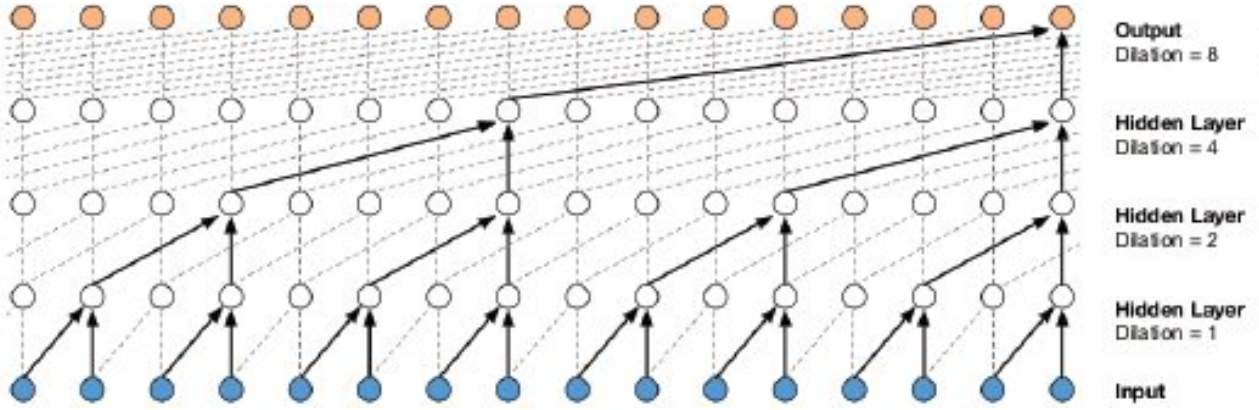

Figure 15.15: Illustration of the wavenet model using dilated (atrous) convolutions, with dilation factors of 1, 2, 4 and 8. From Figure 3 of [oor+16]. Used with kind permission of Aaron van den Oord.

In order to capture long-range dependencies, we can use dilated convolution (Section 14.4.1), as illustrated in Figure 15.15. This model has been successfully used to create a state of the art text to speech (TTS) synthesis system known as wavenet [oor+16]. In particular, they stack 10 causal 1d convolutional layers with dilation rates 1, 2, 4, … , 256, 512 to get a convolutional block with an effective receptive field of 1024. (They left-padded the input sequences with a number of zeros equal to the dilation rate before every layer, so that every layer has the same length.) They then repeat this block 3 times to compute deeper features.

In wavenet, the conditioning information x is a set of linguistic features derived from an input sequence of words; the model then generates raw audio using the above model. It is also possible to create a fully end-to-end approach, which starts with raw words rather than linguistic features (see [Wan+17]).

Although wavenet produces high quality speech, it is too slow for use in production systems. However, it can be "distilled" into a parallel generative model [Oor+18]. We discuss these kinds of parallel generative models in the sequel to this book, [Mur22].

15.4 Attention

In all of the neural networks we have considered so far, the hidden activations are a linear combination of the input activations, followed by a nonlinearity: $z = (Wv)$, where $v \in \mathbb{R}^V$ are the hidden feature vectors, and $W \in \mathbb{R}^{V \times V}$ are a fixed set of weights that are learned on a training set.

However, we can imagine a more flexible model in which we have a set of m feature vectors or values $V \in \mathbb{R}^{m \times v}$, and the model dynamically decides (in an input dependenent way) which one to use, based on how similar the input query vector $q \in \mathbb{R}^q$ is to a set of m keys $K \in \mathbb{R}^{m \times k}$. If q is most similar to key i, then we use value v_i. This is the basic idea behind attention mechanisms. This idea was originally developed for sequence models, and we will therefore explain it in this context. However, it can be more generally applied. Our presentation in the following sections is based on [Zha+20, Chap 10.].

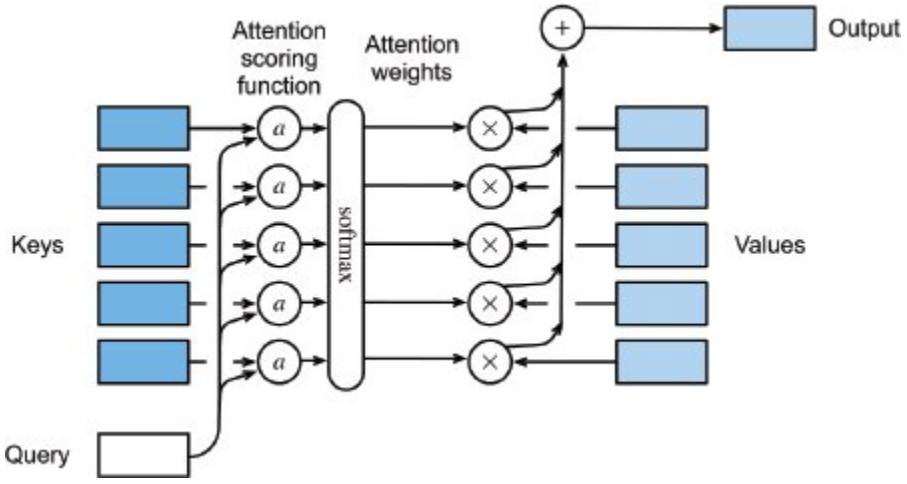

Figure 15.16: Attention computes a weighted average of a set of values, where the weights are derived by comparing the query vector to a set of keys. From Figure 10.3.1 of [Zha+20]. Used with kind permission of Aston Zhang.

15.4.1 Attention as soft dictionary lookup

We can think of attention as a dictionary lookup, in which we compare the query q to each key k_i, and then retrieve the corresponding value v_i. To make this lookup operation differentiable, instead of retrieving a single value v_i, we compute a convex combination of the values, as follows:

$$\text{Attn}(\boldsymbol{q}, (\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_m, \boldsymbol{v}_m)) = \text{Attn}(\boldsymbol{q}, (\boldsymbol{k}_{1:m}, \boldsymbol{v}_{1:m})) = \sum_{i=1}^{m} \alpha_i(\boldsymbol{q}, \boldsymbol{k}_{1:m}) \boldsymbol{v}_i \in \mathbb{R}^v \tag{15.34}$$

where $_i(q, k_{1:m})$ is the i'th attention weight; these weights satisfy 0 $_i(q, k_{1:m})$ 1 for each i and $_i$ $_i(q, k_{1:m}) = 1$.

The attention weights can be computed from an attention score function $a(q, k_i)$ $\mathbb{R}$, that computes the similarity of query q to key k_i. We will discuss several such score function below. Given the scores, we can compute the attention weights using the softmax function:

$$\alpha_i(\boldsymbol{q}, \boldsymbol{k}_{1:m}) = \mathcal{S}_i([a(\boldsymbol{q}, \boldsymbol{k}_1), \ldots, a(\boldsymbol{q}, \boldsymbol{k}_m)]) = \frac{\exp(a(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{j=1}^{m} \exp(a(\boldsymbol{q}, \boldsymbol{k}_j))} \tag{15.35}$$

See Figure 15.16 for an illustration.

In some cases, we want to restrict attention to a subset of the dictionary, corresponding to valid entries. For example, we might want to pad sequences to a fixed length (for efficient minibatching), in which case we should "mask out" the padded locations. This is called masked attention. We can implement this efficiently by setting the attention score for the masked entries to a large negative number, such as 10^6, so that the corresponding softmax weights will be 0. (This is analogous to causal convolution, discussed in Section 15.3.2.)

15.4.2 Kernel regression as non-parametric attention

In Section 16.3.5, we discuss kernel regression, which is a nonparametric model of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i(x, x_{1:n}) y_i \tag{15.36}$$

where $_i(x, x_{1:n})$ 0 measures the normalized similarity of test input x to training input x_i. This similarity measure is usually computed by defining the attention score in terms of a density kernel, such as the Gaussian:

$$\mathcal{K}_\sigma(u) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}u^2} \tag{15.37}$$

where is called the bandwidth. We then define $a(x, x_i) = K(x\ x_i)$.

Because the scores are normalized, we can drop the $\frac{1}{\sqrt{2\pi\sigma^2}}$ term. In addition, to maintain notation consistency with [Zha+20, Ch. 10], we rewrite the term inside the exponential as follows:

$$\mathcal{K}(x; w) = \exp(-\frac{w^2}{2}u^2) \tag{15.38}$$

Plugging this in to Equation (15.36), we get

$$f(x) = \sum_{i=1}^{n} \alpha_i(x, x_{1:n}) y_i \tag{15.39}$$

$$= \sum_{i=1}^{n} \frac{\exp[-\frac{1}{2}((x - x_i)w)^2]}{\sum_{j=1}^{n} \exp[-\frac{1}{2}((x - x_j)w)^2]} y_i \tag{15.40}$$

$$= \sum_{i=1}^{n} \mathcal{S}_i \left[-\frac{1}{2}((x - x_1)w)^2, \cdots, -\frac{1}{2}((x - x_n)w)^2\right] y_i \tag{15.41}$$

We can interpret this as a form of nonparametric attention, where the queries are the test points x, the keys are the training inputs x_i, and the values are the training labels y_i.

If we set $w = 1$, the resulting attention matrix $A_{ji} = {}_i(x_j, x_{1:n})$ for test input j is shown in Figure 15.17a. The resulting predicted curve is shown in Figure 15.17b.

The size of the diagonal band in Figure 15.17a, and hence the sparsity of the attention mechanism, dependends on the parameter w. If we increase w, corresponding to reducing the kernel bandwidth, the band will get narrower, but the model will start to overfit.

15.4.3 Parametric attention

In Section 15.4.2, we defined the attention score in terms of the Gaussian kernel, comparing a scalar query (test point) to each of the scalar values in the training set. This does not scale well to large training sets, or high-dimensional inputs. We will therefore turn our attention to parametric models, where we have a fixed set of keys and values, and where we compare queries and keys in a learned embedding space.

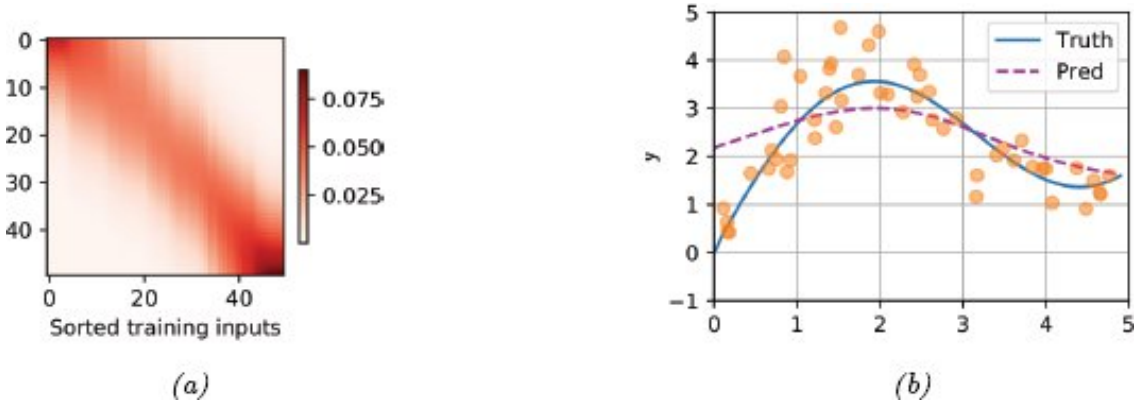

Figure 15.17: Kernel regression in 1d. (a) Kernel weight matrix. (b) Resulting predictions on a dense grid of test points. Generated by code at figures.probml.ai/book1/15.17.

There are several ways to do this. In the general case, the query q $\mathbb{R}^q$ and the key k $\mathbb{R}^k$ may have different sizes. To compare them, we can map them to a common embedding space of size h by computing $W_q q$ and $W_k k$. where W_q $\mathbb{R}^{h \times q}$ and W_k $\mathbb{R}^{h \times k}$. We can then pass these into an MLP to get the following additive attention scoring function:

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_v^\top \tanh(\mathbf{W}_q \boldsymbol{q} + \mathbf{W}_k \boldsymbol{k}) \in \mathbb{R} \tag{15.42}$$

A more computationally efficient approach is to assume the queries and keys both have length d, so we can compute $q^T k$ directly. If we assume these are independent random variables with 0 mean and unit variance, the mean of their inner product is 0, and the variance is d. (This follows from Equation (2.34) and Equation (2.39).) To ensure the variance of the inner product remains 1 regardless of the size of the inputs, it is standard to divide by $\sqrt{d}$. This gives rise to the scaled dot-product attention:

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\mathsf{T} \boldsymbol{k} / \sqrt{d} \in \mathbb{R} \tag{15.43}$$

In practice, we usually deal with minibatches of n vectors at a time. Let the corresponding matrices of queries, keys and values be denoted by $\mathbf{Q} \in \mathbb{R}^{n \times d}$, $\mathbf{K} \in \mathbb{R}^{m \times d}$, $\mathbf{V} \in \mathbb{R}^{m \times v}$. Then we can compute the attention-weighted outputs as follows:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathcal{S}(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v} \tag{15.44}$$

where the softmax fucntion S is applied row-wise. See code.probml.ai/book1/attention_torch for some sample code.

15.4.4 Seq2Seq with attention

Recall the seq2seq model from Section 15.2.3. This uses an RNN decoder of the form $f_d(h_{t-1}^d, y_{t-1}, \boldsymbol{c})$, where c is a fixed-length context vector, representing the encoding of the input $x_{1:T}$. Usually we set $\boldsymbol{c} = h_T^e$, which is the final state of the encoder RNN (or we use a bidirectional RNN with average pooling). However, for tasks such as machine translation, this can result in poor performance, since the output does not have access to the input words themselves. We can avoid this bottleneck by allowing the output words to directly "look at" the input words. But which inputs should it look at? After all, word order is not always preserved across languages (e.g., German often puts verbs at the end of a sentence), so we need to infer the alignment between source and target.

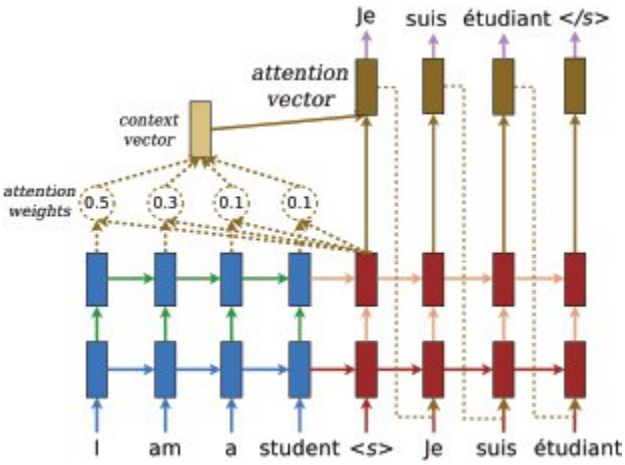

Figure 15.18: Illustration of seq2seq with attention for English to French translation. Used with kind permission of Minh-Thang Luong.

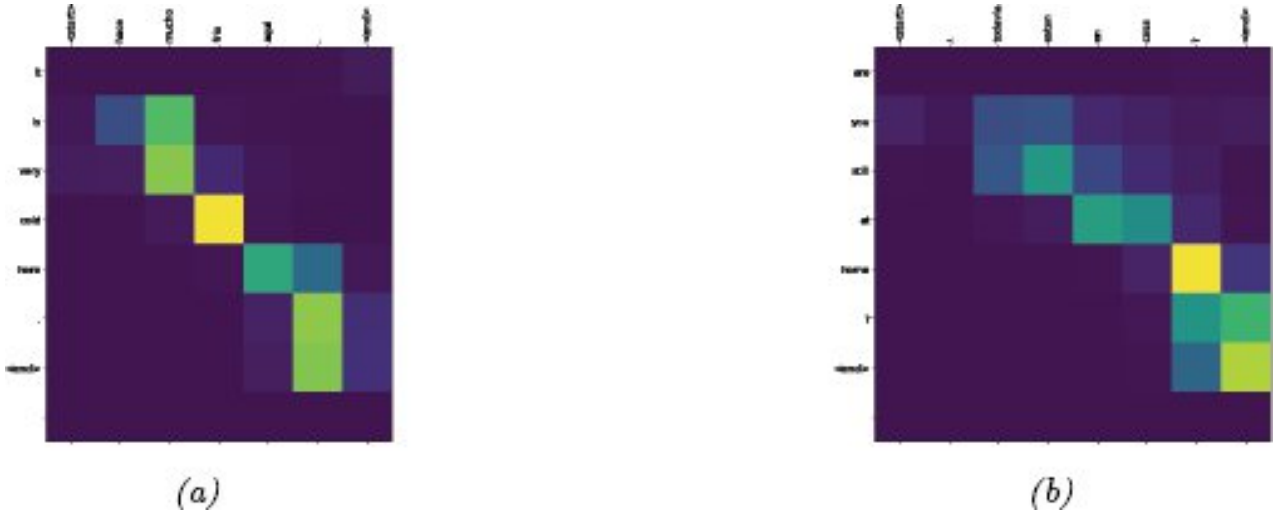

Figure 15.19: Illustration of the attention heatmaps generated while translating two sentences from Spanish to English. (a) Input is "hace mucho frio aqui.", output is "it is very cold here.". (b) Input is "¿todavia estan en casa?", output is "are you still at home?". Note that when generating the output token "home", the model should attend to the input token "casa", but in fact it seems to attend to the input token "?". Adapted from https://www.tensorflow.org/tutorials/text/nmt_with_attention.

We can solve this problem (in a differentiable way) by using (soft) attention, as first proposed in [BCB15; LPM15]. In particular, we can replace the fixed context vector c in the decoder with a dynamic context vector c_t computed as follows:

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \alpha_i(\boldsymbol{h}_{t-1}^d, \boldsymbol{h}_{1:T}^e)\boldsymbol{h}_i^e \tag{15.45}$$

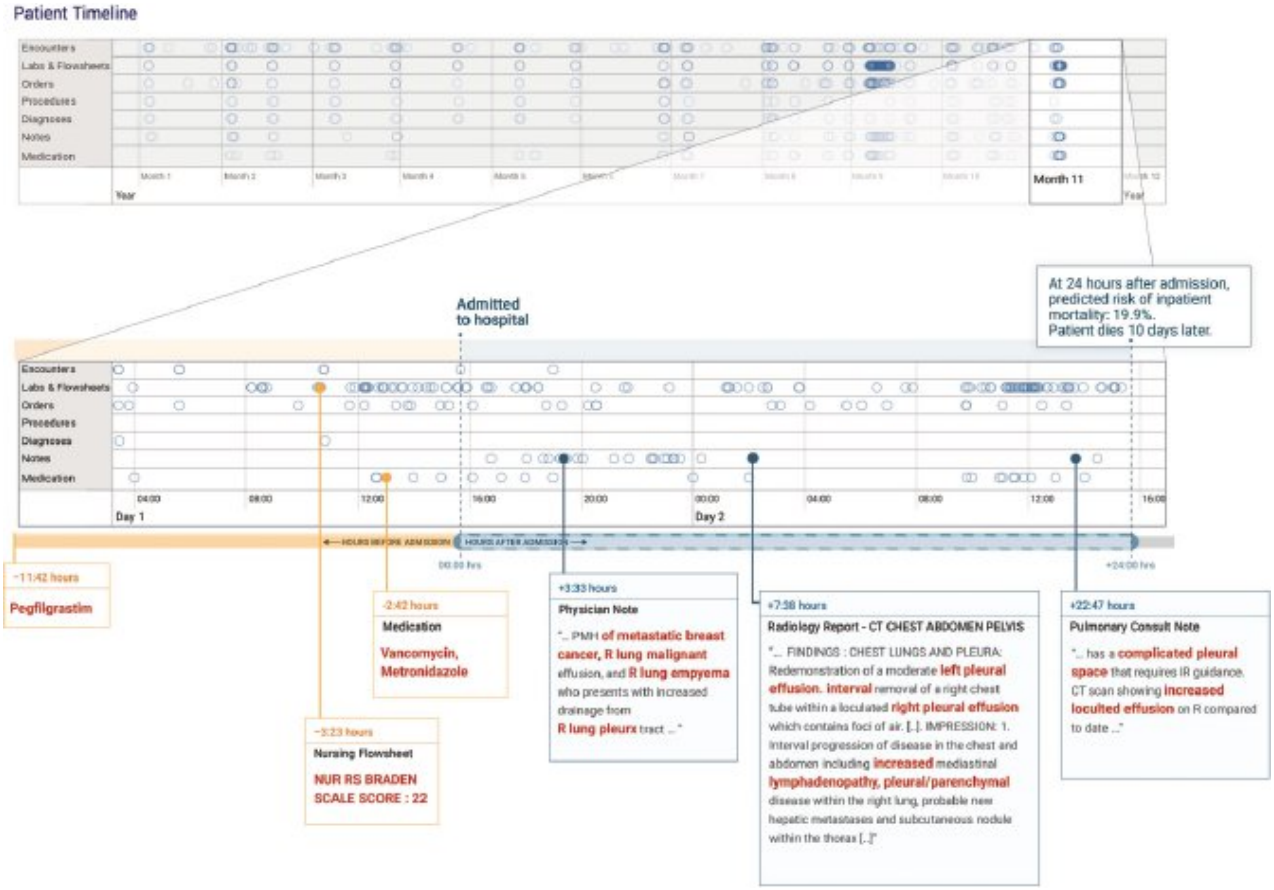

Figure 15.20: Example of an electronic health record. In this example, 24h after admission to the hospital, the RNN classifier predicts the risk of death as 19.9%; the patient ultimately died 10 days after admission. The "relevant" keywords from the input clinical notes are shown in red, as identified by an attention mechanism. From Figure 3 of [Raj+18]. Used with kind permission of Alvin Rakomar.

This uses attention where the query is the hidden state of the decoder at the previous step, $\boldsymbol{h}_{t-1}^{d}$, the keys are all the hidden states from the encoder, and the values are also the hidden states from the encoder. (When the RNN has multiple hidden layers, we usually take the top layer from the encoder, as the keys and values, and the top layer of the decoder as the query.) This context vector is concatenated with the input vector of the decoder, y_{t1}, and fed into the decoder, along with the previous hidden state $\boldsymbol{h}_{t-1}^{d}$, to create $\boldsymbol{h}_{t}^{d}$. See Figure 15.18 for an illustration of the overall model.

We can train this model in the usual way on sentence pairs, and then use it to perform machine translation. (See code.probml.ai/book1/nmt_attention_torch for some sample code.) We can also visualize the attention weights computed at each step of decoding, to get an idea of which parts of the input the model thinks are most relevant for generating the corresponding output. Some examples are shown in Figure 15.19.

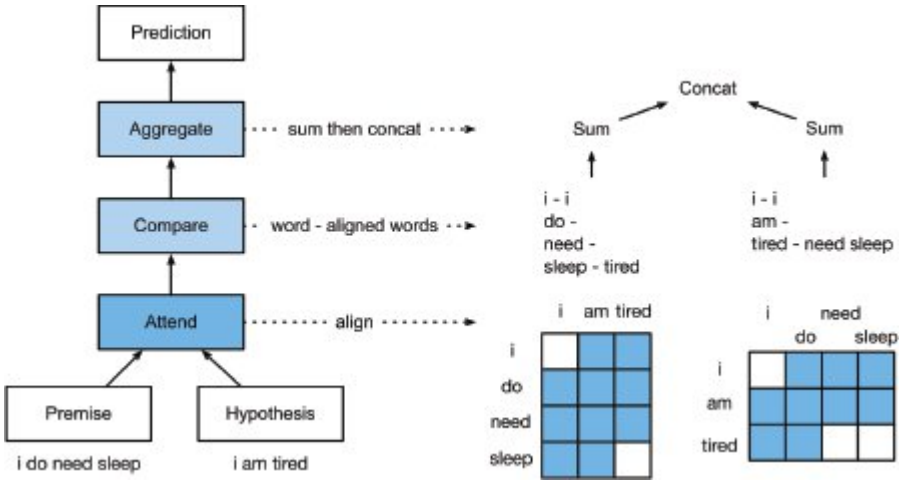

Figure 15.21: Illustration of sentence pair entailment classification using an MLP with attention to align the premise ("I do need sleep") with the hypothesis ("I am tired"). White squares denote active attention weights, blue squares are inactive. (We are assuming hard 0/1 attention for simplicity.) From Figure 15.5.2 of [Zha+20]. Used with kind permission of Aston Zhang.

15.4.5 Seq2vec with attention (text classification)

We can also use attention with sequence classifiers. For example [Raj+18] apply an RNN classifier to the problem of predicting if a patient will die or not. The input is a set of electronic health records, which is a time series containing structured data, as well as unstructured text (clinical notes). Attention is useful for identifying "relevant" parts of the input, as illustrated in Figure 15.20.

15.4.6 Seq+Seq2Vec with attention (text pair classification)

Suppose we see the sentence "A person on a horse jumps over a log" (call this the premise) and then we later read "A person is outdoors on a horse" (call this the hypothesis). We may reasonably say that the premise entails the hypothesis, meaning that the hypothesis is more likely given the premise.[3] Now suppose the hypothesis is "A person is at a diner ordering an omelette". In this case, we would say that the premise contradicts the hypothesis, since the hypothesis is less likely given the premise. Finally, suppose the hypothesis is "A person is training his horse for a competition". In this case, we see that the relationship between premise and hypothesis is neutral, since the hypothesis may or may not follow from the premise. The task of classifying a sentence pair into these three categories is known as textual entailment or "natural language inference". A standard benchmark in this area is the Stanford Natural Language Inference or SNLI corpus [Bow+15]. This consists of 550,000 labeled sentence pairs.

An interesting solution to this classification problem was presented in [Par+16a]; at the time, it was the state of the art on the SNLI dataset. The overall approach is sketched in Figure 15.21. Let A = (a_1, … , a_m) be the premise and B = (b_1, … , b_n) be the hypothesis, where a_i, b_j $\mathbb{R}^E$ are embedding vectors for the words. The model has 3 steps. First, each word in the premise, a_i, attends to each word in the hypothesis, b_j, to compute an attention weight

$$e_{ij} = f(\boldsymbol{a}_i)^\mathsf{T} f(\boldsymbol{b}_j) \tag{15.46}$$

where f : $\mathbb{R}^E$ $\mathbb{R}^D$ is an MLP; we then compute a weighted average of the matching words in the hypothesis,

$$\boldsymbol{\beta}_i = \sum_{j=1}^{n} \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \boldsymbol{b}_j \tag{15.47}$$

Next, we compare a_i with $_i$ by mapping their concatenation to a hidden space using an MLP g : $\mathbb{R}^{2E}$ $\mathbb{R}^H$:

$$\boldsymbol{v}_{A,i} = g([\boldsymbol{a}_i, \boldsymbol{\beta}_i]), \; i = 1, \ldots, m \tag{15.48}$$

Finally, we aggregate over the comparisons to get an overall similarity of premise to hypothesis:

$$\boldsymbol{v}_A = \sum_{i=1}^{m} \boldsymbol{v}_{A,i} \tag{15.49}$$

We can similarly compare the hypothesis to the premise using

$$\boldsymbol{\alpha}_j = \sum_{i=1}^{m} \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{kj})} \boldsymbol{a}_i \tag{15.50}$$

$$\boldsymbol{v}_{B,j} = g([\boldsymbol{b}_j, \boldsymbol{\alpha}_j]), \; j = 1, \ldots, n \tag{15.51}$$

$$\boldsymbol{v}_B = \sum_{j=1}^{n} \boldsymbol{v}_{B,j} \tag{15.52}$$

At the end, we classify the output using another MLP $h : \mathbb{R}^{2H} \; \mathbb{R}^3$:

$$\hat{y} = h([\boldsymbol{v}_A, \boldsymbol{v}_B]) \tag{15.53}$$

See code.probml.ai/book1/entailment_attention_mlp_torch for some sample code.

We can modify this model to learn other kinds of mappings from sentence pairs to output labels. For example, in the semantic textual similarity task, the goal is to predict how semantically related two input sentences are. A standard dataset for this is the STS Benchmark [Cer+17], where relatedness ranges from 0 (meaning unrelated) to 5 (meaning maximally related).

15.4.7 Soft vs hard attention

If we force the attention heatmap to be sparse, so that each output can only attend to one input location instead of a weighted combination of all of them, the method is called hard attention. We compare these two approaches for an image captioning problem in Figure 15.22. Unfortunately, hard attention results in a nondifferentiable training objective, and requires methods such as reinforcement learning to fit the model. See [Xu+15] for the details.

It seems from the above examples that these attention heatmaps can "explain" why the model generates a given output. However, the interpretability of attention is controversial (see e.g., [JW19; WP19; SS19; Bru+19] for discussion).

Figure 15.22: Image captioning using attention. (a) Soft attention. Generates "a woman is throwing a frisbee in a park". (b) Hard attention. Generates "a man and a woman playing frisbee in a field". From Figure 6 of [Xu+15]. Used with kind permission of Kelvin Xu.

15.5 Transformers

The transformer model [Vas+17] is a seq2seq model which uses attention in the encoder as well as the decoder, thus eliminating the need for RNNs, as we explain below. Transformers have been used for many (conditional) sequence generation tasks, such as machine translation [Vas+17], constituency parsing [Vas+17], music generation [Hua+18], protein sequence generation [Mad+20; Cho+20b], abstractive text summarization [Zha+19a], image generation [Par+18] (treating the image as a rasterized 1d sequence), etc.