Deep Learning

Tutorial 5 – Numerical Stability and SGD (student version)

Glenn Angrabeit
Autonomous Vision Group
University of Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN





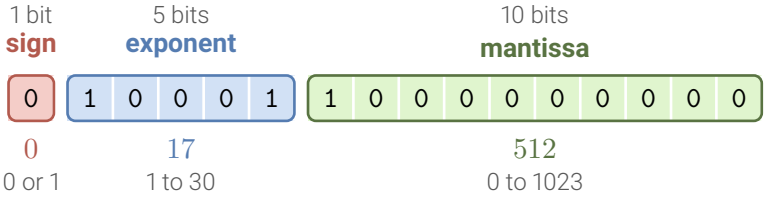
Tübingen AI Center
tuebingen.ai



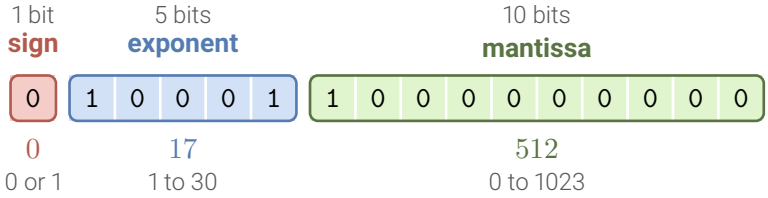
ellis
European Laboratory for Learning and Intelligent Systems

Numerical Stability

How a float16 stores a number

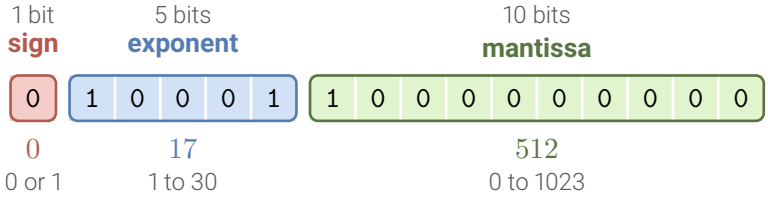


How a float16 stores a number



$$x = (-1)^{\text{sign}} \cdot 2^{\text{exponent} - 15} \cdot \left(1 + \frac{\text{mantissa}}{1024}\right)$$

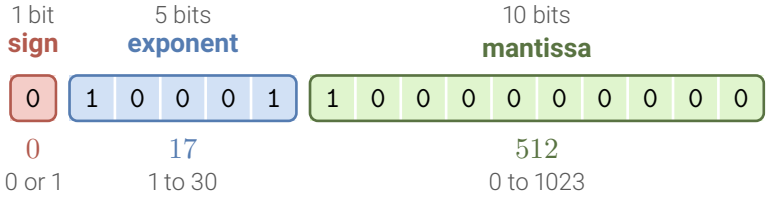
How a float16 stores a number



$$x = (-1)^{\text{sign}} \cdot 2^{\text{exponent} - 15} \cdot \left(1 + \frac{\text{mantissa}}{1024}\right)$$

$$6.0 = (-1)^{0} \cdot 2^{17-15} \cdot \left(1 + \frac{512}{1024}\right) = 1 \cdot 4 \cdot 1.5$$

How a float16 stores a number



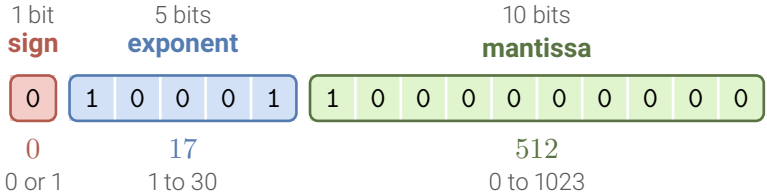
$$x = (-1)^{\text{sign}} \cdot 2^{\text{exponent} - 15} \cdot \left(1 + \frac{\text{mantissa}}{1024}\right)$$

$$6.0 = (-1)^{0} \cdot 2^{17-15} \cdot \left(1 + \frac{512}{1024}\right) \; = \; 1 \cdot 4 \cdot 1.5$$

Special cases:

1) $\text{exponent} = 31$: $x = (-1)^{\text{sign}} \cdot \infty$ $\Rightarrow$ largest $x > 0$: 65504

How a float16 stores a number



$$x = (-1)^{\text{sign}} \cdot 2^{\text{exponent} - 15} \cdot \left(1 + \frac{\text{mantissa}}{1024}\right)$$

$$6.0 = (-1)^{0} \cdot 2^{17-15} \cdot \left(1 + \frac{512}{1024}\right) \; = \; 1 \cdot 4 \cdot 1.5$$

Special cases:

1) exponent $= 31$: $x = (-1)^{\text{sign}} \cdot \infty$ $\Rightarrow$ largest $x > 0$: 65504

2) exponent $= 0$: $x = (-1)^{\text{sign}} \cdot 2^{-14} \cdot \frac{\text{mantissa}}{1024}$ $\Rightarrow$ smallest $x > 0$: $\approx 6 \cdot 10^{-8}$

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

- The value is lost: $70000 \neq 80000,$ but both become ∞
- Arithmetic with ∞ gives NaN

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

- The value is lost: $70000 \neq 80000,$ but both become ∞
- Arithmetic with ∞ gives NaN

> ⚠ **Warning:** a single NaN in the loss makes all gradients and weights NaN.

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

- The value is lost: $70000 \neq 80000,$ but both become ∞
- Arithmetic with ∞ gives NaN

> ⚠ **Warning:** a single NaN in the loss makes all gradients and weights NaN.

Underflow: non-zero numbers too close to 0 are stored as 0

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

- The value is lost: $70000 \neq 80000$, but both become ∞
- Arithmetic with ∞ gives NaN

> ⚠ **Warning:** a single NaN in the loss makes all gradients and weights NaN.

Underflow: non-zero numbers too close to 0 are stored as 0

- Harmless on its own, but a problem once a function needs $x \neq 0$

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

- The value is lost: $70000 \neq 80000$, but both become ∞
- Arithmetic with ∞ gives NaN

> ⚠ **Warning:** a single NaN in the loss makes all gradients and weights NaN.

Underflow: non-zero numbers too close to 0 are stored as 0

- Harmless on its own, but a problem once a function needs $x \neq 0$

Example in float16:

- $e^x = \infty$ for $x > 11.09 \quad \Rightarrow \; e^x - e^x = \infty - \infty = \text{NaN}, \; \ln(e^x) = \ln(\infty) = \infty$

Overflow and underflow

Overflow: numbers outside the range are stored as $\pm\infty$

- The value is lost: $70000 \neq 80000$, but both become ∞
- Arithmetic with ∞ gives NaN

> ⚠ **Warning:** a single NaN in the loss makes all gradients and weights NaN.

Underflow: non-zero numbers too close to 0 are stored as 0

- Harmless on its own, but a problem once a function needs $x \neq 0$

Example in float16:

- $e^x = \infty$ for $x > 11.09 \quad \Rightarrow \; e^x - e^x = \infty - \infty = \text{NaN}, \; \ln(e^x) = \ln(\infty) = \infty$
- $e^x = 0$ for $x < -17.33 \quad \Rightarrow \; \ln(e^x) = \ln(0) = -\infty$

Recap: the sigmoid function

Standard definition: $\sigma(x) = \frac{1}{1 + e^{-x}}$

Recap: the sigmoid function

Standard definition: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

- If $x \gg 0$: $\sigma(x) = \dfrac{1}{1 + e^{-x}} = \dfrac{1}{1 + 0} = 1$

  $\Rightarrow$ e^{-x} underflows along the way, but that is no concern

Recap: the sigmoid function

Standard definition: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

- If $x \gg 0$: $\sigma(x) = \dfrac{1}{1+e^{-x}} = \dfrac{1}{1+0} = 1$

  $\Rightarrow$ e^{-x} underflows along the way, but that is no concern

- If $x \ll 0$: $\sigma(x) = \dfrac{1}{1+e^{-x}} = \dfrac{1}{1+\infty} = \dfrac{1}{\infty} = 0$

  $\Rightarrow$ the result underflows as soon as e^{-x} overflows

Recap: the sigmoid function

Standard definition: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

- If $x \gg 0$: $\sigma(x) = \dfrac{1}{1+e^{-x}} = \dfrac{1}{1+0} = 1$

  $\Rightarrow$ e^{-x} underflows along the way, but that is no concern

- If $x \ll 0$: $\sigma(x) = \dfrac{1}{1+e^{-x}} = \dfrac{1}{1+\infty} = \dfrac{1}{\infty} = 0$

  $\Rightarrow$ the result underflows as soon as e^{-x} overflows

Can we find a mathematically equivalent definition that is more stable for large negative inputs?

Your turn: a second form of the sigmoid

Show that $$\frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}.$$

1. What happens for $x \ll 0$? Did it help?

2. What happens for $x \gg 0$? Does it still work?

Your turn: the log-sigmoid

Losses use $\log \sigma(x)$, but even the stable sigmoid **underflows** for $x \ll 0$:

$$\sigma(x) \to 0 \quad \Rightarrow \quad \log \sigma(x) = \log(0) = -\infty$$

Show that:

i) $\log \sigma(x) = -\log\left(1 + e^{-x}\right)$

ii) $\log \sigma(x) = x - \log\left(1 + e^{x}\right)$

Which one would you choose for $x \geq 0$, and which for $x < 0$?

Stochastic Gradient Descent

True or False?

Statement	**True**	**False**
The gradient descent update equation is given by $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$ where $\mathcal{L}(\cdot)$ is the loss function and $\eta > 0$ is the learning rate.		
If $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \neq \mathbf{0}$, then there always exists a learning rate $\eta > 0$ such that the gradient descent step lowers the loss.		
SGD with batch size equal to the dataset size corresponds to gradient descent.		
In deep learning, an iteration is a complete pass through the training set.		
Stochastic gradient descent computes an approximation to the true gradient.		
Larger batch sizes generally lead to smaller gradient variance.		

Full gradient or stochastic gradient?

Ignoring time, would you always prefer a full gradient descent step over a stochastic gradient descent step?

Old exam question (2024)

a) Consider the model $f(\mathbf{x}) = w_1 x_1^2 + w_2 x_2$, with parameters $\mathbf{w} = (w_1, w_2)^\top$, inputs $\mathbf{x} = (x_1, x_2)^\top$, and ℓ_2 loss function $\mathcal{L} = \frac{1}{2n} \sum_{i=1}^{n} \left(f(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$. Compute the gradient of the loss wrt. the parameters.

Old exam question (2024)

a) Consider the model $f(\mathbf{x}) = w_1 x_1^2 + w_2 x_2$, with parameters $\mathbf{w} = (w_1, w_2)^\top$, inputs $\mathbf{x} = (x_1, x_2)^\top$, and ℓ_2 loss function $\mathcal{L} = \frac{1}{2n} \sum_{i=1}^{n} \left(f(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$. Compute the gradient of the loss wrt. the parameters.

1. Derive the gradient entrywise: $\partial \mathcal{L} / \partial w_k = ?$
2. Define the gradient descent update step entrywise: $w_k^{t+1} = ?$

Old exam question (2024)

b) Using the calculated Jacobian matrix, and the dataset $D = \{(1, 1, 0), (1, -1, 2), (1, 2, -1), (-1, -1, 2)\}$ (the first two elements of each sample are the inputs (x_1, x_2), and the last element is the target y) perform two stochastic gradient descent steps, specifying the values of the parameters. Assume a learning rate of $\alpha = 0.1$, initial parameter values $\mathbf{w}^0 = (0, 0)^\top$, and that the first two data points are sampled in the first, and the second two data points in the second batch. Round to two decimals.

Old exam question (2024)

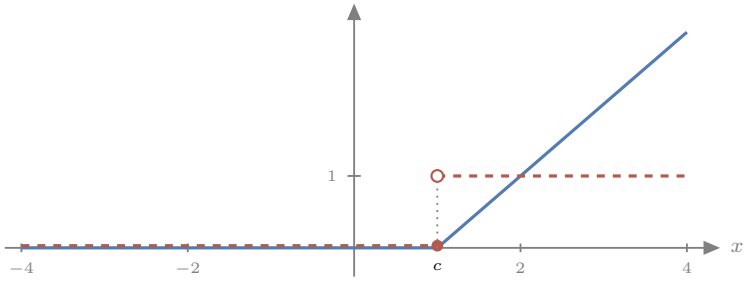
b) Using the calculated Jacobian matrix, and the dataset $D = \{(1, 1, 0), (1, -1, 2), (1, 2, -1), (-1, -1, 2)\}$ (the first two elements of each sample are the inputs (x_1, x_2), and the last element is the target y) perform two stochastic gradient descent steps, specifying the values of the parameters. Assume a learning rate of $\alpha = 0.1$, initial parameter values $\mathbf{w}^0 = (0, 0)^\top$, and that the first two data points are sampled in the first, and the second two data points in the second batch. Round to two decimals.

$$w_1^{t+1} = w_1^t - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(w_1^t \left(x_1^{(i)}\right)^2 + w_2^t \, x_2^{(i)} - y^{(i)} \right) \left(x_1^{(i)}\right)^2$$

$$w_2^{t+1} = w_2^t - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(w_1^t \left(x_1^{(i)}\right)^2 + w_2^t \, x_2^{(i)} - y^{(i)} \right) x_2^{(i)}$$

Exercise 2: Solutions

1.1 a): ReLU with a threshold c



$f(x)$

$f'(x)$

here $c = 1$

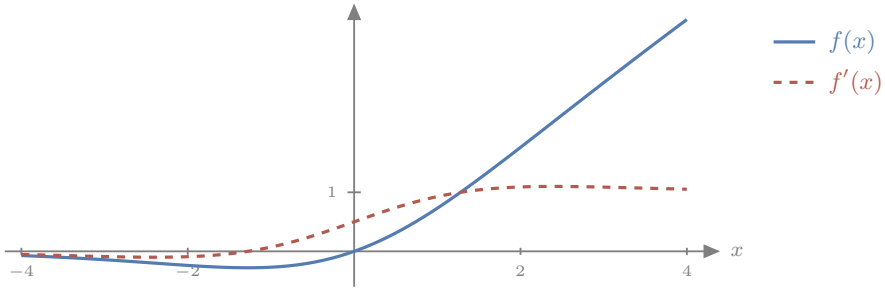
$\mathrm{ReLU}_c(x) = \max(0, x - c)$

$x > c$: $\mathrm{ReLU}_c(x) = x - c$, so $\frac{\partial \mathrm{ReLU}_c(x)}{\partial x} = 1$

$x < c$: $\mathrm{ReLU}_c(x) = 0$, so $\frac{\partial \mathrm{ReLU}_c(x)}{\partial x} = 0$

$x = c$: a kink, not differentiable; by convention take 0

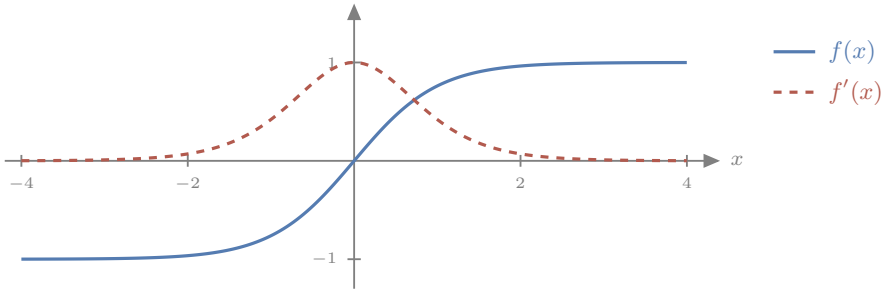
1.1 b): SiLU



$$\mathrm{SiLU}(x) = x\,\sigma(x), \quad \sigma'(x) = \sigma(x)\big(1-\sigma(x)\big)$$

$$\frac{\partial \mathrm{SiLU}(x)}{\partial x} = \sigma(x) + x\,\sigma(x)\big(1-\sigma(x)\big) = \sigma(x)\Big(1 + x\big(1-\sigma(x)\big)\Big)$$

1.1 c): Tanh



$$\text{Tanh}(x) = \frac{2}{1+e^{-2x}} - 1 = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{\partial \text{Tanh}(x)}{\partial x} = \frac{(e^x+e^{-x})(e^x+e^{-x}) - (e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2} = \frac{(e^x+e^{-x})^2 - (e^x-e^{-x})^2}{(e^x+e^{-x})^2}$$

$$= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 = 1 - \text{Tanh}^2(x)$$

1.1 d): the softmax derivative

$$S_j = \mathrm{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{R}, \qquad R = \sum_{i=1}^{n} e^{x_i}, \qquad \frac{\partial R}{\partial x_k} = e^{x_k}$$

$$\boldsymbol{j = k}: \qquad \frac{\partial S_j}{\partial x_j} = \frac{e^{x_j}\, R - e^{x_j}\, e^{x_j}}{R^2} = \frac{e^{x_j}}{R}\left(1 - \frac{e^{x_j}}{R}\right) = S_j\,(1 - S_j)$$

$$\boldsymbol{j \neq k}: \qquad \frac{\partial S_j}{\partial x_k} = \frac{0 \cdot R - e^{x_j}\, e^{x_k}}{R^2} = -\frac{e^{x_j}}{R}\,\frac{e^{x_k}}{R} = -S_j\, S_k$$

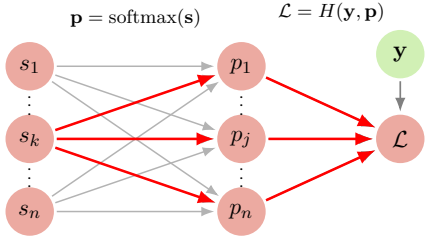
Both cases at once, with the Kronecker delta $\delta_{jk} = 1$ if $j = k$ and 0 otherwise:

$$\frac{\partial S_j}{\partial x_k} = S_j\,(\delta_{jk} - S_k)$$

1.1 e)–f): cross-entropy and its computation graph

e) $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{n} y_j \log p_j \quad \Rightarrow \quad \frac{\partial H}{\partial p_i} = -\frac{y_i}{p_i}$

f) $\mathcal{L}(\mathbf{s}) = H\big(\mathbf{y}, \mathrm{softmax}(\mathbf{s})\big)$



$$\frac{\partial \mathcal{L}}{\partial s_k} = \sum_{j=1}^{n} \frac{\partial \mathcal{L}}{\partial p_j} \frac{\partial p_j}{\partial s_k}$$

1.1 f): cross-entropy on top of the softmax

Both factors are already known:

$$\underbrace{\frac{\partial \mathcal{L}}{\partial p_j} = -\frac{y_j}{p_j}}_{\text{e)}}, \qquad \underbrace{\frac{\partial p_j}{\partial s_k} = p_j \left(\delta_{jk} - p_k\right)}_{\text{d)}}$$

$$\frac{\partial \mathcal{L}}{\partial s_k} = \sum_{j=1}^{n} \Big(-\frac{y_j}{p_j} \Big) p_j \left(\delta_{jk} - p_k\right) = -\sum_{j=1}^{n} y_j \, \delta_{jk} + p_k \underbrace{\sum_{j=1}^{n} y_j}_{=1}$$

$$= -y_k + p_k = p_k - y_k$$

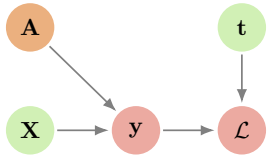
$\Rightarrow$ In vector form $\nabla_{\mathbf{s}} \mathcal{L} = \mathbf{p} - \mathbf{y}$: prediction minus target.

It stays large while the prediction is wrong, even if the softmax saturates.

1.3 a): forward pass of a linear layer

$$\mathbf{A} = (a_1 \;\; a_2) = (3 \;\; 2), \quad \mathbf{X} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -2 & 0 \end{pmatrix}, \quad \mathbf{t} = (15 \;\; 3 \;\; 1), \quad \mathcal{L} = \tfrac{1}{2}\sum_j (y_j - t_j)^2$$

Given: learning rate $\eta = 1$, gradient averaged over the batch of 3



Forward: $\mathbf{y} = \mathbf{AX} = (2a_1 + 4a_2 \quad a_1 - 2a_2 \quad -a_1) = (14 \quad -1 \quad -3)$

$$\mathcal{L} = \tfrac{1}{2}\big((14-15)^2 + (-1-3)^2 + (-3-1)^2\big) = \tfrac{1}{2}(1 + 16 + 16) = 16.5$$

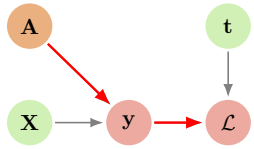
1.3 a): backward pass

- $\mathcal{L} = \frac{1}{2}\sum_j (y_j - t_j)^2$

  $\Rightarrow \dfrac{\partial \mathcal{L}}{\partial \mathbf{y}} = \mathbf{y} - \mathbf{t} = (-1 \quad -4 \quad -4)$

- $\mathbf{y} = \mathbf{A}\mathbf{X} = (2a_1 + 4a_2 \quad a_1 - 2a_2 \quad -a_1)$

  $\Rightarrow \dfrac{\partial \mathbf{y}}{\partial \mathbf{A}} = \begin{pmatrix} \frac{\partial y_1}{\partial a_1} & \frac{\partial y_1}{\partial a_2} \\ \frac{\partial y_2}{\partial a_1} & \frac{\partial y_2}{\partial a_2} \\ \frac{\partial y_3}{\partial a_1} & \frac{\partial y_3}{\partial a_2} \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 1 & -2 \\ -1 & 0 \end{pmatrix}$



Chain rule: $\dfrac{\partial \mathcal{L}}{\partial \mathbf{A}} = \dfrac{\partial \mathcal{L}}{\partial \mathbf{y}} \dfrac{\partial \mathbf{y}}{\partial \mathbf{A}}$

$$\Rightarrow \nabla_{\mathbf{A}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{A}} = (-1 \quad -4 \quad -4) \begin{pmatrix} 2 & 4 \\ 1 & -2 \\ -1 & 0 \end{pmatrix} = (-2 \quad 4)$$

1.3 a)–b): the update and the loss after the step

Update: $\mathbf{A} \leftarrow \mathbf{A} - \eta \frac{1}{3} \nabla_{\mathbf{A}} \mathcal{L} = \begin{pmatrix} 3 + \frac{2}{3} & 2 - \frac{4}{3} \end{pmatrix} = \begin{pmatrix} \frac{11}{3} & \frac{2}{3} \end{pmatrix}$

Forward with the new weights $\mathbf{A} = \begin{pmatrix} \frac{11}{3} & \frac{2}{3} \end{pmatrix}$:

$$\mathbf{y} = \begin{pmatrix} \frac{22}{3} + \frac{8}{3} & \frac{11}{3} - \frac{4}{3} & -\frac{11}{3} \end{pmatrix} = \begin{pmatrix} 10 & \frac{7}{3} & -\frac{11}{3} \end{pmatrix}$$

$$\mathcal{L} = \tfrac{1}{2}\left((10 - 15)^2 + \left(\tfrac{7}{3} - 3\right)^2 + \left(-\tfrac{11}{3} - 1 \right)^2 \right) = \tfrac{1}{2}\left(25 + \tfrac{4}{9} + \tfrac{196}{9} \right) \approx 23.61$$

b) The loss **increases** from 16.5 to 23.61: the learning rate $\eta = 1$ is too large.

Compare statement 8: only a small enough learning rate is guaranteed to lower the loss.

1.3 c): the same layer with a ReLU

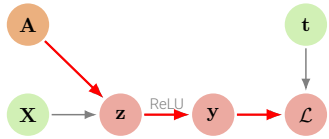
- **Forward:** $\mathbf{z} = \mathbf{A}\mathbf{X} = (14 \quad -1 \quad -3)$

  $\mathbf{y} = \mathrm{ReLU}(\mathbf{z}) = (14 \;\; 0 \;\; 0)$

- $\dfrac{\partial \mathcal{L}}{\partial \mathbf{y}} = \mathbf{y} - \mathbf{t} = (-1 \quad -3 \quad -1)$

- $\dfrac{\partial \mathbf{y}}{\partial \mathbf{z}} = \mathrm{diag}\big(\mathrm{ReLU}'(\mathbf{z})\big) = \mathrm{diag}(1, 0, 0)$

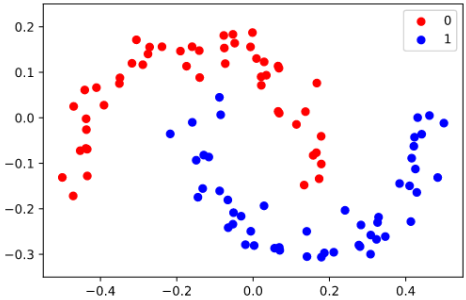
- $\dfrac{\partial \mathbf{z}}{\partial \mathbf{A}}$: as in a)



Chain rule: $\dfrac{\partial \mathcal{L}}{\partial \mathbf{A}} = \dfrac{\partial \mathcal{L}}{\partial \mathbf{y}} \dfrac{\partial \mathbf{y}}{\partial \mathbf{z}} \dfrac{\partial \mathbf{z}}{\partial \mathbf{A}}$

$$\Rightarrow \nabla_{\mathbf{A}} \mathcal{L} = (-1 \quad -3 \quad -1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 4 \\ 1 & -2 \\ -1 & 0 \end{pmatrix} = (-2 \quad -4)$$

2.1: binary classification of a 2D point cloud



- 100 points $\mathbf{x} \in \mathbb{R}^2$, label 0 or 1
- Two **interleaving half moons**
- Goal: classify every point correctly

The tasks:

a) train a logistic regression
b) plot its decision boundary
c) train a 2-layer MLP
d) plot its decision boundary

Can logistic regression separate red from blue?

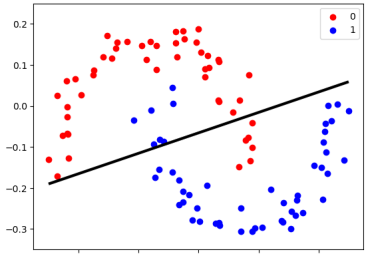
2.1 a)–b): logistic regression

```
params = edf.AffineParams(2, 1)
p = edf.Sigmoid(edf.Affine(params, x_node))
prob_node = SingleProbToProbVector(p)
loss_node = edf.CrossEntropyLoss(prob_node, y_node)
```

b) Evaluate the same graph on a grid:

```
output = edf.Sigmoid(edf.Affine(params, input))
```

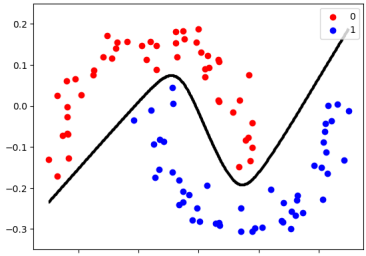
- **Result:** stuck at 13 of 100 points wrong



2.1 c)–d): a two-layer MLP

```
params1 = edf.AffineParams(2, 24)
params2 = edf.AffineParams(24, 1)
h = edf.Sigmoid(edf.Affine(params1, x_node))
p = edf.Sigmoid(edf.Affine(params2, h))
prob_node = SingleProbToProbVector(p)
```

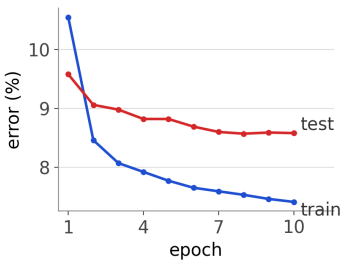
- **Result:** 0 of 100 wrong after 10K iterations
- A hyperplane in $\mathbf{h}$-space is a **curve** in $\mathbf{x}$-space



2.2 a)–b): MNIST, logistic regression with 10 classes

60K training and 10K test images, each 28×28 flattened to $\mathbf{x} \in \mathbb{R}^{784}$

```
edf.clear_compgraph()
x_node = edf.Input()
y_node = edf.Input()
params = edf.AffineParams(784, 10)
prob_node = edf.Softmax(edf.Affine(params, x_node))
loss_node = edf.CrossEntropyLoss(prob_node, y_node)
```



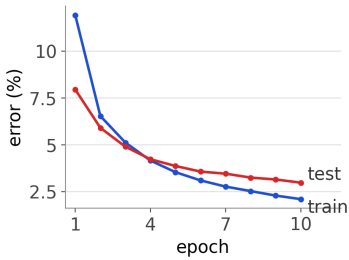
- **Result:** test error 8.58% after 10 epochs

b) Test error of a randomly initialized model, without training?

$\Rightarrow$ About 90%: a random guess is right 1 in 10 times

2.2 c): an MLP with one hidden layer

```
edf.clear_compgraph()
x_node = edf.Input()
y_node = edf.Input()
params1 = edf.AffineParams(784, 64)
params2 = edf.AffineParams(64, 10)
h = edf.Sigmoid(edf.Affine(params1, x_node))
prob_node = edf.Softmax(edf.Affine(params2, h))
loss_node = edf.CrossEntropyLoss(prob_node, y_node)
```



- **Result:** test error 2.98%, about **a third** of logistic regression (8.58%)

2.2 d): other activation functions

Swap `edf.Sigmoid` for `Tanh`, `ReLU` or `SiLU` ($\eta = 0.5$, batch size 64, 10 epochs):

	logistic regression	Sigmoid	Tanh	ReLU	SiLU
train error	7.41%	2.10%	0.66%	0.90%	0.68%
test error	8.58%	2.98%	2.97%	3.04%	3.25%

- On this small problem the activation **hardly matters**

2.2 d): implementing Tanh

Tanh: $\tanh(x) = 2\,\sigma(2x) - 1, \quad \tanh'(x) = 1 - \tanh^2(x)$

```
class Tanh(edf.CompNode):
    def __init__(self, x):
        edf.CompNodes.append(self)
        self.x = x
    def forward(self):
        self.value = 2 / (1 + np.exp(-2 * self.x.value)) - 1
    def backward(self):
        dfdx = 1 - self.value**2
        self.x.addgrad(self.grad * dfdx)
```

- Elementwise $f \Rightarrow$ **diagonal** Jacobian: $\dfrac{\partial \mathcal{L}}{\partial \mathbf{x}} = \dfrac{\partial \mathcal{L}}{\partial \mathbf{f}} \operatorname{diag}\big(f'(\mathbf{x})\big) = \dfrac{\partial \mathcal{L}}{\partial \mathbf{f}} \odot f'(\mathbf{x})$
- So `backward` needs no matrix, just `self.grad * dfdx`

2.2 d): implementing ReLU and SiLU

ReLU: $\mathrm{ReLU}(x) = \max(0, x), \quad \mathrm{ReLU}'(x) = 1 \text{ if } x > 0, \text{ else } 0$

```
def forward(self):
    self.value = np.maximum(0, self.x.value)
def backward(self):
    dfdx = self.x.value > 0    # True/False
    self.x.addgrad(self.grad * dfdx)
```

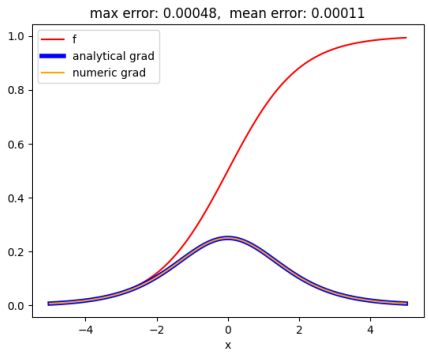
SiLU: $\mathrm{SiLU}(x) = x\,\sigma(x), \quad \mathrm{SiLU}'(x) = \sigma(x)\big(1 + x\,(1 - \sigma(x))\big)$

```
def forward(self):
    self.s = 1 / (1 + np.exp(-self.x.value))   # reused in backward
    self.value = self.x.value * self.s
def backward(self):
    dfdx = self.s * (1 + self.x.value * (1 - self.s))
    self.x.addgrad(self.grad * dfdx)
```

2.2 e): checking the gradients numerically

Symmetric difference quotient with $h = 10^{-4}$: $f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$

```
x1, x2 = edf.Input(), edf.Input()
x1.value = x.value + h
x2.value = x.value - h
f1, f2 = f(x1), f(x2)
edf.Forward()
numeric_grad = (f1.value - f2.value) / (2*h)
```



2.2 f): learning rate

1 hidden layer of 64, 5 epochs, test error:

η	0.1	0.5	1.0
Sigmoid	6.67%	3.70%	$\mathbf{2.97}\%$
ReLU	3.58%	$\mathbf{3.22}\%$	6.72%

- The best η **depends on the architecture**
- $\sigma' \leq \frac{1}{4}$ shrinks the gradients: sigmoid wants larger steps
- Choose η on **validation** data (here: test data, for simplicity)

2.2 g): depth

Hidden layers of 64, best η from f), test error:

hidden layers	2	4	6
Sigmoid ($\eta = 1$)	$\mathbf{3.44\%}$	4.88%	89.72%
ReLU ($\eta = 0.5$)	$\mathbf{3.03\%}$	7.30%	4.80%

- Sigmoid, 6 layers: predicts the **same class** for every image
- Each layer multiplies the gradient by $\sigma' \leq \frac{1}{4}$: **vanishing gradients**
- ReLU still trains, but deeper is not better here

2.2 h)–i): width, and do big networks overfit?

2 hidden layers, 5 epochs, train / test error:

hidden units	32	64	128
Sigmoid ($\eta = 1$)	3.39% / 4.82%	2.40% / 3.44%	2.33% / $\mathbf{3.22}\%$
ReLU ($\eta = 0.5$)	3.37% / 4.70%	2.27% / 3.03%	1.47% / $\mathbf{2.89}\%$

i) 128 units: ≈ 118K parameters, twice the training images. Overfitting?

- No: **train and test error both fall** as the width grows
- Large networks often generalize well. Why is still not fully understood