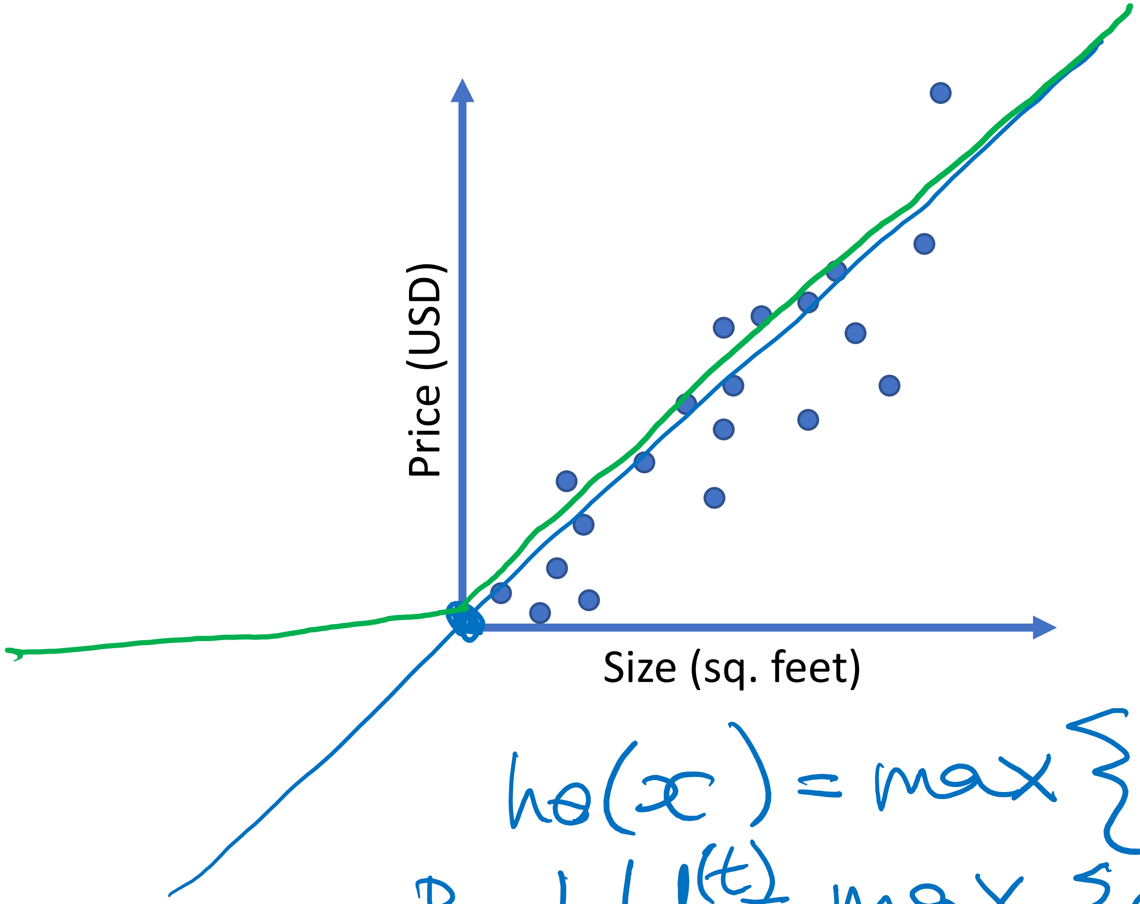
$$\theta := \theta - \frac{\alpha}{B} \sum_{k=1}^{B} \nabla_\theta J^{(j_k)}(\theta)$$

Neural Networks

- How to define $h_\theta(x)$?
  - Neural network!

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

Housing Price Prediction



1 prediction might have a nonlinear relation w/ input

2 can have -ve prices

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

$$\mathrm{ReLU}(t) = \max\{0, t)$$

activation function

$$h_\theta(x) = \mathrm{ReLU}(wx+b)$$

← non-linearity
← one neuron

Housing Price Prediction

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

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

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

We want to stack neurons!

Output of activation $\rightarrow$ input to the next.

Housing Price Prediction

$x \in \mathbb{R}^4$ 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

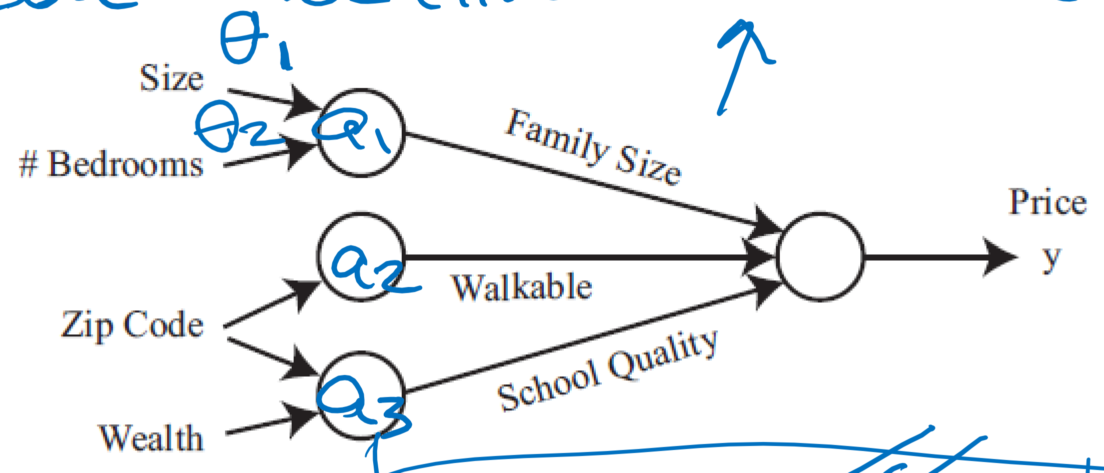
prior knowledge

intermediate variables:

max family size: a_1

walkable: a_2

school quality: a_3



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

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

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

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

Two-Layer Neural Network

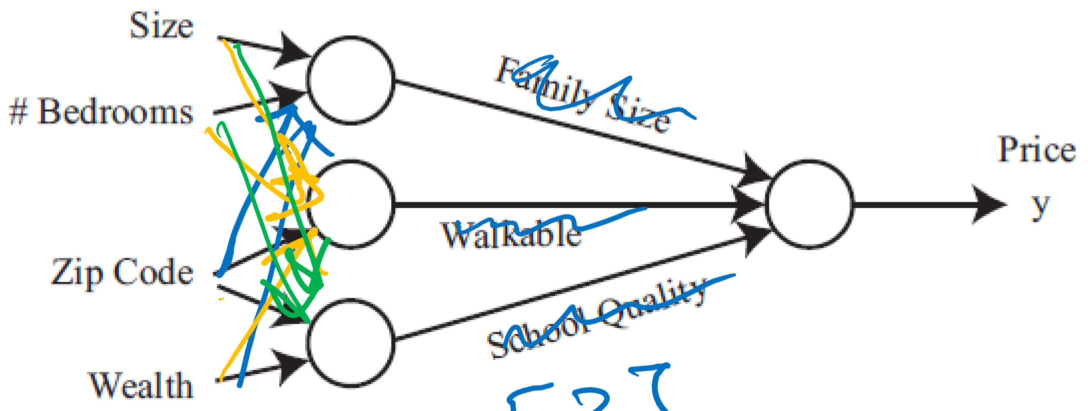
two-layer NN $\updownarrow$ one hidden layer

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 :



$$a_1 = \mathrm{ReLU}(w_1^{[1]T} x + b_1^{[1]})$$
$$a_2 = \mathrm{ReLU}(w_2^{[1]T} x + b_2^{[1]})$$
$$a_3 = \mathrm{ReLU}(w_3^{[1]T} x + b_3^{[1]})$$
$$h_\theta(x) = w^{[2]T} a + b^{[2]},$$

[1]

$w_1^{[1]} \in \mathbb{R}^4,\ x \in \mathbb{R}^4,\ b \in \mathbb{R}$

$w^{[2]} \in \mathbb{R}^3,\ a \in \mathbb{R}^3,\ b^{[2]} \in \mathbb{R}$

Two-Layer Neural Network

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

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

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

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

Vectorization

hidden unit dim.

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

Vectorization

$z_1 = w_1^{[1]^T} x_1 + b_1^{[1]}$

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

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

Vectorization

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

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

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

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

Vectorization helps us parallelize on GPU!

Multi-Layer Fully-Connected Neural Networks

weight matrices

bias

hidden units

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

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

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

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

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

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

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

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

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

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

$\tilde{W}$ —— linear in x —— $\tilde{b}$

Connection to Kernel Methods

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

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

↳ fix β

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

NN: $\phi_\beta(x)$ is learned – best that works for data

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

Summary

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