Gradient Descent and Backpropagation 1

Saeed Saremi

Assigned reading: 6.{2,4}, 7.{1,2}, 5.4.4

September 24, 2024

a summary of the previous lecture

- the geometry of θ in the logistic regression given the following parametrization:[1]

$$p(1|x) = \frac{1}{1+\exp(-\theta^\top x)} =: p(z(x;\theta)), \text{ where } z(x;\theta) = \theta^\top x$$

  θ is perpendicular to the decision boundary and points to class "1".
- softmax function: generalization of the logistic regression to multiclass ($K > 2$) classification via the generative approach, where we assumed $X|k \sim \mathcal{N}(\mu_k, \Sigma)$.
- the negative log-likelihood loss for the logisitic regression and its gradient:

$$\begin{aligned} \mathcal{L}(\theta) &= -y \log p(z(x;\theta)) - (1-y)\log(1-p(z(x;\theta))) \\ \nabla_\theta \mathcal{L}(\theta) &= (p_z - y)\nabla_\theta z = (p_z - y)\, x, \end{aligned}$$

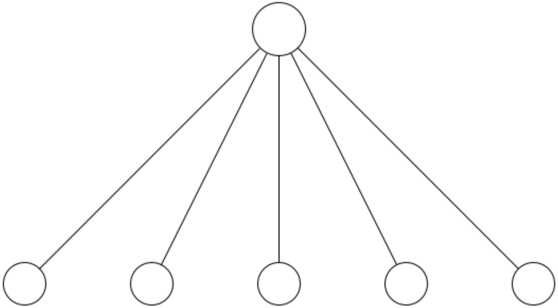
  and we interpreted this gradient geometrically in terms of the parameter updates.
- artificial neural networks as simplified models of biological neural networks

[1] Here, I assume the data is "centered", therefore $\theta_0 = 0$.

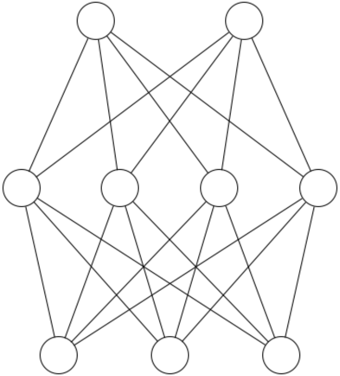
outline

- neural networks as a certain type of nonlinear functions
  - (linear/logisitic regression as single-layer neural network)
  - finish the comparisons with biological neural networks
- training neural networks: stochastic gradient descent
  - gradient of a function
  - the chain rule
  - convex and non-convex functions
  - geometrical interpretation of the gradient
  - stochastic gradient descent (SGD)
- a prelude to backpropagation: the cross-entropy loss for multiclass ($K > 2$) classification and its gradient

single-layer neural networks

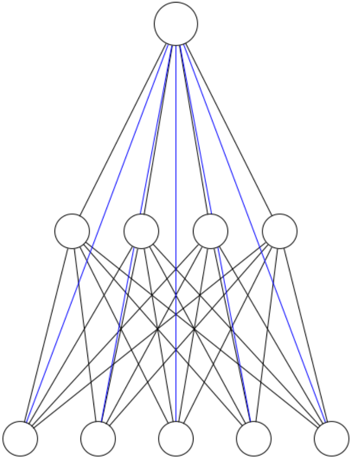


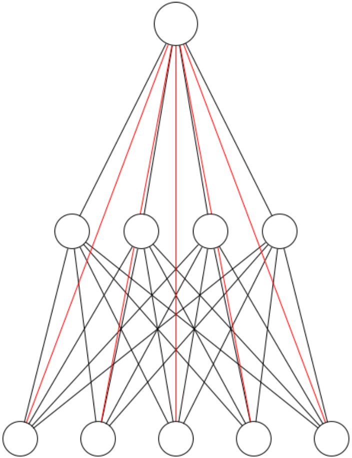
two-layer neural networks



✎ complex neural network architectures:

⊖ anything goes, but “loops” are not allowed!





biological neural networks v.s. deep neural networks

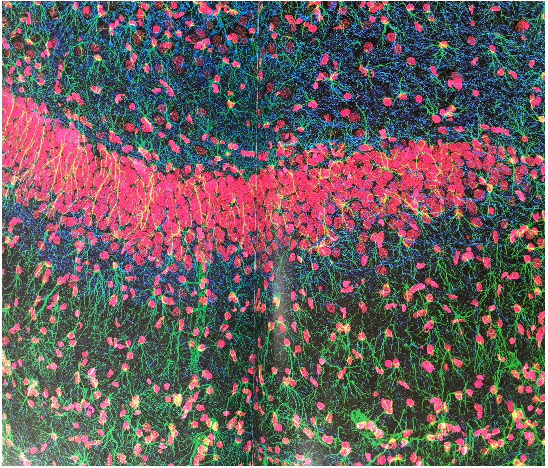


Figure: human brain: 10^{11} neurons, 10^{15} synapses

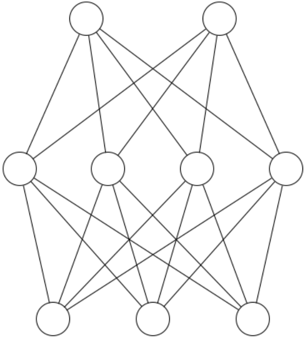
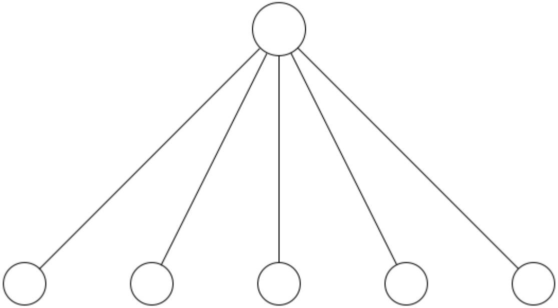


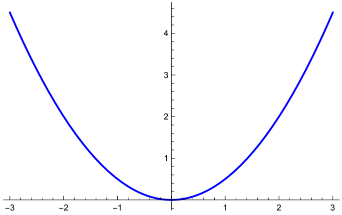
Figure: `GPT-4`: 10^6 neurons, 10^{11} parameters (weights), 100 layers

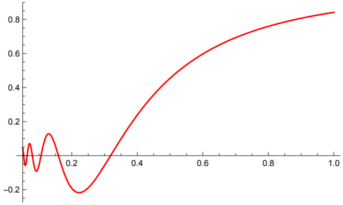
❓ What are we missing in this comparison?

✎ single-layer neural networks: the linear and logistic regression



convex and non-convex functions





Recall from the multivariate calculus that the gradient of the function $f : \mathbb{R}^m \to \mathbb{R}$ is the vector

$$\nabla f(\theta_1, \ldots, \theta_m) = \left(\frac{\partial f}{\partial \theta_1}, \ldots, \frac{\partial f}{\partial \theta_m}\right)^\top$$

- The gradient is the direction that leads to maximal increase of f (steepest ascent). Equivalently, the negative gradient is the direction of steepest descent.
- Formally, a function f is convex if for all θ_1, θ_2 in $\mathbb{R}^m$ and $t \in [0, 1]$:

$$f(t\theta_1 + (1 - t)\theta_2) \leq f(\theta_1) + (1 - t)f(\theta_2)$$

- convex functions have a single (global) minimum.

review: the chain rule from multivariate calculus

Given $u(z_1, z_2) = (z_1 + z_2)^2/2$, where $z_1(x_1, x_2) = x_1 \sin x_2$ and $z_2(x_1, x_2) = (\sin x_2)^2$; determine $\nabla_x u$ using the chain rule:

- brute force:

- chain rule:

the geometrical meaning of the gradient

- Prove that $\nabla f(\theta)$, the gradient of the function $f : \Theta \to \mathbb{R}$ at any point θ, is perpendicular to the level set of f at that point.[2]

(Hint: define a path $t \to \theta$ on the level set and use the fact that by definition of the level set $\partial_t f(\theta(t)) = 0$. Use the chain rule!)

[2]Recall that level sets are defined by $\{\theta' : f(\theta') = f(\theta)\}$ for some constant c.

stochastic gradient descent I

- at a high level the loss is always written as the sum of losses by individual points $i \in [n] := \{1, \ldots, n\}$ in the training set $\mathcal{D} = \{(x_i, y_i)\}_{i \in [n]}$:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \mathcal{L}_i(\theta),$$

  where $\mathcal{L}_i(\theta)$ is short for:

$$\mathcal{L}_i(\theta) := \mathcal{L}(x_i, y_i; \theta).$$

- This is very general, but it's very easy to see where it's coming from in the maximum log-likelihood framework. We always assume the i.i.d. setting:

$$(x_i, y_i) \overset{\text{iid}}{\sim} p_\theta(x, y), \; i \in [n].$$

  Therefore,

$$p(\mathcal{D}|\theta) = \prod_{i=1}^{n} p_\theta(x_i, y_i).$$

  It follows:

$$\mathcal{L}_i(\theta) = -\log p_\theta(x_i, y_i).$$

stochastic gradient descent II

The two prominent examples so far include:

- linear regression:

$$-\log p_\theta(x_i, y_i) = \frac{1}{2\sigma^2}(y_i - \theta^\top x_i^{(0)}) + const$$

- logistic "regression":

$$-\log p_\theta(x_i, y_i) = -y_i \log g(\theta^\top x_i^{(0)}) - (1 - y_i)\log(1 - g(\theta^\top x_i^{(0)})) + const$$

- (Note that the structure of the loss is general: we are getting ready to replace $\theta^\top x_i^{(0)}$ with an L-layer neural network: $f(x_i^{(0)}; \theta^{(1)}, \ldots, \theta^{(L)})$.)

Stochastic Gradient Descent (SGD) in its pure form is defined by the following updates:

$$\theta_{t+1} = \theta_t - \epsilon_t \nabla_\theta \mathcal{L}(x_i, y_i; \theta),$$

where $i \in [n]$ is selected at random (typically without replacement) at each iteration t.

stochastic gradient descent III

- Intuitively (people have tried to study this) in high dimensions the loss landscape is "dominated" by saddle points and the noise in SGD helps to avoid them.
- SGD is convenient in the regime $n \ggg 1$.
- One pass through the data is called an epoch.
- The problem is towards the end of the training (optimization) the noise in SGD will slow down the training.
- In short: noise helps us at the beginning of training, it "hurts" us towards the end.
- Of course, one can find a compromise by dividing the dataset into (random) mini-batches of size b: in this scheme one epoch involves $\lfloor n/b \rfloor$ updates.

? Based on this picture, can you suggest a batching scheme for effective training? [3]

[3] Coming up with a mini-batch / learning rate schedule remains an art and problem dependent.

cross-entropy loss for multiclass classification and its gradient