Deep Learning

Linear Models

> **Linear models** are a class of **regression** and **classification** models in which the prediction is a linear function of the input variables.

Linear Regression: $y = w^T x + b$

- w is a d-dimensional vector of **weights**;
- b is a **bias** - usually included in w as a constant feature $x_0 = 1$;
- Given **training data** $D = \{(x_n, y_n)\}_{n=1}^N$, we want to find the **best** w and b, so we use want to fit the model, i.e. find the best w and b, **minimizing the loss function** - usually, the **squared loss**: $\hat{w} = argmin_w \frac{1}{2} \sum_{n=1}^N (y_n - (w^T x_n + b))^2$;
  - $MSE = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{y}_n)^2$;
- **Closed-form solution**: $w = (X^T X)^{-1} X^T y$, where $X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$;
- **Regularization** is a technique used to **reduce overfitting** by **constraining** the **weights** of the model;
  - If $X^T X$ is not invertible, we can add a regularization term to the loss function - **ridge regression**;
    - $L(w, b) = \sum_{n=1}^N (y_n - (w^T x_n + b))^2 + \lambda ||w||_2^2$;
    - **Closed-form solution**: $\hat{w_{ridge}} = (X^T X + \lambda I)^{-1} X^T y$;
    - l_2 regularization is also known as **weight decay**, or penalized least squares.

Reduce Overfitting

- **Regularization** is a technique used to **reduce overfitting** by **constraining** the **weights** of the model;
- **Early stopping** is a technique used to **reduce overfitting** by **stopping the training** when the **validation error** starts to **increase**;
- **Dropout** is a technique used to **reduce overfitting** by **randomly dropping neurons** during **training - ensemble** of **smaller networks - regularizes** the **network**;

- **Data augmentation** is a technique used to **reduce overfitting** by **increasing the size** of the **training set - artificially generate new data - regularizes** the **network**.

In case of **underfitting**, we can: * **Increase the model capacity - increase the number of parameters**; * **Decrease the regularization - decrease the regularization constant**; * **Increase the training time - increase the number of epochs**.

Maximum A Posteriori Estimation

- A **Bayesian** approach to **linear regression**;
- Assume that the **weights** are **random variables** with a **prior distribution**: $w \sim \mathcal{N}(0, \tau^2 I)$;
- $w\hat{_{MAP}} = argmax_w p(w|y) = argmin_w \lambda ||w||_2^2 + \sum_{n=1}^{N}(y_n - w^T\phi(x_n))^2$;
  - $||w||_2^2$ is the **regularization term**;
  - $\sum_{n=1}^{N}(y_n - w^T\phi(x_n))^2$ is the **squared loss**.

Perceptron: $y = sign(w^T x + b)$

- **Binary classification** model - outputs $+1$ or -1 (**discrete**);
- **Discriminative** model - **directly** models $p(y|x)$;
- Bias b is included in w as a constant feature $x_0 = 1$;
- x can be represented as $\phi(x)$, where ϕ is a **feature map**;
- Algorithm:

```
1. Initialize $w$ to zero: $w_0 = 0$;
2. While not converged, for each $(x_n, y_n)$ in $D$:
    1. Predict: $\hat{y} = sign(w^T x_n)$;
    2. If $\hat{y} \neq y_n$:
        1. Update: $w_{t+1} = w_t + y_n x_n$;
        2. Go to step 2;
```

- **Perceptron convergence theorem**: if the **training data** is **linearly separable**, the **perceptron algorithm** will **converge** in a **finite number of steps**:
- It **cannot** be used for **non-linearly separable data** - XOR problem.

Logistic Regression: $y = \sigma(w^T x + b)$

- **Logistic regression** is a **linear model** for **binary classification** - differs from perceptron, because it uses a **sigmoid function (continuous)** instead of a **sign function (discrete)**;
- **Binary classification** model - outputs $[0, 1]$ (**continuous**);
- **Discriminative** model - **directly** models $p(y|x)$;
- No closed-form solution, so we use **gradient descent** to find the **best** w and b - **stochastic gradient descent** is usually used, because it is **faster**: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$. The **learning rate** η is usually **small**;

    - Different than **gradient descent**: $w_{t+1} = w_t - \eta \nabla_w \sum_{n=1}^{N} L(w_t, x_n, y_n)$, which uses **all** the **training data** to compute the **gradient**.

Multi-class Classification

- **Multi-class classification** is a **classification** task with **more than two classes**, but there are several strategies to **reduce to binary classification**;
- Parametrized by a **weight matrix** $W \in \mathbb{R}^{d \times |Y|}$ and a **bias vector** $b \in \mathbb{R}^{|Y|}$;
- $\hat{y} = argmax_{y \in Y}(W\phi(x) + b)$;
- Multi-class perceptron algorithm:

```
1. Initialize $W$ to zero: $W_0 = 0$;
2. While not converged, for each $(x_n, y_n)$ in $D$:
    1. Predict: $\hat{y} = argmax_{y \in Y}(W\phi(x_n))$;
    2. If $\hat{y_n} \neq y_n$:
        1. Update: $W_{y_n}^{k+1} = W_{y_n}^{k} + \phi(x_n)$
        2. Update: $W_{\hat{y_n}}^{k+1} = W_{\hat{y_n}}^{k} - \phi(x_n)$;
        3. Go to step 2;
```

Neural Networks

- **Neural networks** are a class of **non-linear models** that are **inspired by the brain**;
- Consist of **neurons** that are **connected** to each other;
- **Pre-activation**: $z(x) = w^T x + b = \sum_{i=1}^{d} w_i x_i + b$;
- **Activation**: $h(x) = g(z(x))$; g can be:
    - **Linear**: $g(z) = z$ - linear regression;
        - Derivative: $\frac{d}{dz} linear(z) = 1$;
        - No squashing of the input;
        - Useful to project the input to a lower dimension;
    - **Sigmoid**: $g(z) = \frac{1}{1+e^{-z}}$ - logistic regression;
        - Derivative: $\frac{d}{dz} sigmoid(z) = sigmoid(z)(1 - sigmoid(z))$;
        - Squashes the input to the range $[0, 1]$;
        - Positive, bounded and strictly increasing;
        - Combining layers of sigmoid units increases expressiveness - logistic regression is a network with a single sigmoid unit;
    - **Rectified Linear Unit (ReLU)**: $g(z) = max(0, z)$ - most used because it is **fast** to compute;
        - Derivative: $\frac{d}{dz} ReLU(z) = \begin{cases} 1, & if z > 0 \\ 0, & if z \leq 0 \end{cases}$;
        - Mitigates the **vanishing gradient problem**;
        - Non-negative, increasing but not upper bounded;
        - Not differentiable at $z = 0$;

    - Leads to **sparse representations - sparsity** is a **desirable property** of **representations**;
  - **Hyperbolic Tangent (tanh)**: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$;
    - Derivative: $\frac{d}{dz}tanh(z) = 1 - tanh^2(z)$;
    - Squashes the input to the range $[-1, 1]$;
    - Bounded and strictly increasing;
    - Combining layers of tanh units increases expressiveness;
  - **Softmax**: $g(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$ - used for **multi-class classification**;
    - Derivative: $\frac{d}{dz}softmax(z)_i = softmax(z)_i(1 - softmax(z)_i)$;

  ReLUs are significantly simpler and have a much simpler derivative than the sigmoid, leading to faster computation times. Also, sigmoids are easy to saturate and, when that happens, the corresponding gradients are only residual, making learning slower. ReLUs saturate only for negative inputs, and have constant gradient for positive inputs, often exhibiting faster learning

Feed-forward Neural Networks

- A **feed-forward neural network** is a **neural network** where the **neurons** are **organized in layers** - there are **hidden layers** between the **input layer** and the **output layer**;
  - **Input layer**: x - vector of features;
  - **Hidden layers**: $h^{(1)}, h^{(2)}, ..., h^{(L)}$;
  - **Output layer**: y - vector of predictions;
  - **Weights**: $W^{(1)}, W^{(2)}, ..., W^{(L)}$ - each weight matrix is between two layers;
  - **Biases**: $b^{(1)}, b^{(2)}, ..., b^{(L)}$ - each bias vector is between two layers;
  - **Hidden layer pre-activation**: $z^{(l)}(x) = W^{(l)}h^{(l-1)}(x) + b^{(l)}$;
  - **Hidden layer activation**: $h^{(l)}(x) = g(z^{(l)}(x))$;
  - **Output layer activation**: $f(x) = h^{(L)}(x)$;
- **Universal approximation theorem**: a **feed-forward neural network** with a **single hidden layer** and **non-linear** activation can **approximate any function** - given enough **neurons**;

Training Neural Networks

- Training consists of **finding the best parameters** θ - weights and biases, that **minimize the loss function**: $L(\theta) := \lambda\omega(\theta) + \frac{1}{N}\sum_{n=1}^{N} L(f(x_n;\theta), y_n)$;
- λ is the **regularization constant**;
- $\omega(\theta)$ is the **regularization term**;
- $L(f(x_n;\theta), y_n)$ is the **loss function**;
  - **Mean squared error**: $L(f(x_n;\theta), y_n) = \frac{1}{2}(f(x_n;\theta) - y_n)^2$ - used for **regression**;

        - The $\frac{1}{2}$ factor is included for **convenience**, so that the derivative of the loss function is $y - \hat{y}$.
    - **Cross-entropy loss** (negative log-likelihood): $L(f(x_n;\theta), y_n) = -\sum_{i=1}^{k} y_{n,i} log(f(x_n;\theta)_i) = -log((softmax(z(x))_y))$ - used for **classification**, usually **logistic regression**;
- **Backpropagation** is a technique used to **compute the gradients** of the **loss function** with respect to the **parameters - chain rule**.

Automatic Differentiation

- **Automatic differentiation** is a **technique** for **computing derivatives** of **functions**;
    - Forward propagation can be represented as a **computation graph** - a **directed acyclic graph** (DAG) that represents the **computation** of the **function**;
        - Each box can be an object with a `fprop` method that computes the **forward pass**;
        - Calling the `fprop` method of each box in the **topological order** of the graph computes the **forward pass**;
    - **Backpropagation** is also implemented as a **computation graph** - a **directed acyclic graph** (DAG) that represents the **computation** of the **gradients**;
        - Each box can be an object with a `bprop` method that computes the **loss gradient** w.r.t. its parents, given the **loss gradient** w.r.t. to the output of the box;
        - Calling the `bprop` method of each box in the **reverse topological order** of the graph computes the **backward pass**.
    - There are several **Autodiff** strategies:
        - **Symbol-to-Number Differentiation**
            - Take a computational graph and numerical inputs;
            - Returns a set of numerical outputs describing the gradient at those inputs;
            - **Advantage**: simpler to implement and debug;
            - **Disadvantage**: only works for first-order derivatives;
            - **Example**: Caffe, Torch, PyTorch, ...
        - **Symbol-to-Symbol Differentiation**
            - Take a computational graph and add additional nodes to the graph that provide a symbolic description of the gradient;
            - **Advantage**: works for higher-order derivatives;
            - **Disadvantage**: more complex to implement and debug;
            - **Example**: Theano, TensorFlow, ...

Regularization

- **Regularization** is a technique used to **reduce overfitting** by **constraining** the **weights** of the model - $\Omega(\theta)$ is the **regularization term**;

- **L2 regularization**: $\Omega(\theta) = \frac{1}{2}\sum_{li=1}^{L+1} ||W^{(l)}||^2$ - **weight decay** - penalizes **large weights**;
  - Equivalent to **Gaussian prior** on the weights;
- **L1 regularization**: $\Omega(\theta) = \sum_{li=1}^{L+1} ||W^{(l)}||$ - **sparse weights** - promotes **sparsity** of the weights;
  - Equivalent to **Laplace prior** on the weights;
- **Dropout**: randomly **drop** some **neurons** during **training** - **ensemble** of **smaller networks** - **regularizes** the **network**;
  - There is a dropout probability p for each neuron;
  - Usually implemented with random binary masks;
  - The hidden layer activations becomes: $h^{(l)}(x) = g(z^{(l)}(x)) \odot m^{(l)}$, where $m^{(l)} \in \{0,1\}^{K^{(l)}}$ is a **random binary mask** with p probability of being 1;

Representing Learning - Auto-encoders

- **Auto-encoders** are feed-forward NNs trained to reproduce its input at its output layer;
  - Useful for **dimensionality reduction** and **unsupervised pre-training**;
- **Encoder** - maps input to a hidden representation : $h = g(Wx + b)$;
- **Decoder** - maps hidden representation to a reconstruction : $\hat{x} = W^T h(x) + c$;
- **Loss function** - $\mathcal{L}(\hat{x}, x) = \frac{1}{2}||\hat{x} - x||^2$;
- **Objective** - $\hat{W} = argmin_W \sum_i ||W^T g(Wx_i) - x_i||^2$ - drop the bias term b.

  Auto-encoders are networks that are trained to learn the identify function, i.e., to reconstruct in the output what they see in the input. This is done by imposing some form of constraint in the hidden representations (e.g. lower dimensional or sparse). They are useful to learn good representations of data in an unsupervised manner, for example to capture a lower-dimensional manifold that approximately contains the data.

Single Value Decomposition (SVD)

- **SVD** is a matrix factorization method that decomposes a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ into the product of three matrices U, Σ and V such that $A = U\Sigma V^T$;
  - $U \in \mathbb{R}^{m \times m}$ - columns are an orthonormal basis of $R(A)$ (left singular vectors);
  - $\Sigma \in \mathbb{R}^{m \times n}$ - diagonal matrix with singular values of A;
  - $V \in \mathbb{R}^{n \times n}$ - columns are an orthonormal basis of $R(A^T)$ (right singular vectors);

- $sigma_1 \geq ... \geq \sigma_r$ - square roots of the eigenvalues of A^TA or AA^T - **singular values** of A;
- $U^TU = I$ and $V^TV = I$.

Linear Auto-Encoder

- Let $X \in \mathbb{R}^{N \times D}$ be a data matrix with N samples and D features ($N > D$);
- Assume $W \in \mathbb{R}^{K \times D}$ ($K < D$);
- We want to minimize $\sum_{i=1}^{N} ||x_i - \hat{x}_i||_2^2 = ||X - XW^TW||_F^2$;
  - $|| \cdot ||_F^2$ - **Frobenius norm**;
  - W^TW has rank K;
- From the **Eckart-Young theorem**, the minimizer is **truncated SVD** of X^T;
  - $\hat{X}^T = U_K \Sigma_K V_K^T$;
  - $W = U_K^T$;
- This is called **Principal Component Analysis (PCA)** - fits a **linear manifold** to the data - auto-encoder with **linear activations**.
- By using **non-linear activations**, we obtain more sophisticated codes (representations).

There are some variants of auto-encoders:

- **Sparse auto-encoders** - add a **sparsity penalty** $\Omega(h)$ to the loss function;
  - Typically the number of hidden units is larger than the number of inputs;
  - The sparsity penalty is a **regularization** term that encourages the hidden units to be **sparse**;
- **Stochastic auto-encoders** - encoder and decoder are **not deterministic**, but involve some **noise/randomness**;
  - Uses distribution $p_encoder(h|x)$ for the encoder and $p_decoder(x|h)$ for the decoder;
  - The auto-encoder can be trained to minimize $-log(p_decoder(x|h))$;
- **Denoising auto-encoders** - use a **perturbed version of the input** $\tilde{x} = x + n$, where n is a **random noise**;
  - Instead of minimizing $\frac{1}{2}||\hat{x} - x||^2$, we minimize $\frac{1}{2}||\hat{x} - \tilde{x}||^2$;
  - This is a form of implicit regularization that ensures **smoothness**: it forces the system to represent well not only the data points, but also their perturbations;
- **Stacked auto-encoders** - several layers of auto-encoders stacked together;
- Variational auto-encoders - learn a **latent variable model** of the data;
  - They maximize the **evidence lower bound (ELBO)** of the **data likelihood** - GANs do not.

Regularized Auto-Encoders

- We need some sort of **regularization** to avoid **overfitting**;
- To regularize auto-encoders, **regularization** is added to the loss function;
- The goal is then to minimize $\mathcal{L}(\hat{x}, x) + \Omega(h, x)$;
- For example:
  - Regularizing the code: $\Omega(h, x) = \lambda ||h||^2$;
  - Regularizing the derivatives: $\Omega(h, x) = \lambda \sum_i ||\nabla_x h_i||^2$.

  One use of auto-encoders is **unsupervised pre-training** of **deep neural networks**.

Word Embeddings

- **Word embeddings** are a **representation** of a **word** in a **vector space - distributed representation**;
- **word2vec** is a **neural network** that learns **word embeddings - unsupervised** - considers a context window around each word in the sentence; it comes with two variantes:
  - **skip-gram**: given a word, predict the words around it;
  - **continuous bag-of-words (CBOW)**: given the words around a word, predict the word;
- **Global Vectors (GloVe)** is a **neural network** that learns **word embeddings - unsupervised** - considers the **co-occurrence matrix** of words in the corpus;
- **Contextualized word embeddings** are **word embeddings** that are **context-dependent - ELMo** and **BERT**.

Convolutional Neural Networks

- **Convolutional neural networks (CNNs)** are a class of **deep neural networks** that are **specialized** for **processing data** that has a **grid-like topology**, such as **images**;
- Equivalent to **translations** of the **input**;
- Convolutional and pooling layers exploit the fact that the same feature may appear in different **parts of the image**;
  - **Pooling layer** provides **invariance** - this means that the **output** is **invariant** to **small translations/shifts** of the input; They also make the representation **smaller** and **easier to process**; **Max pooling** is the most common, and offers **scaling, shifting and rotation invariance**;
- Lower layers of a CNN learn **local features** (e.g. edges), while higher layers learn **global features** (e.g. objects);
- **Convolution layers** are alternated with **pooling layers - convolution** is a **linear operation** that **preserves the grid-like topology** of the input;
- **Activation maps** are the **output** of a **convolutional layer**;

- Properties of CNNs:
  - Pooling layers are **Invariant** - the output is **invariant** to **small translations/shifts** of the input;
  - Convolution layers provide **translation and scale equivariance** but not **rotation equivariance** - the output features appear in the same relative positions and scale as the input features;
  - **Locality** - the output is **only affected** by a **small region** of the input;
  - **Sparse interactions** - each output value is the result of a **small number of interactions** with the input;
  - **Parameter sharing** - the **same parameters** are used for **different parts** of the input.

Size and Complexity

- Given an image of size $N \times N \times D$,
- A **Filter/kernel** is a **small matrix** that is **convolved** with the **input** to produce an **activation map** - $F \times F \times D$;
  - **Number of channels** is the **number of filters/kernels** - K;
- **Stride** is the **step size** of the **convolution** - S;
- **Padding** is the **number of zeros** added to the **input** - P;
  - A common padding size is $P = \frac{F-1}{2}$, which preserves the spatial size of the input $M = N$;
- The **output** of a **convolutional layer** is of size $M \times M \times K$, where:
  - $M = \frac{N-F+2P}{S} + 1$;
- **Number of trainable weights**: $NumberOfFilters \times (FilterHeight \times FilterWidth \times NumberOfChannels + Bias)$;
  - Other formula: $KernelSize^2 \times NumberOfInputChannels \times NumberOfOutputChannels$;
  - If we have a **FeedForward NN**, the number of parameters is: $NumberOfUnits \times ((InputWidth \times InputHeight \times InputChannels) + Bias)$;
- **Number of biases**: $NumberOfInputChannels$;
- **Number of units within a layer**: $M \times M \times K$;

  For linear layers, the number of parameters is $inputflatsize \times outputs + outputs$.

Residual Networks (ResNets)

- **Residual networks** are a class of **neural networks** that **skip connections** - tend to lead to more stable learning;
- Key motivation: **mitigate the vanishing gradient problem**;
- With $H(x) = \mathcal{F}(x) + \lambda x$, the gradient backpropagation becomes:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial L}{\partial H}\left(\frac{\partial \mathcal{F}}{\partial x} + \lambda\right)$$

A Residual Neural Network is a deep learning model in which the weight layers learn residual functions with reference to the layer inputs. A Residual Network is a network with skip connections that perform identity mappings, merged with the layer outputs by addition. This enables deep learning models with tens or hundreds of layers to train easily and approach better accuracy when going deeper.

They are used to allow gradients to flow through a network directly, without passing through non-linear activation functions. Non-linear activation functions, by nature of being non-linear, cause the gradients to explode or vanish (depending on the weights).

Recurrent Neural Networks

- RNNs allow to take advantage of **sequential data** - words in text, DNA sequences, sound waves, etc;
  - $h_t = g(Vx_t + Uh_{t-1} + c)$ - h_t is the **hidden state** at time t;
  - $\hat{y}_t = Wh_t + b$ - $\hat{y}_t$ is the **output** at time t;
- Used to generate, tag and classify sequences, and are trained using **back-propagation through time - BPTT**;
  - Parameters V, U, W, c and b are **shared** across **time steps** - **parameter sharing**;
  - **Exploding gradients** are a problem - **gradient clipping** is used to avoid this problem (truncating the gradient if it exceeds some magnitude);
  - **Vanishing gradients** are a problem - **LSTMs** and **GRUs** are used to avoid this problem;
    - More frequent and hard to deal with;
    - Vanishing gradients problem of RNNs: the **gradient** of the **loss function** with respect to the **parameters vanishes** as the **sequence length** increases;
    - GRUs solve this problem by **skipping** the **hidden state - gating mechanism**;
    - LSTMs solve this problem by using **memory cells** (propagated additively) and **gating functions** that control how much information is propagated from the previous state to the current and how much input influences the current state.
- They have **unbounded memory - recurrent connections** allow information to persist - however, for long sequences, they have a tendency to remember less accurately the initial words they have generated;
- **Bidirectional RNNs** combine **left-to-right RNN** (encoder) and **right-to-left RNN** (decoder) - **encoder-decoder** architecture;
  - Used as sequence encoders, and their main advantage is that each state contains **contextual information coming from both sides**;

  - Unlike **standard RNNs**, they have the same focus on the beginning and the end of the input sequence.

Applications:

- **Sequence generation** - generate a sequence of words - **auto-regressive models**;
  - Generating each word depends on all the previous words, but this doesn't scale well, we would need too many parameters. One solution is to use limited memory with an order-m Markov model (n-grams). Alternative: consider all the history, but compress it into a vector (this is what RNNs do). Sequence generataion RNNs are typically trained with maximum likelihood estimation (cross-entropy), so it intuitively measures how "perplexed/surprised" the model is.
  - We can also have an RNN over characters instead of words. Advantage: can generate any combination of characters, not just words in a closed vocabulary and much smaller set of output symbols; Disadvantage: need to remember deeper back in history;
  - **Teacher forcing** is a technique used to train **sequence generation** models - **feed the correct previous word** to the **decoder** at each step;
    * Causes **exposure bias** at run time: the model will have trouble recovering from mistakes early on, since it generates histories that it has never observed before.
- **Sequence tagging** - assign a label to each element in a sequence - use Bidirectional RNN;
- **Pooled classification** - classify a sequence as a whole;

  Standard RNNs suffer from vanishing and exploding gradients - alternative parameterizations like **LSTMs** and **GRUs** are used to avoid this problem;

- **Gated Recurrent Units (GRUs)** are a type of **recurrent neural network** that are **simpler** than **LSTMs** and **perform better** than **standard RNNs** - idea is to create some **shortcuts** in the **standard RNN**;
  - $u_t = \sigma(V_u x_t + U_u h_{t-1} + b_u)$ - **update gate**;
  - $r_t = \sigma(V_r x_t + U_r h_{t-1} + b_r)$ - **reset gate**;
  - $\tilde{h}_t = tanh(vx_t + U(r_t \odot h_{t-1}) + b)$ - **candidate hidden state**;
  - $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$ - **hidden state**;
- **Long Short-Term Memory (LSTM)** is a type of **recurrent neural network** that are **more complex** than **GRUs** and **perform better** than **standard RNNs** - idea is to use **memory cells** c_t to **store information**;
  - $i_t = \sigma(V_i x_t + U_i h_{t-1} + b_i)$ - **input gate**;
  - $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$ - **forget gate**;
  - $o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$ - **output gate**;
  - $\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b)$ - **candidate cell state**;
  - $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ - **cell state**;

  - $h_t = o_t \odot tanh(c_t)$ - **hidden state**.
  - A **Bidirectional LSTM (BiLSTM)** is a **recurrent neural network** that **processes** the **input sequence forward** and **backward** and **concatenates** the **outputs - encoder-decoder** architecture;
    * $h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t}$ - $\oplus$ is the **concatenation** operator;
    * $c_t = \overrightarrow{c_t} \oplus \overleftarrow{c_t}$ - $\oplus$ is the **concatenation** operator;
    * It is used for **sequence tagging** and **sequence classification**;
    * Better than **standard LSTMs** because they can **access future information**.

Sequence to Sequence Models

- **Sequence-to-sequence models** are a class of **neural networks** that are used to **map sequences** to **sequences - encoder-decoder** architecture;
  - Used for **machine translation**, **speech recognition**, **image captioning**, etc;
- A **Neural Machine Translation (NMT)** system is a **sequence-to-sequence model** that is used to **translate** a **sequence** in one **language** to a **sequence** in another **language - encoder-decoder** architecture;
  - **Encoder** RNN encodes source sentence into a **vector state** - $h_t = f(x_t, h_{t-1})$;
    * e.g. $h1 = relu(W_{hx}x_1 + W_{hh}h_0 + b_h)$, where $h_0 = 0$, W_{hx} is the **input weight matrix**, W_{hh} is the **hidden weight matrix** and b_h is the **bias vector**;
  - **Decoder** RNN decodes the **vector state** into a **target sentence** - $y_t = g(y_{t-1}, s_t)$;
    * e.g. $y_1 = argmax(W_{yh}h_1 + b_y)$, where W_{yh} is the **output weight matrix** and b_y is the **bias vector**;
- Representing the **input sequence** as a **single vector** is a **bottleneck - attention mechanisms** are used to **improve performance** - focus on different parts of the input;
- **Greedy decoding** is a **decoding strategy** that **greedily** picks the **most likely** output at each step;
- **Exposure bias** is the tendency of sequence-to-sequence models to be exposed only to correct target sequence prefixes at training time, and never to their own predictions. This makes them having trouble to recover from their own incorrect predictions at test time, if they are produced early on in the sequence. Exposure bias is caused by auto-regressive teacher forcing, where models are trained to maximize the probability of target sequences and are always assigned the previous target symbols as context;
- Sentences are **sorted** by **length** to **improve performance - batching**;
  - Within the same batch, all sequences must have the same length, and for this reason they must be padded with padding symbols for making them as long as the longest sentence in the batch. Since

in NLP sentences can have very different lengths, if we don't sort sentences by length, we can end up with very unbalanced batches, where some sentences are very short and others are very long, which makes it necessary to add a lot of padding symbols. This process is inefficient and can make training more time consuming. For this reason, sentences are usually sorted by length, which makes each batch more balanced.

RNN and Sequence-to-Sequence Models Exam Exercises

- Calculate $z_t = W_{hh}h_{t-1} + W_{hx}x_t + b_h$;
- Calculate $h_t = tanh(z_t)$;
- $LabelProbabilities = softmax(z_t)$;
- $CrossEntropyLoss = log(LabelProbabilities) \cdot y_t$;

Attention Mechanisms and Transformers

Translation model - models how words are translated - learn from parallel data. * **Encoder** - encodes the source sentence into a vector state; * **Decoder** - decodes the vector state into a target sentence; * **Beam search** is a **decoding strategy** that **keeps track** of the k **most likely** output sequences at each step - **greedy search/decoding** is a special case of **beam search** with $k = 1$;

Language model - models how words are generated - learn from monolingual data - learning this requires a large amount of data, and can be done with a markov model n-gram or a neural network.

- We want to **automatically weight** input relevance, to improve performance, reduce the number of parameters, faster training and inference (easy parallelization);
- Encoders/decoders can be RNNs, CNNs or **self-attention layers**;
- **Self-attention** is a **linear operation** that **maps** a **sequence** of **vectors** to a **sequence** of **vectors** - **encoder-decoder** architecture;
  - **Query** vector q_t - $Q = XW_q$ - W_q is the **query weight matrix**;
  - **Key** vectors $k_1, k_2, ..., k_n$ - $K = XW_k$ - W_k is the **key weight matrix**;
  - **Value** vectors $v_1, v_2, ..., v_n$ - $V = XW_v$ - W_v is the **value weight matrix**;
  - **Attention weights** $P = softmax(\frac{QK^T}{\sqrt{d_k}})$ - d_k is the **dimensionality** of the **query** and **key** vectors;
  - **Output** vector $Z = PV$;
  - Or it can be written without matrix multiplication:
    - $e_{i,j} = \frac{x_i^T x_j}{\sqrt{d_k}}$ - **score**;

        * $\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum_{j=1}^{n} exp(e_{i,j})}$ - **attention weights**;
        * $z_i = \sum_{j=1}^{n} \alpha_{i,j} v_j$ - **output**;

  Self-attention encoders are better than RNN encoders, since they can better capture long-range relations between elements of a sequence, since information does not propagate sequentially, but in parallel. This makes them more efficient and easier to parallelize - faster training and inference.

- **Transformers**: **encoder-decoder** architecture with **self-attention** layers instead of **RNNs**;
    - Uses **scaled dot-product attention** and **multi-head attention**;
    - **Encoder**: **self-attention** layers;
    - **Decoder**: **self-attention** layers (**masked needed**) + **encoder-decoder attention** layers;
        * They need **causal masking** at training time to avoid **cheating - mask** the **future** - reproduce test time conditions;
        * The difference between self-attention and masked self-attention is that in the latter the **attention weights** are **masked** to **avoid cheating - mask** the **future**;
    - *The self-attention in transformers allows any word to attend to any other word, both in the source and on the target. When the model is generating a sequence left-to-right it cannot attend at future words, which have not been generated yet. At training time, causal masking is needed in the decoder self-attention to mask future words, to reproduce test time conditions.*
- **Scaled dot-product attention** is a **self-attention** layer with **multiple heads**:
    - **Scores** given by $z = \frac{Xq}{\sqrt{d_k}}$; d_k is the **dimensionality** of the **query** and **key** vectors;
    - Apply **softmax** to get **attention probabilities**: $\alpha = softmax(z)$;
    - The **output** vector is $c = X^T \alpha$;
- **Multi-head attention** is a **self-attention** layer with **multiple heads - parallel** self-attention layers;
    - **Query** vectors q_t - $Q = XW_q$ - W_q is the **query weight matrix**;
    - **Key** vectors $k_1, k_2, \ldots, k_n$ - $K = XW_k$ - W_k is the **key weight matrix**;
    - **Value** vectors $v_1, v_2, \ldots, v_n$ - $V = XW_v$ - W_v is the **value weight matrix**;
    - **Attention weights** $P = softmax(\frac{QK^T}{\sqrt{d_k}})$ - d_k is the **dimensionality** of the **query** and **key** vectors;
    - **Output** vector $Z = PV$;
    - Or it can be written without matrix multiplication:
        * $e_{i,j}^{(h)} = \frac{(W_q^{(h)} x_i)^T (W_k^{(h)} x_j)}{\sqrt{d_k}}$ - **score**;

    - $\alpha_{i,j}^{(h)} = \frac{exp(e_{i,j}^{(h)})}{\sum_{j=1}^{n} exp(e_{i,j}^{(h)})}$ - **attention weights**;
    - $z_i^{(h)} = \sum_{j=1}^{n} \alpha_{i,j}^{(h)}(W_v^{(h)}x_j)$ - **output**;
    - $z_i = concat(z_i^{(1)}, z_i^{(2)}, \ldots, z_i^{(H)})W_o$ - **output**;
- **Positional encoding** is a technique used to **encode the position** of each **word** in a **sequence**;
  - Without positional encodings, the self-attention in transformers is insensitive to the word positions being queried: permuting the words leads to a similar permutation in the self-attention responses. In order for transformers to be sensitive to the word order, each word embedding is augmented with a positional embedding

Self-attention vs. Multi-head attention

- **Self-attention** is a **linear operation** that **maps** a **sequence** of **vectors** to a **sequence** of **vectors - encoder-decoder** architecture;
- **Multi-head attention** is a **self-attention** layer with **multiple heads - parallel** self-attention layers;
- **Scaled dot-product attention** is a **self-attention** layer with **multiple heads**.

Computational Cost

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-attention	$O(n^2 d_k)$	$O(1)$	$O(1)$
Recurrent	$O(nd^2)$	$O(n)$	$O(n)$
Convolutional	$O(k^2 d)$	$O(1)$	$O(log_k n)$
Self-attention restricted	$O(rnd_k)$	$O(1)$	$O(n/r)$

Self-Supervised Learning and Large Pretrained Models

- **Contextualized representations** are **embeddings** that **depend on the context**;
- Words can have different meanings depending on the context;
- **ELMo** is a model that learned **context-dependent embeddings**;
  - **E**mbeddings from **L**anguage **Mo**dels;

  - **ELMo** is a **bidirectional LSTM** model - BiLSTM;
  - Save all parameters at all layers;
  - Then, for your downstream task, tune a scalar parameter for each layer. and pass the entire sentence through this encoder.
- Pretraining large models and fine-tuning them to a specific task is a common practice in deep learning:
  - **Pretraining** is a technique used to **initialize** the **parameters** of a **neural network - self-supervised learning**;
  - **Fine-tuning** is a technique used to **adapt** the **parameters** of a **neural network** to a **specific task**;
    * **Limitations**: fine-tuning a large model to several tasks can be very expensive and requires a copy of the model for each task;
- **Dangers of large pretrained models**:
  - For many existing models, data was not properly curated or representative of the world's population;
  - Current models are English-centric; other languages are poorly represented;
  - They may propagate biases and discriminate against minorities;
  - They may disclose private information (maybe some private information was in the training data, and models can expose it);
  - Their output is uncontrolled – it can be toxic or offensive;
  - They can provide misleading information with unpredictable consequences;
- Models: ELMo, BERT, GPT, etc;
  - **GPT-3** - decoder-only transformer;
    * **Few-shot learning** - can be trained with **few examples**;
    * ChatGPT is a **chatbot** based on GPT-3;
    * Pretrained with causal language modeling;
  - **BERT (Bidirectional Encoder Representations from Transformers)** - encoder-only transformer, learn contextyualized word representations;
    * Pretrained on **masked language modeling** and **next sentence prediction**;
  - **T5** - encoder-decoder transformer;
    * Span corruption as an auxiliary task (replace a span of text with a mask token and train to predict the original span);
    * Suitable for classification and generation tasks;
- **Adapters** and **prompting** are other strategies more parameter-efficient than fine-tuning;
  - Adapters are small modules that are plugged into a pretrained model and trained on a specific task; their advantage over fine-tuning is that they require fewer parameters and less training time;
  - Prompting is usually done with models such as GPT, which are trained with a causal language modeling objective;
- Current models exhibit **few-shot learning** capabilities - can be trained with **few examples**.

Deep Generative Models

...

Derivatives and Gradients

- **Derivatives** are a **measure of how a function changes** when its **inputs change**;
- **Gradients** are a **generalization of derivatives** to **multiple dimensions - vector of partial derivatives**;
- **Partial derivatives** are the **derivatives** of a **function** with **respect to one variable**, while **holding the other variables constant**.

Derivative Rules

- $\frac{d}{dx}c = 0$;
- $\frac{d}{dx}x^n = nx^{n-1}$;
- $\frac{d}{dx}e^x = e^x$;
- $\frac{d}{dx}log(x) = \frac{1}{x}$;
- $\frac{d}{dx}sin(x) = cos(x)$;
- $\frac{d}{dx}cos(x) = -sin(x)$;
- $\frac{d}{dx}f(g(x)) = f'(g(x))g'(x)$;
- $\frac{d}{dx}f(x)g(x) = f'(x)g(x) + f(x)g'(x)$;
- $\frac{d}{dx}\frac{f(x)}{g(x)} = \frac{f'(x)g(x)-f(x)g'(x)}{g(x)^2}$;
- $\frac{d}{dx}\sum_{i=1}^{n} f_i(x) = \sum_{i=1}^{n} \frac{d}{dx}f_i(x)$;
- $\frac{d}{dx}\prod_{i=1}^{n} f_i(x) = \sum_{i=1}^{n} \frac{d}{dx}f_i(x)\prod_{j\neq i} f_j(x)$.

Chain Rule

- The **chain rule** is a **formula** for computing the **derivative** of the **composition** of **two or more functions**;
- If $y = f(u)$ and $u = g(x)$, then $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$.

Squared Loss Derivative

$$\nabla_w L(y, \hat{y}) = \nabla_w \frac{1}{2}(y-\hat{y})^2 = \frac{1}{2}\cdot 2(y-\hat{y})\nabla_w(y-\hat{y}) = (y-\hat{y})\nabla_w(y-w^T x) = (y-\hat{y})(\nabla_w y-\nabla_w w^T x) = (y-\hat{y})(-x) =$$

Cross-Entropy Loss Derivative

The sum was omitted, because it doesn't change the process of taking the derivative (the derivative is a linear transformation). A sigmoid activation function is assumed here:

$\nabla_W ylog(\sigma(W^Tx)) + \nabla_W(1-y)log(1-\sigma(W^Tx)) = \; = y\nabla_W\sigma(W^Tx)\sigma(W^Tx) + (1-y)\nabla_W(1-\sigma(W^Tx))1-\sigma(W^Tx)$

Gradient

- The **gradient** of a **scalar function** $f : \mathbb{R}^n \to \mathbb{R}$ is the **vector of partial derivatives** of f with respect to each of its **input variables**;
- $\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$;
- Rules:
- $\nabla_x c = 0$;
- $\nabla_x x^T Ax = (A + A^T)x$;
- If A is **symmetric**, then $\nabla x^T Ax = 2Ax$;
- Particular case: $f(x) = x^T x = ||x||^2$, then $\nabla f(x) = 2x$.
- If $f(x) = x^T b = b^T x$, then $\nabla f(x) = b$.
- If $g(x) = f(Ax), then \nabla g(x) = A^T \nabla f(Ax)$.
- If $g(x) = f(a \cdot x)$, then $\nabla g(x) = a \cdot \nabla f(a \cdot x)$.

Algorithmic Complexities

$M_1 \in \mathbb{R}^{N\times D}$ and $M_2 \in \mathbb{R}^{D\times M}$, so $M_1 M_2 \in \mathbb{R}^{N\times M}$, has time complexity $O(NDM)$.

Inner Product or **Dot Product** is a matrix multiplication when vectors are expanded so that $M_1 \in \mathbb{R}^{1\times D}$ and $M_2 \in \mathbb{R}^{D\times 1}$.

Outer product is a matrix multiplication when vectors are expanded so that $M_1 \in \mathbb{R}^{N\times 1}$ and $M_2 \in \mathbb{R}^{1\times M}$.

$M^\top, M \in \mathbb{R}^{N\times D}$ has time complexity $O(ND)$.

$diag(M), M \in \mathbb{R}^{N\times M}$ has time complexity $O(min(N, M))$ for diagonal extraction and $O(NM)$ for diagonal matrix creation.

Hadamard product = **elementwise multiplication** = $M \odot N, M, N \in \mathbb{R}^{N\times M}$ has time complexity $O(NM)$.

All other elementwise operations, such as $exp(M)$ softmax(M), etc.. for $M \in \mathbb{R}^{N\times M}$ have time complexity $O(NM)$.

Deep Learning
MSc in Computer Science and Engineering
MSc in Electrical and Computer Engineering

Final exam — February 12, 2022

Version A

Instructions

- You have 120 minutes to complete the exam.
- Make sure that your test has a total of 10 pages and is not missing any sheets, then write your full name and student n. on this page (and your number in all others).
- The test has a total of 19 questions, with a maximum score of 100 points. The questions have different levels of difficulty. The point value of each question is provided next to the question number.
- Please provide your answer in the space below each question. If you make a mess, clearly indicate your answer.
- The exam is open book and open notes. You may use a calculator, but any other type of electronic or communication equipment is not allowed.
- Good luck.

Part 1	**Part 2**	**Part 3, Pr. 1**	**Part 3, Pr. 2**	**Total**
32 points	18 points	25 points	25 points	100 points

Part 1: Multiple Choice Questions (32 points)

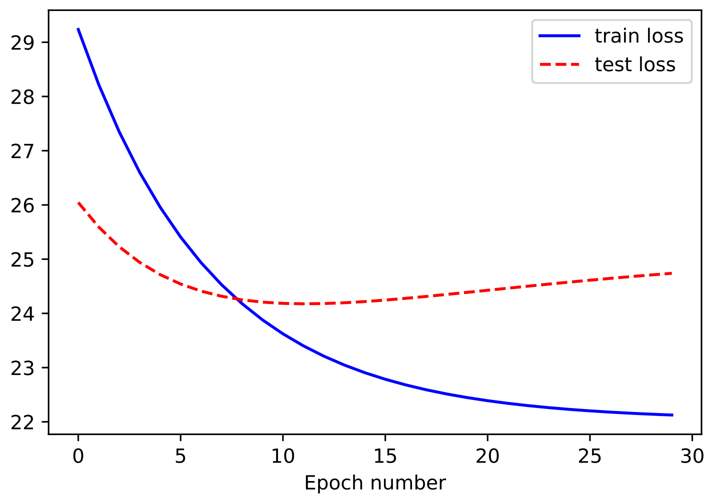
In each of the following questions, indicate your answer by *checking a single option.*

1. (4 points) An RNN-based sequence-to-sequence model with an attention mechanism translates an input sentence of M words into an output sentence with N words. How does the number of computational operations (algorithmic complexity) increase as a function of M and N?

   - ☐ $O(M + N)$
   - ■ $O(MN)$
   - ☐ $O(\max(M, N)^2)$
   - ☐ $O(M^N)$

   Solution: The correct option is $O(MN)$, since for each of the N generated words we need to attend to M representations for the source words.

2. (4 points) A model is trained for 30 epochs with gradient descent and it leads to the following plot for its training and test losses:



   Which of the following statements is a plausible explanation for what could be happening?

   - ☐ The model is underfitting the training data.
   - ■ **The model is overfitting the training data.**
   - ☐ The model generalizes well to unseen examples.
   - ☐ None the above.

   Solution: The training error is decreasing, while the test error is increasing, which suggests that the model is overfitting the training data.

3. (4 points) A neural network is overfitting its training data. What strategies could mitigate this?

   - ■ **Increase the dropout probability.**

- ☐ Decrease the amount of training data.
- ☐ Increase the number of hidden units.
- ☐ All the above.

Solution: More regularization should help, and this can be achieved by increasing the dropout probability.

4. (4 points) Let $\vee, \wedge, \oplus$ denote respectively the OR, AND, and XOR Boolean logical operators, and $\neg$ denote Boolean negation. Assume Boolean values are represented as -1 (False) and $+1$ (True). Which of these logical functions cannot be learned by a single perceptron with inputs A and B?

   - ■ $(A \wedge \neg B) \vee (\neg A \wedge B)$
   - □ $(A \oplus B) \wedge A$
   - □ $A \vee B$
   - □ $\neg A \wedge B$

   Solution: The answer is $(A \wedge \neg B) \vee (\neg A \wedge B)$, which is equal to $A \oplus B$.

5. (4 points) Let $L(\boldsymbol{w}) = \frac{1}{2}\sum_i (y_i - \boldsymbol{w}^\top \boldsymbol{\phi}(x_i))^2$ be the loss function corresponding to a linear regression problem. Which equation represents the stochastic gradient descent update for $\boldsymbol{w}$?

   - ■ $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta(y_i - \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_i))\boldsymbol{\phi}(x_i)$
   - □ $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta \sum_i (y_i - \boldsymbol{w}^\top \boldsymbol{\phi}(x_i))\boldsymbol{\phi}(x_i)$
   - □ $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta(y_i - \mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{\phi}(x_i)))\boldsymbol{\phi}(x_i)$, where $\mathrm{sign}(\cdot)$ is the sign function
   - □ $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta(y_i - \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}(x_i)))\boldsymbol{\phi}(x_i)$, where $\sigma(z) = 1/(1+e^{-z})$ is the sigmoid function.

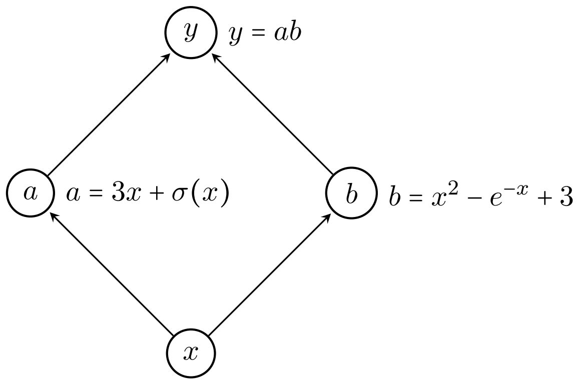
   Solution: Stochastic gradient updates depend only on a single example (or a mini-batch of examples). The option $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta \sum_i (y_i - \boldsymbol{w}^\top \boldsymbol{\phi}(x_i))\boldsymbol{\phi}(x_i)$ corresponds to gradient descent on the full batch.

6. (4 points) Which one of the following statements is true?

   - □ Convolutional layers are equivariant to translations and rotations.
   - □ Neural networks with a single hidden layer with linear activations are universal approximators.
   - □ Auto-encoders with non-linear activations and a squared loss are equivalent to PCA.
   - ■ **None of the above.**

   Solution: Convolutional layers are equivariant to translations, but not rotations. Neural networks with a single hidden layer with **linear** activations are equivalent to linear classifiers, which are not universal approximators. Auto-encoders with **linear** activations would correspond to PCA.

7. (4 points) Consider the following computation graph, where $\sigma(z) = 1/(1+e^{-z})$ is the sigmoid function. What is the derivative of y with respect to x?



- ■ **$b(3 + \sigma(x)(1 - \sigma(x))) + a(2x + e^{-x})$.**
- □ $a(3 + \sigma(x)(1 - \sigma(x))) + b(2x + e^{-x})$.
- □ $3 + \sigma(x)(1 - \sigma(x)) + 2x + e^{-x}$.
- □ 0.

Solution: It is $b(3 + \sigma(x)(1 - \sigma(x))) + a(2x + e^{-x})$:

$$\frac{\partial y}{\partial a} = b, \quad \frac{\partial y}{\partial b} = a$$
$$\frac{\partial a}{\partial x} = 3 + \sigma(x)(1 - \sigma(x)), \quad \frac{\partial b}{\partial x} = 2x + e^{-x}$$
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial y}{\partial b}\frac{\partial b}{\partial x} = b(3 + \sigma(x)(1 - \sigma(x))) + a(2x + e^{-x}).$$

8. (4 points) Which one of the following statements is **false**?
   - ■ **Gradient clipping can prevent vanishing gradients.**
   - □ Transformer models can be used for computer vision applications.
   - □ Distributed representations generally require fewer dimensions than local (one-hot) representations.
   - □ Upper level layers (closer to the output) tend to learn more abstract representations (shapes, forms, objects) compared to bottom level layers.

   Solution: Gradient clipping can prevent exploding gradients, not vanishing gradients.

Part 2: Short Answer Questions (18 points)

Please provide **brief** answers (1-2 sentences) to the following questions.

1. (6 points) Explain how dropout regularization works.

   Solution: At training time, for each example neurons are dropped randomly with probability p (i.e. their activations are masked to become zero) and the remaining activations are scaled by $1/(1-p)$. This forces each neuron to depend less on other neurons' activations.

2. (6 points) Explain the role and need for positional encoding in transformers.

Solution: Without positional encodings, the self-attention in transformers is insensitive to the word positions being queried: permuting the words leads to a similar permutation in the self-attention responses. In order for transformers to be sensitive to the word order, each word embedding is augmented with a positional embedding.

3. (6 points) Mention one advantage of contextualized word embeddings (e.g. BERT) over static word embeddings (e.g. word2vec or GloVe).

   Solution: Contextualized word embeddings can assign different representations to the same word being used in different contexts; this is particularly useful for polysemic words (such as "bank" which can be a river bank or a financial institution).

Part 3: Problems (50 points)

Problem 1: Convolutional Neural Networks (25 points)

In their retail store, Yolanda and Zach currently use a card punching system to register the entry and exit times of their 6 employees. However, they heard about recent advances in computer vision systems and decided to replace that system by face recognition using CNNs.

To train the system, they collected a large dataset of pictures from their 6 employees, Alice, Berta, Chad, Diane, Eric, and Frank. Each picture in the dataset is a 192×256 grayscale picture, similar to those depicted in Fig. 1, and is labeled according to the corresponding employee.

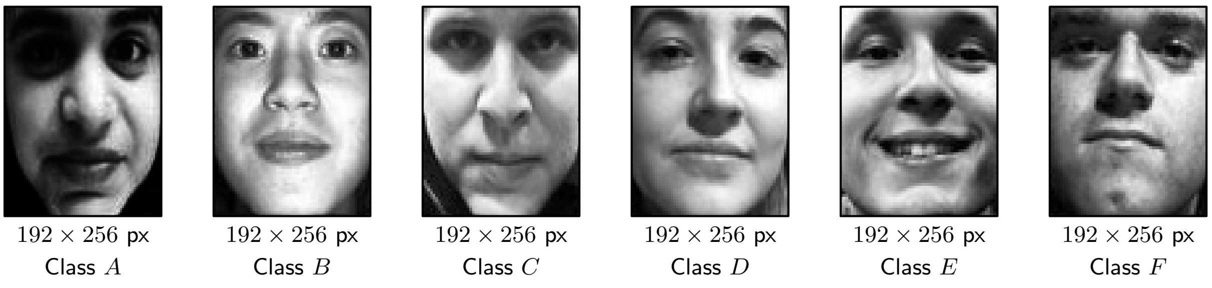
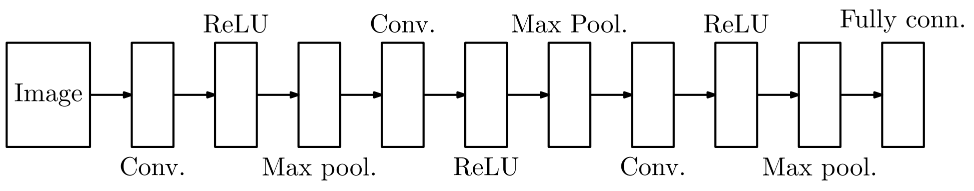


Figure 1: Sample pictures from the 6 classes that the CNN must recognize. Alice corresponds to class A, Berta to class B, etc.

1. (4 points) Briefly explain in 1-2 sentences why a CNN is an adequate choice of architecture for Yolanda and Zach's task (image classification).

   Solution: CNNs take advantage of the spacial structure of the image, unlike standard feed-forward networks. Moreover, convolutional and pooling layers exploit the fact that the same feature may appear in different parts of the image, enabling the network to process those occurrences in a similarly way.

2. (7 points) Suppose that, in their classifier, their use the following architecture:



which is specified using the following Pytorch code snippet:

```
nn.Sequential(
    nn.Conv2d(1, 5, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(5, 10, kernel_size=5, stride=1, padding=0),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(10, 20, kernel_size=2, stride=2, padding=0),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=5, stride=2),
    nn.Flatten(),
    nn.Linear(2800, 6))
```

Fill in the following table with the adequate values.

Layer	**Output size**	**N. weights**	**N. biases**
Input	$192 \times 256 \times 1$	0	0
1st conv. layer	$192 \times 256 \times 5$	45	5
1st pooling layer	$96 \times 128 \times 5$	0	0
2nd conv. layer	$92 \times 124 \times 10$	1250	10
2nd pooling layer	$46 \times 62 \times 10$	0	0
3rd conv. layer	$23 \times 31 \times 20$	800	20
3rd pooling layer	$10 \times 14 \times 20$	0	0
Output layer	6×1	$16,800$	6

3. (7 points) Consider the diagram in Fig. 2, containing the brightness values for the first window of pixels in one of the images in the dataset.

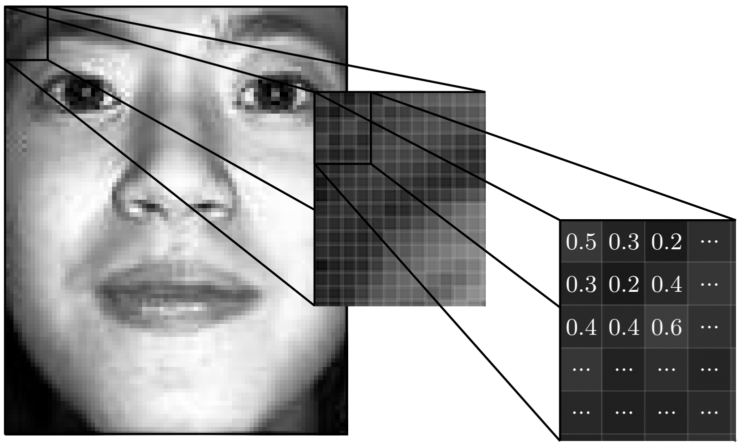


Figure 2: Brightness values for one of the images in the dataset.

Suppose that, after training, one of the filters in the first convolutional layer is defined by the parameters

$$\boldsymbol{K} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad\qquad b = 0.5.$$

For the filter provided, compute the the top-left-most value after the pooling layer. Do not forget that the first convolutional layer includes a padding of size 1 (use zeros as the padding value).

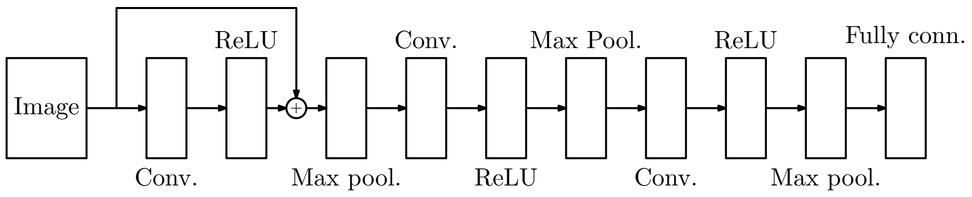
Solution: The input of the max-pool layer corresponding to the provided pixels is given by

$$\begin{aligned}\boldsymbol{h}_{\text{conv}} &= \text{ReLU}\left(\begin{bmatrix} -0.5 & 0.2 \\ -0.9 & 0.0 \end{bmatrix} + 0.5\right) \\ &= \begin{bmatrix} 0.0 & 0.7 \\ 0.0 & 0.5 \end{bmatrix}.\end{aligned}$$

At the output of the pooling layer, we thus have

$$h_{\text{pool}} = 0.7.$$

4. (7 points) Suppose that Yolanda and Zach decide to add some skip connections to their network, as indicated in the diagram:



Repeat Question 3, but now considering the skip connection indicated in the diagram above.

Solution: The input of the first max-pool layer corresponding to the provided pixels is given by

$$\begin{aligned}\boldsymbol{h}_{\text{conv}} &= \text{ReLU}\left(\begin{bmatrix} -0.5 & 0.2 \\ -0.9 & 0.0 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.3 \\ 0.3 & 0.2 \end{bmatrix} + 0.5\right) \\ &= \begin{bmatrix} 0.5 & 1.0 \\ 0.0 & 0.7 \end{bmatrix}.\end{aligned}$$

At the output of the pooling layer, we thus have

$$h_{\text{pool}} = 1.0.$$

Problem 2: Sequence-to-Sequence Models (25 points)

Bartholomew (known to his friends as Bart) had an idea for a project: building a system to summarize news articles into a short sentence (e.g., a tweet). He collected a dataset with news documents and their corresponding tweets, which he will use to train a summarization model.

1. (7 points) Bart's sister (Lisa) is taking a course on *deep learning* and she recommended using a sequence-to-sequence architecture based on a *recurrent neural network* (RNN) for this problem. Bart tested a simple RNN-based sequence-to-sequence model (without any attention mechanism) on a small-scale experiment. He is using a very small vocabulary (7 words, including the <STOP> symbol), shared between the source and target, and using the same embedding vectors for both sides. The embedding vectors are

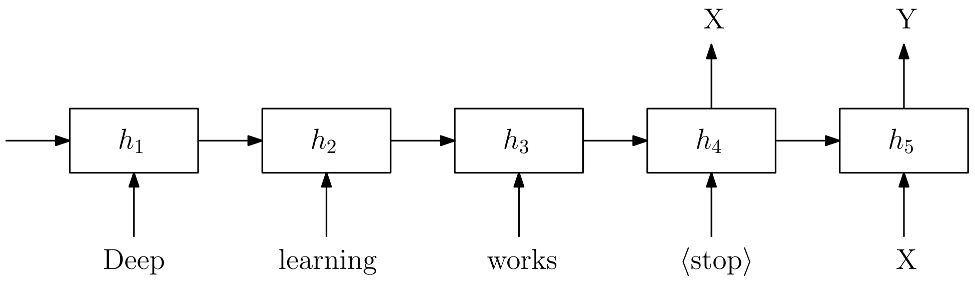
$$\boldsymbol{x}_{\text{Deep}} = [0,1]^\top, \quad \boldsymbol{x}_{\text{learning}} = [1,0]^\top, \quad \boldsymbol{x}_{\text{works}} = [-1,-1]^\top,$$
$$\boldsymbol{x}_{\text{!!!}} = [2,-1]^\top, \quad \boldsymbol{x}_{\text{\#deep}} = [-1,2]^\top, \quad \boldsymbol{x}_{\text{lol}} = [0,-1]^\top, \quad \boldsymbol{x}_{\text{<stop>}} = [1,1]^\top.$$

The initial hidden state of the RNN, $\boldsymbol{h}_0$, is all-zeros. The input-to-hidden matrix is

$$\boldsymbol{W}_{hx} = \begin{bmatrix} 0 & -1 \\ 2 & 0 \\ 1 & 1 \end{bmatrix}.$$

The recurrent matrix $\boldsymbol{W}_{hh}$ is the identity matrix. All biases are vectors of zeros. The RNN uses `relu` activations.

Compute the last state of the encoder RNN, $\boldsymbol{h}_4$, for the input document "Deep learning works". Show all your calculations.



Solution: We have:

$$\begin{aligned}
\boldsymbol{h}_1 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{Deep}} + \boldsymbol{W}_{hh}\boldsymbol{h}_0) \\
&= \text{relu}([-1,0,1]^\top + [0,0,0]^\top) \\
&= [0,0,1]^\top. \\
\boldsymbol{h}_2 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{learning}} + \boldsymbol{W}_{hh}\boldsymbol{h}_1) \\
&= \text{relu}([0,2,1]^\top + [0,0,1]^\top) \\
&= [0,2,2]^\top. \\
\boldsymbol{h}_3 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{works}} + \boldsymbol{W}_{hh}\boldsymbol{h}_2) \\
&= \text{relu}([1,-2,-2]^\top + [0,2,2]^\top) \\
&= [1,0,0]^\top. \\
\boldsymbol{h}_4 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{<stop>}} + \boldsymbol{W}_{hh}\boldsymbol{h}_3) \\
&= \text{relu}([-1,2,2]^\top + [1,0,0]^\top) \\
&= [0,2,2]^\top.
\end{aligned}$$

Therefore the last state is $\boldsymbol{h}_4 = [0,2,2]^\top$.

2. (8 points) Assume that in the previous question we obtained $\boldsymbol{h}_4 = [0, 2, 2]^\top$. Assume that the decoder RNN has the same parameters as the encoder RNN, and the hidden-to-output matrix is

$$\boldsymbol{W}_{yh} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -2 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \\ -2 & 0 & 1 \end{bmatrix}.$$

The target word probabilities at time step t are given by $\mathrm{softmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_t)$, where $\boldsymbol{h}_t$ is the corresponding state of the decoder RNN.

Compute the first two words of the generated tweet using **greedy decoding**.

Solution: We have:

$$\begin{aligned} \boldsymbol{y}_1 &= \mathrm{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_4) \\ &= \mathrm{argmax}([-2, 0, 2, -2, 4, 0, 2]) = \#\mathrm{deep}. \\ \boldsymbol{h}_5 &= \mathrm{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\#\mathrm{deep}} + \boldsymbol{W}_{hh}\boldsymbol{h}_4) \\ &= \mathrm{relu}([-2, -2, 1]^\top + [0, 2, 2]^\top) \\ &= [0, 0, 3]^\top. \\ \boldsymbol{y}_2 &= \mathrm{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_5) \\ &= \mathrm{argmax}([0, -3, 0, -3, 0, 0, 3]) = <\mathrm{stop}>. \end{aligned}$$

The generated words are "#deep <stop>".

3. (6 points) After playing with this network for a while, Bart realized that it didn't work well for long documents and therefore decided to add an attention mechanism. In the first decoding step, using **scaled dot-product attention** with $\boldsymbol{h}_4$ as the query vector and $\boldsymbol{h}_1$, $\boldsymbol{h}_2$, $\boldsymbol{h}_3$ as the key and value vectors, compute the attention probabilities and the resulting context vector (use $\boldsymbol{h}_1 = [0, 0, 1]^\top$, $\boldsymbol{h}_2 = [0, 2, 2]^\top$, $\boldsymbol{h}_3 = [1, 0, 0]^\top$, $\boldsymbol{h}_4 = [0, 2, 2]^\top$).

Solution: We have:

$$\begin{aligned} s_1 &= \frac{1}{\sqrt{3}}[0, 2, 2]^\top[0, 0, 1] = \frac{2}{\sqrt{3}} \\ s_2 &= \frac{1}{\sqrt{3}}[0, 2, 2]^\top[0, 2, 2] = \frac{8}{\sqrt{3}} \\ s_3 &= \frac{1}{\sqrt{3}}[0, 2, 2]^\top[1, 0, 0] = 0 \\ Z &= \exp\left(\frac{2}{\sqrt{3}}\right) + \exp\left(\frac{8}{\sqrt{3}}\right) + \exp(0) = 105.546 \\ \boldsymbol{p} &= \mathrm{softmax}([s1, s2, s3]) = \exp([s_1, s_2, s_3])/Z = [.030, .960, .009] \\ \boldsymbol{c} &= \boldsymbol{h}_1 p_1 + \boldsymbol{h}_2 p_2 + \boldsymbol{h}_3 p_3 = [.009, 1.921, 1.951]. \end{aligned}$$

4. (4 points) Lisa's friend, Allison, who is also knowledgeable about deep learning, told Bart about transformers and large pretrained models. Give one example of a pretrained model that Bart could use for this task and the necessary steps to use it.

   Solution: Bart could use a pretrained decoder-only (e.g., GPT) or encoder-decoder model (e.g., T5, BART) and fine-tune it on the data he has available. Alternatively he could use the model without any fine-tuning and use prompting at test time. A possible prompt would be "<document> TL;DR: <answer>". Note: an encoder-only model (e.g. BERT) would not be suitable, since this an auto-regressive generation task.

Deep Learning

MSc in Computer Science and Engineering

MSc in Electrical and Computer Engineering

Final exam — January 25, 2023

Version A

Instructions

- You have 120 minutes to complete the exam.
- Make sure that your test has a total of 14 pages and is not missing any sheets, then write your full name and student n. on this page (and your number in all others).
- The test has a total of 17 questions, with a maximum score of 100 points. The questions have different levels of difficulty. The point value of each question is provided next to the question number.
- Please provide your answer in the space below each question. If you make a mess, clearly indicate your answer.
- The exam is open book and open notes. You may use a calculator, but any other type of electronic or communication equipment is not allowed.
- Good luck.

Part 1	**Part 2**	**Part 3, Pr. 1**	**Part 3, Pr. 2**	**Total**
32 points	18 points	25 points	25 points	100 points

Part 1: Multiple Choice Questions (32 points)

In each of the following questions, indicate your answer by *checking a single option.*

1. (4 points) The softplus function is defined as $\mathrm{softplus}(t) = \log(1 + \exp(t))$. The logistic function is $\sigma(t) = 1/(1 + \exp(-t))$. Which of the following options is the derivative of softplus at t?

   - ☐ $\sigma(t)(1 - \sigma(t))$
   - ■ $\sigma(t)$
   - ☐ $\sigma(-t)$
   - ☐ $\mathrm{softplus}(t)(1 - \mathrm{softplus}(t))$

   Solution: The derivative of softplus is the logistic function, $\sigma(t) = 1/(1 + \exp(-t))$.

2. (4 points) Which of the following pairs of functions always give the same result when commuted?

   - ■ **ReLU activations and max-pooling layer.**
   - ☐ Convolutional layer and max-pooling layer.
   - ☐ Convolutional layer and ReLU activations.
   - ☐ None of the above.

   Solution: Applying ReLU activations followed by a max-pooling layer gives the same result as a max-pooling layer followed by ReLU activations.

3. (4 points) Let $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})_{i=1}^N\}$ be a linearly separable dataset for binary classification, and let $\mathcal{D}' = \{(2\boldsymbol{x}^{(i)}, y^{(i)})_{i=1}^N\}$ be a scaled version of the same dataset where the input vectors are scaled by a factor of 2. Consider the perceptron algorithm without a bias parameter and where all the weights are initialized to zero. On which of the datasets the perceptron will make more mistakes until it converges?

   - ☐ The number of mistakes can be larger for $\mathcal{D}$ than for $\mathcal{D}'$.
   - ☐ The number of mistakes can be smaller for $\mathcal{D}$ than for $\mathcal{D}'$.
   - ■ **The number of mistakes must be the same for both datasets.**
   - ☐ There can be infinitely many mistakes for both datasets.

   Solution: For $\mathcal{D}$, the updates will be $\boldsymbol{w} \leftarrow \boldsymbol{w} + y^{(i)}\boldsymbol{x}^{(i)}$. For $\mathcal{D}'$, the updates will be $\boldsymbol{w} \leftarrow \boldsymbol{w} + 2y^{(i)}\boldsymbol{x}^{(i)}$. Since $\boldsymbol{w}$ is initialized to 0, the mistakes will be exactly the same for both datasets and the final $\boldsymbol{w}$ for $\mathcal{D}'$ will be twice the final $\boldsymbol{w}$ for $\mathcal{D}$.

4. (4 points) Assume a transformer with two attention heads, one with projection matrices $\boldsymbol{W}_Q^{(1)}$, $\boldsymbol{W}_K^{(1)}$, $\boldsymbol{W}_V^{(1)}$ and another with projection matrices $\boldsymbol{W}_Q^{(2)}$, $\boldsymbol{W}_K^{(2)}$, $\boldsymbol{W}_V^{(2)}$ where $\boldsymbol{W}_Q^{(2)} = -\boldsymbol{W}_Q^{(1)}$, $\boldsymbol{W}_K^{(2)} = -\boldsymbol{W}_K^{(1)}$, and $\boldsymbol{W}_V^{(2)} = -\boldsymbol{W}_V^{(1)}$. Let $\boldsymbol{P}^{(1)}$ and $\boldsymbol{P}^{(2)}$ be the attention probability matrices and $\boldsymbol{Z}^{(1)}$ and $\boldsymbol{Z}^{(2)}$ be the context representations obtained by the two attention heads. What is the relation between $\boldsymbol{P}^{(1)}$ and $\boldsymbol{P}^{(2)}$ and between $\boldsymbol{Z}^{(1)}$ and $\boldsymbol{Z}^{(2)}$?

   - ☐ $\boldsymbol{P}^{(1)} = \boldsymbol{P}^{(2)}$ and $\boldsymbol{Z}^{(1)} = \boldsymbol{Z}^{(2)}$.
   - ☐ $\boldsymbol{P}^{(1)} = -\boldsymbol{P}^{(2)}$ and $\boldsymbol{Z}^{(1)} = \boldsymbol{Z}^{(2)}$.

■ $\boldsymbol{P}^{(1)} = \boldsymbol{P}^{(2)}$ **and** $\boldsymbol{Z}^{(1)} = -\boldsymbol{Z}^{(2)}$**.**

□ None of the above.

Solution: We have $\boldsymbol{P}^{(2)} = \text{Softmax}(\boldsymbol{Q}^{(2)}(\boldsymbol{K}^{(2)})^\top/\sqrt{K}) = \text{Softmax}(-\boldsymbol{Q}^{(1)}(-\boldsymbol{K}^{(1)})^\top/\sqrt{K}) = \text{Softmax}(\boldsymbol{Q}^{(1)}(\boldsymbol{K}^{(1)})^\top/\sqrt{K}) = \boldsymbol{P}^{(1)}$. We have $\boldsymbol{Z}^{(2)} = \boldsymbol{P}^{(2)}\boldsymbol{V}^{(2)} = -\boldsymbol{P}^{(1)}\boldsymbol{V}^{(1)} = -\boldsymbol{Z}^{(1)}$.

5. (4 points) Consider a neural network layer with preactivation $\boldsymbol{z}^{(\ell)} = \mathbf{W}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}$ and activation $\boldsymbol{h}^{(\ell)} = \boldsymbol{g}(\boldsymbol{z}^{(\ell)})$ where the activation function $g(z_i) = \log(1 + \mathrm{relu}(z_i))$ is applied elementwise. Let L be the loss function associated to this neural network. Which of these expressions is the correct derivative of L with respect to $\boldsymbol{h}^{(\ell-1)}$?

   - ☐ $\frac{\partial L}{\partial \boldsymbol{h}^{(\ell-1)}} = \boldsymbol{W}^\top \left(\frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}} \odot 1(\boldsymbol{z}^{(\ell)} \geq \mathbf{0})\right)$.
   - ■ $\frac{\partial L}{\partial \boldsymbol{h}^{(\ell-1)}} = \boldsymbol{W}^\top \left(\frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}} \odot \frac{1(\boldsymbol{z}^{(\ell)} \geq \mathbf{0})}{1+\mathrm{relu}(\boldsymbol{z}^{(\ell)})}\right)$.
   - ☐ $\frac{\partial L}{\partial \boldsymbol{h}^{(\ell-1)}} = \boldsymbol{W}^\top \left(\frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}} \odot (1 - (\boldsymbol{h}^{(\ell)})^2)\right)$.
   - ☐ $\frac{\partial L}{\partial \boldsymbol{h}^{(\ell-1)}} = \boldsymbol{W}^\top \frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}}$.

   Solution: We have $\frac{\partial L}{\partial z_i^{(\ell)}} = \frac{\partial L}{\partial h_i^{(\ell)}} g'(z_i^{(\ell)})$, i.e., $\frac{\partial L}{\partial \boldsymbol{z}^{(\ell)}} = \frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}} \odot \boldsymbol{g}'(\boldsymbol{z}^{(\ell)})$ and $\frac{\partial L}{\partial h_i^{(\ell-1)}} = \sum_j \frac{\partial L}{\partial z_j^{(\ell)}} W_{ji}$, i.e., $\frac{\partial L}{\partial \boldsymbol{h}^{(\ell-1)}} = W^\top \frac{\partial L}{\partial \boldsymbol{z}^{(\ell)}}$. We have $g'(z_i) = \frac{1(z_i \geq 0)}{1+\mathrm{relu}(z_i)}$, i.e., $\boldsymbol{g}'(\boldsymbol{z}) = \frac{1(\boldsymbol{z} \geq \mathbf{0})}{1+\mathrm{relu}(\boldsymbol{z})}$. Therefore:

$$\frac{\partial L}{\partial \boldsymbol{h}^{(\ell-1)}} = \boldsymbol{W}^\top \left(\frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}} \odot \boldsymbol{g}'(\boldsymbol{z}^{(\ell)})\right) = \boldsymbol{W}^\top \left(\frac{\partial L}{\partial \boldsymbol{h}^{(\ell)}} \odot \frac{1(\boldsymbol{z}^{(\ell)} \geq \mathbf{0})}{1 + \mathrm{relu}(\boldsymbol{z}^{(\ell)})}\right).$$

6. (4 points) Assume an auto-encoder that learns representations of 28×28 images using 3 hidden layers of sizes 250, 75, 250, respectively. Which of the following statements about the architecture of the auto-encoder is correct?

   - ☐ The output layer has K (number of classes) units; the latent representation has dimension 75; the loss function of the auto-encoder is the cross-entropy loss.
   - ■ **The output layer has 784 units; the latent representation has dimension 75; the loss function is the mean squared error.**
   - ☐ The output layer has 784 units; the latent representation has dimension 75; the loss function is the cross-entropy loss.
   - ☐ The output layer has 784 units; the latent representation has dimension 250; the loss function is the mean squared error.

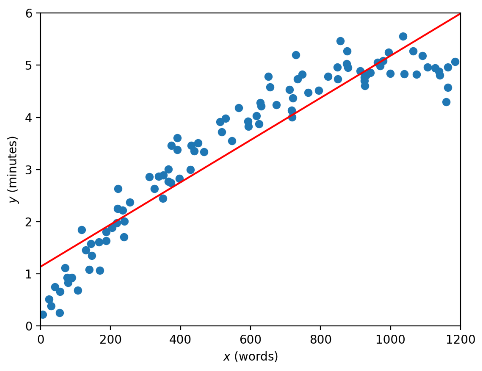
   Solution: We train the auto-encoder on the task of predicting the same image it received as input, *i.e.*, on the regression task of predicting the value of each pixel value, therefore the output layer will have 784 units and the loss function will be the mean squared error. The learned representation will be encoded in the network's bottleneck: 75 units.

7. (4 points) Which of these layers computes its output using the dot product between patches of the input and different filters?

   - ■ **Convolution layer.**
   - ☐ Max-pooling layer.
   - ☐ Fully-connected layer.
   - ☐ None of the above.

   Solution: The convolution layer does exactly what is described.

8. (4 points) The figure below represents a regression model (in red) fit to a given training dataset by minimizing the mean squared error loss. Can you describe what is happening and how would you fix it?



- ☐ Underfitting. Utilize dropout.
- ☐ Overfitting. Utilize ℓ_2-regularization.
- ■ **Underfitting. Increase the complexity of the model.**
- ☐ Overfitting. Train the model for a longer duration.

Solution: We need to increase the complexity of the model (e.g. the degree of the polynomial), since the model is underfitting the data.

Part 2: Short Answer Questions (18 points)

Please provide **brief** answers (1-2 sentences) to the following questions.

1. (6 points) What is exposure bias in sequence-to-sequence models and what causes it?

   Solution: Exposure bias is the tendency of sequence-to-sequence models to be exposed only to correct target sequence prefixes at training time, and never to their own predictions. This makes them having trouble to recover from their own incorrect predictions at test time, if they are produced early on in the sequence. Exposure bias is caused by auto-regressive teacher forcing, where models are trained to maximize the probability of target sequences and are always assigned the previous target symbols as context.

2. (6 points) In sequence models for natural language processing, why are sentences usually sorted by length when batching?

   Solution: Within the same batch, all sequences must have the same length, and for this reason they must be padded with padding symbols for making them as long as the longest sentence in the batch. Since in NLP sentences can have very different lengths, if we don't sort sentences by length, we can end up with very unbalanced batches, where some sentences are very short and others are very long, which makes it necessary to add a lot of padding symbols. This process is inefficient and can make training more time consuming. For this reason, sentences are usually sorted by length, which makes each batch more balanced.

3. (6 points) Mention two advantages of self-attention encoders over RNN encoders.

   Solution: They can better capture long-range relations between elements of the sequence, since information does not propagate sequentially (words can be compared with a constant number of operations). This usually reflects in more accurate models. Also, the training is usually faster, since the self-attention computation can be parallelized, unlike RNNs, that require sequential computation.

Part 3: Problems (50 points)

Problem 1: Convolutional Neural Networks (25 points)

GoogLeNet is a convolutional neural network architecture that makes use of a particular block structure known as *inception block*. It was proposed by Szegedy and collaborators in 2015 in an article published at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. The architecture of GoogLeNet is summarized in Fig. 1. Assume that all convolution layers shown include a ReLU non-linearity, besides the convolution operation.

Each inception block in the network (gray blocks) corresponds to a number of parallel convolutional branches, and its architecture is detailed in Fig. 2, where we write $\oplus$ to denote concatenation along the channel dimension and, as before, all convolution layers include a ReLU non-linearity, besides the convolution operation.

1. (5 points) The inception block includes some 1×1 convolutional layers. What does such a layer do, and why may it be useful?

   Note: Note that, for a $H \times W \times C$ image, a 1×1 convolutional layer comprises a filter with a $1 \times 1 \times C$.

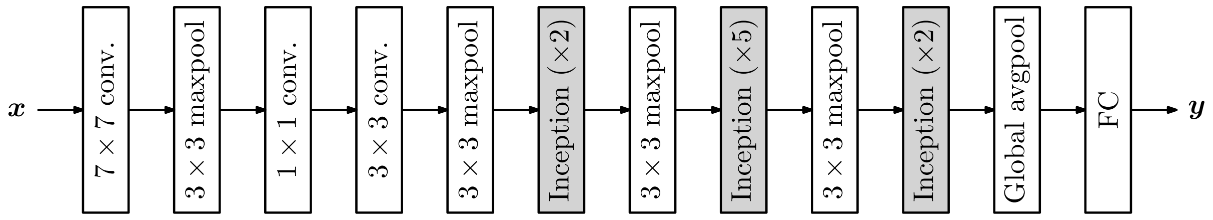


Figure 1: GoogLeNet architecture.

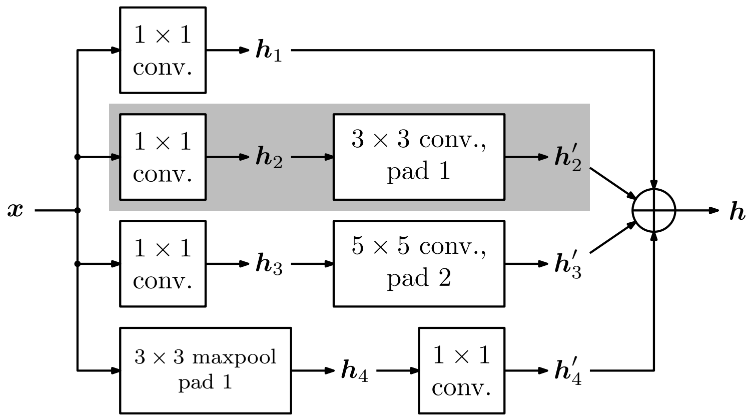


Figure 2: Inception block. The operation $\oplus$ denotes *concatenation* along the channel dimension. The branch used in question (c) is outlined in gray.

Solution: A 1×1 convolutional layer maps each input "pixel" (with all its channels) to one output pixel in a nonlinear way. It can be thought of as applying a fully connected layer to each input "pixel". This can be used, for example, to reduce or enlarge the number of channels in an image in a "pixelwise" manner, performing a nonlinear transformation on each of them.

2. (8 points) Consider an inception block where all convolutions have a single output channel. Suppose that the input $\boldsymbol{x}$ for such block is an $H \times W \times C$ image (C corresponds to the number of channels). Fill in the following table, where the dimensions refer to the different computed elements in Fig. 2 ($\boldsymbol{h}_1$, $\boldsymbol{h}_2$, $\boldsymbol{h}'_2$, etc.), and the number of parameters refer to the layers that compute them. Assume that stride = 1 and padding = 0, **unless where explicitly stated otherwise**. Indicate all relevant computations.

	# param.	**Dimension**
$\boldsymbol{x}$	–	$H \times W \times C$
$\boldsymbol{h}_1$	$C + 1$	$H \times W$
$\boldsymbol{h}_2$	$C + 1$	$H \times W$
$\boldsymbol{h}'_2$	$3 \times 3 + 1 = 10$	$H \times W$
$\boldsymbol{h}_3$	$C + 1$	$H \times W$
$\boldsymbol{h}'_3$	$5 \times 5 + 1 = 26$	$H \times W$
$\boldsymbol{h}_4$	0	$H \times W \times C$
$\boldsymbol{h}'_4$	$C + 1$	$H \times W$
$\boldsymbol{h}$	–	$H \times W \times 4$

3. (12 points) Consider once again the inception block in Fig. 2 and an input image $\boldsymbol{x}$ with dimensions $3 \times 3 \times 2$. Suppose that, after training, the parameters of the second branch in the block (outlined in gray) are

$$\boldsymbol{K}_{1\times 1} = \left[\left[0.5\right], \left[0.5\right]\right], \qquad b_{1\times 1} = 0,$$

for the 1×1 convolutional layer, and

$$\boldsymbol{K}_{3\times 3} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \qquad b_{3\times 3} = 0,$$

for the 3×3 convolutional layer. Compute $\boldsymbol{h}_2'$ for the input

$$\boldsymbol{x} = \left[\begin{bmatrix} 1.03 & 0.94 & 0.98 \\ 0.78 & 0.49 & 0.82 \\ 0.94 & 0.77 & 0.85 \end{bmatrix}, \begin{bmatrix} 0.97 & 1.06 & 1.02 \\ 0.82 & 0.51 & 0.78 \\ 0.86 & 0.83 & 0.95 \end{bmatrix}\right].$$

Indicate all relevant computations.

Note: If you are unable to compute $\boldsymbol{h}_2$, you can use

$$\boldsymbol{h}_2 = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.8 & 0.5 & 0.8 \\ 0.9 & 0.8 & 0.9 \end{bmatrix}$$

in the computation of $\boldsymbol{h}_2'$.

Solution: We can observe, from $\boldsymbol{K}_{1\times 1}$, that the 1×1 convolutional layer merely computes the average of the two input channels for the image, yielding

$$\boldsymbol{h}_2 = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.8 & 0.5 & 0.8 \\ 0.9 & 0.8 & 0.9 \end{bmatrix}.$$

To compute $\boldsymbol{h}_2'$ we first add a padding of 1 and then apply the convolution to the resulting image:

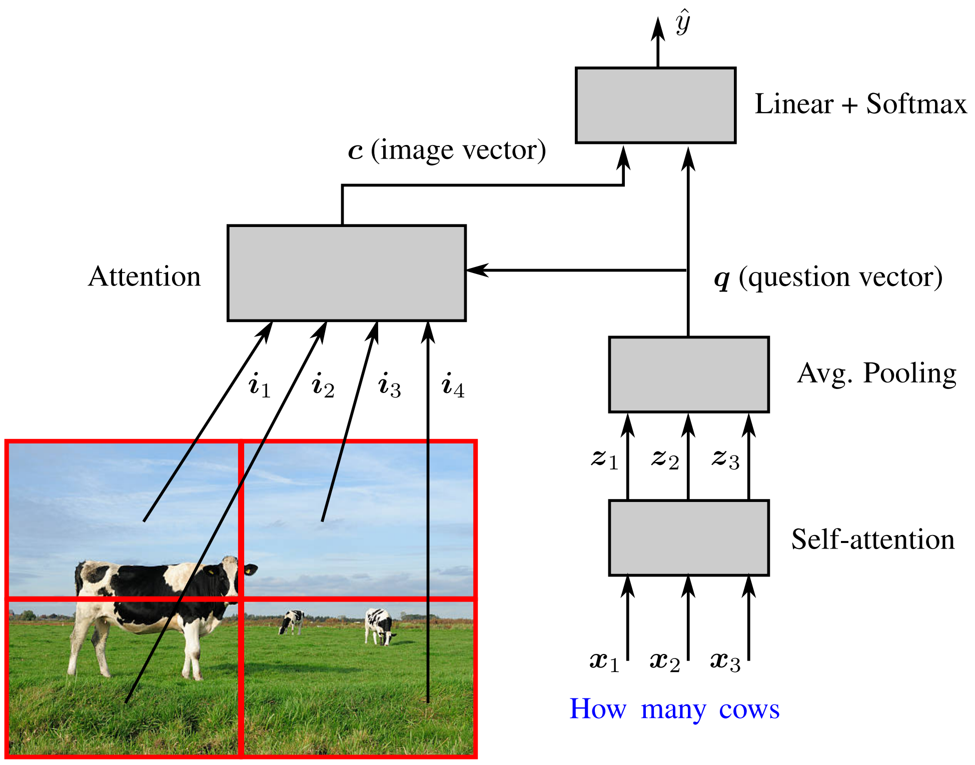
$$\boldsymbol{z}_2' = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.8 & 0.5 & 0.8 & 0.0 \\ 0.0 & 0.9 & 0.8 & 0.9 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1.3 & 2.1 & 1.3 \\ -0.3 & -0.4 & -0.3 \\ -1.3 & -2.1 & -1.3 \end{bmatrix}.$$

Finally, applying the ReLU, we get

$$\boldsymbol{h}_2' = \text{ReLU}\left(\begin{bmatrix} 1.3 & 2.1 & 1.3 \\ -0.3 & -0.4 & -0.3 \\ -1.3 & -2.1 & -1.3 \end{bmatrix}\right) = \begin{bmatrix} 1.3 & 2.1 & 1.3 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}.$$

Problem 2: Sequence-to-Sequence Models (25 points)

Consider the network represented below for a visual question answering task. The task is as follows: given an image (in the example, a picture of cows) and a natural language question about the image (such as "How many cows?"), the goal is to output an answer from a predefined list of answers—this can be seen as a classification task.



$\hat{y}$
Linear + Softmax
$\boldsymbol{c}$ (image vector)
Attention
$\boldsymbol{q}$ (question vector)
Avg. Pooling
$\boldsymbol{i}_1$ $\boldsymbol{i}_2$ $\boldsymbol{i}_3$ $\boldsymbol{i}_4$
$\boldsymbol{z}_1$ $\boldsymbol{z}_2$ $\boldsymbol{z}_3$
Self-attention
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$
How many cows

The network has the following architecture:

- The image is processed by a convolution neural network (not represented), resulting in 4 feature representations $\boldsymbol{i}_1, \boldsymbol{i}_2, \boldsymbol{i}_3, \boldsymbol{i}_4$, where each $\boldsymbol{i}_i \in \mathbb{R}^2$ as shown in the figure.

- The words of the question are encoded as word embeddings, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$, added to positional encodings $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$, and provided as input to a small transformer model. Each $\boldsymbol{x}_i = \boldsymbol{w}_i + \boldsymbol{p}_i \in \mathbb{R}^3$. In the example, $n = 3$ (there is no stop symbol for simplicity).

- The transformer model has one layer of self-attention with a single head, with 2×3 projection matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$. Scaled dot product attention is used to compute the attention probabilities. There is no feed-forward layer after the self-attention, no layer normalization, and no residual connections.

- The representations in the final layer of the transformer, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$, are average-pooled, leading to a single vector that represents the question, $\boldsymbol{q} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{z}_i$.

- This question vector $\boldsymbol{q}$ is then used in another attention mechanism as a query to attend over the image representations $\boldsymbol{i}_1, \boldsymbol{i}_2, \boldsymbol{i}_3, \boldsymbol{i}_4$ (again with scaled dot product attention), which are used both as keys and values. The result of this attention operation is an image vector $\boldsymbol{c}$.

- Finally, $\boldsymbol{q}$ and $\boldsymbol{c}$ are concatenated and go through an output linear layer, leading to a vector of answer probabilities

$$\operatorname{softmax}\left(\mathbf{A}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{c}\end{bmatrix}+\boldsymbol{b}\right),$$

  where $\mathbf{A}$ and $\boldsymbol{b}$ are model parameters.

- The model parameters are as follows. The embedding vectors are

$$\boldsymbol{w}_{\text{How}} = [-1,0,1]^\top, \quad \boldsymbol{w}_{\text{many}} = [1,-1,0]^\top, \quad \boldsymbol{w}_{\text{cows}} = [-1,-1,-1]^\top,$$
$$\boldsymbol{p}_1 = [1,0,0]^\top, \quad \boldsymbol{p}_2 = [0,1,0]^\top, \quad \boldsymbol{p}_3 = [0,0,1]^\top.$$

  The projection matrices of the transformer are

$$\mathbf{W}_Q = \mathbf{W}_K = \mathbf{W}_V = \begin{bmatrix} 0 & 2\\ -1 & 0\\ 2 & 1\end{bmatrix}.$$

  The parameters of the last output layer are:

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 1 & 0\\ -1 & -2 & 0 & 2\\ 0 & -1 & 2 & 1\end{bmatrix}, \quad \boldsymbol{b} = \mathbf{0},$$

  where the rows of $\mathbf{A}$ correspond (respectively) to the words "One", "Two", "Three".

1. (5 points) Compute the query matrix $\mathbf{Q}$, the key matrix $\mathbf{K}$, and the value matrix $\mathbf{V}$ associated to the words in the question. (Note: don't forget to account for the positional encodings.)

   Solution: By summing the word and positional encodings, we obtain the following embedding matrix:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1\\ 1 & 0 & 0\\ -1 & -1 & 0\end{bmatrix}$$

   The query, key, and value matrices are:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q = \begin{bmatrix} 0 & 0 & 1\\ 1 & 0 & 0\\ -1 & -1 & 0\end{bmatrix}\begin{bmatrix} 0 & 2\\ -1 & 0\\ 2 & 1\end{bmatrix} = \begin{bmatrix} 2 & 1\\ 0 & 2\\ 1 & -2\end{bmatrix}.$$

   Since all projection matrices are the same, we have $\mathbf{Q} = \mathbf{K} = \mathbf{V}$.

2. (10 points) Assume that in the previous question we obtained

$$\mathbf{Q} = \mathbf{K} = \mathbf{V} = \begin{bmatrix} 2 & 1\\ 0 & 2\\ 1 & -2\end{bmatrix}.$$

   Compute the question vector $\boldsymbol{q}$.

Solution: The attention probabilities are

$$\mathbf{P} = \text{Softmax}\left(\frac{1}{\sqrt{2}}\mathbf{Q}\mathbf{K}^\top\right) = \text{Softmax}\left(\frac{1}{\sqrt{2}}\begin{bmatrix} 2 & 1 \\ 0 & 2 \\ 1 & -2 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 0 & 2 \\ 1 & -2 \end{bmatrix}^\top\right) = \text{Softmax}\left(\frac{1}{\sqrt{2}}\begin{bmatrix} 5 & 2 & 0 \\ 2 & 4 & -4 \\ 0 & -4 & 5 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.87 & 0.10 & 0.03 \\ 0.20 & 0.80 & 0.00 \\ 0.03 & 0.00 & 0.97 \end{bmatrix}.$$

The output if the self-attention is:

$$\mathbf{Z} = \mathbf{P}\mathbf{V} = \begin{bmatrix} 1.77 & 1.03 \\ 0.39 & 1.79 \\ 1.03 & -1.91 \end{bmatrix}.$$

Finally, the query vector is

$$\boldsymbol{q} = \frac{1}{3}\mathbf{Z}^\top\mathbf{1} = [1.06, 0.30]^\top.$$

3. (10 points) Assume that in the previous question we obtained $\boldsymbol{q} = [1.06, 0.30]^\top$. Let the image feature maps be:

$$\boldsymbol{i}_1 = [1,1]^\top, \quad \boldsymbol{i}_2 = [1,1]^\top, \quad \boldsymbol{i}_3 = [0,0]^\top, \quad \boldsymbol{i}_4 = [4,4]^\top.$$

Compute the attention probabilities over the image feature maps (using scaled dot product attention) and the most probable answer returned by the system.

Solution: We can form the key matrix (which equals the value matrix):

$$\mathbf{I} = [\boldsymbol{i}_1^\top, \boldsymbol{i}_2^\top, \boldsymbol{i}_3^\top, \boldsymbol{i}_4^\top]^\top = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 4 & 4 \end{bmatrix}.$$

The attention probabilities are

$$\boldsymbol{a} = \text{softmax}\left(\frac{1}{\sqrt{2}}\mathbf{I}\boldsymbol{q}\right) = \text{softmax}\left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 4 & 4 \end{bmatrix}\begin{bmatrix} 1.06 \\ 0.30 \end{bmatrix}\right) = [0.05, 0.05, 0.02, 0.88]^\top.$$

The image vector is:

$$\boldsymbol{c} = \mathbf{I}^\top\boldsymbol{a} = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 1 & 1 & 0 & 4 \end{bmatrix}\begin{bmatrix} 0.05 \\ 0.05 \\ 0.02 \\ 0.88 \end{bmatrix} = [3.63, 3.63]^\top.$$

The logits for the answer are

$$\boldsymbol{s} = \mathbf{A}\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{c} \end{bmatrix} + \boldsymbol{b} = \begin{bmatrix} 2 & 0 & 1 & 0 \\ -1 & -2 & 0 & 2 \\ 0 & -1 & 2 & 1 \end{bmatrix}\begin{bmatrix} 1.06 \\ 0.30 \\ 3.63 \\ 3.63 \end{bmatrix} = [5.8, 5.6, 10.6]^\top.$$

Therefore, the answer is “Three”.

Deep Learning

MSc in Computer Science and Engineering

MSc in Electrical and Computer Engineering

Final exam — February 26, 2022

Version A

Instructions

- You have 120 minutes to complete the exam.
- Make sure that your test has a total of 9 pages and is not missing any sheets, then write your full name and student n. on this page (and your number in all others).
- The test has a total of 19 questions, with a maximum score of 100 points. The questions have different levels of difficulty. The point value of each question is provided next to the question number.
- Please provide your answer in the space below each question. If you make a mess, clearly indicate your answer.
- The exam is open book and open notes. You may use a calculator, but any other type of electronic or communication equipment is not allowed.
- Good luck.

Part 1	**Part 2**	**Part 3, Pr. 1**	**Part 3, Pr. 2**	**Total**
32 points	18 points	25 points	25 points	100 points

Part 1: Multiple Choice Questions (32 points)

In each of the following questions, indicate your answer by *checking a single option*.

1. (4 points) Which of the following is the derivative of the tanh activation function?
   - ■ $1 - \tanh(s)^2$
   - □ $\tanh(s)(1 - \tanh(s))$
   - □ 1 if $s > 0$, 0 otherwise.
   - □ None of the above.

   Solution: The correct option is $1 - \tanh(s)^2$.

2. (4 points) Your model is overfitting. Which of these strategies could mitigate the problem?
   - ■ **Augment your training set with more labeled data.**
   - □ Decrease regularization.
   - □ Increase the learning rate.
   - □ All of the above.

   Solution: Annotate more data. Regularization should increase, not decrease.

3. (4 points) A **transformer-based** sequence-to-sequence model translates an input sentence of M words into an output sentence with N words. How does the number of computational operations (algorithmic complexity) increase as a function of M and N?
   - □ $O(M + N)$
   - □ $O(MN)$
   - ■ $O(\max(M, N)^2)$
   - □ $O(M^N)$

   Solution: The correct option is $O(\max(M, N)^2)$, since self-attention has a quadratic cost.

4. (4 points) Which of the following sentences is **true**?
   - □ Neural networks work well in practice because their loss function is convex.
   - ■ **Variational auto-encoders are an instance of a latent variable model.**
   - □ Generative adversarial networks maximize a lower bound of the data log-likelihood.
   - □ Convolutional neural networks are equivariant to rotations and scalings.

   Solution: Variational auto-encoders are an instance of a latent variable model. They maximize a lower bound of the data log-likelihood, GANs do not.

5. (4 points) Consider a linear model used in a binary classification task, where the output corresponds to the probability of $y = +1$ given the input, and trained with a binary cross-entropy loss. Which of these statements is **false**?

   - ☐ The model corresponds to a logistic regression classifier.
   - ■ **The model is trained with the perceptron algorithm.**
   - ☐ The decision boundary for the network is a hyperplane in feature space.
   - ☐ The network can be trained using stochastic gradient descent.

   Solution: The binary cross-entropy loss indicates that the model is trained as a logistic regression classifier, which differs from the perceptron algorithm.

6. (4 points) Which of these statements is **false**?

   - ☐ A common used objective in GANs is

     $$\min_G \max_D \mathbb{E}_{\mathbb{P}_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{\mathbb{P}_{\boldsymbol{\theta}}(h)}[\log(1 - D(G(h)))],$$

     where G is the generator and D is the discriminator.
   - ■ **GPT-3 is an encoder-only model trained on a masked language modeling objective.**
   - ☐ VAEs often suffer from posterior collapse.
   - ☐ A common ethical concern with existing deep learning systems is their bias against minority groups not well represented in their training data.

   Solution: GPT-3 is a decoder-only model. **BERT** is an encoder-only model trained on a masked language modeling objective.

7. (4 points) The first layer in AlexNet can be specified by the following code line in Pytorch:

   ```
   nn.Conv2d(3, 96, kernel_size=11, stride=4)
   ```

   where the first two parameters correspond, respectively, to the number of input and output channels, and no padding is used. Suppose that the input images are $223 \times 223 \times 3$. What is the size of the output after the above convolutional layer?

   - ☐ $213 \times 213 \times 96$
   - ☐ $54 \times 54 \times 11$
   - ■ $54 \times 54 \times 96$
   - ☐ None of the above.

   Solution: The final dimension is $M' \times M' \times F$, where $M' = \lfloor (M - K)/S \rfloor + 1$ and F is the number of filters. In our case, $M = 223$, $K = 11$, $S = 4$ and $F = 96$, yielding $M' = (223 - 11)/4 + 1 = 54$.

8. (4 points) Suppose that a **max pooling** layer with a 2×2 kernel and stride of 1 receives the following input:

   $$\boldsymbol{x}_{\text{in}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

   What is the output of the pooling layer?

☐ $\boldsymbol{x}_{\text{out}} = \begin{bmatrix} 3 & 4 \end{bmatrix}$.

☐ $\boldsymbol{x}_{\text{out}} = \begin{bmatrix} 1.5 & 2.5 \\ 4.5 & 5.5 \end{bmatrix}$.

■ $\boldsymbol{x}_{\text{out}} = \begin{bmatrix} 5 & 6 \end{bmatrix}$.

☐ $\boldsymbol{x}_{\text{out}} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$.

Part 2: Short Answer Questions (18 points)

Please provide **brief** answers (1-2 sentences) to the following questions.

1. (6 points) Explain which problem long-short term memories (LSTMs) try to solve and how they do it.

   Solution: LSTMs solve the vanishing gradient problem of RNNs. They do it by using memory cells (propagated additively) and gating functions that control how much information is propagated from the previous state to the current and how much input influences the curreent state.

2. (6 points) What is an auto-encoder and why it can be useful?

   Solution: Auto-encoders are networks that are trained to learn the identify function, i.e., to reconstruct in the output what they see in the input. This is done by imposing some form of constraint in the hidden representations (e.g. lower dimensional or sparse). They are useful to learn good representations of data in an unsupervised manner, for example to capture a lower-dimensional manifold that approximately contains the data.

3. (6 points) Explain why transformers need causal masking at training time.

   Solution: The self-attention in transformers allows any word to attend to any other word, both in the source and on the target. When the model is generating a sequence left-to-right it cannot attend at future words, which have not been generated yet. At training time, causal masking is needed in the decoder self-attention to mask future words, to reproduce test time conditions.

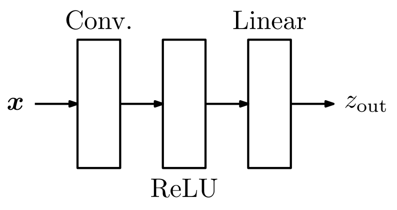
Part 3: Problems (50 points)

Problem 1: Convolutional Neural Networks (25 points)

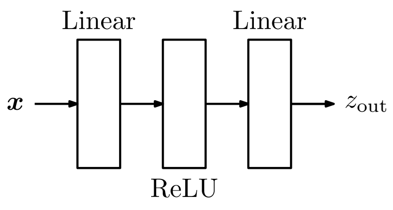
Consider the two networks depicted in Fig. 1. In the network of Fig. 1a the first block corresponds to a convolutional layer with a single 2×2 filter, no padding and stride $s = 1$. In the network of Fig. 1b the first block corresponds to a hidden layer with 4 units and ReLU activation.

In both networks we denote by $\boldsymbol{z}_1$ the input to the ReLU, by $\boldsymbol{h}_1$ the input to the rightmost linear layer, which in both cases comprises a single unit. We denote by z_{out} the scalar output, such that $\sigma(z_{\text{out}}) = \mathbb{P}[y = +1 \mid \boldsymbol{x}]$, where σ is the sigmoid function.

Note: Assume that, before the linear layer, $\boldsymbol{h}_1$ is flattened into a single column vector by stacking all columns of $\boldsymbol{h}_1$ together, as in the following diagram:

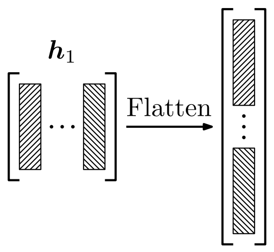


(a) Convolutional network.



(b) Fully connected network.

Figure 1: Two network architectures to process the input $\boldsymbol{x}$.



1. (5 points) Suppose that both networks expect as input a 3×3 matrix, which in the case of the network in Fig. 1b has been previously flattened. Fill in the table below. When counting the number of parameters in each network, make sure to consider the bias terms.

	Conv.	**Fully conn.**
Dimensions of $\boldsymbol{x}$	3×3	9×1
Dimensions of $\boldsymbol{z}_1$	2×2	4×1
Dimensions of $\boldsymbol{h}_1$	2×2	4×1
Dimensions of z_{out}	1	1
N. param. first layer	5	40
N. param. last layer	5	5

2. (8 points) Suppose that, after training, the parameters of the convolutional network in Fig. 1a are

$$\boldsymbol{K}_1 = \begin{bmatrix} -1 & -2 \\ 1 & 2 \end{bmatrix}, \qquad b_1 = 0 \qquad \boldsymbol{w}_{\text{out}} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \qquad b_{\text{out}} = -5,$$

where $\boldsymbol{K}_1$ and b_1 are the parameters of the convolutional layer, and $\boldsymbol{w}_{\text{out}}$ and b_{out} the parameters of the linear layer. Compute the output z_{out} of the network for the input

$$\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Solution: Computing the output of the convolutional layer for the provided input, we get

$$\boldsymbol{z}_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + 0 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \qquad \boldsymbol{h}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Then,

$$z_{\text{out}} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} - 5 = 5 - 5 = 0.$$

3. (8 points) Suppose that, after some training iterations, the parameters for the rightmost linear layer of the fully connected network in Fig. 1b are

$$\boldsymbol{w}_{\text{out}} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \qquad b_{\text{out}} = -5.$$

Let $(\boldsymbol{x}, y)$ be a sample in the dataset for which $y = +1$, and suppose that, when the input is $\boldsymbol{x}$, we have $\boldsymbol{h}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^\top$. Perform one update of stochastic gradient descent to $\boldsymbol{w}_{\text{out}}$ and b_{out}, using a stepsize $\eta = 1$ and knowing that the loss considered is the negative log likelihood, i.e.,

$$L(z_{\text{out}}; y) = \begin{cases} -\log \sigma(z_{\text{out}}) & \text{if } y = +1; \\ -\log(1 - \sigma(z_{\text{out}})) & \text{if } y = -1. \end{cases}$$

Recall that $\sigma(z) = 1/(1 + \exp(-z))$ and $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

Solution: We have that

$$z_{\text{out}} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} - 5 = 5 - 5 = 0$$

Also,

$$\nabla_{\boldsymbol{w}_{\text{out}}} L(z_{\text{out}}; y = +1) = (\sigma(z_{\text{out}}) - 1) \nabla_{\boldsymbol{w}_{\text{out}}} z_{\text{out}} = (\sigma(z_{\text{out}}) - 1) \boldsymbol{h}_1,$$
$$\nabla_{b_{\text{out}}} L(z_{\text{out}}; y = +1) = (\sigma(z_{\text{out}}) - 1) \nabla_{b_{\text{out}}} z_{\text{out}} = (\sigma(z_{\text{out}}) - 1).$$

Since $\sigma(z_{\text{out}}) = 0.5$,

$$\nabla_{\boldsymbol{w}_{\text{out}}} L(z_{\text{out}}; y = +1) = \begin{bmatrix} -0.5 \\ 0 \\ 0 \\ -0.5 \end{bmatrix},$$
$$\nabla_{b_{\text{out}}} L(z_{\text{out}}; y = +1) = -0.5,$$

yielding

$$\boldsymbol{w}_{\text{out}} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} - \begin{bmatrix} -0.5 \\ 0 \\ 0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 2 \\ 3 \\ 4.5 \end{bmatrix},$$
$$b_{\text{out}} = -5 - (-0.5) = 5.5.$$

4. (4 points) Briefly explain why using a ReLU activation may be preferable over a sigmoid activation in very deep networks.

   Solution: ReLUs are significantly simpler and have a much simpler derivative than the sigmoid, leading to faster computation times. Also, sigmoids are easy to saturate and, when that happens, the corresponding gradients are only residual, making learning slower. ReLUs saturate only for negative inputs, and have constant gradient for positive inputs, often exhibiting faster learning.

Problem 2: Sequence Classification (25 points)

Ada is developing a system for quote attribution, where the input is a sentence (a quote) and the output should be the author of that sentence, among three possibilities: Mark Twain (class 1), Albert Einstein (class 2), and GPT-3 (class 3). Ada trains an RNN-based classifier on a corpus of quotes, obtaining the following model parameters:

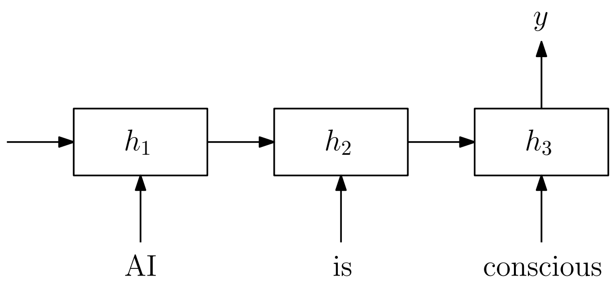
- The initial hidden state of the RNN, $\boldsymbol{h}_0$, is all-zeros.
- The input-to-hidden matrix and the hidden-to-output matrix are respectively:

$$\boldsymbol{W}_{hx} = \begin{bmatrix} 0 & -1 & 2 \\ 1 & -2 & 0 \end{bmatrix}, \quad \boldsymbol{W}_{yh} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 2 & -1 \end{bmatrix}.$$

- The recurrent matrix $\boldsymbol{W}_{hh}$ is the identity matrix.
- All biases are vectors of zeros.
- The RNN uses `relu` activations.

She now wants to use her model to predict the author of the quote “AI is conscious”. The relevant word embeddings are:

$$\boldsymbol{x}_{\text{AI}} = [0, 1, 1]^\top, \quad \boldsymbol{x}_{\text{is}} = [1, 0, -1]^\top, \quad \boldsymbol{x}_{\text{conscious}} = [0, -1, 0]^\top.$$



1. (8 points) Assume Ada’s model uses the last state of the RNN ($\boldsymbol{h}_3$) to make the prediction. Who is the predicted author of the quote? Show all your calculations.

Solution: We have:

$$\begin{aligned}
\boldsymbol{h}_1 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{AI}} + \boldsymbol{W}_{hh}\boldsymbol{h}_0)\\
&= \text{relu}([1,-2]^\top + [0,0]^\top)\\
&= [1,0]^\top.\\
\boldsymbol{h}_2 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{is}} + \boldsymbol{W}_{hh}\boldsymbol{h}_1)\\
&= \text{relu}([-2,1]^\top + [1,0]^\top)\\
&= [0,1]^\top.\\
\boldsymbol{h}_3 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_{\text{conscious}} + \boldsymbol{W}_{hh}\boldsymbol{h}_2)\\
&= \text{relu}([1,2]^\top + [0,1]^\top)\\
&= [1,3]^\top.\\
\hat{\boldsymbol{y}} &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_3)\\
&= \text{argmax}([-3,1,-1]) = 2 \quad \Rightarrow \quad \text{Albert Einstein.}
\end{aligned}$$

2. (8 points) Ada realized that her model is much better if she uses a simple attention mechanism (instead of using the last state of the RNN) as the pooling strategy. She adds a query vector $\boldsymbol{q}$ as an extra model parameter, and trains the entire model, obtaining the same parameters as above and $\boldsymbol{q} = [2,-1]^\top$. This model uses as keys and values the RNN states $\boldsymbol{h}_1 = [1,0]^\top$, $\boldsymbol{h}_2 = [0,1]^\top$, $\boldsymbol{h}_3 = [1,3]^\top$. Using **scaled dot product attention**, what is the new prediction for the same quote and which word receives the largest attention probability? Show all your calculations.

   Solution: The states $\boldsymbol{h}_1$, $\boldsymbol{h}_2$, $\boldsymbol{h}_3$ are the same as in the previous exercise. The attention scores are:

$$\begin{aligned}
s_1 &= \frac{1}{\sqrt{2}}\boldsymbol{q}^\top\boldsymbol{h}_1 = \frac{1}{\sqrt{2}}[2,-1]^\top[1,0] = \frac{2}{\sqrt{2}}\\
s_2 &= \frac{1}{\sqrt{2}}\boldsymbol{q}^\top\boldsymbol{h}_2 = \frac{1}{\sqrt{2}}[2,-1]^\top[0,1] = \frac{-1}{\sqrt{2}}\\
s_3 &= \frac{1}{\sqrt{2}}\boldsymbol{q}^\top\boldsymbol{h}_3 = \frac{1}{\sqrt{2}}[2,-1]^\top[1,3] = \frac{-1}{\sqrt{2}}\\
\boldsymbol{p} &= \text{softmax}([s_1,s_2,s_3]) = [0.807, 0.097, 0.097]\\
\boldsymbol{c} &= p_1\boldsymbol{h}_1 + p_2\boldsymbol{h}_2 + p_3\boldsymbol{h}_3 = [0.903, 0.387]\\
\hat{\boldsymbol{y}} &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{c})\\
&= \text{argmax}([-0.387, 0.903, 1.420]) = 3 \quad \Rightarrow \quad \text{GPT-3.}
\end{aligned}$$

3. (4 points) Is there any possible input for which any of the two models above (with the given parameters) can assign probability higher than $\frac{1}{3}$ to Mark Twain? What if we replace relu by tanh activations? Justify your answer.

   Solution: There is no such input. Since the output of relu is non-negative, given any $\boldsymbol{c} = [c_1, c_2]^\top$ with $c_1, c_2 \geq 0$, we have the logits $\boldsymbol{W}_{yh}\boldsymbol{c} = [-c_2, c_1, 2c_1 - c_2]^\top$, therefore the probability of the first class (Mark Twain) is always below or equal to the probability of the first class and below or equal to the probability of the second class, which implies it is less than or equal to $\frac{1}{3}$. This does not happen (necessarily) if we replace relu by tanh, since in that case c_1 and/or c_2 can be negative.

4. (5 points) Give one example of a transformer-based pretrained model that Ada could use for this task and the necessary steps to use it.

   Solution: Ada could use a pretrained encoder-only (e.g., BERT) or encoder-decoder model (e.g., T5, BART) and fine-tune it on the data she has available. Note: a decoder-only model (e.g. GPT-3) would not be the best choice, since this a classification task.

Deep Learning

MSc in Computer Science and Engineering

MSc in Electrical and Computer Engineering

Final exam — February 10, 2023

Version A

Instructions

- You have 120 minutes to complete the exam.
- Make sure that your test has a total of 14 pages and is not missing any sheets, then write your full name and student n. on this page (and your number in all others).
- The test has a total of 17 questions, with a maximum score of 100 points. The questions have different levels of difficulty. The point value of each question is provided next to the question number.
- Please provide your answer in the space below each question. If you make a mess, clearly indicate your answer. Please use a pen, not a pencil.
- The exam is open book and open notes. You may use a calculator, but any other type of electronic or communication equipment is not allowed.
- Good luck.

Part 1	Part 2	Part 3, Pr. 1	Part 3, Pr. 2	Total
32 points	18 points	25 points	25 points	100 points

Part 1: Multiple Choice Questions (32 points)

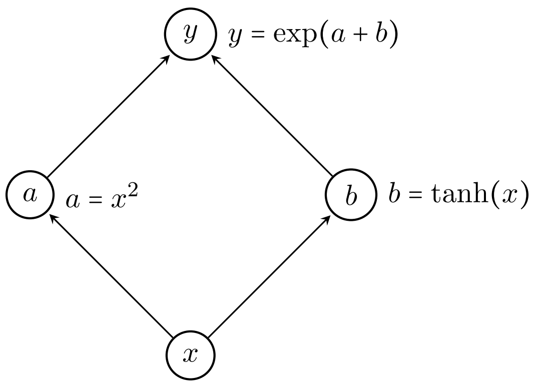
In each of the following questions, indicate your answer by *checking a single option.*

1. (4 points) Which framework is most suitable for a classification task with 3 labels?
   - ☐ Linear regression.
   - ☐ Binary logistic regression.
   - ■ **Multi-class logistic regression.**
   - ☐ None of the above is suitable.

2. (4 points) Which activation function best handles the problem of vanishing gradients?
   - ☐ Softmax.
   - ☐ Hyperbolic tangent.
   - ☐ Sigmoid function.
   - ■ **Rectified linear unit.**

3. (4 points) Which of the following statements about autoregressive RNN-based language models is correct?
   - ☐ Just like n-gram models, RNNs make a Markov assumption, hence they can only remember the last n words they generate, for a given $n \in \mathbb{N}$.
   - ☐ RNNs have unbounded memory, however they have a tendency to "remember" less accurately the most recent words they generate.
   - ■ **RNNs have unbounded memory, however for long sequences they have a tendency to "remember" less accurately the initial words they have generated.**
   - ☐ RNNs suffer from vanishing gradients, but this can be mitigated with gradient clipping.

4. (4 points) Consider the following computation graph. What is the derivative of y with respect to x?



   - ■ $\left(2x + 1 - \tanh^2(x)\right) y$.
   - ☐ $(2x + \tanh(x)\,(1 - \tanh(x)))\exp(a + b)$.
   - ☐ $2x + 1 - \tanh^2(x)$.
   - ☐ $\exp(i\pi) + 1$.

Solution: It is $\big(2x+1-\tanh^2(x)\big)\,y$:

$$\begin{aligned}
&\frac{\partial y}{\partial a} = \frac{\partial y}{\partial b} = \exp(a+b) = y,\\
&\frac{\partial a}{\partial x} = 2x, \quad \frac{\partial b}{\partial x} = 1-\tanh^2(x),\\
&\frac{\partial y}{\partial x} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial y}{\partial b}\frac{\partial b}{\partial x} = \big(2x+1-\tanh^2(x)\big)\,y.
\end{aligned}$$

5. (4 points) Let $\boldsymbol{s} \in \mathbb{R}^{10000}$ be a vector of logits where $s_1 = 0$ and $s_j = 1$, for all $j \in \{2, 3, \dots, 10000\}$. A probability vector $\boldsymbol{p} = [p_1, \dots, p_{10000}]^\top$ is obtained by taking the softmax transformation of $\boldsymbol{s}$. What is the value of p_1?

   - ☐ 0.
   - ☐ 1.
   - ☐ 10000^{-1}.
   - ■ **None of the above.**

   Solution: We have $p_1 = 1/(1 + 9999 \exp(1))$, which does not match any of the first three options.

6. (4 points) Assume we train an LSTM model on English to Portuguese translation. We ensure the training dataset contains the same amount of masculine/feminine pronouns. The resulting model:

   - ☐ Is not biased since the training dataset was balanced with respect to gender pronoun frequencies.
   - ☐ Can still be biased if the model was not trained for enough epochs.
   - ■ **Can still be biased if the co-occurence frequency of masculine/feminine pronouns with non-gendered words such as {doctor, nurse} is not balanced in the training data.**
   - ☐ Can still biased because LSTM cells include a bias vector.

   Solution: The marginal frequency of male/female pronouns is not sufficient to ensure that there is no bias. It is important to check the co-occurrence with non-gendered words.

7. (4 points) Consider a sequence of length L leading to a matrix of token representations $\boldsymbol{X} \in \mathbb{R}^{L \times D}$. What is the output of a self-attention layer when $\boldsymbol{X}$ is provided as input, using only a single attention head and where the projection matrices are $\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{W}_V = \boldsymbol{I}$ (the identity matrix)?

   - ☐ A vector representation in $\mathbb{R}^D$ that summarizes the most relevant vectors of $\boldsymbol{X}$ through an attention layer in which the query is the first row of $\boldsymbol{X}$.
   - ☐ A vector representation in $\mathbb{R}^L$ that summarizes the most relevant vectors of $\boldsymbol{X}$ through an attention layer in which the query is the first column of $\boldsymbol{X}$.
   - ☐ The sequence is fed through an LSTM and the last hidden state $\boldsymbol{h}_L \in \mathbb{R}^D$ is considered the output of the self-attention layer.
   - ■ **A matrix representation in $\mathbb{R}^{L \times D}$, where each row $\boldsymbol{z}_i \in \mathbb{R}^D$ results from doing attention over the whole original sequence in which the query is $\boldsymbol{x}_i \in \mathbb{R}^D$.**

8. (4 points) Which of the following statements is true?

   - ☐ The family of encoder-decoder large language models, of which T5 is an example, is suitable for classification tasks but unsuitable for generation tasks.
   - ■ **Encoder-only models can be pretrained with masked language modeling. An example is BERT.**
   - ☐ Decoder-only models are usually pretrained with masked language modeling. An example is GPT.

- ☐ Prompting is a more efficient way to adapt models than fine-tuning, but it can only be done if the pretrained model is trained with a masked language modeling objective.

Solution: The family of encoder-decoder large language models is suitable to both classification and generation. Decoder-only models, including GPT, are trained with a causal language modeling objective. Prompting is a more efficient way to adapt models than fine-tuning, but it is usually done with models such as GPT, which are trained with a causal language modeling objective.

Part 2: Short Answer Questions (18 points)

Please provide **brief** answers (1-2 sentences) to the following questions.

1. (6 points) Explain how a bidirectional RNN works and when they are useful.

   Solution: Bidirectional RNNs combine a left-to-right RNN with a right-to-left RNN. After propagating the states in both directions, the states of the two RNNs are concatenated. Bidirectional RNNs are used as sequence encoders, their main advantage is that each state contains contextual information coming from both sides. Unlike unidirectional RNNs, they have the same focus on the beginning and on the end of the input sequence.

2. (6 points) Explain the difference between self-attention and masked self-attention, as well as why and where the masked version is used.

   Solution: The self-attention in transformers allows any word to attend to any other word, both in the source and on the target. When the model is generating a sequence left-to-right it cannot attend at future words, which have not been generated yet. At training time, causal masking is needed in the decoder self-attention to mask future words, to reproduce test time conditions.

3. (6 points) Explain what an adapter is and what the advantages over fine-tuning are.

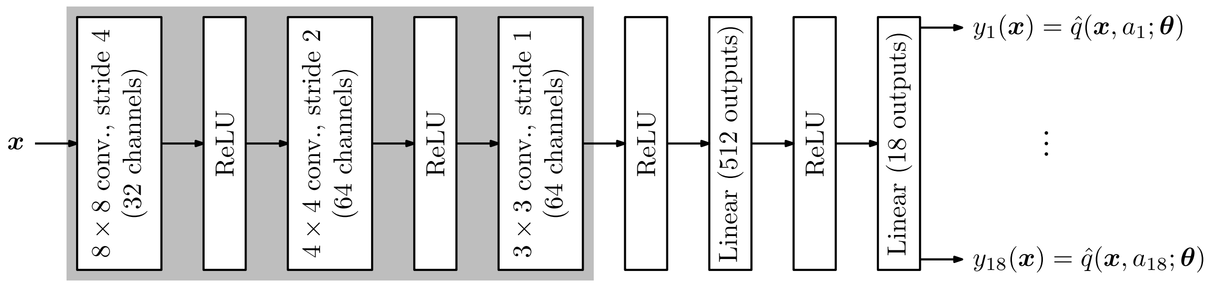
   Solution: TO DO

Part 3: Problems (50 points)

Problem 1: DQN (25 points)

The *Deep Q network* (DQN) is a convolutional neural network proposed in a pioneer work by DeepMind that first combined deep convolutional networks with reinforcement learning. DQN was used to learn how to play a number of games from the Atari 2600 console using only information from the pixels and the points in the game. DQN is summarized in Fig. 1, where the nonlinearities have been explicitly represented.

The input to the network consists of an image $\boldsymbol{x}$ with 84×84 pixels and 4 channels. The network has 18 outputs, where output i corresponds to a scalar value $\hat{q}(\boldsymbol{x}, a_i; \boldsymbol{\theta})$; $\boldsymbol{\theta}$ represents the parameters in the network and a_i is one among 18 possible actions in the game. Let $\mathcal{A}$ denote the set of such actions.



$\boldsymbol{x}$
8 × 8 conv., stride 4 (32 channels)
ReLU
4 × 4 conv., stride 2 (64 channels)
ReLU
3 × 3 conv., stride 1 (64 channels)
ReLU
Linear (512 outputs)
ReLU
Linear (18 outputs)
$y_1(\boldsymbol{x}) = \hat{q}(\boldsymbol{x}, a_1; \boldsymbol{\theta})$
$y_{18}(\boldsymbol{x}) = \hat{q}(\boldsymbol{x}, a_{18}; \boldsymbol{\theta})$

Figure 1: DQN architecture.

In very broad terms, to train the network, a set of samples of the form $(\boldsymbol{x}, a, target)$ is collected, where *target* is a scalar value and $a \in \mathcal{A}$. The loss associated with one such sample is then given by

$$L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta})) = \frac{1}{2}(target - \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))^2. \tag{1}$$

1. (10 points) Compute the total number of parameters in the network. Do not forget to account for the bias terms. Indicate all relevant computations.

   Solution: We have:

   - In the first convolutional layer, the number of parameters is $(8 \times 8 \times 4 + 1) \times 32 = 8,224$. The output has a dimension of $20 \times 20 \times 32$.
   - In the second convolutional layer, the number of parameters is $(4 \times 4 \times 32 + 1) \times 64 = 32,832$. The output has a dimension of $9 \times 9 \times 64$.
   - In the third convolutional layer, the number of parameters is $(3 \times 3 \times 64 + 1) \times 64 = 36,928$. The output has a dimension of $7 \times 7 \times 64$.

   The (flattened) input to the first linear layer is a vector with $7 \times 7 \times 64 = 3,136$ elements. The first linear layer has, therefore, a total of $3,136 \times 512 + 512 = 1,606,144$ parameters. Finally, the output layer has a total of $512 \times 18 + 18 = 9,234$ parameters.

   Summarizing, the network has a total of $8,224 + 32,832 + 36,928 + 1,606,144 + 9,234 = 1,693,362$ parameters.

2. (7 points) Suppose that you want to replace the layers in the shaded block in Fig. 1 by a single linear layer. Compute the number of parameters of the resulting neural network and compare them with the number of parameters of the original network.

   Note: If you did not answer Question 1, consider that the number of parameters of the original network is, approximately, 1.7×10^6 parameters.

   Solution: As seen in 1, the dimension of the output of the shaded is $7 \times 7 \times 64$ or, when flattened, $3,136 \times 1$. If we replace the shaded block with a linear layer, the input for such layer would have a dimension $(84 \times 84 \times 4) \times 1 = 28,224 \times 1$. The total number of parameters in that linear layer would thus be $28,224 \times 3,136 = 88,510,464$.

   The total number of parameters of the resulting network would thus be $88,510,464 + 1,606,144 + 9,234 = 90,125,842$ parameters, which is over 53 times larger than the original network.

3. (8 points) Suppose that, given a sample $(\boldsymbol{x}, a, target)$, we let

$$g_k = \frac{\partial L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))}{\partial \theta_k},$$

where $L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))$ is the loss defined in (1). Further suppose you want to include L_2 regularization when training the network. Indicate how the loss $L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))$ should be modified to include such regularization, and compute the derivative with respect to θ_k as a function of g_k.

Solution: The new loss would be

$$L_{\text{new}}(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta})) = L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta})) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2,$$

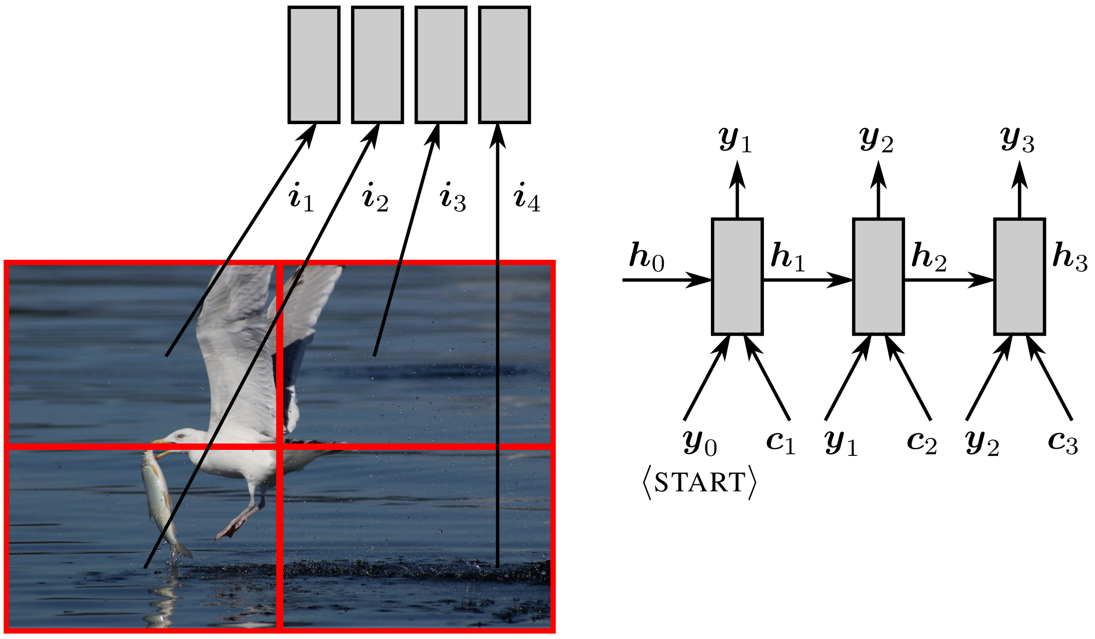
where N is the number of points in the dataset and $\|\cdot\|_2$ indicates the 2-norm of a vector, defined for an arbitrary vector $\boldsymbol{u}$ as $\|\boldsymbol{u}\|_2^2 = \boldsymbol{u}^T\boldsymbol{u}$.

The derivative with respect to θ_k becomes

$$\begin{aligned}\frac{\partial L_{\text{new}}(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))}{\partial \theta_k} &= \frac{\partial L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))}{\partial \theta_k} + \lambda\theta_k \\ &= g_k + \lambda\theta_k.\end{aligned}$$

Problem 2: Image Captioning (25 points)

Consider the problem shown in the figure, where an image is given and the task is to generate a descriptive natural language caption for the image.



The image is processed by a convolutional neural network (not represented), resulting in 4 feature representations $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, $\boldsymbol{i}_3$, $\boldsymbol{i}_4$, where each $\boldsymbol{i}_j \in \mathbb{R}^2$ as shown in the figure. Then, the caption is generated auto-regressively by an RNN-based decoder, conditioned on the image feature representations. The output vocabulary contains only 6 words, including the ⟨START⟩ and ⟨STOP⟩ symbols, with the following embedding vectors:

$$\boldsymbol{y}_{\langle\text{START}\rangle} = [0,0,0]^\top, \qquad \boldsymbol{y}_{\langle\text{STOP}\rangle} = [1,1,1]^\top, \qquad \boldsymbol{y}_{\text{fish}} = [-1,2,0]^\top,$$
$$\boldsymbol{y}_{\text{seagull}} = [1,-2,0]^\top, \qquad \boldsymbol{y}_{\text{flying}} = [0,-1,-1]^\top, \qquad \boldsymbol{y}_{\text{fishing}} = [0,2,1]^\top.$$

1. (10 points) Let the input image be the one shown in the figure, with the following image feature maps:

$$\boldsymbol{i}_1 = [4,0]^\top, \qquad \boldsymbol{i}_2 = [0,4]^\top, \qquad \boldsymbol{i}_3 = [0,0]^\top, \qquad \boldsymbol{i}_4 = [0,0]^\top.$$

At each time step t, the RNN-based decoder receives as input $\boldsymbol{x}_t \in \mathbb{R}^5$, which is a **concatenation** of the previous output embedding $\boldsymbol{y}_{t-1} \in \mathbb{R}^3$ and an image representation $\boldsymbol{c}_t \in \mathbb{R}^2$ (by this order), and uses this input and the previous hidden state $\boldsymbol{h}_{t-1}$ to compute the new state $\boldsymbol{h}_t$. This is followed by a linear output layer with hidden-to-output matrix

$$\boldsymbol{W}_{yh} = \begin{bmatrix} -5 & -5 \\ 0 & 3 \\ 1 & 2 \\ 2 & 2 \\ 3 & -1 \\ 2.9 & 0 \end{bmatrix} \begin{array}{l} \#\ \langle\text{START}\rangle \\ \#\ \langle\text{STOP}\rangle \\ \#\ \text{fish} \\ \#\ \text{seagull} \\ \#\ \text{flying} \\ \#\ \text{fishing} \end{array},$$

where each row of this matrix corresponds to the words stated above. The input-to-hidden matrix and the recurrence matrix are given respectively by

$$\boldsymbol{W}_{hx} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}, \qquad \boldsymbol{W}_{hh} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Assume that the RNN uses `relu` activations, that all biases are vectors of zeros, and that the initial hidden state $\boldsymbol{h}_0$ is a vector of zeros.

In this question, assume that $\boldsymbol{c}_t := \boldsymbol{c}$ is **constant for all time steps** and obtained through average pooling of the image feature representations, $\boldsymbol{c} = \frac{1}{4}\sum_{j=1}^{4} \boldsymbol{i}_j$, without any attention mechanism. Compute the first three words of the caption using **greedy decoding**.

Solution: We have:

$$
\begin{aligned}
\boldsymbol{c} &= \frac{1}{4}\sum_{j=1}^{4} \boldsymbol{i}_j = \frac{1}{4}[4,4]^\top = [1,1]^\top. \\
\boldsymbol{x}_1 &= \text{concat}(\boldsymbol{y}_{\langle \text{START} \rangle}, \boldsymbol{c}) = [0,0,0,1,1]^\top. \\
\boldsymbol{h}_1 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_1 + \boldsymbol{W}_{hh}\boldsymbol{h}_0) \\
&= \text{relu}([1,1]^\top + [0,0]^\top) = [1,1]^\top. \\
y_1 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_1) \\
&= \text{argmax}([-10,3,3,4,2,2.9]) = 4 \quad \Rightarrow \quad \text{seagull}. \\
\boldsymbol{x}_2 &= \text{concat}(\boldsymbol{y}_{\text{seagull}}, \boldsymbol{c}) = [1,-2,0,1,1]^\top. \\
\boldsymbol{h}_2 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_2 + \boldsymbol{W}_{hh}\boldsymbol{h}_1) \\
&= \text{relu}([2,-1]^\top + [1,1]^\top) = [3,0]^\top. \\
y_2 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_2) \\
&= \text{argmax}([-15,0,3,6,9,8.7]) = 5 \quad \Rightarrow \quad \text{flying}. \\
\boldsymbol{x}_3 &= \text{concat}(\boldsymbol{y}_{\text{flying}}, \boldsymbol{c}) = [0,-1,-1,1,1]^\top. \\
\boldsymbol{h}_3 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_3 + \boldsymbol{W}_{hh}\boldsymbol{h}_2) \\
&= \text{relu}([0,-1]^\top + [0,3]^\top) = [0,2]^\top. \\
y_3 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_3) \\
&= \text{argmax}([-10,6,4,4,-2,0]) = 2 \quad \Rightarrow \quad \langle \text{STOP} \rangle.
\end{aligned}
$$

2. (5 points) Assume now that, instead of using a constant $\boldsymbol{c}_t$ for all time steps, the RNN-based decoder has a **scaled dot-product** attention mechanism that attends to the image feature representations. For each time step, the query vector is $\boldsymbol{h}_{t-1}$ and the image feature representations $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, $\boldsymbol{i}_3$, $\boldsymbol{i}_4$ are used as both keys and values. Assume again that $\boldsymbol{h}_0$ is a vector of zeros. For the first time step ($t = 1$), compute the attention probabilities and the resulting image vector $\boldsymbol{c}_1$. Will the first word be the same or different from the one in the previous question?

Solution: We have:

$$
\begin{aligned}
\boldsymbol{I} &= [\boldsymbol{i}_1, \boldsymbol{i}_2, \boldsymbol{i}_3, \boldsymbol{i}_4]^\top = \begin{bmatrix} 4 & 0 \\ 0 & 4 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}. \\
\boldsymbol{p}_1 &= \text{softmax}\left(\frac{1}{\sqrt{2}}\boldsymbol{I}\boldsymbol{h}_0\right) = [.25,.25,.25,.25]^\top. \\
\boldsymbol{c}_1 &= [1,1]^\top.
\end{aligned}
$$

The first word will be the same (“seagull”) since the attention mechanism gives uniform probabilities due to the query being all-zeros, hence the first step is exactly the same as before.

3. (10 points) Compute the second word of the caption using the attention-based RNN decoder.

Solution: The first word is “seagull”, as seen before, and the first state is $\boldsymbol{h}_1 = [1,1]$ as in the first exercise. For the second and time step, we have:

$$
\begin{aligned}
\boldsymbol{p}_2 &= \text{softmax}\,(\boldsymbol{I}\boldsymbol{h}_1) = \text{softmax}(\frac{1}{\sqrt{2}}[4,4,0,0]^\top) = [0.47209639, 0.47209639, 0.02790361, 0.02790361]^\top. \\
\boldsymbol{c}_2 &= \boldsymbol{I}^\top \boldsymbol{p}_2 = [1.88838556, 1.88838556]^\top. \\
\boldsymbol{x}_2 &= \text{concat}(\boldsymbol{y}_{\text{seagull}}, \boldsymbol{c}) = [1, -2, 0, 1.888, 1.888]^\top. \\
\boldsymbol{h}_2 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_2 + \boldsymbol{W}_{hh}\boldsymbol{h}_1) \\
&= \text{relu}([2.888, -0.111]^\top + [1,1]^\top) = [3.888, 0.888]^\top. \\
y_2 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_2) \\
&= \text{argmax}([-23.9, 2.7, 5.7, 9.6, 10.8, 11.3]) = 6 \quad \Rightarrow \quad \text{fishing.}
\end{aligned}
$$