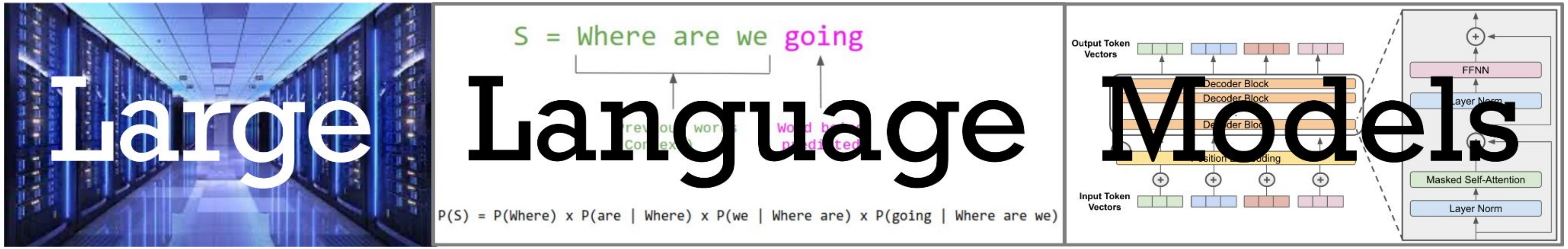


CIS 7000 - Fall 2024

Background

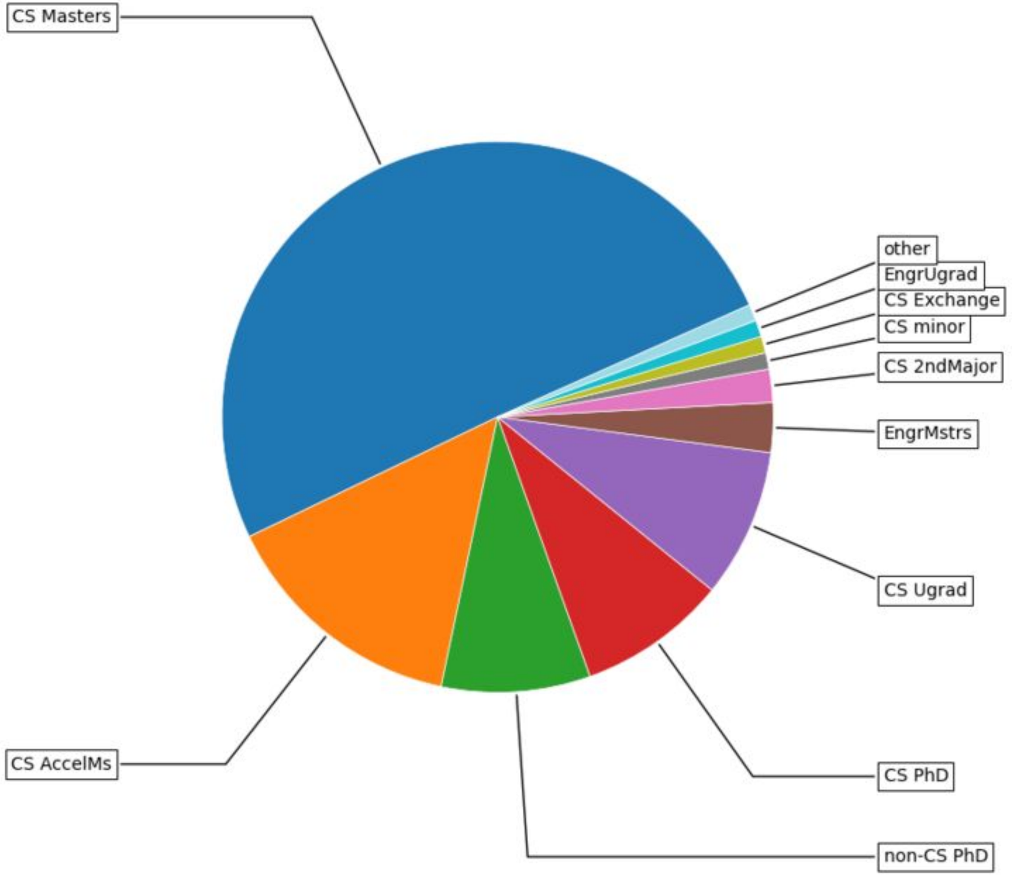
Professor Mayur Naik

Slides adapted in part from UC Berkeley's CS182: Designing, Visualizing and Understanding Deep Neural Networks (Spring'21) and Chapter 7 of Jurafsky/Martin's book "Speech and Language Processing" (3rd ed).

Recap of Last Lecture

- The Turing Test
- Overview of LLMs
  - How do LLMs work, What LLMs can do, Limitations of LLMs, What is the future
- Course Logistics

Who’s In CIS 7000?



Announcements

- **Homework 0** “Exploring LLMs” is due on **Sunday Sept 8 at 11:59 pm ET**. Available via Canvas and https://llm-class.github.io/homeworks.html. Late submissions will not be accepted!

- **Homework 1** “Transformer from Scratch” will be released on **Monday Sept 9**; much more work than Homework 0! Due in 3 parts: **Wed Sept 18** (Part 1), **Wed Sept 25** (Part 2), and **Wed Oct 2 at 11:59 pm ET** (Final).

Today’s Agenda

- Language Modeling
- Perplexity Evaluation
- Feedforward Neural Networks
  - Forward pass
  - Loss function
  - Back-propagation

What is Language Modeling?

The task of computing $P(w \mid h)$, the probability distribution of possible words w from a vocabulary given some history (sequence of words) h.

… and thanks for all the _____

fish	0.70
memories	0.10
support	0.05
help	0.05
love	0.04
time	0.02
work	0.02
fun	0.01
…	…

What is Language Modeling?

The task of computing $P(w \mid h)$, the probability distribution of possible words w from a vocabulary given some history (sequence of words) h.

Enables to compute probabilities of entire sentences by applying **chain rule of probability**:

$P(w_1\ w_2 \ldots w_k) = P(w_1).P(w_2|w_1).P(w_3|w_{1:2})\ldots P(w_k|w_{1:k-1}) = \Pi_{i=1}^{k} P(w_i|w_{1:i-1})$

Question: Why is this task important?

Motivation for Language Modeling

- Generating more plausible sentences.
    - *P*("I saw a van") > *P*("eyes awe of an")
    - Many NLP applications: correcting grammar or spelling errors, machine translation, speech recognition, content summarization, conversational agents, etc.

- More importantly, Large Language Models are built by training them on this task!

An Approximation

- A problem: computing *P(w | h)* exactly is infeasible for arbitrary history *h* since language is **creative** and *h* might have never occurred before!

- Idea: the Markov assumption: approximate the history by just the last few words.

$$P(w_k|w_{1:k-1}) \approx P(w_k|w_{k-n+1:k-1})$$

- Example: n-gram models: look at n-1 words in the history. n=2 is bigrams, n=3 is trigrams, etc.

  LLMs use *much* larger n, in the thousands or even millions!

Estimating Probabilities

Goal: Train a language model P_θ with parameters (weights) θ such that $P_\theta(h)$ computes a probability distribution over the vocabulary of all possible words.

Start with a dataset $D_{train} = \{ (h_1, w_1), \ldots, (h_n, w_n) \}$.

Assumption: i.i.d. (independent and identically distributed)

Question: What is the objective of this model?

A good model is one that makes the data look probable. Therefore, choose θ such that

$$P(D_{train}) = \prod_{i=1}^{n} P(h_i).P_\theta(w_i \mid h_i) \text{ is maximized.}$$

Multiplying Probabilities

$$P(D_{train}) = \prod_{i=1}^{n} P(h_i).P_\theta(w_i \mid h_i)$$

$\leftarrow$ Multiplying together many numbers <= 1

$$\log P(D_{train}) = \sum_{i=1}^{n} \log P(h_i) + \log P_\theta(w_i \mid h_i) = \sum_{i=1}^{n} \log P_\theta(w_i \mid h_i) + \cancel{\text{const}}$$

$$\theta^* \leftarrow \underset{\theta}{\arg\max} \sum_{i=1}^{n} \log P_\theta(w_i \mid h_i)$$ maximum likelihood estimation (MLE)

This is our **loss function**

$$\theta^* \leftarrow \underset{\theta}{\arg\min} -\frac{1}{n} \sum_{i=1}^{n} \log P_\theta(w_i \mid h_i)$$ negative log likelihood (NLL)

Also called *cross-entropy*

Evaluation: How Good is Our Model?

Suppose we train P_θ (more on how to do this later in today's lecture!)

How do we tell how good our LM is?

Does our LM prefer good sentences over bad ones?

- Assign higher probability to real or frequently observed sentences than ungrammatical or rarely observed ones?

Train the parameters of the LM on a **training set** (D_{train}).

Test the LM's performance on data we haven't seen.

- A **test set** is an unseen dataset different from D_{train}, totally unused.
- An **evaluation metric** tells us how well our model does on test set.

Extrinsic Evaluation of LMs

Best evaluation for comparing LMs A and B.

Put each model in a task (e.g. spelling corrector, speech recognizer, machine translation system).

Run the task and get an accuracy for A and B.

- (e.g. how many misspelled words corrected properly, how many words translated properly).

Whichever model has higher accuracy is better.

Problem: Time-consuming.

Intrinsic Evaluation of LMs

Common intrinsic evaluation metric for LMs: **perplexity**.

Bad approximation unless the test data looks *just* like the training data.

Generally only useful in pilot experiments.

But it is helpful to think about (as long as extrinsic evaluation is also done).

Let’s look at different intuitions of perplexity and define it!

Intuition of Perplexity

The Shannon Game: how well can we predict the next word in a given sentence?

I always order pizza with cheese and ______

The 33rd President of the US was ______

I saw a _____

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1e-100

A good LM is one that assigns a higher probability to the word that actually occurs.

Perplexity

The best LM is one that best predicts an unseen test set D_{test}.

Perplexity is 2^J where J is cross entropy loss on test set: $J = -\frac{1}{n}\sum_{i=1}^{n} \log P_\theta(w_i \mid h_i)$

Perplexity >= 1. Lower is better.

Equivalently, perplexity of a sentence $w_1\ w_2 \ldots w_n$ is $P_\theta(w_1\ w_2 \ldots w_n)^{-\frac{1}{n}}$ which is the same as $\sqrt[n]{\frac{1}{\Pi_{i=1}^{n} P_\theta(w_i|w_{1:i-1})}}$

That is, it is the probability of the sentence normalized by the number of words.

Another Intuition of Perplexity

Perplexity is the **average branching factor** at any point in a sentence.

Example 1: task is to recognize sentence of random digits. Average branching factor at each step is 10, so perplexity is 10.

Example 2: task is to recognize Operator (1 in 4), Sales (1 in 4), Tech Support (1 in 4), and 30,000 names (1 in 120,000 each). Perplexity is ~ 53.

Lower Perplexity = Better Model

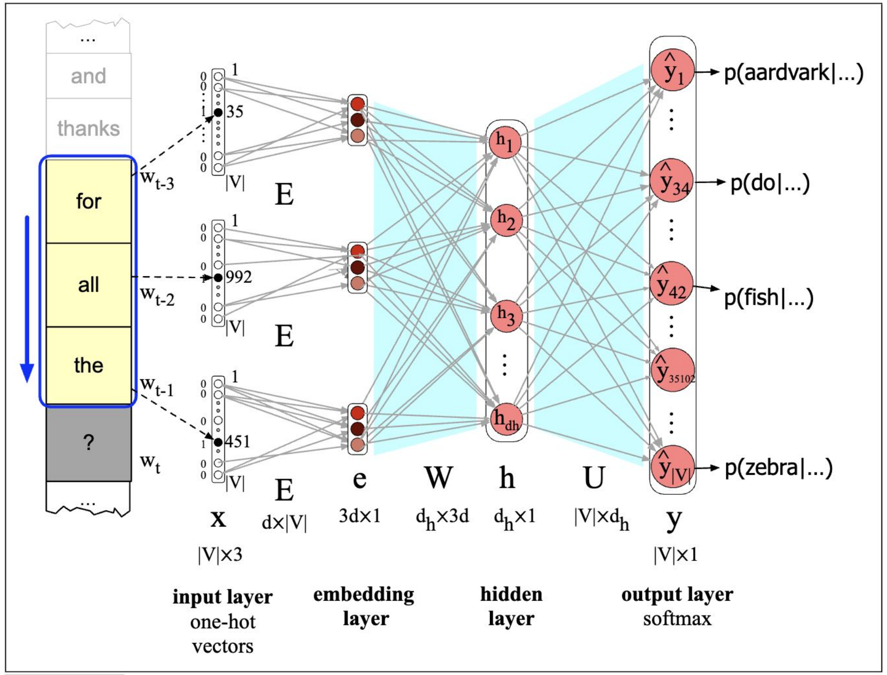
Training on 38 million words and testing on 1.5 million words from the WSJ.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Further Reading: Hugging Face doc article on Perplexity of fixed-length models.

Feedforward Neural Network for Language Modeling

Architecture and Illustration of Forward Pass

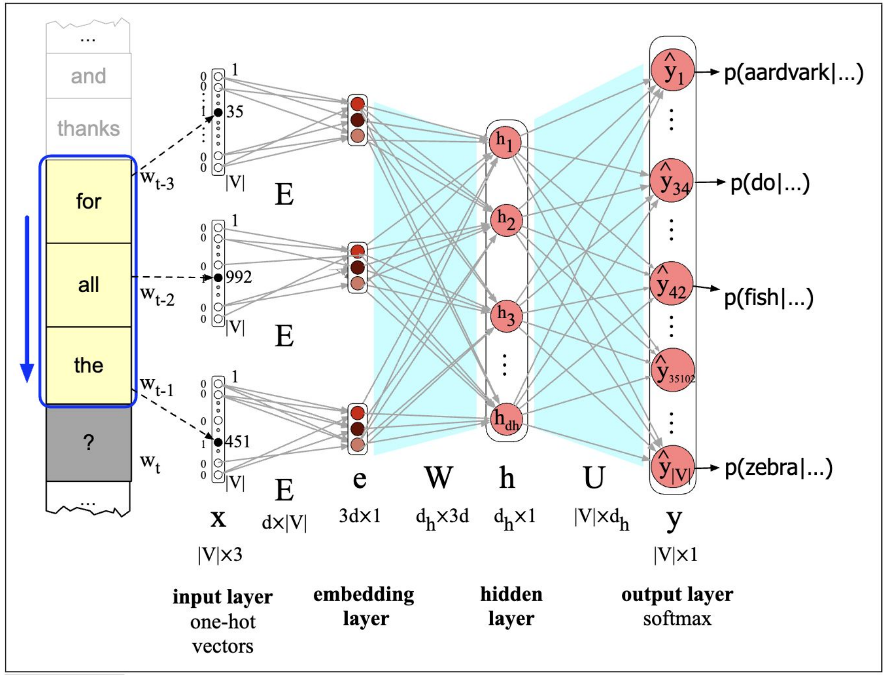


Sketch of feedforward neural language model with 3-word input context.

Picture Credit: Chapter 7 of Jurafsky/Martin's book "Speech and Language Processing" (3rd ed).

Y. Benjio et al. [A Neural Probabilistic Language Model](). JMLR 2003.

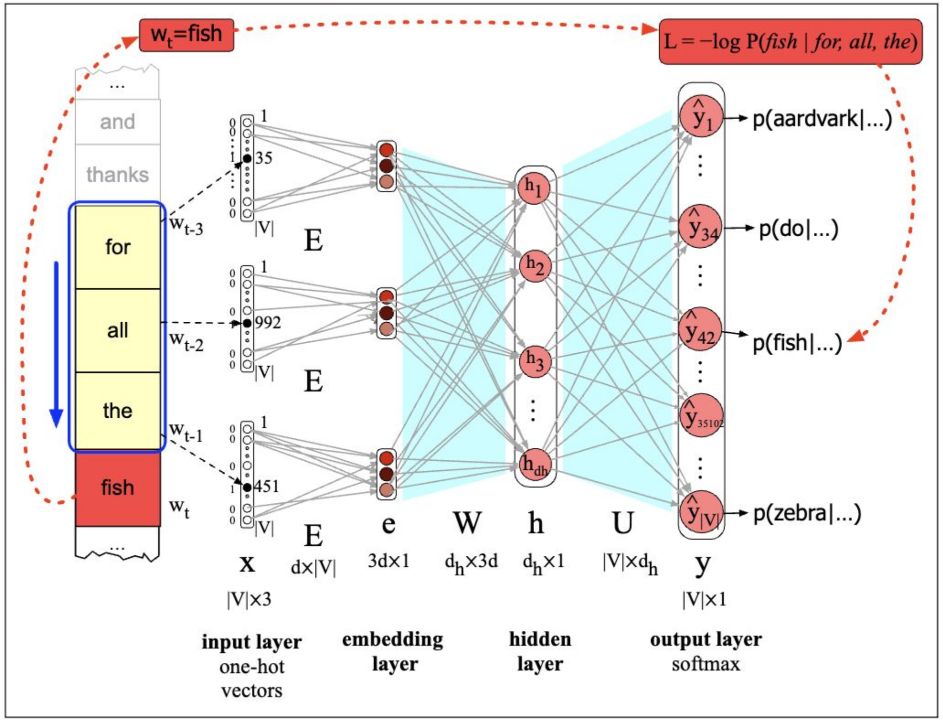
Architecture and Illustration of Forward Pass



At each timestep **t**, the network:

1. computes a **d**-dimensional embedding for each context word (by multiplying a one-hot vector by embedding matrix **E**)
2. concatenates the 3 resulting embeddings to produce embedding layer **e**
3. **e** is multiplied by a weight matrix **W** and then an activation function is applied element-wise to produce hidden layer **h**
4. **h** is then multiplied by another weight matrix **U**
5. finally, a softmax output layer predicts at each node **i** the probability that the next word $\mathbf{w_t}$ will be vocabulary word $\mathbf{V_i}$

Loss Function: Negative Log Likelihood (NLL)

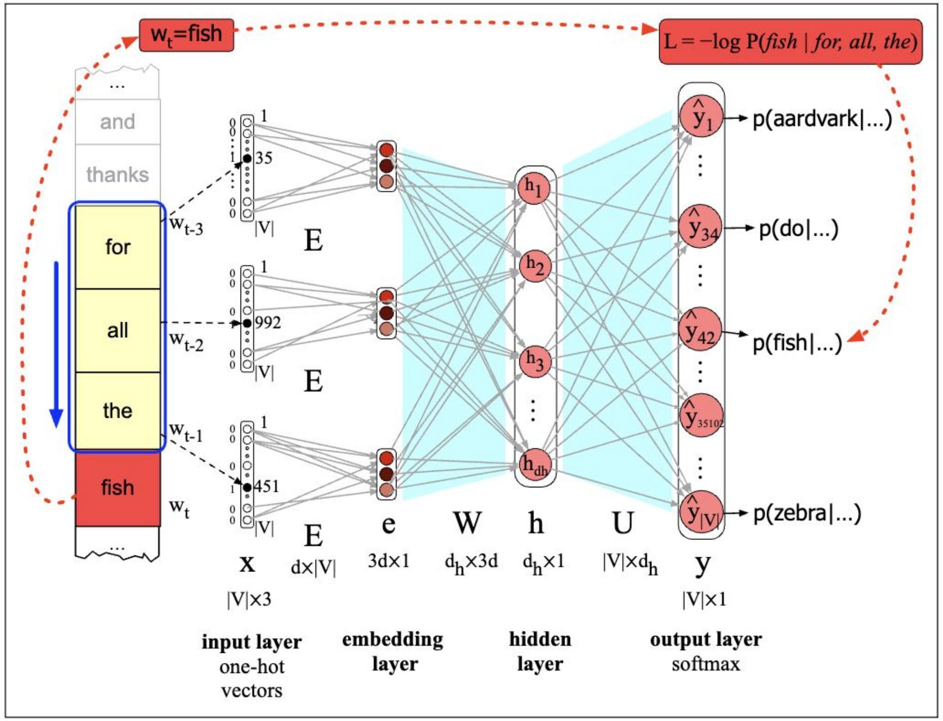


The parameter update for stochastic gradient descent for cross-entropy loss L from step s to s+1 is:

$$\theta^{s+1} = \theta^{s} - \eta \frac{\partial\left[-\log p(w_t|w_{t-1},\ldots,w_{t-n+1})\right]}{\partial\theta}$$

This gradient can be computed in any standard neural network framework (e.g. Pytorch) which will then backpropagate through θ = **E**, **W**, **U**.

Forward Pass and Loss Function in Equations



$\mathbf{e} = [\mathbf{E}\,\mathbf{x}_{t-3}; \mathbf{E}\,\mathbf{x}_{t-2}; \mathbf{E}\,\mathbf{x}_{t-1}]$

$\mathbf{h} = \sigma\,(\mathbf{W}\,\mathbf{e})$

$\mathbf{z} = \mathbf{U}\,\mathbf{h}$

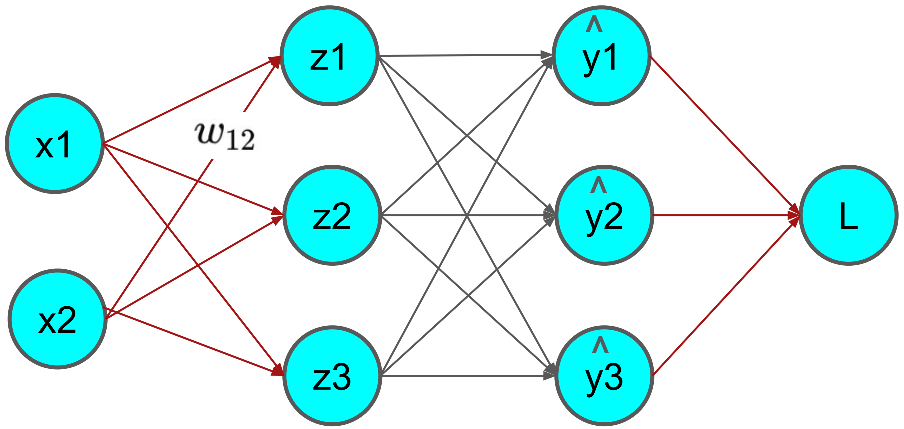
$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$

$L = -\,\mathbf{y}\,(\log \hat{\mathbf{y}})^{T}$

where $\mathbf{y}$ is one-hot vector representing ground truth.

Short Primer on Back-Propagation

Let's consider a simple feedforward network:



$$\mathbf{z} = \mathbf{W}\,\mathbf{x}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

$$L = -\,\mathbf{y} \log \hat{\mathbf{y}}$$

where **y** is one-hot vector representing ground truth.

where $\hat{\mathbf{y}}_i = \frac{e^{z_i}}{\sum_{j=1}^{3} e^{z_j}}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix} \quad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix} \quad L = -\sum_{i=1}^{3} y_i \log(\hat{y}_i)$$

Short Primer on Back-Propagation

Computation Graph: (linear in this example but a DAG in general)

$$\begin{tikzcd}
\mathbf{W} \arrow[r] & \mathbf{z} \arrow[r] & \hat{\mathbf{y}} \arrow[r] & \mathbf{L}
\end{tikzcd}$$

$$\boxed{\begin{aligned} &\mathbf{z} = \mathbf{W}\,\mathbf{x} \\ &\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}) \\ &\mathbf{L} = -\,\mathbf{y}\log\hat{\mathbf{y}} \end{aligned}}$$

Backward Differentiation:

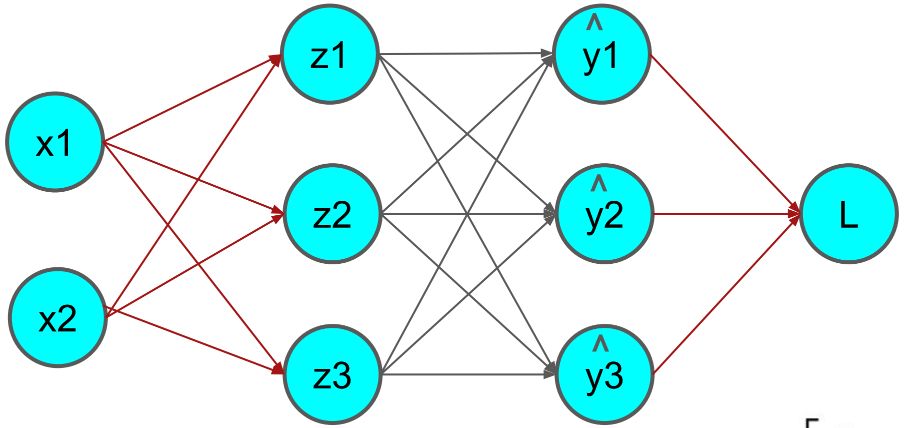
$$\begin{tikzcd}
\mathbf{W} \arrow[r] & \mathbf{z} \arrow[r] \arrow[l, bend left, "\frac{\partial \mathbf{z}}{\partial \mathbf{W}}"] & \hat{\mathbf{y}} \arrow[r] \arrow[l, bend left, "\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}}"] & \mathbf{L} \arrow[l, bend left, "\frac{\partial L}{\partial \hat{\mathbf{y}}}"]
\end{tikzcd}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

(by chain rule of probability)

Short Primer on Back-Propagation

Gradient of Loss with Respect to Predicted Probabilities:



$\mathbf{z} = \mathbf{W}\,\mathbf{x}$

$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{z})$

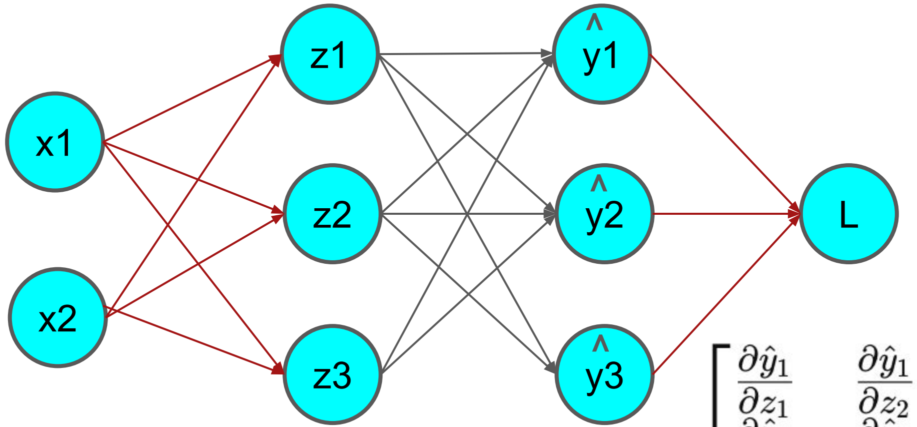
$L = -\,\mathbf{y} \log \hat{\mathbf{y}}$

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} \to \begin{bmatrix} \frac{\partial L}{\partial \hat{y}_1} & \frac{\partial L}{\partial \hat{y}_2} & \frac{\partial L}{\partial \hat{y}_3} \end{bmatrix} = \begin{bmatrix} -\frac{y_1}{\hat{y}_1} & -\frac{y_2}{\hat{y}_2} & -\frac{y_3}{\hat{y}_3} \end{bmatrix}$$

since $L = -\sum_{i=1}^{3} y_i \log(\hat{y}_i)$ and $\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$

Short Primer on Back-Propagation

Jacobian of Softmax with Respect to Logits:



$\mathbf{z} = \mathbf{W}\,\mathbf{x}$

$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$

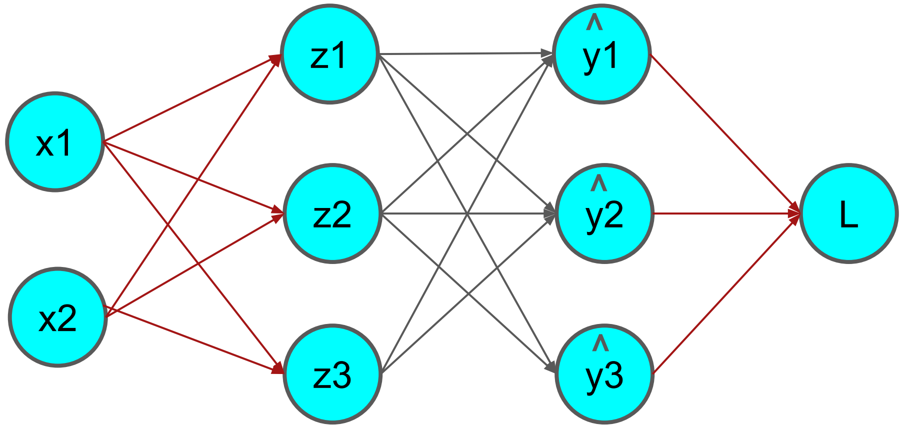
$L = -\,\mathbf{y} \log \hat{\mathbf{y}}$

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}} \longrightarrow \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial z_1} & \frac{\partial \hat{y}_1}{\partial z_2} & \frac{\partial \hat{y}_1}{\partial z_3} \\ \frac{\partial \hat{y}_2}{\partial z_1} & \frac{\partial \hat{y}_2}{\partial z_2} & \frac{\partial \hat{y}_2}{\partial z_3} \\ \frac{\partial \hat{y}_3}{\partial z_1} & \frac{\partial \hat{y}_3}{\partial z_2} & \frac{\partial \hat{y}_3}{\partial z_3} \end{bmatrix} = \begin{bmatrix} \hat{y}_1(1-\hat{y}_1) & -\hat{y}_1\hat{y}_2 & -\hat{y}_1\hat{y}_3 \\ -\hat{y}_2\hat{y}_1 & \hat{y}_2(1-\hat{y}_2) & -\hat{y}_2\hat{y}_3 \\ -\hat{y}_3\hat{y}_1 & -\hat{y}_3\hat{y}_2 & \hat{y}_3(1-\hat{y}_3) \end{bmatrix}$$

since $\hat{y}_i = \dfrac{e^{z_i}}{\sum_{k=1}^{3} e^{z_k}}$ and $\dfrac{\partial \hat{y}_i}{\partial z_j} = \begin{cases} \hat{y}_i(1-\hat{y}_i) & \text{if } i = j \\ -\hat{y}_i\hat{y}_j & \text{if } i \neq j \end{cases}$

Short Primer on Back-Propagation

Jacobian of Logits with Respect to Weights:



$\mathbf{z} = \mathbf{W}\,\mathbf{x}$

$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$

$L = -\,\mathbf{y} \log \hat{\mathbf{y}}$

$$\frac{\partial \mathbf{z}}{\partial \mathbf{W}} \quad \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \frac{\partial z_1}{\partial W_{12}} & \frac{\partial z_1}{\partial W_{21}} & \frac{\partial z_1}{\partial W_{22}} & \frac{\partial z_1}{\partial W_{31}} & \frac{\partial z_1}{\partial W_{32}} \\ \frac{\partial z_2}{\partial W_{11}} & \frac{\partial z_2}{\partial W_{12}} & \frac{\partial z_2}{\partial W_{21}} & \frac{\partial z_2}{\partial W_{22}} & \frac{\partial z_2}{\partial W_{31}} & \frac{\partial z_2}{\partial W_{32}} \\ \frac{\partial z_3}{\partial W_{11}} & \frac{\partial z_3}{\partial W_{12}} & \frac{\partial z_3}{\partial W_{21}} & \frac{\partial z_3}{\partial W_{22}} & \frac{\partial z_3}{\partial W_{31}} & \frac{\partial z_3}{\partial W_{32}} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{bmatrix}$$

since $z_j = \sum_{n=1}^{2} W_{jn} x_n$ and $\frac{\partial z_j}{\partial W_{mn}} = \begin{cases} x_n & \text{if } j = m \\ 0 & \text{if } j \neq m \end{cases}$

Short Primer on Back-Propagation

Putting it all together:

$\mathbf{z} = \mathbf{W}\,\mathbf{x}$

$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$

$\mathbf{L} = -\,\mathbf{y}\log\hat{\mathbf{y}}$

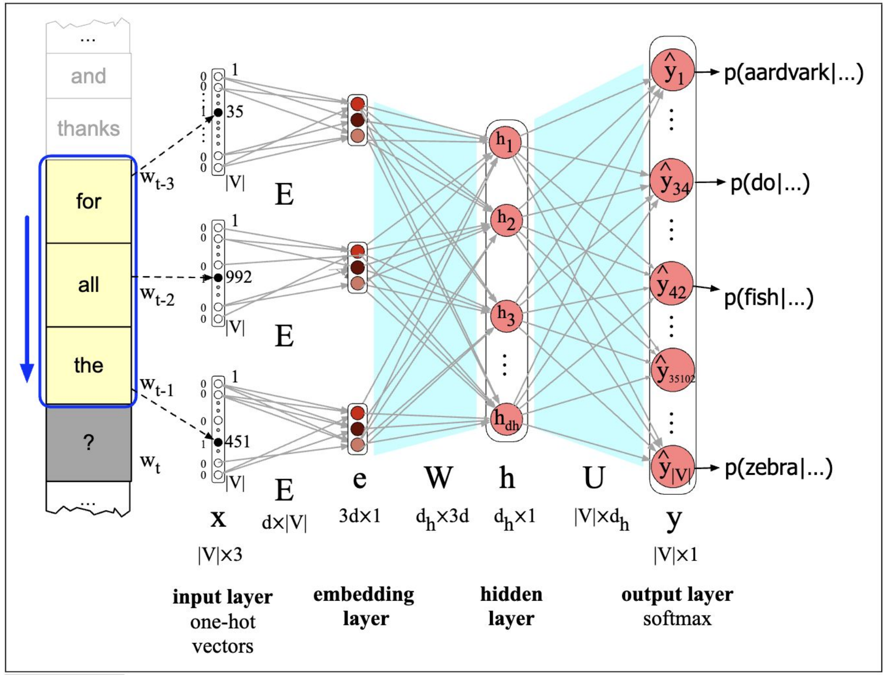
$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

$$= \begin{bmatrix} \frac{\partial L}{\partial \hat{y}_1} & \frac{\partial L}{\partial \hat{y}_2} & \frac{\partial L}{\partial \hat{y}_3} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial z_1} & \frac{\partial \hat{y}_1}{\partial z_2} & \frac{\partial \hat{y}_1}{\partial z_3} \\ \frac{\partial \hat{y}_2}{\partial z_1} & \frac{\partial \hat{y}_2}{\partial z_2} & \frac{\partial \hat{y}_2}{\partial z_3} \\ \frac{\partial \hat{y}_3}{\partial z_1} & \frac{\partial \hat{y}_3}{\partial z_2} & \frac{\partial \hat{y}_3}{\partial z_3} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial z_1}{\partial W_{11}} & \frac{\partial z_1}{\partial W_{12}} & \frac{\partial z_1}{\partial W_{21}} & \frac{\partial z_1}{\partial W_{22}} & \frac{\partial z_1}{\partial W_{31}} & \frac{\partial z_1}{\partial W_{32}} \\ \frac{\partial z_2}{\partial W_{11}} & \frac{\partial z_2}{\partial W_{12}} & \frac{\partial z_2}{\partial W_{21}} & \frac{\partial z_2}{\partial W_{22}} & \frac{\partial z_2}{\partial W_{31}} & \frac{\partial z_2}{\partial W_{32}} \\ \frac{\partial z_3}{\partial W_{11}} & \frac{\partial z_3}{\partial W_{12}} & \frac{\partial z_3}{\partial W_{21}} & \frac{\partial z_3}{\partial W_{22}} & \frac{\partial z_3}{\partial W_{31}} & \frac{\partial z_3}{\partial W_{32}} \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{y_1}{\hat{y}_1} & -\frac{y_2}{\hat{y}_2} & -\frac{y_3}{\hat{y}_3} \end{bmatrix} \cdot \begin{bmatrix} \hat{y}_1(1-\hat{y}_1) & -\hat{y}_1\hat{y}_2 & -\hat{y}_1\hat{y}_3 \\ -\hat{y}_2\hat{y}_1 & \hat{y}_2(1-\hat{y}_2) & -\hat{y}_2\hat{y}_3 \\ -\hat{y}_3\hat{y}_1 & -\hat{y}_3\hat{y}_2 & \hat{y}_3(1-\hat{y}_3) \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{bmatrix}$$

$$= \begin{bmatrix} (\hat{y}_1 - y_1)x_1 & (\hat{y}_1 - y_1)x_2 & (\hat{y}_2 - y_2)x_1 & (\hat{y}_2 - y_2)x_2 & (\hat{y}_3 - y_3)x_1 & (\hat{y}_3 - y_3)x_2 \end{bmatrix}$$

Pros/Cons of Feedforward Neural Network as Language Model



Improvements over n-gram LM:

- No sparsity problem.
- No need to store all observed n-grams.

Remaining problems:

- Fixed window is too small.
- Model size (**W**) increases for longer input context.
- Window can never be large enough!
- Each $\mathbf{x}_i$ is multiplied by completely different weights in **W**, so no symmetry in how the inputs are processed.

Up Next …

- **Homework 1** “Transformer from Scratch” will be released on **Sept 9**.
- **Sept 9** Lecture: From Pytorch to Hugging Face: How to run your own LLM

  Involves hands-on Jupyter Notebook exercises by TAs!