NUMERISCHE MATHEMATIK I | FROM-SCRATCH MLP, BACKPROPAGATION, AND GRADIENT DESCENT

JONAS BRESCH

ABSTRACT. These notes introduce the theoretical background and practical derivations for implementing a fully connected multilayer perceptron (MLP) from scratch. Hidden layers use the *softplus* activation (a smooth, everywhere-differentiable alternative to ReLU). We derive forward and backward passes, prove the required gradient identities (softmax + cross-entropy, linear layer, softplus derivative), discuss full-batch gradient descent and provide proofs of the basic decrease lemma under smoothness, and conclude with practical considerations and gradient checking.

1. PROBLEM SETUP

We consider a supervised multi-class classification problem. Given

(1) training examples $(x^{(i)}, y^{(i)})$ for $i = 1, \dots, N$,
(2) with input $x^{(i)} \in \mathbb{R}^D$,
(3) and labels $y^{(i)} \in \{1, \dots, K\}$

Our goal is to learn parameters θ of a function

$$f_\theta : \mathbb{R}^D \to \{1, \dots, K\}$$

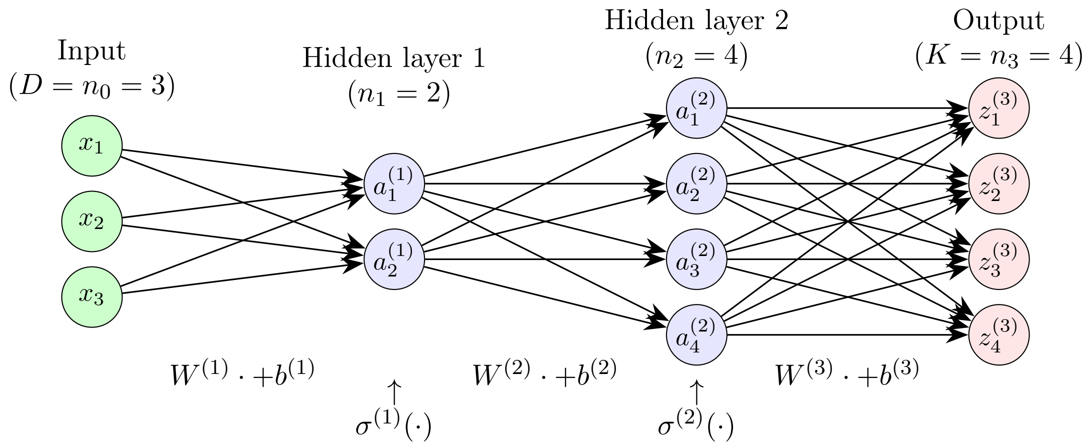
that predicts the labels. In the exercise we restrict to digit recognition (MNIST) with classes $\{1, 5, 7\}$ mapped to $\{0, 1, 2\}$, so $K = 3$.

2. MODEL: MULTILAYER PERCEPTRON (MLP)

The function $f(\theta, \cdot)$ is constructed as follows: There is a number of layers L with fixed layer sizes $n_0 = D, n_1, \dots, n_L = K$. For each of the layer $\ell = 1, \dots, L$, we have

Date: November 13, 2025.

FIGURE 1. Selection of MNIST digest, each sample is of size 28×28, i.e. $D = 28^2 = 784$.



(1) weight matrix $W^{(\ell)} \in \mathbb{R}^{n_{\ell-1} \times n_\ell}$,
(2) and bias vector $b^{(\ell)} \in \mathbb{R}^{n_\ell}$.

Together, they define the searched parameter $\theta = (W^{(1)}, b^{(1)}, ..., W^{(L)}, b^{(L)})$. The network computes, for input $x \in \mathbb{R}^{n_1}$,

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}, \qquad a^{(0)} = x,$$
$$a^{(\ell)} = \sigma^{(\ell)}(z^{(\ell)}), \quad \ell = 1, \ldots, L, \qquad z^{(L)} = f(\theta; x).$$

Here, for incorporating some (component-wise) non-affine actions in the network, we use some *activation functions* $\sigma^{(\ell)} : \mathbb{R} \to \mathbb{R}$. In summary, we have

$$f(\theta; x) = \sigma^{(L)}(W^{(L)}\sigma^{(L-1)}(W^{(L-1)}...\sigma^{(2)}(W^{(2)}\sigma^{(1)}(W^{(1)}x + b^{(1)}) + b^{(2)}) + ... + b^{(L-1)}) + b^{(L)}.$$

Possible choices for the activation functions for each layer are

(1) the *rectified linear unit* function: $\mathrm{ReLu}(x) := \max\{x, 0\}$,
(2) the *parametric rectified linear unit* function: $\mathrm{PReLu}(x) := \max\{x, 0\} + \alpha \max\{-x, 0\}$, for some choice of $\alpha > 0$,
(3) the *softplus* functions (properties $\sigma(x) \geq \mathrm{ReLU}(x), |\sigma(x) - \mathrm{ReLU}(x)| \to 0, |x| \to \infty$), one example is $\sigma(x) = \log(1 + \exp(x))$,
(4) the *sigmoid* function: $\sigma(x) = \frac{1}{1+\exp(-x)} = 1 - \sigma(-x)$,
(5) the *tangens hyperbolicus* $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

In the following, we will use for all hidden layers the softplus activation function $\sigma^{(\ell)}(x) = \sigma(x) := \log(1 + \exp(x))$. For the last layer, we use the identity $\sigma^{(L)}(x) = x$ and hence $z^{(L)} = a^{(L)}$ is the output.

Lemma 2.1. *The softplus activation function* $\sigma := \log(1 + \exp(x))$ *is smooth and has derivative*

$$\sigma'(x) = \tfrac{1}{1+e^{-x}}, \qquad x \in \mathbb{R}.$$

Proof. This can be shown by simple arguments from Analysis I. □

3. Loss and objective

For finding the optimal parameter θ, so that the neuronal network is doing what we aim to predict on the trainings set $(x^{(i)}, y^{(i)})$, we have to define a so-called *loss function*

$$\ell(\theta; x, y) = \ell(f(\theta; x), y)$$

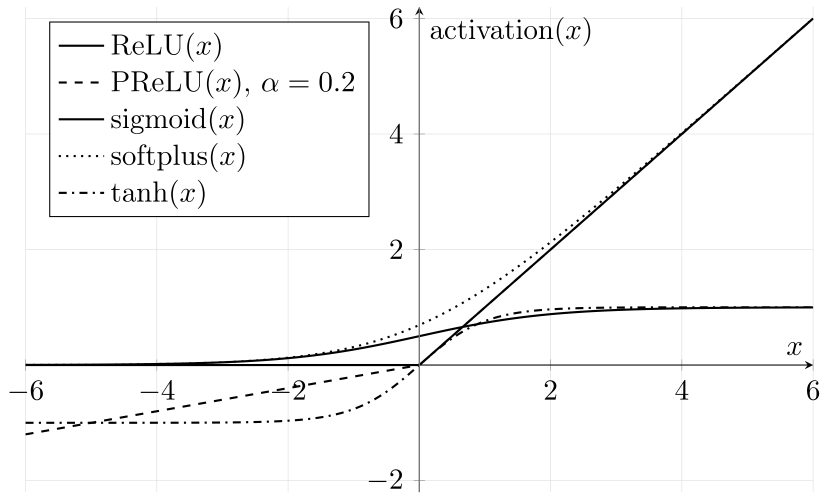


FIGURE 2. Visualization of the different activation functions (1)–(5).

for all given $(x, y) \in \{(x^{(i)}, y^{(i)}) : 1 \leq i \leq N\}$. The loss function for the entire network is then defined by the following sum

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(\theta; x^{(i)}), y^{(i)}),$$

Sometimes, regularizing might improve the results. Hence, one might use the following version for some family of regularization parameters $\{\lambda^{(i)} : 1 \leq i \leq N\}$

$$\mathcal{L}_\lambda(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(\theta; x^{(i)}), y^{(i)}) + \frac{\lambda}{2} \sum_{\ell=1}^{L} \|W^{(\ell)}\|_F^2.$$

For the appropriate definition of a loss function on $\mathbb{R}^K$ we use the so-called one-hot encoded labels $y^{(i)} \in \mathbb{R}^K$, defined by

$$y_j^{(i)} := \delta_{j, y^{(i)}}, \quad j \in \{1, ..., K\}, \quad 1 \leq i \leq N.$$

Possible choices for the losses are

(1) $\ell(s, y) = \|s - y\|^2$,
(2) $\ell(s, y) = -\sum_{j=1}^{K} y_j \log(p_j(s))$, where $p_j(s) := \frac{\exp(s_j)}{\sum_{k=1}^{K} \exp(s_k)}$.

4. FORWARD PROPAGATION — IMPLEMENTATION NOTES

- Compute pre-activations $z^{(\ell)}$ and activations $a^{(\ell)}$ for all layers and store them for backpropagation.
- For numerical stability, compute softmax on logits s by subtracting the row-wise maximum: $\text{softmax}(s) = \text{softmax}(s - m)$ with $m = \max_j s_j$. For softplus and sigmoid, use numerically stable routines (e.g. `log1p` for $\log(1 + e^u)$).

axis = 0 columwise
axis = 1 rowwise

5. BACKWARD PROPAGATION — DERIVATIONS

The crucial idea is to use the recursice definition of the neuronal network to obtain the derivative if the loss function with respect to the parameters. This gradient can be used to perform a *Gradient Descent* of the following form to update in each *epoch* the learned parameters θ in the following manner:

$$\theta^{(t+1)} := \theta^{(t)} - \alpha^{(t)} \nabla_\theta \mathcal{L}(\theta^{(t)}), \quad t = 0, 1, 2, ...$$

where $\alpha^{(t)}$ is the chosen step size, also called the *learning rate*. By the definition of the loss function we immediately have

$$\nabla_\theta \mathcal{L}(\theta^{(t)}) = \sum_{i=1}^{N} \nabla_\theta \ell(f(\theta^{(t)}; x^{(i)}), y^{(i)}).$$

For large $N \gg 1$ performing a "full" gradient descent calculation and update might be too computational expansive. Hence, stochastic versions of the gradient descent (SGD) are used, where only a (random) selection of the parameters are updated, the so-called *batches*. Anyway, for the derivative of $\mathcal{L}$ w.r.t. the parameters θ, we have to calculate the derivative of ℓ w.r.t. each of $\{W^{(\ell)}, b^{(\ell)}\}$. For this the recursive architecture of the network comes in to play using, mainly, the cain role for the Frechet or Total derivative from Analysis II.

We first notice that

$$\nabla_\theta \ell(f(\theta, x^{(i)}), y^{(i)}) = \nabla_s \ell(s, y^{(i)})[f(\theta, x^{(i)})] \circ \nabla_\theta f(\theta, x^{(i)}).$$

Second, the derivative of the trainings function $f(\theta, x^{(i)})$ is now recursively obtained, by

$$\nabla_\theta f(\theta, x^{(i)}) = \nabla_\theta z^{(L)} = \nabla_\theta W^{(L)} a^{(L)}(\theta) + b^{(L)}.$$

Starting with the derivatives with resect to $W^{(L)}$ and $b^{(L)}$, we obtain

$$\nabla_{W^{(L)}} W^{(L)} a^{(L)}(\theta) + b^{(L)} = a^{(L)}(\theta)^T, \quad \nabla_{b^{(L)}} W^{(L)} a^{(L)}(\theta) + b^{(L)} = 1.$$

Now, we get

$$\nabla_{b^{(\ell)}} a^{(\ell)}(\theta) = \nabla_{b^{(\ell)}} \sigma(W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}) = \sigma'(z^{(\ell)}) \nabla_{b^{(\ell)}} z^{(\ell)} = \sigma'(z^{(\ell)})$$

and

$$\nabla_{W^{(\ell)}} a^{(\ell)}(\theta) = \nabla_{W^{(\ell)}} \sigma(W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}) = \sigma'(z^{(\ell)}) \nabla_{W^{(\ell)}} z^{(\ell)} = \sigma'(z^{(\ell)})(a^{(\ell-1)})^T.$$

As we have

$$\nabla_{b^{(\ell-1)}} z^{(\ell)} = \nabla_{b^{(\ell-1)}} W^{(\ell)} a^{(\ell-1)}(\theta) + b^{(\ell)} = (W^{(\ell)})^T \nabla_{b^{(\ell-1)}} a^{(\ell-1)}(\theta),$$

we obtain the recursive scheme

$$\nabla_{b^{(\ell-1)}} f(\theta, x) = \sigma'(z^{(\ell-1)}(W^{(\ell)})^T \nabla_{b^{(\ell)}} f(\theta, x), \qquad \nabla_{W^{(\ell-1)}} f(\theta, x) = \nabla_{b^{(\ell-1)}} f(\theta, x)(a^{(\ell-2)})^T.$$

For the two loss function, we have

$$\begin{aligned} \nabla_s \|s - x\|^2 &= 2(s - x), \\ \nabla_s - \sum_{j=1}^{K} y_j \log(p_j(s)) &= -\nabla_s \sum_{j=1}^{K} y_j (s_j - \log(\sum_{k=1}^{K} e^{s_k})) = [-y_j + \sum_{j=1}^{K} y_j e^{s_j} / \sum_{k=1}^{K} e^{s_k}]_j. \end{aligned}$$

5.1. **Putting it together: full backpropagation, $\ell = \|\cdot - \cdot\|^2$.** Starting with $\nabla_{B^{(L)}} \ell = \Delta^{(L)} = \frac{1}{N}(f(\theta; x) - y)$ (gradient w.r.t. final logits), and $\nabla_{W^{(L)}} \ell = \Delta^{(L)}(a^{(L-1)})^T$ for $\ell = L, L-1, \ldots, 1$ compute

$$\begin{aligned} \nabla_{b^{(\ell-1)}} &= \operatorname{diag}(\sigma'(z^{(\ell-1)})(W^{(\ell)})^T \Delta^{(\ell)} =: \Delta^{(\ell-1)}, \\ \nabla_{W^{(\ell-1)}} \ell &= \Delta^{(\ell-1)}(a^{(\ell-2)})^\top (+\lambda W^{(\ell)}). \end{aligned}$$

6. Gradient descent: one-step analysis and decrease lemma

At each epoch we perform the update from above an assume now that $\alpha^{(t)} = \alpha$ is a constant step size (learning rate). We now state a standard result that gives a sufficient condition for decrease when the objective has Lipschitz-continuous gradient.

Theorem 6.1 (Descent lemma)**.** *Assume $\mathcal{L}$ is differentiable and its gradient is L-Lipschitz: for all u, v,*

$$\|\nabla\mathcal{L}(u) - \nabla\mathcal{L}(v)\| \leq L\|u - v\|.$$

Then for any $\alpha \in (0, 2/L)$,

$$\mathcal{L}(\theta - \alpha\nabla\mathcal{L}(\theta)) \leq \mathcal{L}(\theta) - \frac{\alpha(2 - \alpha L)}{2}\|\nabla\mathcal{L}(\theta)\|^2.$$

Proof. From smoothness we have

$$\mathcal{L}(\theta - \alpha g) \leq \mathcal{L}(\theta) - \alpha g^\top g + \frac{L\alpha^2}{2}\|g\|^2,$$

with $g = \nabla\mathcal{L}(\theta)$. Rearranging yields the stated inequality. $\square$

This theorem explains why a sufficiently small learning rate yields a guaranteed decrease per step; however for neural networks the assumptions needed (global Lipschitz gradient, convexity) do not hold in general and empirical tuning is required.

7. Complexity and memory

For one forward/backward pass on N samples, cost is $\mathcal{O}(N\sum_{\ell=1}^{L} n_{\ell-1}n_\ell)$ in arithmetic operations. Memory to store activations for backprop grows with $\mathcal{O}(N\sum_{\ell=1}^{L-1} n_\ell)$.

8. Practical tips

- Use stable softmax (subtract max per row) and add a small epsilon when computing log in cross-entropy.
- For full-batch GD, use smaller learning rates than for SGD.
- Monitor train and validation loss; use early stopping.
- If training diverges decrease learning rate or check gradient signs with gradient checking.

9. References and further reading

- Goodfellow, Bengio, Courville: "Deep Learning" (MIT Press) — Chapters on optimization and neural networks.
- LeCun et al.: "Efficient BackProp" (1998) — initialization and practical tips.
- Papers on Adam, BatchNorm, etc., for advanced optimizers and tricks.