LANGUAGE & AI: DEEP LEARNING

Chris Emmery
Department of Cognitive Science & AI
Tilburg University

@cmry • @_cmry • @cmry • cmry.github.io

♻️ RECAP PREVIOUS LECTURES

- We looked at how language might be **noisy** as input.
- We discussed several way to **represent** and **model** language: count and prediction-based, and sequential.
- We looked at framing **prediction** tasks around those respective representations.

> *Today, we discuss Deep Learning: end-to-end representation learning and predictions in one black box.*

🔁 RECURRENT MODELS

⬅️ RELATION TO MATERIAL

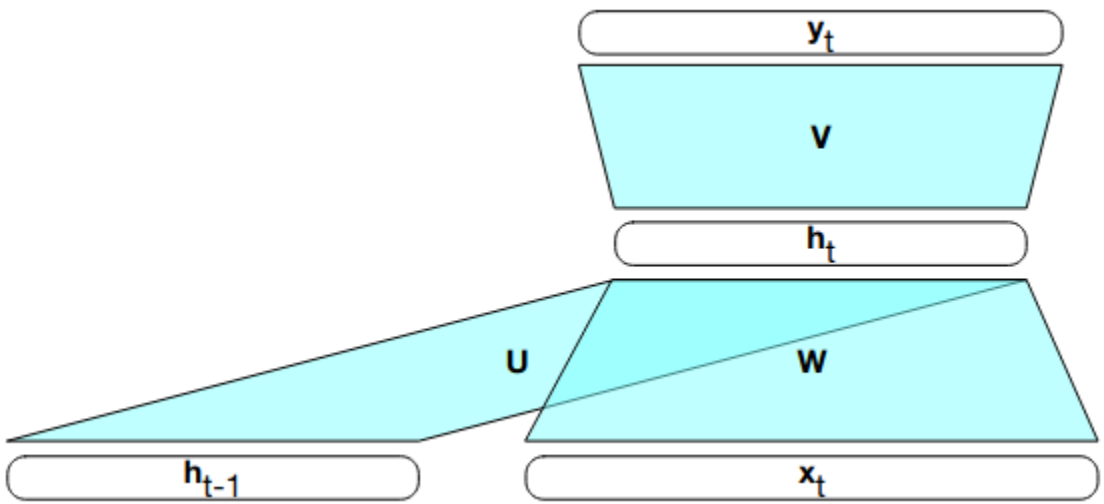
- Representing word similarity (Weeks 1 and 4).
- Predicting outputs given input (Weeks 3 and 5).
- Sequential (or temporal) structure (Week 5).
- Semantic representations (Week 4)
- Context windows (Week 4).

RECURRENT NEURAL NETWORKS (RNNS)

- Recurrent unit: encodes earlier steps in future decisions via 'lookback' for the hidden layers.

🐓 HOW DO THEY WORK?

Magic? No, just **U**.



🥚 FORMALLY

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{z} = \mathbf{U}\mathbf{h}$$
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

$\rightarrow$

$$\mathbf{h}_t = g\left(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t\right)$$
$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

- Dimensionality given d_{in}, d_h, and d_{out}:
- $\mathbf{W} \in \mathbb{R}^{d_h \times d_{in}}$,
- $\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$, and
- $\mathbf{V} \in \mathbb{R}^{d_{\text{out}} \times d_h}$.

RNNS AS LMS

$$\mathbf{e} = \left[\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}\right]$$
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e} + \mathbf{b})$$
$$\mathbf{z} = \mathbf{U}\mathbf{h}$$
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g\left(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t\right)$$
$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

Actual probabilities as dot product:

$$P(w_{t+1} = i \mid w_1, \ldots, w_t) = \mathbf{y}_t[i]$$

$$P(w_{1:n}) = \prod_{i=1}^{n} \mathbf{y}_i[w_i]$$

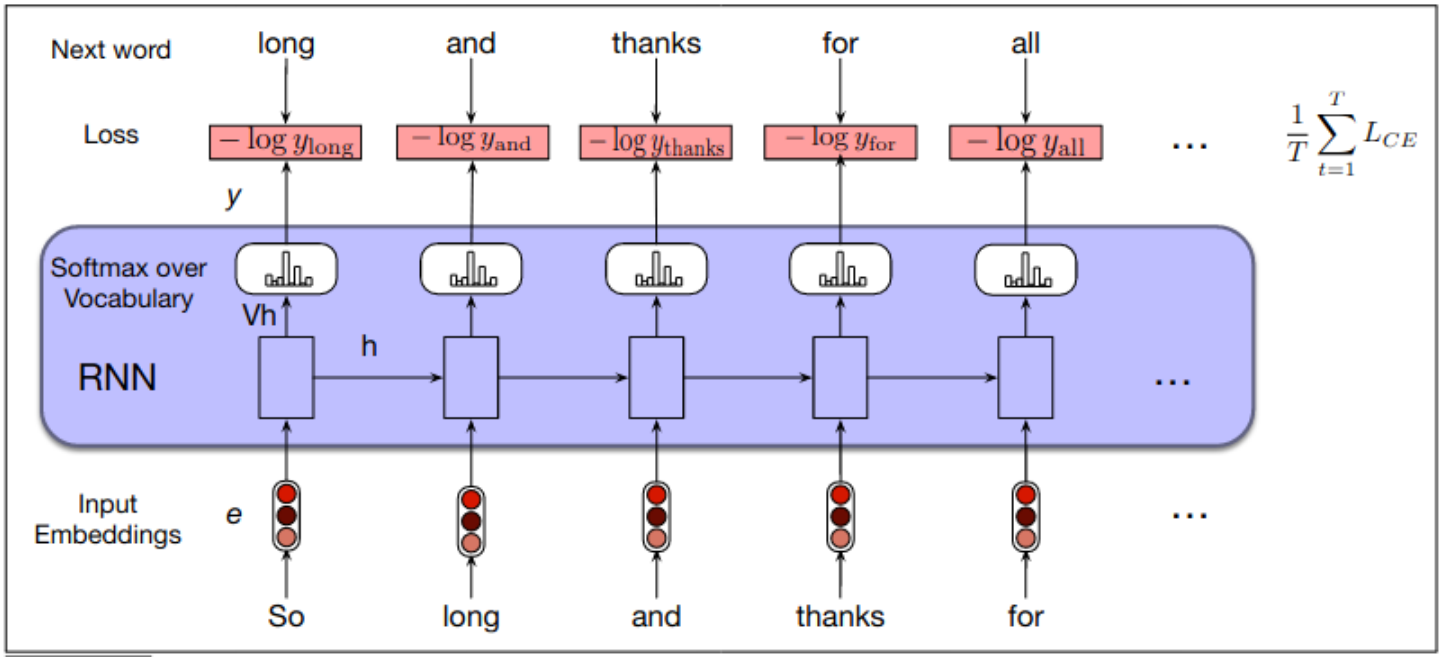
Cross-entropy loss over predicted probabilities:

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

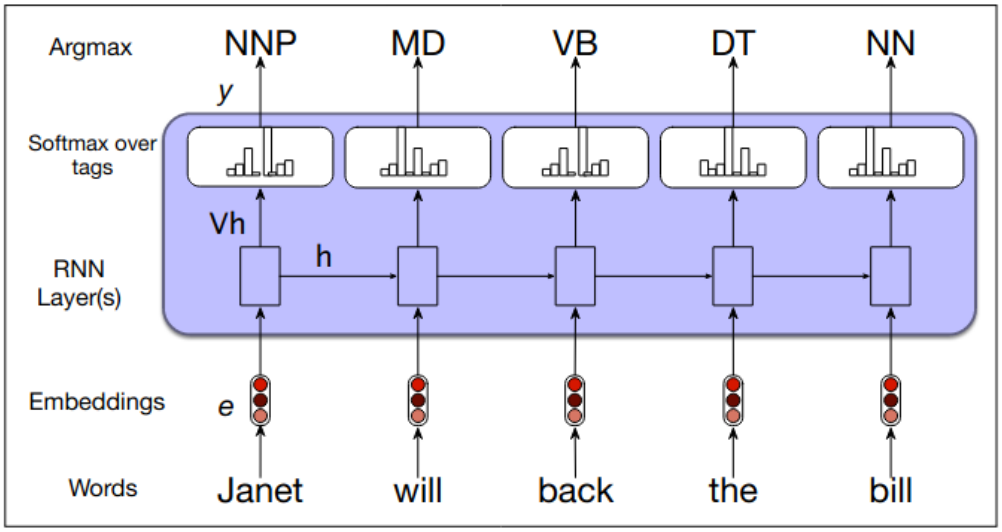
$$L_{CE}\left(\hat{\mathbf{y}}_t, \mathbf{y}_t\right) = -\log \hat{\mathbf{y}}_t\left[w_{t+1}\right]$$

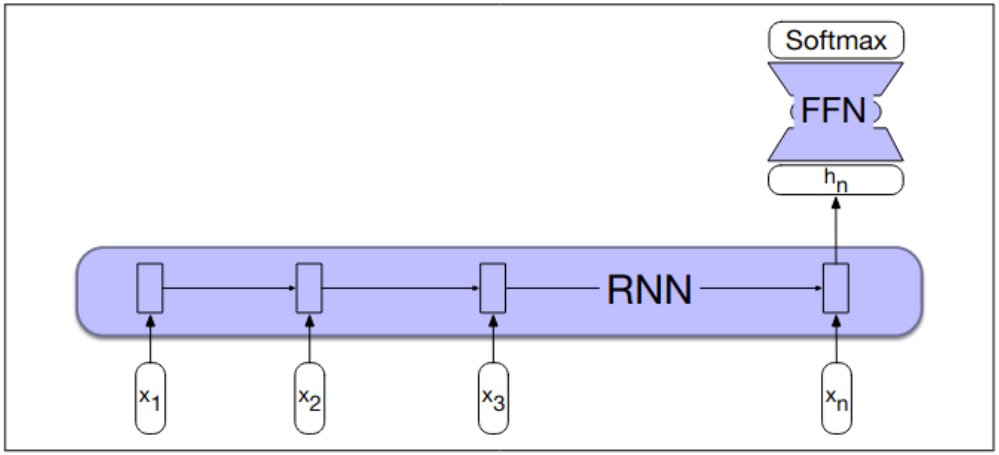
*LM Cross-entropy loss for = probability the model assigns to the **correct next word** (one-hot vector).*

🥣 PUTTING IT TOGETHER



GENERALIZING RNN ARCHITECTURE TO OTHER TASKS





🥪 LET'S INCREASE THE COMPLEXITY!

- Like feed-forward layers: **stack** RNNs, use top as input.
- **Bidirectional** RNNs (every step / last): $\mathbf{h}_t = [\mathbf{h}_t^f ; \mathbf{h}_t^b]$.
- !! Remaining issues:
  - Two tasks in one:
    - Carrying info forward.
    - Representation for current task.
  - Short context focus.
  - Vanishing gradients.

🥖 LONG SHORT-TERM MEMORY (LSTM) NETWORK

- Manage information required in the long term.
- Adds context vector and masking gates to the representation.

🍪 REMAINING LIMITATIONS

- Recurrence leads to information loss and training issues.
- Sequential nature blocks parallelism (as in e.g. CNNs).

🤖 TRANSFORMERS

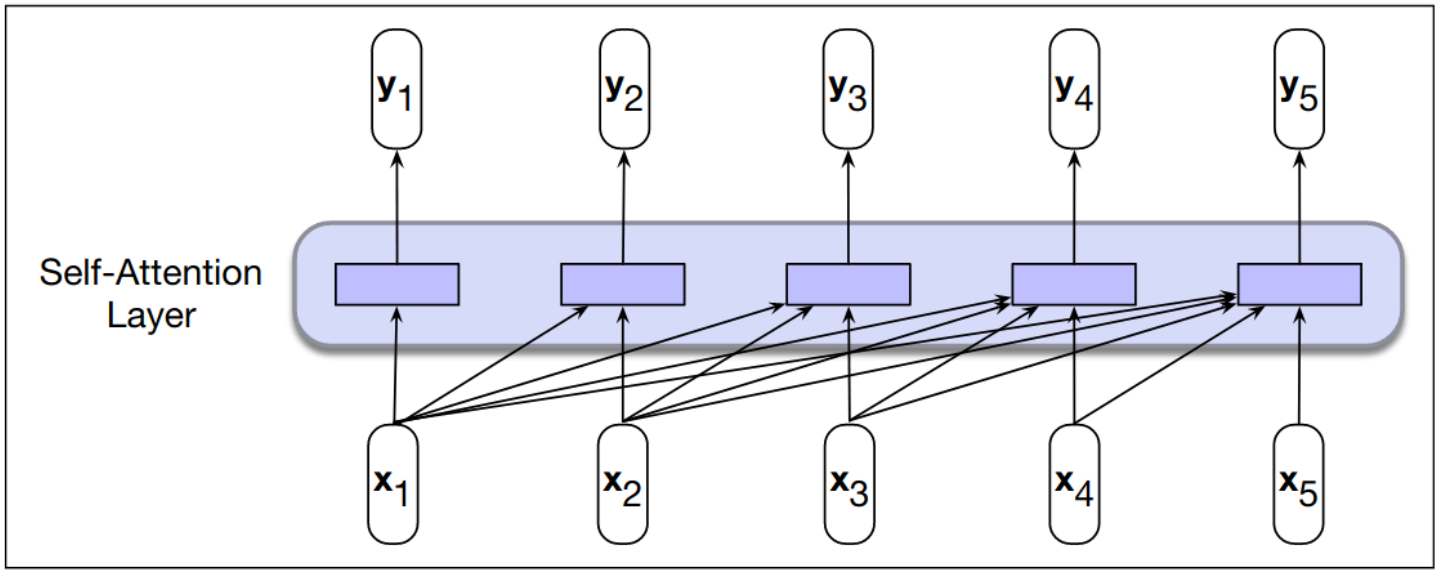
👀 ATTENTION

$$\alpha_{ij} = \text{softmax}\left(\text{score}\left(\mathbf{x}_i, \mathbf{x}_j\right)\right) \forall j \leq i$$

$$= \frac{\exp\left(\text{score}\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \forall j \leq i$$

$$\text{score}\left(x_i, x_j\right) = x_i \cdot x_j$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$



🤖👀 TRANSFORMER ATTENTION

- **query** ($\mathbf{q}_i$): attention focus wrt preceding inputs -- $\mathbf{W}^{\mathbf{Q}}$
- **key** ($\mathbf{k}_i$): preceding input being wrt attention focus -- $\mathbf{W}^{\mathbf{K}}$
- **value** ($\mathbf{v}_i$): computes the output for attention focus -- $\mathbf{W}^{\mathbf{V}}$

Project $\mathbf{x}_i$ *:*

$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i$ *;* $\mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i$ *;* $\mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$ *, where dims:* $1 \times d$*. Later:* $\mathbf{W}^{Q} \in \mathbb{R}^{d \times d}$ *,* $\mathbf{W}^{K} \in \mathbb{R}^{d \times d}$ *, and* $\mathbf{W}^{V} \in \mathbb{R}^{d \times d}$ *.*

🧮👀 MATRIX SELF-ATTENTION

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

$$\downarrow$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

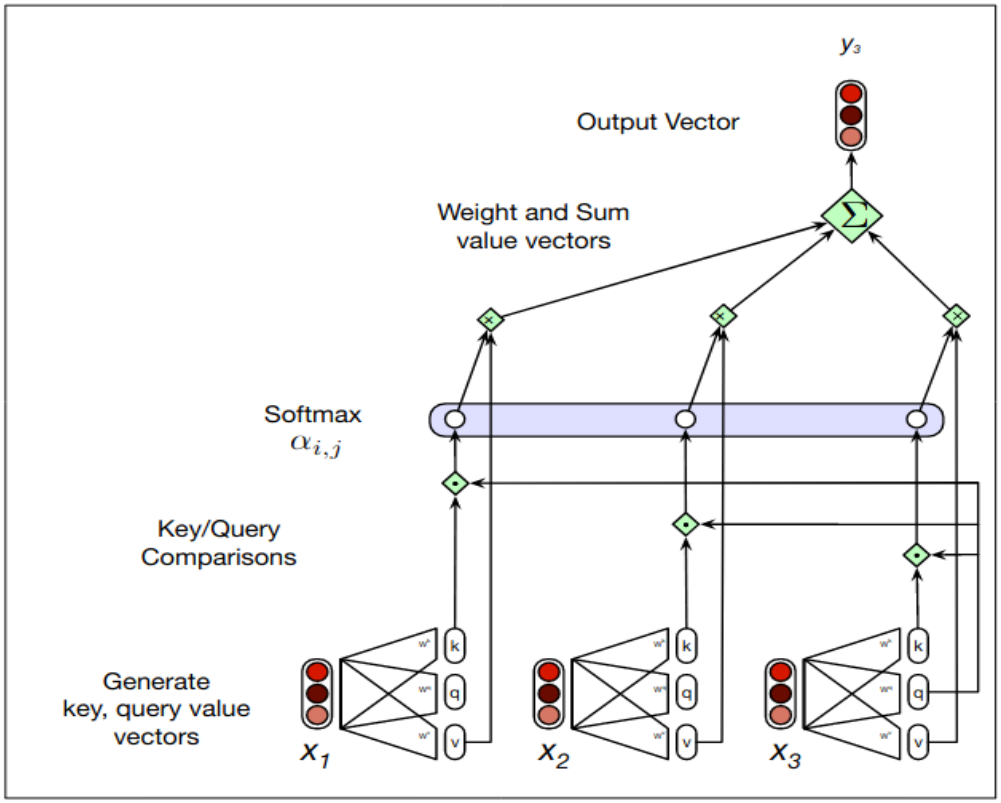
Actually: $\mathbf{X} \in \mathbb{R}^{N \times d}$ (embeddings),
so: $\mathbf{Q} \in \mathbb{R}^{N \times d}$, $\mathbf{K} \in \mathbb{R}^{N \times d}$,
and $\mathbf{V} \in \mathbb{R}^{N \times d}$, giving:

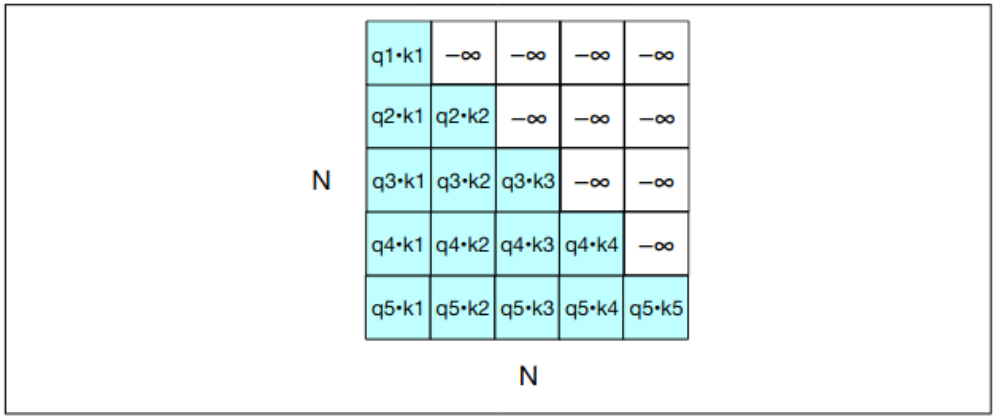
$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

which gives:

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

🏞️ VISUALLY





Setting weights to -inf to hide future.

🚗 TRANSFORMER BLOCKS

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{Self Attn}(\mathbf{x}))$$

$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z}))$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma} \qquad \text{LayerNorm} = \gamma\hat{\mathbf{x}} + \beta$$

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

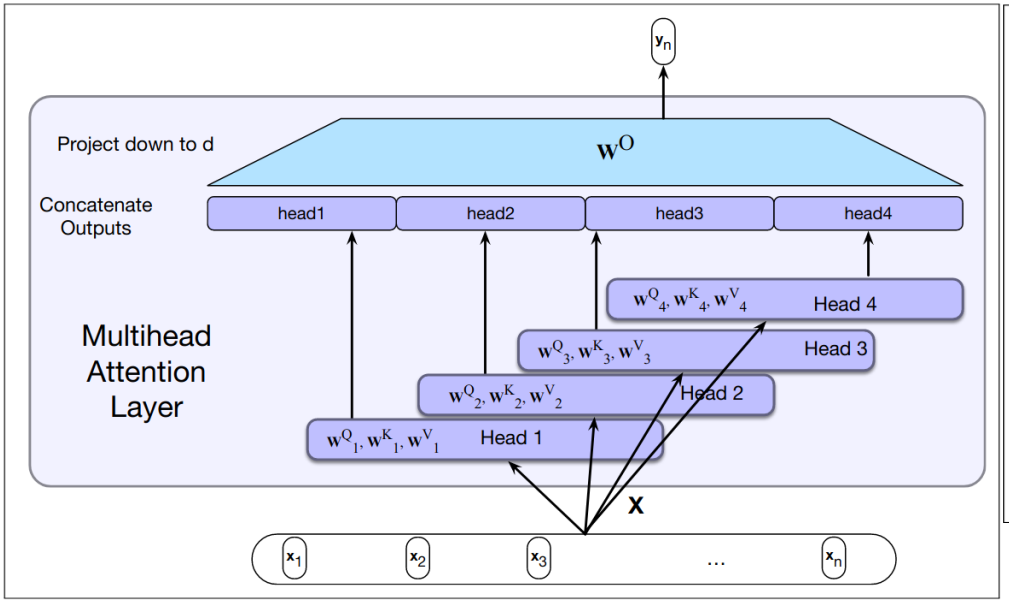
$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2}$$

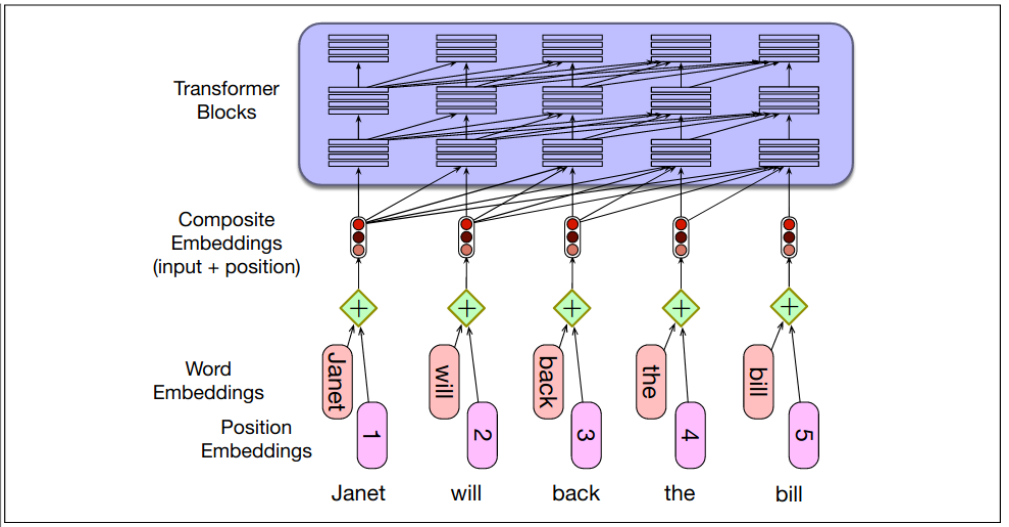
MORE TRICKS

$$\text{MultiHeadAttn}(\mathbf{X}) = (\text{head}_1 \oplus \text{head}_2 \ldots \oplus \text{head}_h)\,\mathbf{W}^O$$

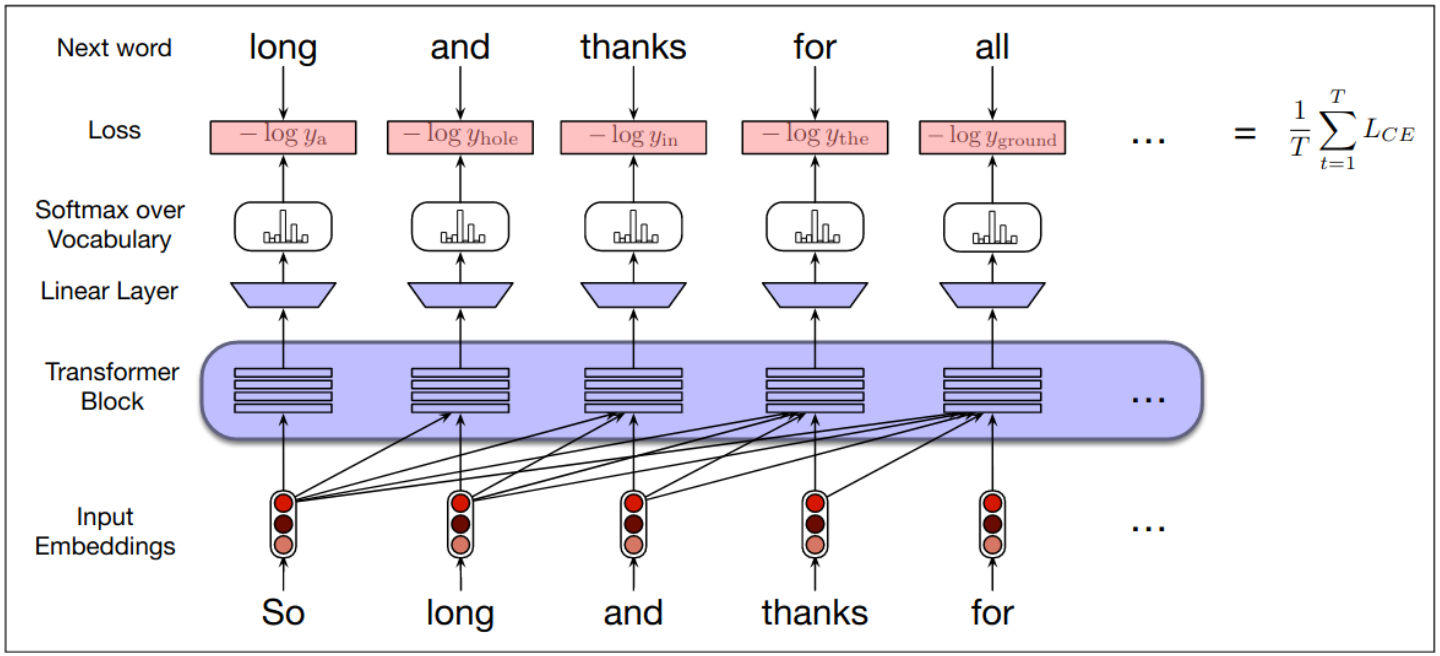
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

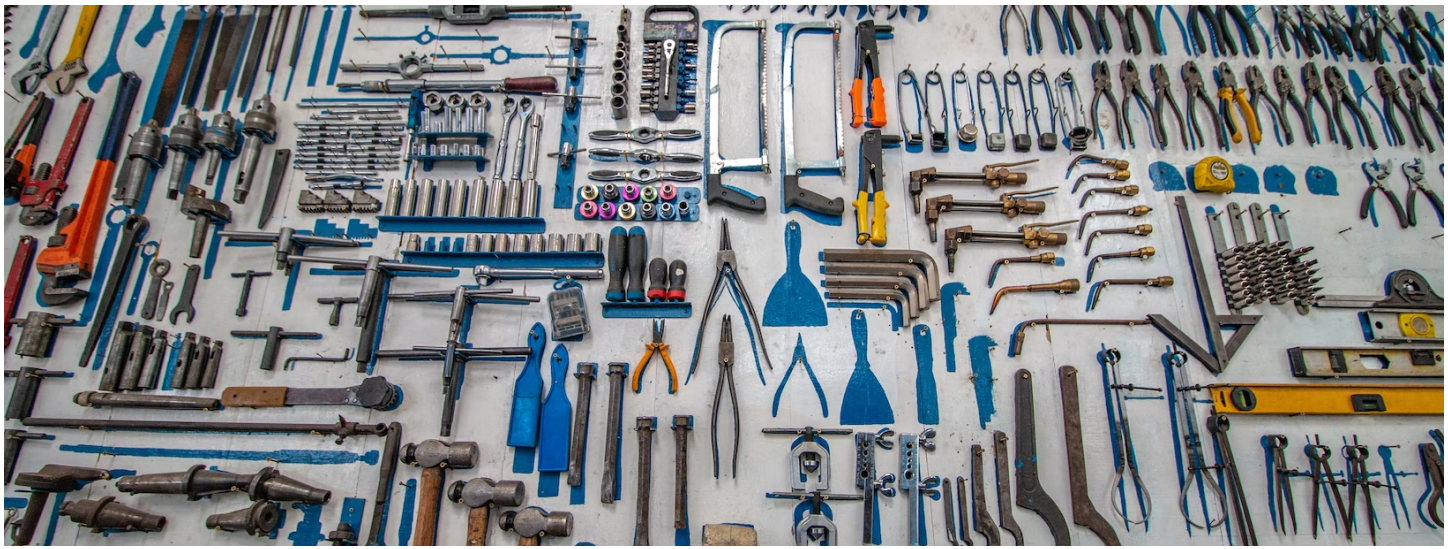




🔨 APPLIED


Next word
long
and
thanks
for
all
Loss
$-\log y_{\text{a}}$
$-\log y_{\text{hole}}$
$-\log y_{\text{in}}$
$-\log y_{\text{the}}$
$-\log y_{\text{ground}}$
...
=
$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$
Softmax over Vocabulary
Linear Layer
Transformer Block
...
Input Embeddings
...
So
long
and
thanks
for

🏞 DEEP LEARNING LANDSCAPE



🤦 QUESTIONS

Post on the Discussion board and join class on Thursdays!