DATA.ML.200
Deep Learning

List of topics covered

Note: this material is not supposed to be an self-explanatory material for self-studying, but just a list of topics that can help to check the coverage of topics that were covered.

Week 1

Where to use deep learning?

- Difficult tasks
- High-dimensional data (audio, images, videos, medical signals, text, etc.) which have some structure in it
- Tasks for which there is large quantities of data available
- Loss functions that are differentiable
- Multiple processing stages dealing with different phenomena (temporal, spatial, etc. processing) by end-to-end learning

Supervised learning

- Training data consisting of pairs of inputs and outputs (x,y). Aim to generalize to new data (x,y).
- Binary classification, one output variable (class 0: y = 0; class 1: y=1)
- Multi-class classification. One-hot encoded vector y.
- Regression: vector with continuous values as target

Perceptron

- Mathematical model: output is weighted sum of inputs, followed by binarization of the output
- Linear decision boundary

Multilayer perceptron

- Idea: include one more hidden layers with nonlinearities
- Forward pass:
- Neuron model: pre-activations are weighted sum of inputs
- Activation obtained by applying nonlinear activation function on pre-activations
- Activation functions: sigmoid, rectified linear unit (ReLU), tanh

Output layer in classification / regression

- Last layer: linear in case of regression
- Classification: outputs are probabilities
- Binary: one output unit
- Sigmoid maps values between 0 and 1
- Multi-class: outputs are probabilities corresponding to each class
- Softmax layer: outputs sum to one
- Single-label (softmax) vs. multilabel (sigmoid)

Estimating the weights: losses

- Regression: mean-squared error MSE (maximum likelihood estimator in Gaussian noise)
- Binary classification: binary cross-entropy
- Single-label multi-class classification: categorical cross-entropy
- Multi-label multi-class classification: binary-cross-entropy

Optimization of MLP

- Non-convex optimization problem: no algorithm guaranteed to find globally optimal solution
- Gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent
- Epoch

Use of MLPs in deep learning

- As last layers of more complex systems (1-2 layers often enough)
- Effective for vector representations for finalizing decisions
- Used for changing dimensionalities of data
- Not very effective with multi-dimensional data (tensors, time-series data) and using redundancies within it -> other methods to deal with these
- Often called “fully connected layers”, FC layers

Gradient calculation in MLP

- Efficient gradient calculation: backpropagation algorithm
- Uses multiple times the chain rule

Use of MLPs in deep learning

- As last layers of more complex systems (1-2 layers often enough)
- Used for mapping the dimensionalities of data
- Not very effective with multi-dimensional data (tensors, time-series data) and using redundancies within it -> other methods to deal with the
- Often called “fully connected layers”, FC layers

Week 2

CNNs

Background

- Data often high-dimensional (images, audio, medical signals)
- MLPs do not model different translations of data
- Convolutions: equivariance

CNN basics

- Convolutional kernerls for obtaining feature maps
- Stride
- Translation invariance by pooling
- Max pooling
- Subsampling

The role of CNNs in deep learning

- Typically as the first layers

CNN receptive field

U-net

- Encoder-decoder
- Deconvolution / transposed convolution

Choosing the hyperparameters

- Kernel size, number of kernels per each layer etc.

1D vs 2D vs 3D convolution

- 1D: time series. E.g. audio
- 2D: Images, spectrograms
- 3D: E.g., MRI imaging (height, width, depth)

Separable convolution

Causal convolution

Week 3

Training / validation / testing splits

- Data splits (training & validation & testing) and role of each of them
- Overfitting
- Generalization, overfitting
- Choosing hyperpameters based on the validation data
- Early stopping

Regularization

- Early stopping
- Model averaging (ensemble)
- Dropout
- Limit the model size (with small data)
- Weight regularization
- Data augmentation

Gradient explosion / vanishing in deep networks

- Effect of non-linearities , ReLU, sigmoind, tanh)

Network architectures

- Skip connections (can skip a layer and use the values later. Example: U-net)
- Residual connection: (one type of skip connection) output = input + f(input)
- Gated connections: output = sigmoid(input) * input

Normalizations

- Input normalization
- Layer normalization (separate statistics for each sample, summing over nodes)
- Batch normalization (separate statistics for each node (channel in CNN), summing over samples in minibatch).

Optimization

- Stochastic gradient descent, Adaptive moment estimation (Adam)
- Step size

Week 4

RNNs

Why?

- Sequence modeling problems (machine translation, audio/visual captioning, language queries in LLMs) require complex sequential mappings
- CNNs / MLPs cannot handle this (they have fixed linear operations in each layer). Receptive field is fixed

Sequence modeling

- Problem formulations:
- Sequence to sequence (can be different sizes)
- Sequence to vector
- Vector to sequence

- Two main techniques RNNs / transformers

Building a sequence model

- Concepts: input sequence, output sequence

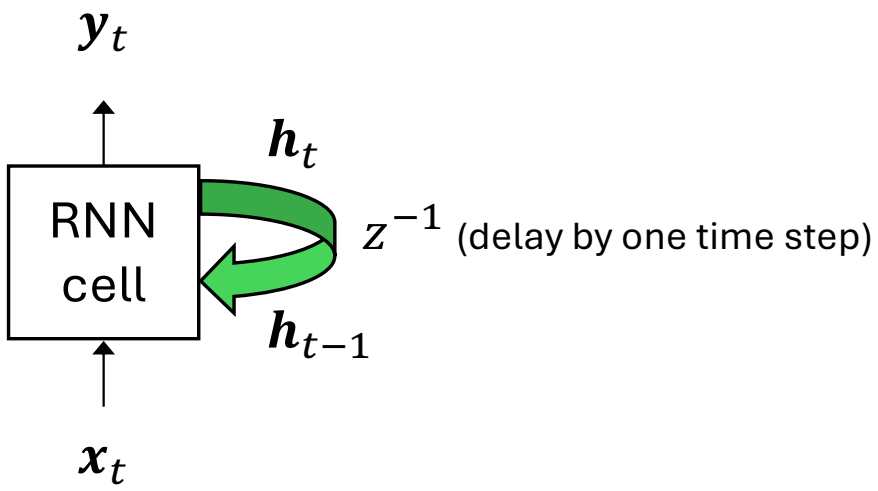
- Desirable properties: being able to use information from any input token to produce any output token

RNNs

- Two ideas: memory in the network & recurrent connections
- Memory needed to process data from multiple time steps (each token is a separate input)

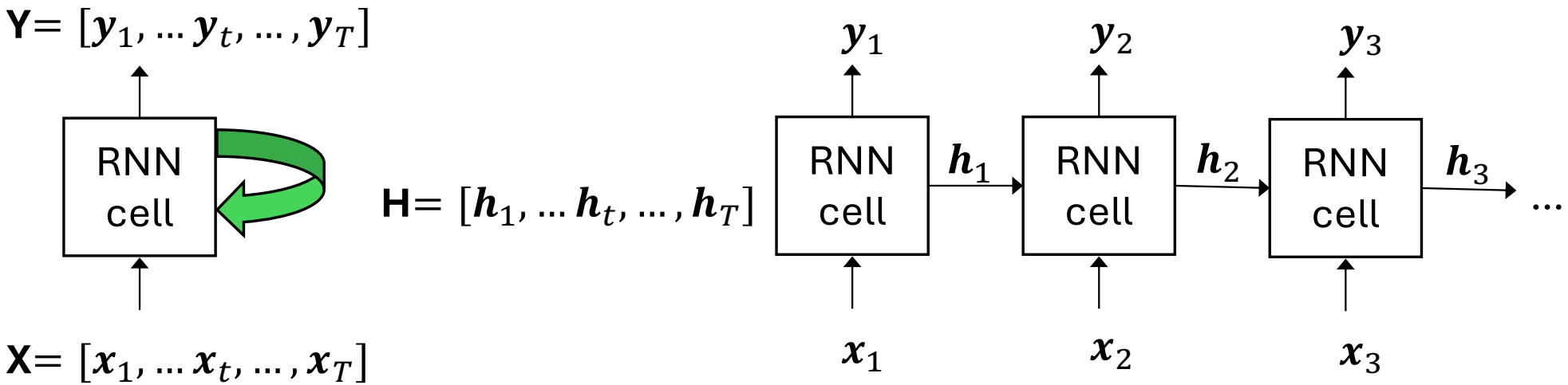
Recurrent neural networks (RNNs)

- RNN cell has two inputs: input and recurrent input
- Two outputs: hidden output and recurrent output
- Sharing of weights/parameters through time
- Suitable for modelling long-term dependencies



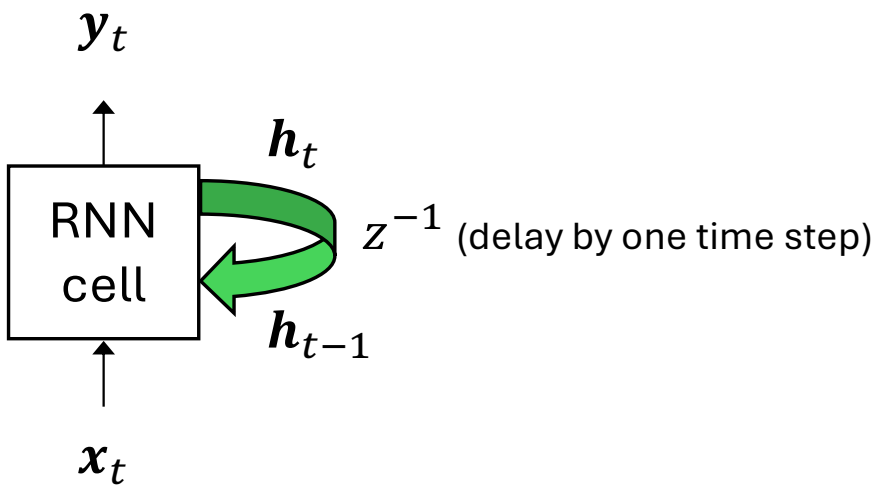
With material from CSC2535 2013: Advanced Machine Learning, Lecture 10: Recurrent neural networks, Geoffrey Hinton, University of Toronto.

RNN unrolled over time



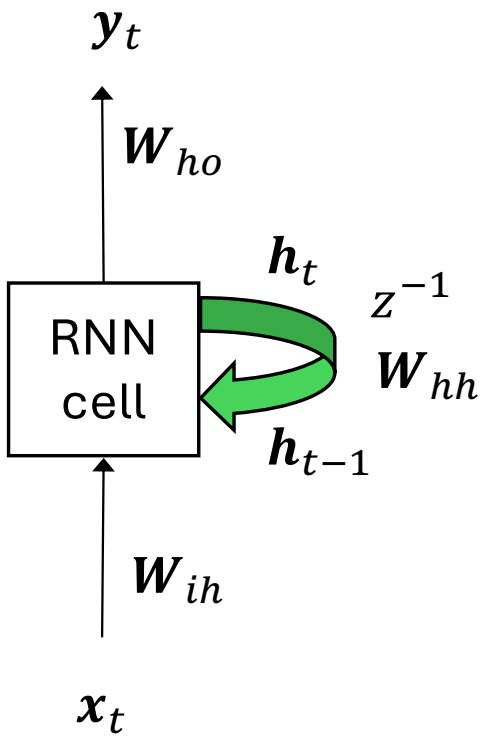
RNN hidden vector

- "Stored" in the memory of the cell while passing to the next time step
  - Allows storing past information
  - Updates of the hidden vector based on input and previous hidden vector
  - Recurrent connection: feedback loop



RNNs (continued)

- "Vanilla" RNN weights matrices (and associated biases):
- Input → hidden: $\mathbf{W}_{ih}$
- Hidden → hidden: $\mathbf{W}_{hh}$
- Hidden → output: $\mathbf{W}_{ho}$

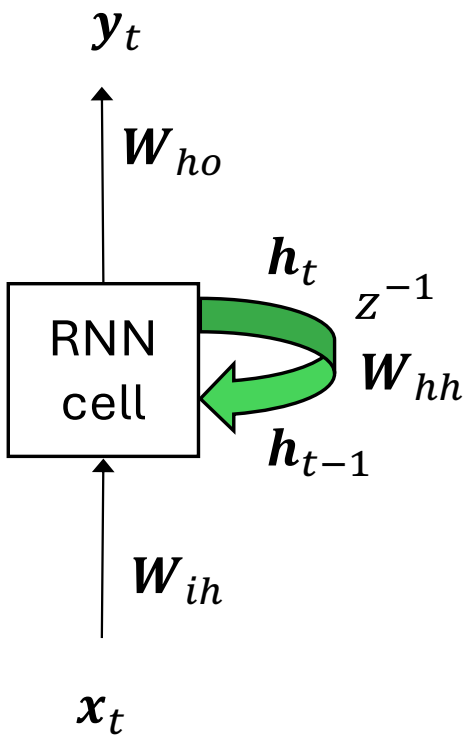


Vanilla RNN forward pass

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{b}_{ih} + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hh})$$

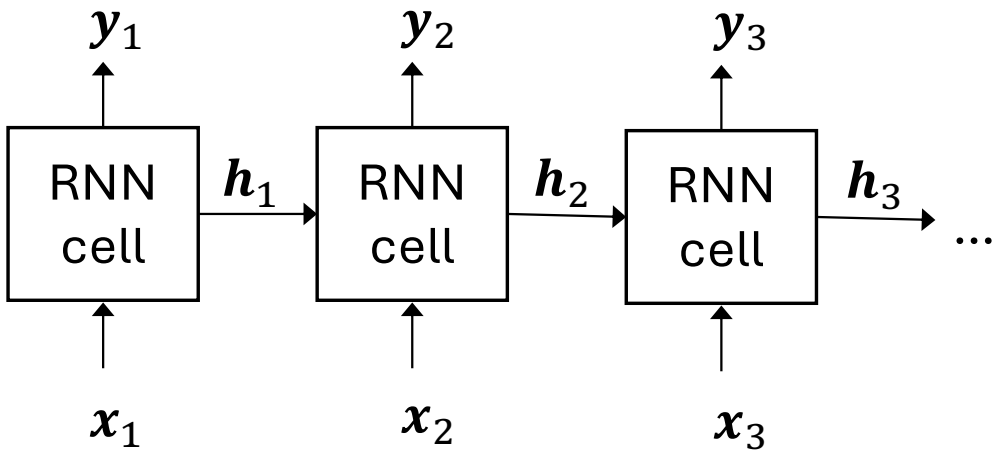
$$\boldsymbol{y}_t = \boldsymbol{W}_{ho}\boldsymbol{h}_t$$

$$\boldsymbol{h}_0 = \boldsymbol{0}$$



Training RNNs

- Trained with **b**ack**p**ropagation **t**hrough **t**ime (BPTT)
- Problems in training due to gradient
  - Vanishing → small gradients lead to exponential shrinkage
  - Exploding → big gradients lead to exponentially large increment



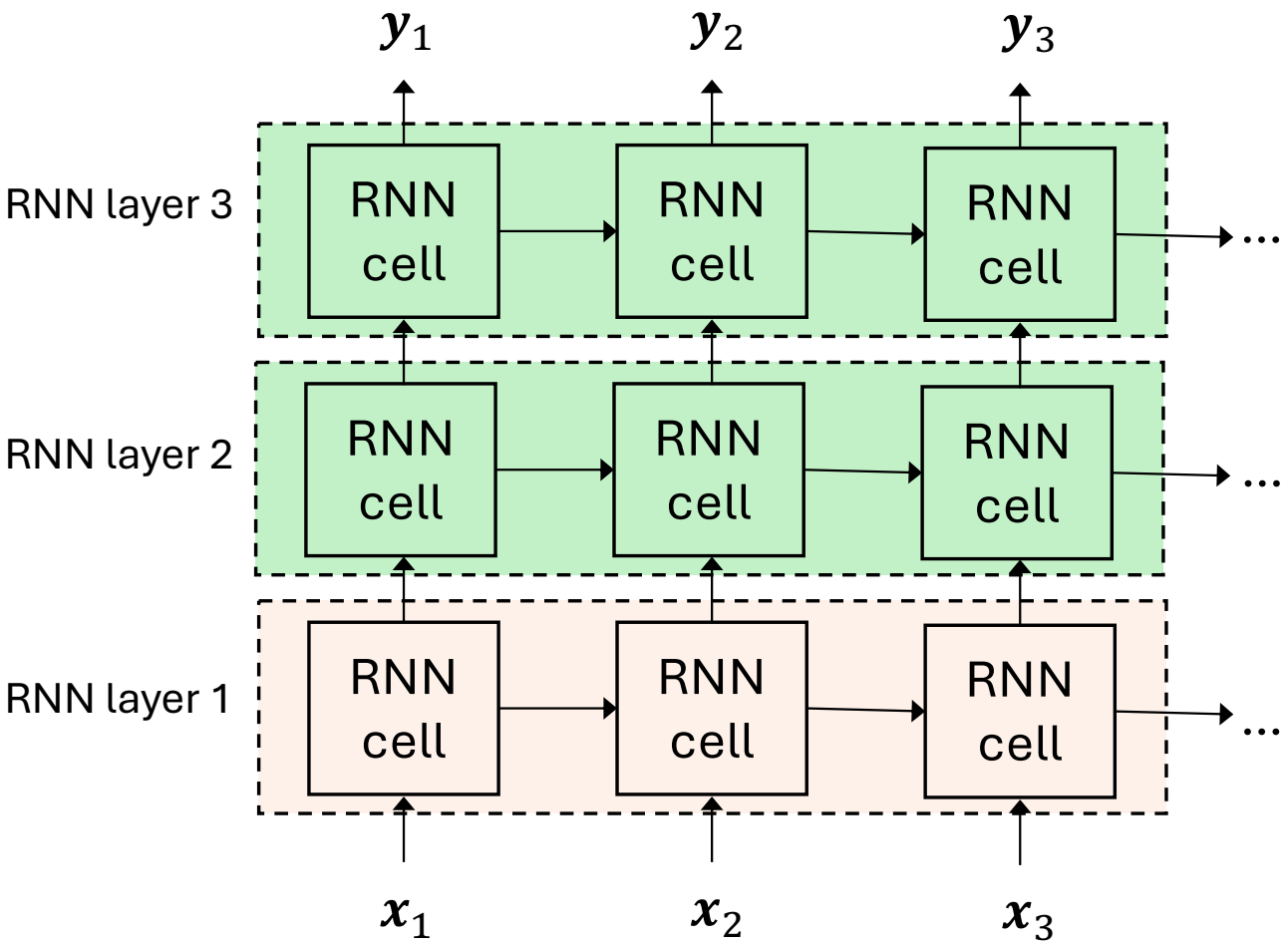
Solution to gradient problems: variants of vanilla RNN

- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

- The cell state vector is directly copied from the previous time step, with a *gating* mechanism that regulates how much new information is *added* to it
- Better gradient flow than vanilla RNN
- Allow training RNNs to model longer sequencies and dependencies in them

Deep recurrent neural networks

- Similarly to fully connected or convolutional layers, RNN layers can be stacked to obtain deep recurrent networks



RNNs

- Sequence to vector output using RNNs
- Vector to sequence using RNNs

Bidirectional recurrent neural network

- Two parallel recurrent layers where the information is flowing to different direction -> *bidirectional* recurrent network
- *Forward layer* processes the input in the order $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T$
- *Backward layer* processes the input in the order $\boldsymbol{x}_T, \boldsymbol{x}_{T-1}, \ldots, \boldsymbol{x}_1$
- Outputs of the two layers concatenated to get the final output
- Enables modeling future context when causal processing is not needed

Example: RNN based machine translation

- Encoder-decode structure
- Training: text sequencies, output shifted by one
- Inference: autoregressive generation
- Network used to train predict word / character / subword classes

Example: RNN based language model

- Can be used for language generation, also modeling probabilities
- Training: text sequencies, output shifted by one
- Network used to train predict word / character classes

Week 5

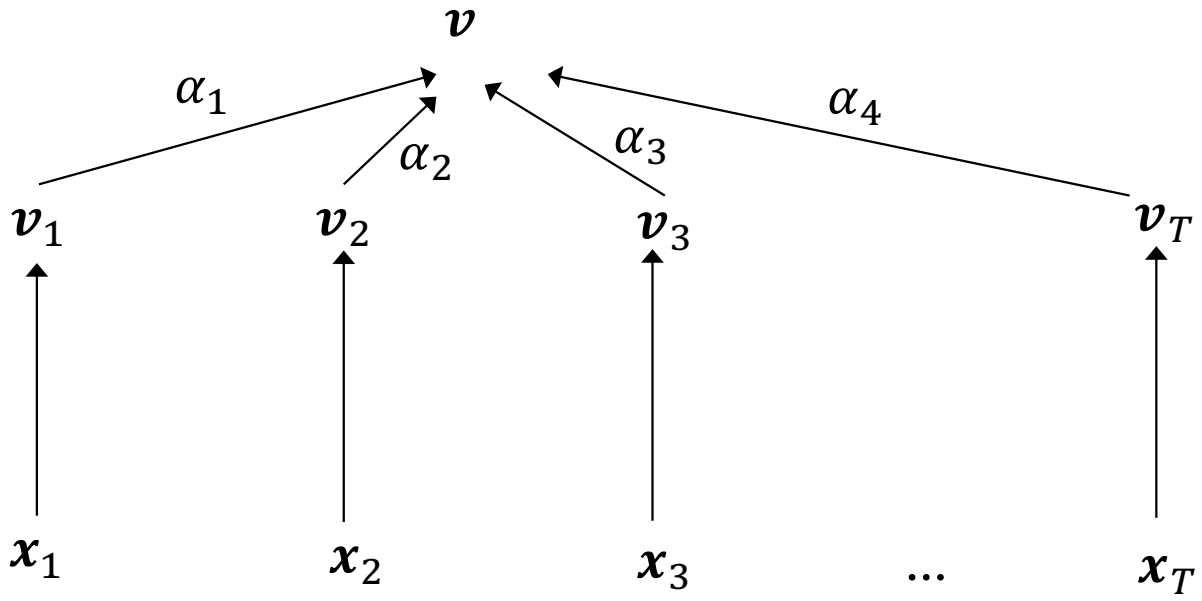
Attention

- Basic idea: different parts of a sequence (or a feature map in more general) are *weighted* when merging information
- Weighted sum, where the weights are *adaptive* – calculated based on the data on the fly -> ”attention pooling”

$$\boldsymbol{v} = \sum_t \boldsymbol{v}_t \alpha_t$$

$$\boldsymbol{v}_t = f(\boldsymbol{x}_t)$$

$$\alpha_t = \mathrm{softmax}_t(g(\boldsymbol{x}_t))$$



Attention

- Simple attention pooling useful in models where not all the instances contribute to an output equally
  - For example: classification when the target sound is not active during the whole sequence
- Lots of applications in sequence to sequence tasks where the labels are not synchronous (automatic speech recognition, machine translation, etc.)

Self-attention

- Attention score and weight calculated for each pair of instances in an input sequence
- Simple principle which variants are used e.g. in powerful NLP models (GPT-2, GPT-3, BERT)
- GPT: Generative pretrained transformer
- Transformer = multi-head self-attention (multiple parallel self-attention mechanisms)
- Powerful sequence model where basic computation operations can be done in parallel
- Applicaple in audio decoders: CNN encoder extract high-level features, and transformer decoders are used to produce the output

Self-attention

- Attention score and weight calculated for each pair if inputs instances of an sequence

  $\boldsymbol{x}_i, \boldsymbol{x}_j \rightarrow e_{i,j} \rightarrow \mathrm{softmax_j} \rightarrow \alpha_{i,j}$

- Value vector:

  $\boldsymbol{x}_j \rightarrow \boldsymbol{v}_j$

- Output sequence:

  $\boldsymbol{y}_i = \sum_j \boldsymbol{v}_j \alpha_{i,j}$

Self-attention

- Self-attention layer in matrix form:

$$\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

- $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_q$ is the query matrix (time x dimensions)
- $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_k$ is the key matrix (time x dimensions)
- $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_v$ is the value matrix (time x dimensions)
- $\boldsymbol{X}$ is the input feature sequence (time x dimensions) represented using a matrix

Self-attention

- Self-attention layer in matrix form:

$$\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

- Simple operation where operations can be calculated in parallel very effectively -> allows training with large datasets effectively
- Above operation presents the processing of one *attention head*. Typically several of these are used in one layer (transformer = multi-head self-attention) and their outputs are concatenated

Transformer: multi-head self attention

Transformer decoder

- Autoregressive model, similar to RNN decoder
- GPT: generative **pre-trained** transformer
- Pre-training: (self-supervised learning): input: text, output: text shifted by one. Implemented by masked prediction to force causality (in an RNN it was the uni-directional architecture)
- Fine-tuned to specific tasks

Transformer encoder

- Bidirectional sequence to sequence model (predict probabilities of tokes)
- Trained in **self-supervised** manner by masking individual input tokens (words)
- Masked words presented by “mask” symbol and the masked words are predicted by the model

Positional encoding

- Positions of the feature vectors in the sequence does not matter -> the position needs to be represented explicitly (**positional encoding** added to the original feature vectors)

Transformer: multi-head self attention

- Multiple attention heads with different weight matrices for query, key, value
- Outputs of all heads are concatenated -> linear layer reduces dimensionality

Masked attention / causal attention

- Pretraining / inference

Transformer encoder-decoder

- Sequence to sequence model

Working with text data

- Characters: each character a separate class -> one-hot encoding
- Words: **embedding** for each word
- Subword tokens (one-hot encoding or embeddings)

Working with images / audio

- The number of samples / pixes too many to use directly with transformers
- Options: image (divide into patches e.g. 16 x 16, stack each into a vector that is a token to a DNN)

- Audio: CNN first, then transformer
- Conformer: transformer and convolutional layers alternating

Representation learning

- Raw data is often high-dimensional and unstructured.
- Good representations capture semantic meaning and reduce complexity.
- They allow models to generalize better across tasks.

- can learn from unlabeled data

- Trained representations can be **fine-tuned** to specific tasks

Self-supervised learning

- One of the most powerful techniques for representation learning
- Input Data itself used to generate training targets

- Several approaches, “Pre-text tasks”
- One of them: masking individual input tokens (words).”

- Related to how the brain processes sensory signals (the brain predicts them, to anticipate the environment, be energy-efficient, handle uncertainty, faster reaction, and learning

Week 6

GANs

Generative modeling

- Goal: we want to generate data based on some model G (and possibly condition model the model e.g. on text G(c))
- Generative modeling = model the distribution of training samples P(X), and sample from the distribution
- Basic setup: have a set of training samples, learn to generate similar samples.
- Problem: we have no way to directly measure the distribution in case of large-dimensional data (in sequence models it was possible because of some assumptions)

- How to make a generative model that has the same PDF as the training samples? $P_{Gen}(X) = P_{Real}(X)$

- Main methods: generative adversarial learning, diffusion models, normalizing flows
- We will study GANs (they are easier to understand, and computationally less expensive)

Generative adversarial learning

- How to tell difference between real and generated?
- Solution: Train a discriminator network: takes real and fake as input, does binary classification

- Generator tries to fool the discriminator

- Two networks: generator and discriminator that are competing with each other

Generative adversarial learning

- How generator can generate different outputs? Idea: real data lies in low-dimensional manifold, which can be represented with a low-dimensional latent vector. Use low-dimensional random latent vectors as input
- Discriminator: takes real and fake as input, does binary classification
- Generator and discriminator structures domain specific. For example CNNs

Training GANs

- Discriminator trained by minimizing the BCE loss
- Generator ties to MAXIMIZE the loss. Can be implemented by gradient reversal layer. Alternatively, inverse the fake labels when training the generator
- Training is very sensitive to the parameters (e.g. step size).
- Can be controlled by 1) scheduling (how many iterations each model is trained), 2) learning rates, or 3) magnitude of the gradient reversal
- Validation: best to train a separate discriminator

Week 7

Transfer learning

- Deep learning benefits from training models with large datasets. Requires lots of data and compute
- In some tasks, large-scale data is not available
- Transfer learning: transfer knowledge from one task (**source task** where resources are available) to other task (**target task**).
- Source task -> target task
- Differences in data: space of inputs, space of outputs, distribution of inputs & outputs

Source models / tasks

1. One source task, supervised training
2. Pre-training: training a generic source model. Self-supervised learning, pre-text task
3. Multi-task learning
4. In-context learning (task defined by the input)

How to do transfer learning

1. Fine-tuning.
    - Take the source model and train with the target task data
2. Using the representation from the source task
    - Take the source model, remove the last layer, and train a new layer to fit to the target task
    - Representation learning
    - Using representation, “embeddings” from the source model
3. Low-rank adaptation (LoRA)

Related concepts:

- Continual learning, incremental learning
- Domain adaptation: label set the same, distribution different.

Contrastive learning

- Positive and negative examples
- Similarity / distance calculation
- Example: InfoNCE
- Semantic representation space
- Cross-modal learning (mapping data from different modalities to the same shared space -> similarity calculation in the shared space -> cross modal retrieval etc.)