CS-E4891 Deep Generative Models
Lecture 5: Autoregressive Models

Lauri Juvela

Department of Information and Communications Engineering (DICE)
Aalto University

April 28, 2025

Lauri Juvela – Introduction

- Assistant professor at Aalto since 2023, **Speech Synthesis Group** at ELEC / DICE , PhD 2020
- Machine learning researcher at Neural DSP 2019-2023, working on guitar amplifier modeling with neural networks
- Research:
  - Deep generative models for speech and audio
  - Controllable and interpretable neural speech synthesis, differentiable DSP
  - Audio effects modeling
  - Watermarking and deepfake detection

Outline

- Autoregressive models
- Distributions used in autoregressive modeling
- Deep neural networks for autoregression
- Autoregreressive Transformer language models
- Tokenisation and byte pair encoding

- Reading: Autoregressive models (Section 22 from Murphy, 2023), Transformer language model (Section 12.1 from Bishop 2024)

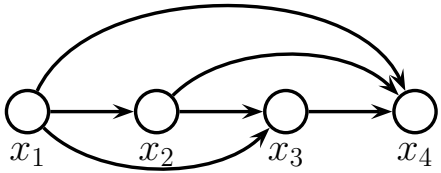
Autoregressive models

- We can factorize any sequence of probabilities as an autoregressive model using the chain rule of probability

$$p(\boldsymbol{x}_{1:T}) = p(\boldsymbol{x}_1)p(\boldsymbol{x}_2|\boldsymbol{x}_1)p(\boldsymbol{x}_3|\boldsymbol{x}_2,\boldsymbol{x}_1)p(\boldsymbol{x}_4|\boldsymbol{x}_3,\boldsymbol{x}_2,\boldsymbol{x}_1)\cdots = \prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{x}_{1:t-1})$$

In this lecture we consider explicit generative models

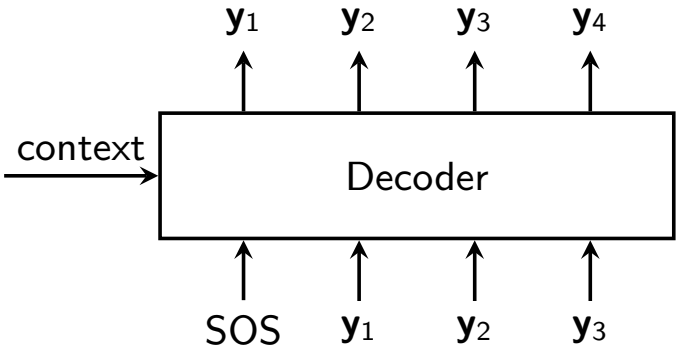
- Density model $p_\theta(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{\theta})$ has an explicit parametric form
- A trained model can be used to generate new examples from $p_\theta(\boldsymbol{x})$.



Fully connected autoregressive model
Figure 22.1 from (Murphy, 2023)

Deep autoregressive models

- Autoregressive models are common in sequence-to-sequence translation problems
- Typical architectures: RNN, CNN, Transformer



First order linear autoregressive AR(1) process

Time series x_t is governed by an AR(1) process

$$x_t = \phi x_{t-1} + \varepsilon_t, \text{where } \varepsilon_t \sim \mathcal{N}(0, \sigma^2)$$

Linear predictive model:

$$\hat{x}_t = \hat{\phi} x_{t-1}$$

- Least-squares solution for the optimal parameter $\hat{\theta}$ is the maximum likelihood estimate (MLE) (home exercise)

$$\text{MSE}(x, x_{t-1}; \theta) = \sum_{t=1}^{T} (x_t - \theta x_{t-1})$$

- Linear AR models have closed form MLE solutions, in this case

$$\hat{\phi}_{\text{MLE}} = \frac{\sum_{t=1}^{T-1} x_t x_{t-1}}{\sum_{t=1}^{T-1} x_t^2}$$

- You can also estimate $\hat{\theta}$ using gradient descent (home exercise). This becomes very useful when the model is not linear, such as neural networks.

Markov Models

Markov property: variable $\boldsymbol{x}_t$ only depends on the previous time-step $\boldsymbol{x}_{t-1}$

$$p(\boldsymbol{x}_{1:T}) = p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$$

Recurrent neural networks are non-linear Markov models

$$\boldsymbol{h}_t = \mathsf{RNN}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

In practice, Markovian systems can model longer dependencies, if the state can pack information from multiple time steps.

N-grams are simple language models (Murphy 2.6.2)

- Bi-grams: probability of the word pair (w_i, w_j) occurring in text.
- Maximum likelihood estimate: count the occurences in a text corpus

$$p(w_j|w_i) = \frac{\sum_{n=1}^{N} \mathbb{I}(w_n = w_j)\mathbb{I}(w_{n-1} = w_i)}{N}$$

- Distribution $p(w_n|w_{n-1})$ is the collection of all bi-gram point probabilities

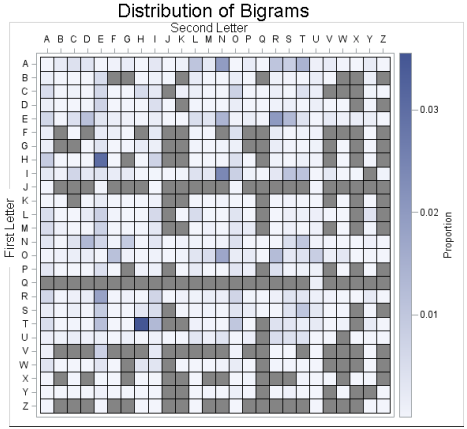


Figure: English letter Bi-gram frequency matrix (`https://blogs.sas.com/content/iml/2014/09/26/bigrams.html`)

N-grams are simple language models (Murphy 2.6.2)

- Tri-grams: probability of the ordered word triplet (w_i, w_j, w_k) occurring in text.
- Maximum likelihood estimate: count the occurences in a text corpus

$$p(w_k|w_i, w_j) = \frac{\sum_{n=1}^{N} \mathbb{I}(w_n = w_k)\mathbb{I}(w_{n-1} = w_j)\mathbb{I}(w_{n-2} = w_i)}{N}$$

- Distribution $p(w_n|w_{n-1}, w_{n-2})$ is the collection of all tri-gram point probabilities
- N-grams generalize the idea to the context length N

$$p(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i})}{|V| \sum_{w_i} c(w_{i-n+1}^{i})}$$

- Unseen N-grams will have a probability zero, which can cause problems in language modeling
- Practical N-gram models apply some kind of smoothing to the counts, for example Laplace smoothing adds one to all counts

$$p(w_i|w_{i-n+1}^{i-1}) = \frac{1 + c(w_{i-n+1}^{i})}{|V| \sum_{w_i} c(w_{i-n+1}^{i})}$$

Higher order Markov models (Murphy 2.6.2)

- N-grams can be seen as higher-order Markov models with order N
- Higher order Markov property: variable $\boldsymbol{x}_t$ depends on the previous time-steps $\boldsymbol{x}_{t-n:t-1}$

$$p(\boldsymbol{x}_{1:T}) = \prod_{t=n+1}^{T} p(\boldsymbol{x}_t | \boldsymbol{x}_{t-n:t-1})$$

- Causal convolution nets have a finite receptive field R and can also be considered higher-order non-linear Markov models

$$\boldsymbol{x}_t = \text{CNN}(\boldsymbol{x}_{t-1:t-R})$$

Neural autoregressive distribution estimators (NADE) (Murphy 22.2)

- Extension of Mixture Density Networks (MDN) to work with autoregressive models

$$p(x_t|\boldsymbol{x}_{1:t-1}) = \sum_{k=1}^{K} \pi_{t,k} \mathcal{N}(\boldsymbol{x}_t|\mu_{t,k}, \sigma^2_{t,k})$$

- Distribution parameters are generated by a network

$$[\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t, \boldsymbol{\pi}_t] = f_t(\boldsymbol{x}_{1:t-1}, \boldsymbol{\theta}_t)$$

- Re-parametrisation is similar to VAEs: split network output to three chunks and re-parameterize variables

$$\boldsymbol{\mu}_t = \boldsymbol{z}_{\mu,t}, \boldsymbol{\sigma}_t = \exp(\boldsymbol{z}_{\sigma,t}), \boldsymbol{\pi}_t = \mathrm{softmax}(\boldsymbol{z}_{\pi,t})$$

- Unlike VAEs, **data likelihood is directly tractable**
- Typically parameters $\boldsymbol{\theta}_t$ are shared for different time steps f_t: use convolution nets, RNNs, Transformers etc.
- Works when there is a natural linear ordering, e.g., time series data
- Not so natural for images

Loss function for NADE

- Maximum likelihood training: minimize negative log-likelihood

$$p(x_t|\boldsymbol{x}_{1:t-1}) = \sum_{k=1}^{K} \pi_{t,k} \mathcal{N}(\boldsymbol{x}_t|\mu_{t,k}, \sigma_{t,k}^2)$$

- In practice, covariances are chosen as diagonal, probability for component k is

$$p_k(x_t|\boldsymbol{x}_{1:t-1}) = \pi_{t,k} \frac{1}{\sqrt{2\pi\sigma_{t,k}^2}} \exp\left(-\frac{(x_t - \mu_{t,k})^2}{2\sigma_{t,k}^2}\right)$$

- Log-likelihood for a single component becomes

$$\mathcal{L}_k(x_t|\boldsymbol{x}_{1:t-1}) = \log(p_k(x_t|\boldsymbol{x}_{1:t-1})) = \log(\pi_{t,k}) - \frac{1}{2}\log(2\pi\sigma_{t,k}^2) - \frac{1}{2}\frac{(x_t - \mu_{t,k})^2}{\sigma_{t,k}^2}$$

- Compute negative log likelihood loss from (use log-sum-exp trick for numerical stability)

$$\text{NLL}(x_t|\boldsymbol{x}_{1:t-1}) = -\log\left(\sum_{k=1}^{K} \exp\left(\mathcal{L}_k(x_t|\boldsymbol{x}_{1:t-1})\right)\right)$$

WaveNet

- Convolutional network for speech and audio using dilated convolutions
- Unconditional model $p(x_t|\boldsymbol{x}_{1:t-1}) = f_t(\boldsymbol{x}_{1:t-1})$ ("decoder-only" in Transformer terminology)
- Conditional version of the model $p(x_t|\boldsymbol{x}_{1:t-1}, c) = f_t(\boldsymbol{x}_{1:t-1}, c)$ can be used for text-to-speech (TTS) synthesis
- Feedforward WaveNets are useful in audio effects modeling (for e.g., guitar amplifiers)

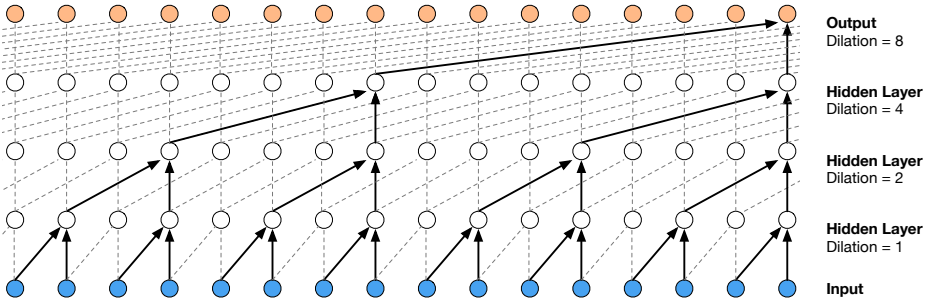
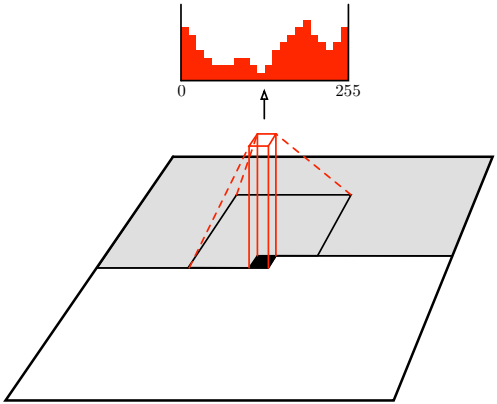


Illustration: Figure 22.2 from Murphy, 2023, orig. van den Oord 2016

PixelCNN

- Autoregressive model for images
- Quantize pixel intensity values and model these as a categorical distribution
- Requires a sequential ordering for pixels, rasterization is a design choice
- Use a 2D CNN architecture with causal mask corresponding to the rasterisation: don't look at the "future" pixels



Language models and Tokenisation

- Language models operate on sequences of discrete **tokens**
- Use the **categorical distribution** to represent token probabilities at each time step (similar to multiclass classification)
- Different approaches to tokenization: character based – word based or something in between

Bernoulli distribution for binary classification)

- The Bernoulli distribution is defined on the domain $z \in (0, 1)$ and has a single parameter λ that denotes the probability of observing $z = 1$. It follows that the probability of observing $z = 0$ is $1 - \lambda$.

$$p(y|\lambda) = \begin{cases} 1-\lambda & y = 0, \\ \lambda & y = 1 \end{cases}$$

or equivalently (in branchless programming mode)

$$p(x|\lambda) = (1-\lambda)^{1-x}\lambda^{x}$$

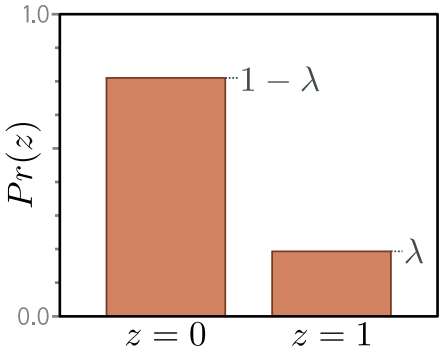


Figure 5.4 from Prince, 2024.

Bernoulli distribution and Binary Cross Entropy loss

A neural network f predicts output $\hat{y}$, given input x and parameters $\boldsymbol{\theta}$

$$\hat{y} = \text{sigmoid}(f(\boldsymbol{x}; \boldsymbol{\theta}))$$

Substitute probability λ with network prediction

$$p(y|\boldsymbol{x}) = (1 - \hat{y})^{1-y} \hat{y}^{y}$$

Log-likelihood over training examples $i = 1, \ldots, I$

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{I} (1 - y_i) \log(1 - \hat{y}_i) + y_i \log(\hat{y}_i)$$

Negative log-likelihood is equivalent to the Binary Cross Entropy Loss

$$\text{NLL}(\boldsymbol{\theta}) = \sum_{i=1}^{I} = -(1 - y_i) \log(1 - \hat{y}_i) y_i \log(\hat{y}_i)$$

Categorical distribution for multiclass classification

Probability of class k in a categorical distribution is

$$p(y = k) = \lambda_k$$

Use the softmax function to normalize network outputs to sum to one

$$p(y = k|\boldsymbol{x}) = \text{softmax}\left(f(\boldsymbol{x}, \boldsymbol{\theta})\right) = \frac{\exp(z_k)}{\sum_{k'}^{K} \exp(z_{k'})}$$

Negative log-likelihood gives the categorical cross entropy loss

$$\begin{aligned} -\mathcal{L}(\boldsymbol{\theta}) &= -\sum_{i=1}^{I} \log(\text{softmax}_{y_i}(f(\boldsymbol{x}, \boldsymbol{\theta}))) \\ &= -\sum_{i=1}^{I} \left(f_{y_i}(\boldsymbol{x}_i, \boldsymbol{\theta}) - \log \left[\sum_{k'=1}^{K} \exp(f_{k'=1}(\boldsymbol{x}_i, \boldsymbol{\theta})) \right] \right) \end{aligned}$$

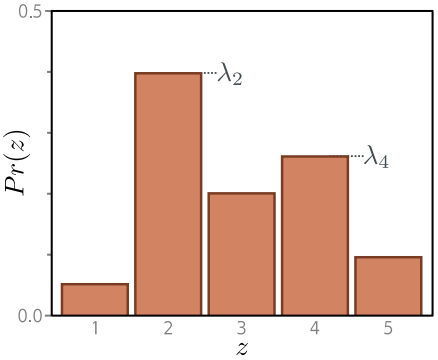


Figure 5.9 from Prince, 2024.

Cross Entropy and Perplexity (Murphy, section 5.2.4)

- KL divergence can be split into cross-entropy and self-entropy

$$D_{\mathbb{KL}}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = \mathbb{H}_{\mathsf{CE}}(p, q) - \mathbb{H}(p)$$

- $\mathbb{H}_{\mathsf{CE}}(p, q)$ is the **cross entropy**

$$\mathbb{H}_{\mathsf{CE}}(p, q) = -\sum_x p(x) \log q(x)$$

- **Entropy** measures the expected number of bits (in base 2, nats in base e) required to encode the outcome of a random variable.
- Perplexity is another commonly used measure to quantify the model's "surprisal" when seeing new data
- Perplexity using natural logarithms is defined as

$$\text{perplexity}_{\text{nats}}(p, q) \triangleq e^{\mathbb{H}_{\mathsf{CE}}(p,q)}$$

- Often perplexity is defined in bits, but this requires using $\log_2$ in entropy calculations

$$\text{perplexity}_{\text{bits}}(p, q) \triangleq 2^{\mathbb{H}_{\mathsf{CE}}(p,q)}$$

Perplexity for empirical distribution

Approximate the true distribution by sampling data from p

$$p_{\mathcal{D}}(x|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(x = x_n)$$

Cross entropy becomes

$$H = \frac{1}{N}\sum_{n=1}^{N}\log p(x_n) = \frac{1}{N}\log\prod_{n=1}^{N}p(x_n)$$

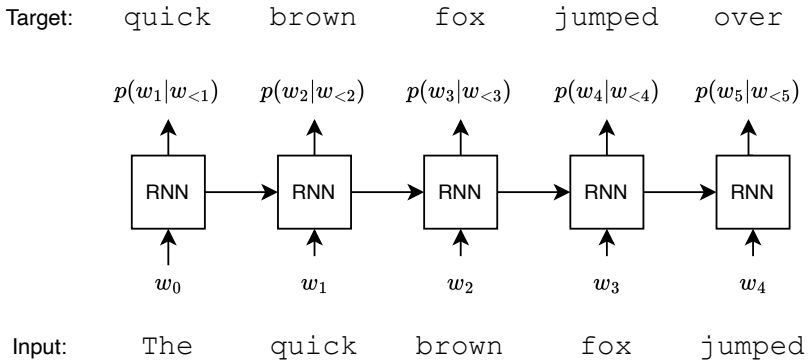
The corresponding perplexity becomes

$$\text{perplexity}(p_{\mathcal{D}}, p) = \exp\left[-\frac{1}{N}\log\left(\prod_{n=1}^{N}p(x_n)\right)\right] = \exp\left[\log\left(\prod_{n=1}^{N}p(x_n)^{-\frac{1}{N}}\right)\right]$$

$$= \left(\prod_{n=1}^{N}p(x_n)^{-\frac{1}{N}}\right) = \sqrt[N]{\prod_{n=1}^{N}\frac{1}{p(x_n)}}$$

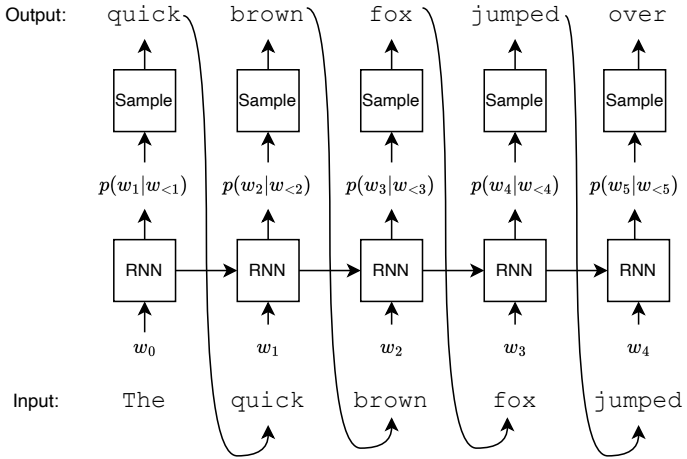
Training autoregressive models with Teacher Forcing

- Target and input are shifted versions of the same sequence
- During training, the entire sequence is observed and next token predictions can be computed in parallel.
- **Teacher forcing** refers to conditioning on the ground-truth observed context history.



Autoregressive inference

- At inference, the model has to rely on previously generated inputs for context
- The distribution of generated sequences may differ from the real data distribution
- This contributes to language model **hallucination** effects (i.e., generating unlikely outputs)



Architecture and data distribution choice for autoregression

- Distribution needs to have tractable explicit density function

$$p_k(x_t|\boldsymbol{x}_{1:t-1}, \boldsymbol{\phi}_t) = \dots$$

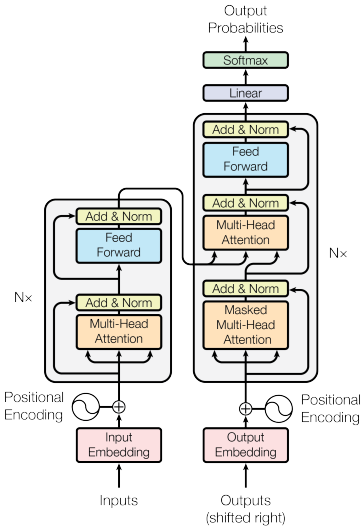
- Mixtures of Gaussians, Logistics or other members of the location-scale familiy
- Bernoulli, Categorical or other members of the Multinoulli family
- Neural network predicts distribution parameters, re-parametrize so that distribution parameter properties are fulfilled

$$\boldsymbol{\phi}_t = f(\boldsymbol{x}_{1:t-1}; \boldsymbol{\theta})$$

- Causality is really the only requirement. Typical implementations use combinations of RNNs, Causal CNNs, and Masked Attention Transformers.
- Train the model using a NLL loss
- During inference, sample from the distribution and feed back generated samples

Transformer language models

- Only use the autoregressive decoder from the Transformer (right side)
- Masked self-attention: causal mask only allows looking at past time-steps
- Add & Norm - Residual connection and layer normalization
- No cross-attention needed in this case
- Feed-forward block is a simple MLP without temporal connectivity



Self-attention (Bishop 2024, section 12.1.)

- Positional encoding is defined as

$$r_{ni} = \begin{cases} \sin\left(\frac{n}{L^{i/D}}\right), & \text{if } i \text{ is even,} \\ \cos\left(\frac{n}{L^{(i-1)/D}}\right), & \text{if } i \text{ is odd.} \end{cases}$$

- n is timestep, i indexes feature dimension, L is expected maximum length (10000), D is the total number of dimensions
- This corresponds to multiple clock signals that wrap around at different rates

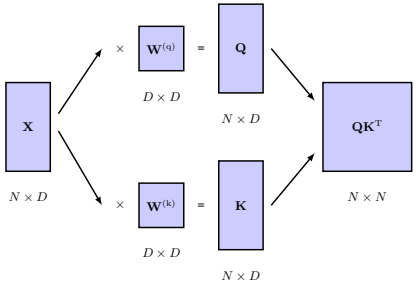


Figure 12.4 from Bishop, 2024.

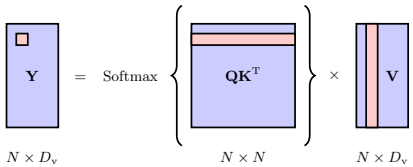


Figure 12.5 from Bishop, 2024.

Positional encoding (Bishop section 12.1.9)

- Positional encoding is defined as

$$r_{ni} = \begin{cases} \sin\left(\frac{n}{L^{i/D}}\right), & \text{if } i \text{ is even,} \\ \cos\left(\frac{n}{L^{(i-1)/D}}\right), & \text{if } i \text{ is odd.} \end{cases}$$

- n is timestep, i indexes feature dimension, L is expected maximum length (10000), D is the total number of dimensions
- This corresponds to multiple clock signals that wrap around at different rates

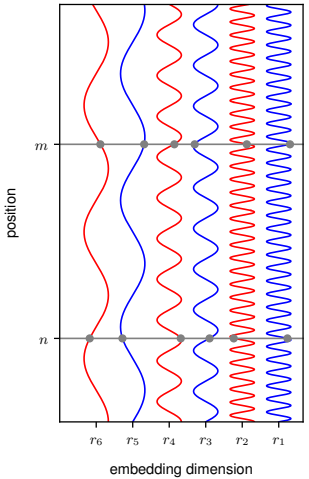


Figure 12.10a from Bishop, 2024.

Tokenisation

- Autoregressive language models operate with categorical distributions over a set of symbols, i.e., tokens
- Word-based tokenisation leads to large dictionary sizes (typically 50-100k in English)
- There will still be many unknown tokens for missing words, and it's difficult to handle named entities, etc.
- Agglutinative languages like Finnish are especially bad in word form combinations
- Character-level tokenisations are simple and cover everything, but they throw out all semantics
- Purely character based sequences are also longer and harder to model (remember $\mathcal{O}(T^2)$ scaling with Attention)
- How about something in-between - how to construct sub-word tokenisations?

Byte pair encoding

- Method for automatically creating sub-word tokenisations for language models, used in e.g., GPT models
- Start by splitting the text corpus into individual characters
- Count the frequencies of token pairs and merge the most common pair into a new token
- Repeat counting and merging until desired vocabulary size is reached

Byte pair encoding

- Method for automatically creating sub-word tokenisations for language models, used in e.g., GPT models
- Start by splitting the text corpus into individual characters
- Count the frequencies of token pairs and merge the most common pair into a new token
- Repeat counting and merging until desired vocabulary size is reached

Example:

- Let's look at a text corpus that consists of the following words
- `"hug", "pug", "pun", "bun", "hugs"`
- Base vocabulary will then be
- `["b", "g", "h", "n", "p", "s", "u"]`

Source `https://huggingface.co/learn/llm-course/chapter6/5`

Byte pair encoding example continued

- Let's look at a text corpus that consists of the following words
- `"hug", "pug", "pun", "bun", "hugs"`
- Base vocabulary will then be
- `["b", "g", "h", "n", "p", "s", "u"]`
- Assume the words have the following counts
- `("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`

Byte pair encoding example continued

- Let's look at a text corpus that consists of the following words
- `"hug", "pug", "pun", "bun", "hugs"`
- Base vocabulary will then be
- `["b", "g", "h", "n", "p", "s", "u"]`
- Assume the words have the following counts
- `("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`
- Then the token counts will be `("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)`
- Lets count token pairs, for example the pair `("h", "u")` is present in the words "hug" and "hugs", so 15 times total in the corpus.
- What pair has the highest frequency?

Byte pair encoding example continued

- Let's look at a text corpus that consists of the following words
- `"hug", "pug", "pun", "bun", "hugs"`
- Base vocabulary will then be
- `["b", "g", "h", "n", "p", "s", "u"]`
- Assume the words have the following counts
- `("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`
- Then the token counts will be `("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)`
- Lets count token pairs, for example the pair `("h", "u")` is present in the words "hug" and "hugs", so 15 times total in the corpus.
- What pair has the highest frequency?
- ("u", "g") is present in "hug", "pug", and "hugs", for a total of 20 times in the vocabulary.
- Merge `("u", "g") -> "ug"` into a new token

Byte pair encoding example continued

- After merging, our vocabulary and corpus look like this

  Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug"]`

  Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`
- Count pair frequencies again, e.g., `("h", "ug")` now occurs 15 times
- What is the most frequent pair now?

Byte pair encoding example continued

- After merging, our vocabulary and corpus look like this

  Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug"]`

  Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`
- Count pair frequencies again, e.g., `("h", "ug")` now occurs 15 times
- What is the most frequent pair now?
- `("u", "n")` occurs 16 times, merge `("u", "n") -> "un"`
- After merging, our vocabulary and corpus look like this

  Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug", "un"]`

  Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)`
- Now the most frequent pair is `("h", "ug")`, so we apply the merge rule `("h", "ug") -> "hug"`,

Byte pair encoding example continued

- After merging, our vocabulary and corpus look like this

  Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug"]`

  Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`
- Count pair frequencies again, e.g., `("h", "ug")` now occurs 15 times
- What is the most frequent pair now?
- `("u", "n")` occurs 16 times, merge `("u", "n") -> "un"`
- After merging, our vocabulary and corpus look like this

  Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug", "un"]`

  Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)`
- Now the most frequent pair is `("h", "ug")`, so we apply the merge rule `("h", "ug") -> "hug"`,
- After merging, our vocabulary and corpus look like this

  Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]`

  Corpus: `("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)`
- Continue this until target vocabulary size is reached. **Try this yourself in the home exercise**.

Temperature

- Temperature is a smoothing parameter used to adjust the level of randomness in a categorical distribution
- Scale the logits with temperature parameter T and apply softmax to normalize

$$\text{softmax}(\boldsymbol{z}, T) = \frac{\exp(z_i/T)}{\sum_{k=1}^{K} \exp(z_k/T)}$$

- Example: softmax$(\boldsymbol{a}/T)$ distribution where $\boldsymbol{a} = (3, 0, 1)$

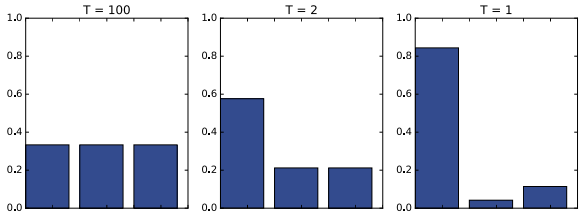


Figure: Figure 14.3. in Murphy, 2023

Summary

- Autoregressive models
- Distributions used in autoregressive modeling
- Deep neural networks for autoregression
- Autoregreressive Transformer language models
- Tokenisation and byte pair encoding

References

- Murphy K, Probabilistic Machine Learning: Advanced Topics, 2023.
- Prince SJD, Understanding Deep Learning, The MIT Press, 2023.
- Bishop, Christopher M., and Hugh Bishop. Deep learning: Foundations and concepts. Springer Nature, 2023.