DPL 2025 **Name:** ____________________

Midterm test

1. **Exponential Time-to-Failure — Quadratic Failure Rate Model**: : We consider a network of sensors. For each sensor we measure a feature vector $x \in \mathbb{R}^d$ (e.g. vibration, temperature, ...). We want to predict the failure rate (hazard) using a quadratic model

$$\lambda(x) = (w^\top x)^2,$$

where $w \in \mathbb{R}^d$ are unknown weights. The observed time-to-failure for one sensor is denoted $y \geq 0$ (seconds). We assume that conditional on x the failure time y follows an exponential distribution

$$p(y \mid x, w) = \lambda(x)\, e^{-\lambda(x)\, y} = (w^\top x)^2\, e^{-(w^\top x)^2 y} \qquad y \geq 0.$$

- 1.1 **Loss function.** You are given training data $\mathcal{D} = \{(x_1, y_1) \dots (x_N, y_N)\}$, where x_i is measured features and corresponding failure time y_i. Derive the **loss function** which minimizes the negative log-likelihood of the training data $\mathcal{D}$ of the above model $p(y \mid x, w)$.

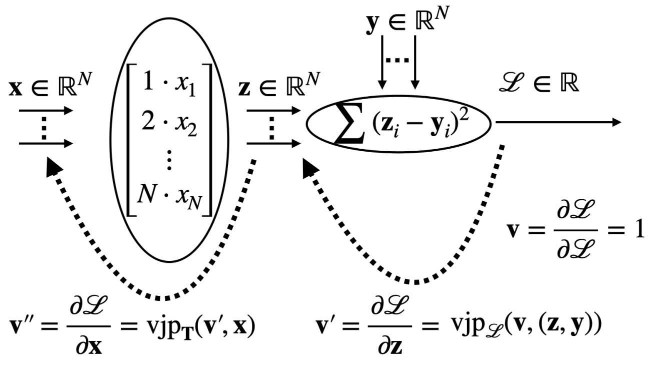
- 1.2 **Draw computational graph** of the loss function for a single training example (x, y) and compute gradient of the loss with respect to w.

- 1.3 **Backpropagation:** Assume $x = [2, 1], y = 1, w = [1, -1]$ and compute feed-forward pass and local gradients to all relevant edges in the computation graphs. Finally, estimate the gradient of the loss with respect to w (the outcome is intentionally suspicious).

- 1.3 **Interpretation:** Explain (by a simple sentence) what the gradient value says about the local shape (around the point $w = [1, -1]$) of the loss landscape.

2. **Vector-jacobian-product:**

   Consider the following computational graph:


$\mathbf{y} \in \mathbb{R}^N$
$\mathbf{x} \in \mathbb{R}^N$
$\begin{bmatrix} 1 \cdot x_1 \\ 2 \cdot x_2 \\ \vdots \\ N \cdot x_N \end{bmatrix}$
$\mathbf{z} \in \mathbb{R}^N$
$\sum (\mathbf{z}_i - \mathbf{y}_i)^2$
$\mathscr{L} \in \mathbb{R}$
$\mathbf{v} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} = 1$
$\mathbf{v}'' = \frac{\partial \mathscr{L}}{\partial \mathbf{x}} = \text{vjp}_{\mathbf{T}}(\mathbf{v}', \mathbf{x})$
$\mathbf{v}' = \frac{\partial \mathscr{L}}{\partial \mathbf{z}} = \text{vjp}_{\mathscr{L}}(\mathbf{v}, (\mathbf{z}, \mathbf{y}))$

   Graph consists of two following layers:

   a) **First layer:** Transformation layer that multiplies elements of input vector $\mathbf{x}$ by the sequence of integer numbers, such that $z_1 = 1 \cdot x_1, \quad z_2 = 2 \cdot x_2, \quad \ldots z_N = N \cdot x_N$.

   b) **Second layer:** L2-loss that compute element-wise square differences between $\mathbf{z} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^N$.

- 2.1 Compute **Jacobian** of the first layer.

- 2.2 Compute **Jacobian** of the second layer.

- 2.3 Suggest an efficient implementation of **vector-Jacobian-product** $\mathbf{v}'' = \frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathrm{vjp}_{\mathbf{T}}(\mathbf{v}', \mathbf{x})$

- 2.3 What will be ratio $r(N) = M_1/M_2$ between number of multiplications of (i) your efficient implementation of $\mathrm{vjp}_{\mathbf{T}}$, denoted as M_1, and (ii) naive implementation through Jacobian multiplications denoted as M_2. For example, ratio $r(N) < 1$ would denote that your implementation is more efficient than a naive solution.

$r(N) =$

3. **Losses and Overfitting:**

- 3.1 **What are the fundamental factors that cause neural networks to overfit?** Select none, one, or multiple answers.
  - It's because we are using neural networks; other models do not overfit.
  - There are many explanations consistent with our observations (training data).
  - The learning rate is too high.
  - The training dataset is too large.
  - The testing dataset is too large.
  - We are minimizng the KL-divergence between our model and the data, but the true data distribution is unknown.
  - The GPUs do not have sufficient computational power.
- 3.2 **How can I reduce overfitting?** Select none, one, or multiple answers.
  - Always model high-dimensional output by a discrete (non-parametric) distribution as it never suffers from curse-of-dimensionality and consequent overfitting.
  - Always assume Gaussian noise
  - Never assume Gaussian noise
  - Use a very large (ideally infinite) dataset.
  - The weights in the classifier/regressor must always be positive.
  - The weights in the classifier/regressor must always be negative.
  - Use the deepest possible convolutional networks.
  - Use the correct model that includes as much prior knowledge about the problem as possible.
  - Avoid a sharp minimum in the loss function.
  - Avoid a flat minimum in the loss function.
  - Use a fully differentiable loss function.
  - Implement all Vector-Jacobian-Product functions on the GPU.

- 3.3 Mark which of the following statements are TRUE/FALSE:
  - 2D line fitting with the L1-norm corresponds to minimizing the KL-divergence between the true data distribution p_{data} and a Laplace distribution with a linear mean constructed as follows:

$$p(\mathbf{x}, y|\mathbf{w}) = \text{Laplace}\big(y;\ w_1 x + w_0,\ b\big) = \frac{1}{2b}\exp\left(-\frac{|y-(w_1 x + w_0)|}{b}\right)$$

  - The value of the global optimum of the following problem is always equal to zero:

$$\min_{\mathbf{w}} D_{KL}\big(p_{\text{data}}(\mathbf{x}, y) \parallel p(\mathbf{x}, y|\mathbf{w})\big) = 0$$

  - The value of the global optimum of the following problem is always negative:

$$\min_{\mathbf{w}} D_{KL}\big(p_{\text{data}}(\mathbf{x}, y) \parallel p(\mathbf{x}, y|\mathbf{w})\big) < 0$$

  - Least-square regression of polynomial function $y = w_0 + w_1 \cdot x + w_2 \cdot x^2 + \cdots + w_N \cdot x^N$, where weights w_i are unknown parameters cannot be computed in closed-form, because the function is non-linear.
  - Least-square regression of function performing rotation $\mathbf{y} = \mathbf{R}(w) \cdot \mathbf{x}$, where $\mathbf{R}(w)$ is the rotation matrix with the angle w is linear least-squares problem.
  - VJP of a convolutional layer can be computed as a convolution.
  - Every input pixel to a convolutional layer always affects all pixels in the output feature map.
  - Consider 1×1 convolution layer with a weight $w \in \mathbb{R}$ and non-zero input image. The change in the w value always changes all output values.

4. **Convolution:** Assume that the input feature map to a convolutional layer is $3 \times 10 \times 10$ (channels$\times$height$\times$width).

- 4.1 What is the size and number of convolution kernels/filters if the output is $10 \times 3 \times 3$? (Assume stride=1, padding=0.)

- 4.2 What is the padding of the convolutional layer if the kernel/filter size is $3 \times 3 \times 3$ and the output is $7 \times 12 \times 12$ (stride=1)?

- 4.3 Now consider a single-channel input $1 \times 10 \times 10$. Suppose we have a network consisting of two convolutional layers (without any activation function). Each layer has a single 3×3 kernel of the following form:

$$\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}, \ \mathbf{v} = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{bmatrix}$$

Can these two layers be replaced by a fully connected layer? If yes, what would its weights be? If not, explain why.

5. **Logistic loss:**

   Assume binary classification problem with the classifier modeling probability as follows:

$$p(y|\mathbf{x}, \mathbf{w}) = \begin{cases} \sigma(f(\mathbf{x}, \mathbf{w})) & y = 1 \\ 1 - \sigma(f(\mathbf{x}, \mathbf{w})) & y = -1 \end{cases}$$

   Show that minimizing negative-log-likelihood leads to the logistic loss:

$$\mathcal{L}(\mathbf{W}) = \sum_i \log \Big(1 + \exp(-y_i f(\mathbf{x}_i, \mathbf{w}))\Big)$$

6. Tell us what you liked and what you disliked so far in the subject and how we should change it.

   *If you have enough time, draw a picture on a topic **"Deep learning and Halloween"**. The best drawings will get bonus points and will be published on the course website.*