Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2024

Prof. R. Wattenhofer
Saku Peltonen

Computational Thinking
Exercise 11

1 Limitations of Neural Networks

Which of the following functions can theoretically be approximated arbitrarily well by a sufficiently large neural network?

a) $f(x) = x^2$ for $x \in [0, 1]$

b) $f(x) = |x|$ for $x \in [-1, 1]$

c) $x \in [0, 100]$ and $f(x) = \begin{cases} 1 \text{ for } x \in \mathbb{N} \\ 0 \text{ else} \end{cases}$

d) $x \in [-10, 10]$ and $f(x) = \begin{cases} 3x^4 + 5x & \text{for } x > 0 \\ -3x^3 + 7x^2 & \text{else} \end{cases}$

e) $x \in [-10, 10]$ and $f(x) = \begin{cases} 4x^3 + 7x + 2 & \text{for } x > 0 \\ -3x^3 + 8x & \text{else} \end{cases}$

2 An Ill-Designed Network

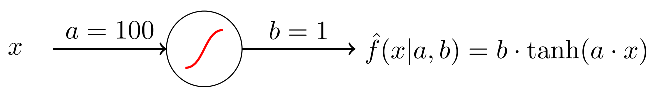

Figure 1: A simple neural network

Figure 1 shows a simple neural network with a single hidden node that applies the hyperbolic tangent non-linearity $\tanh(ax) = \frac{\exp(ax) - \exp(-ax)}{\exp(ax) + \exp(-ax)}$. You want to train the network with stochastic gradient descent to approximate the identity function $f(x) = x$ for inputs $x \in [-1, 1]$.

a) Given the weights a and b as in the figure, calculate the output $\hat{f}(x|a, b)$ for the input $x = 0.9$

b) Calculate the numerical gradient of the MSE regression loss $L = \frac{1}{2}(f(x) - \hat{f}(x|a, b))^2$ with respect to b with your result from before, i.e., for $x = 0.9$.

c) Calculate the numerical gradient of the same loss with respect to the parameter a.
Hint: The derivative of the hyperbolic tangent is given by $\frac{d}{dz} \tanh(z) = 1 - \tanh^2(z)$

d) Given a learning rate $\alpha = 0.1$, update the parameters with the calculated gradients. What issue do you see?

e) If you instead start with $a = 1$ and $b = 100$, what other issue will arise?

Bonus Can you give a parametrization that would give a decent approximation?

3 Gradient Descent with Momentum

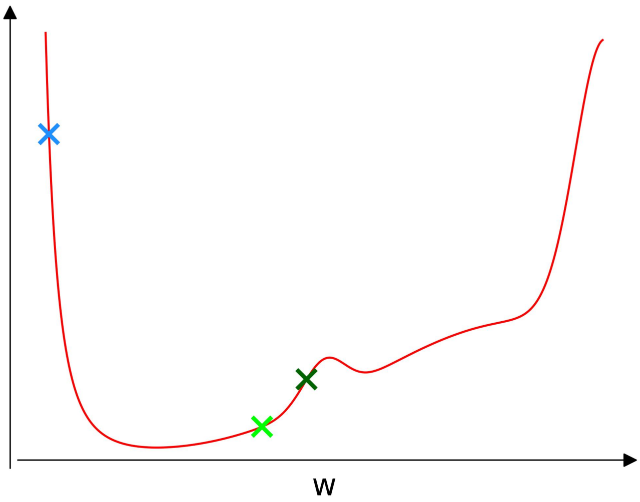

Figure 2: Loss surface and initialization point of a parameter within a neural network.

Gradient descent presents some difficulties, such as setting an appropriate learning rate. Here we introduce a heuristic that helps to overcome some of these difficulties: Momentum. Recall that in gradient descent the update of a parameter w is $w := w - \alpha \cdot g_w$ where g_w is the gradient $\frac{\partial}{\partial w} L(\hat{f}, D)$. Gradient descent with momentum stores an auxiliary variable m_w for each parameter w and updates the parameters in two steps: First, the momentum parameter is updated as $m_w := \beta \cdot m_w + (1 - \beta) \cdot g_w$, where $\beta \in [0, 1)$ is an additional hyperparameter. Second, the model parameter is updated as $w := w - \alpha \cdot m_w$.

a) For which value of β is gradient descent with momentum equivalent to standard gradient descent?

b) Figure 2 shows the loss of a neural network with respect to a single parameter w of the network. We first look at the green x's. The dark green x marks the initial value of the parameter w, the light green x marks its value after a first gradient descent step. Roughly mark in the figure where the next update will end up if we were to follow normal gradient decent.

c) Now what if we use momentum? Roughly mark in the figure where the next update will end up if we follow gradient decent with momentum for $\beta = 0.99$

d) Next we look at the blue x, which marks the initial value of the parameter in another run. Mark in the figure, where gradient descent on w with a sufficiently small learning rate α will end up on this loss surface (after several updates).

e) What happens in the case of gradient descent with momentum for the initial value marked by the blue x? Does it still reach the global optimum?