Contents

Blogs

Contents

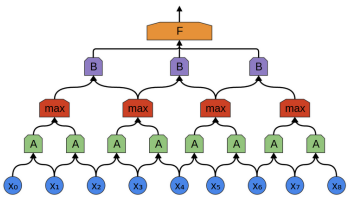
Blogs **December 21, 2016**

Conv Nets: A Modular Perspective

Table of Contents Local *Written by Brandon McKinzie*

From this post on Colah's Blog.

The title is inspired by the following figure. Colah mentions how groups of neurons, like A, that appear in multiple places are sometimes called **modules**, and networks that use them are sometimes called modular neural networks. You can feed the output of one convolutional layer into another. With each layer, the network can detect higher-level, more abstract features.



⟶ Function of the A neurons: compute certain *features*.

⟶ Max pooling layers: kind of "zoom out". They allow later convolutional layers to work on larger sections of the data. They also make us invariant to some very small transformations of the data.

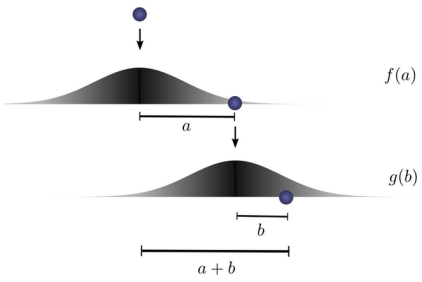
Understanding Convolutions

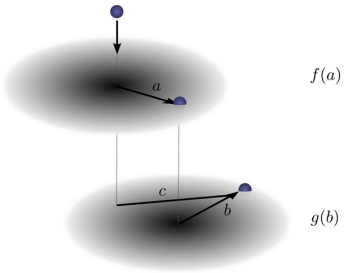
Written by Brandon McKinzie

From Colah's Blog.

Ball-Dropping Example. The posed problem:

> Imagine we drop a ball from some height onto the ground, where it only has one dimension of motion. How likely is it that a ball will go a distance c if you drop it and then drop it again from above the point at which it landed?





From basic probability, we know the result is a sum over possible outcomes, constrained by $a + b = c$. It turns out this is actually the definition of the convolution of f and g.

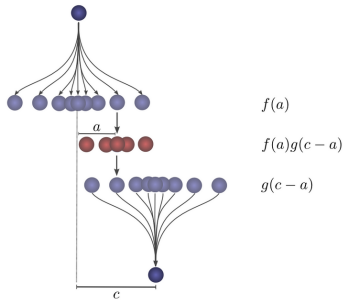
$$\Pr(\text{a} + \text{b} = \text{c}) = \sum_{a+b=c} f(a) \cdot g(b) \tag{1}$$

$$(f * g)(c) = \sum_{a+b=c} f(a) \cdot g(b) \tag{2}$$

$$= \sum_{a} f(a) \cdot g(c - a) \tag{3}$$

Visualizing Convolutions. Keeping the same example in the back of our heads, consider a few interesting facts.

- **Flipping directions**. If $f(x)$ yields the probability of landing a distance x away from where it was dropped, what about the probability that it was dropped a distance x from where it *landed*? It is $f(-x)$.
- Above is a visualization of one term in the summation of $(f * g)(c)$. It is meant to show how we can move the bottom around to think about evaluating the convolution for different c values.



0	0	0	0	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	-1	1	0	0
0	0	0	0	0
0	0	0	0	0

We can relate these ideas to image recognition. Below are two common kernels used to convolve images with.

On the left is a kernel for *blurring* images, accomplished by taking simple averages. On the right is a kernel for *edge detection*, accomplished by taking the difference between two pixels, which will be largest at edges, and essentially zero for similar pixels.

Blogs **December 23, 2016**

Deep Reinforcement Learning

Table of Contents Local *Written by Brandon McKinzie*

Link to tutorial – Part I of "Demystifying deep reinforcement learning."

Reinforcement Learning. Vulnerable to the *credit assignment problem* - i.e. unsure which of the preceding actions was responsible for getting some reward and to what extent. Also need to address the famous *explore-exploit dilemma* when deciding what strategies to use.

Markov Decision Process. Most common method for representing a reinforcement problem. MDPs consist of states, actions, and rewards. Total reward is sum of current (includes previous) and *discounted* future rewards:

$$R_t = r_t\gamma(r_{t+1} + \gamma(r_{t+2} + \ldots)) = r_t + \gamma R_{t+1} \tag{4}$$

Q - learning. Define function $Q(s, a)$ to be best possible score at end of game after performing action a in state s; the "quality" of an action from a given state. The recursive definition of Q (for one transition) is given below in the *Bellman equation*.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

and updates are computed with a learning rate α as

$$Q(s_t, a_t) = (1 - \alpha) \cdot Q(s_{t-1}, a_{t-1}) + \alpha \cdot (r + \gamma \max_{a'} Q(s'_{t+1}, a'_{t+1}))$$

Deep Q Network. Deep learning can take deal with issues related to prohibitively large state spaces. The implementation chosen by DeepMind was to represent the Q-function with a neural network, with the states (pixels) as the input and Q-values as output, where the number of output neurons is the number of possible actions from the input state. We can optimize with simple squared loss:

$$L = \tfrac{1}{2}[\underbrace{r + max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

and our algorithm from some state s becomes

1. **First forward pass** from s to get all predicted Q-values for each possible action. Choose action corresponding to max output, leading to next s'.

2. **Second forward pass** from s' and again compute $\max_{a'} Q(s', a')$.
3. **Set target output** for each action a' from s'. For the action corresponding to max (from step 2) set its target as $r + \gamma \max_{a'} Q(s', a')$, and for all other actions set target to same as originally returned from step 1, making the error 0 for those outputs. (Interpret as update to our guess for the best Q-value, and keep the others the same.)
4. **Update weights** using backprop.

Experience Replay. This the most important trick for helping convergence of Q-values when approximating with non-linear functions. During gameplay all the experience $< s, a, r, s' >$ are stored in a replay memory. When training the network, random minibatches from the replay memory are used instead of the most recent transition.

Exploration. One could say that initializing the Q-values randomly and then picking the max is essentially a form of exploitation. However, this type of exploration is *greedy*, which can be tamed/fixed with **ε-greedy exploration**. This incorporates a degree of randomness when choosing next action at *all* time-steps, determined by probability ε that we choose the next action randomly. For example, DeepMind decreases ε over time from 1 to 0.1.

Deep Q-Learning Algorithm.

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
      select an action a
            with probability ε select a random action
            otherwise select a = argmax_a' Q(s,a')
      carry out action a
      observe reward r and new state s'
      store experience <s, a, r, s'> in replay memory D

      sample random transitions <ss, aa, rr, ss'> from replay memory D
      calculate target for each minibatch transition
            if ss' is terminal state then tt = rr
            otherwise tt = rr + γmax_a' Q(ss', aa')
      train the Q network using (tt - Q(ss, aa))^2 as loss

      s = s'
until terminated
```

Blogs **January 15, 2017**

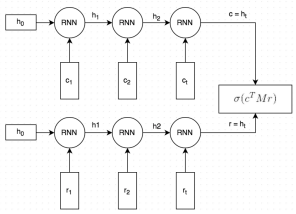
Deep Learning for Chatbots (WildML)

Table of Contents Local *Written by Brandon McKinzie*

Overview.

- **Model**. Implementing a retrieval-based model. Input: conversation/context c. Output: response r.
- **Data**. Ubuntu Dialog Corpus (UDC). 1 million examples of form (context, utterance, label). The label can be 1 (utterance was actual response to the context) or a 0 (utterance chosen randomly). Using NLTK, the data has been . . .
  → **Tokenized**: dividing strings into lists of substrings.
  → **Stemmed**. **IDK**
  → **Lemmatized**. **IDK**
  The test/validation set consists (context, ground-truth utterance, [9 distractors (incorrect utterances)]). The distractors are picked at random[1]

Dual-Encoder LSTM.



1. **Inputs**. Both the context and the response text are split by words, and each word is embedded into a vector and fed into the same RNN.
2. **Prediction**. Multiply the [vector representation ("meaning")] c with param matrix M to predict some response r'.
3. **Evaluation**. Measure similarity of predicted r' to actual r via simple dot product. Feed this into sigmoid to obtain a probability [of r' being the correct response]. Use (binary) cross-entropy for loss function:

$$L = -y \cdot \ln(y') - (1 - y) \cdot \ln(1 - y') \tag{5}$$

where y' is the predicted probability that r' is correct response r, and $y \in \{0, 1\}$ is the true label for the context-response pair (c, r).

[1] Better example/approach: Google's Smart Reply uses clustering techniques to come up with a set of possible responses.

Data Pre-Processing. Courtesy of WildML, we are given 3 files after preprocessing: train.tfrecords, validation.tfrecords, and test.tfrecords, which use TensorFlow's 'Example' format. Each Example consists of . . .

- context: Sequence of word ids.
- context_len: length of the aforementioned sequence.
- utterance: seq of word ids representing utterance (response).
- utterance_len.
- label: only in training data. 0 or 1.
- distractor_[N]: Only in test/validation. N ranges from 0 to 8. Seq of word ids reppin the distractor utterance.
- distractor_[N]_len.

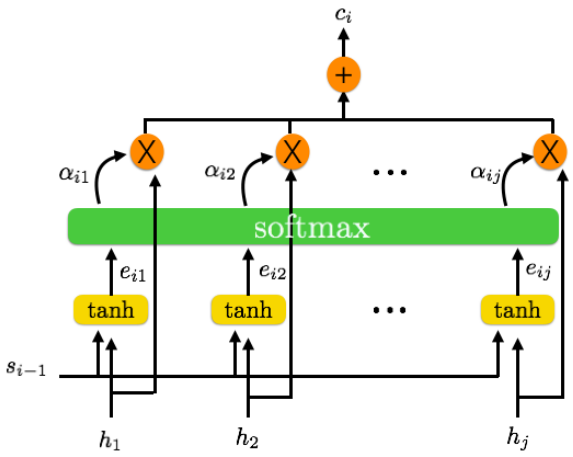
Blogs **April 04, 2017**

Attentional Interfaces – Neural Perspective

Table of Contents Local *Written by Brandon McKinzie*

[Link to article]

Attention Mechanism. Below is a close-up view/diagram of an attention layer. Technically, it only corresponds to a single time step i; we are using the previous decoder state s_{i-1} to compute the ith context vector c_i which will be fed as an input to the decoder for step i.



For convenience, I'll rewrite the familiar equations for computing quantities at some step i.

$$\textbf{[decoder state]} \qquad s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{6}$$

$$\textbf{[context vect]} \qquad c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{7}$$

$$\textbf{[annotation weights]} \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{8}$$

$$e_{ij} = a(s_{i-1}, h_j) \tag{9}$$

Now we can see just how simple this really is. Recall that Bahdanau *et al.*, 2015 use the wording: "e_{ij} *is an alignment model which scores how well the inputs around position* j *and the output at position* i *match*." But we can see an example implementation of an alignment model above: the *tanh* function (that's it).

Appendix

Contents

Common Distributions and Models

Written by Brandon McKinzie

$(\forall n \in \mathbb{Z}^+)\Gamma(n) = (n-1)!$

$\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}\mathrm{d}x$

Continuous Distributions.

Distribution	Density Function	Notation
Gaussian	$p(\theta) = (2\pi)^{-d/2}\det(\boldsymbol{\Sigma})^{-1/2}\, e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$	$\theta \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

Distributions with support $\theta > 0$:

Distribution	Density Function	Notation
Chi-Square	$p(\theta) = \frac{2^{-\nu/2}}{\Gamma(\nu/2)}\theta^{\nu/2-1}e^{-\theta/2}$	$\theta \sim \chi^2_\nu$
Gamma	$p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{\alpha-1}e^{-\beta\theta}$	$\theta \sim \text{Gamma}(\alpha, \beta)$
Inverse-gamma	$p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{-\alpha-1}e^{-\beta/\theta}$	$\theta \sim \text{Inv-gamma}(\alpha, \beta)$
Inverse-chi-square	$p(\theta) = \frac{2^{-\nu/2}}{\Gamma(\nu/2)}\theta^{-\nu/2-1}e^{-\frac{1}{2\theta}}$	$\theta \sim \text{Inv-}\chi^2_\nu$

Distributions with support $\theta \in [0, 1]$:

Distribution	Density Function	Notation
Beta	$p(\theta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$	$\theta \sim \text{Beta}(\alpha, \beta)$
Dirichlet	$p(\theta) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)}\prod_k \theta_k^{\alpha_k-1}, \quad \sum_k \theta_k = 1$	$\theta \sim \text{Dirichlet}(\alpha_1, \ldots, \alpha_K)$

Discrete Distributions.

Distribution	Density Function	Notation
Bernoulli	$p(x;\theta) = \theta^{\mathbb{1}_{x=1}}(1-\theta)^{\mathbb{1}_{x=0}}$	$X \sim Ber(\theta)$
Binomial	$p(x;n) = \binom{n}{x}p^x(1-p)^{n-x}$	$x \sim \text{Bin}(n, p)$
Multinomial	$p(x_1, \ldots, x_k; n) = \frac{n!}{\prod_i^k x_i!}\prod_i^k p_i^{x_i}$	

Logistic Regression. Perhaps the simplest *linear method*[2] for classification is **logistic regression**. Let K be the number of classes that y can take on. The model is defined as

$$\Pr[y = k \mid \boldsymbol{x}] = \frac{\exp(\boldsymbol{\theta}_k^T\boldsymbol{x})}{1 + \sum_{\ell=1}^{K-1}\exp(\boldsymbol{\theta}_\ell^T\boldsymbol{x})}, \quad \text{for } 1 \le k \le K-1 \tag{10}$$

$$\Pr[y = K \mid \boldsymbol{x}] = \frac{1}{1 + \sum_{\ell=1}^{K-1}\exp(\boldsymbol{\theta}_\ell^T\boldsymbol{x})} \tag{11}$$

and we often denote $\Pr[y = k \mid \boldsymbol{x}]$ under the entire set of parameters $\boldsymbol{\theta}$ simply as $p_k(\boldsymbol{x}; \boldsymbol{\theta})$ or just $p_k(\boldsymbol{x})$. The decision boundaries are the set of points in the domain of $\boldsymbol{x}$ for which

[2] We say a classification method is *linear* if its **decision boundary is linear**.

some $p_k(\boldsymbol{x}) = p_{j\neq k}(\boldsymbol{x})$. Equivalently, the model can be specified by $K-1$ log-odds or logit transformations of the form

$$\log\left(\frac{p_i(\boldsymbol{x})}{p_K(\boldsymbol{x})}\right) = \boldsymbol{\theta}_i^T\boldsymbol{x} \quad \text{for } 1 \leq i \leq K-1 \tag{12}$$

Also note that the parameter vectors are orthogonal to the K-1 decision boundaries. For any x, x' on the decision boundary defined as the set of points $\{x : p_a(x) = p_b(x)\}$, we know that the vector $x - x'$ is parallel to the decision boundary (by definition), and can derive

$$\frac{p_a(x)}{p_b(x)} = 1 = \exp(\theta_a^T x - \theta_b^T x) \implies \theta_a^T x = \theta_b^T x \tag{13}$$

$$\therefore \theta_a^T(x - x') = \theta_b^T(x - x') = 0 \tag{14}$$

and thus θ_a and θ_b are both perpendicular to the decision boundary where $p_a(x) = p_b(x)$.

Fancy math definitions/concepts for fancy authors who require fancy terminology.

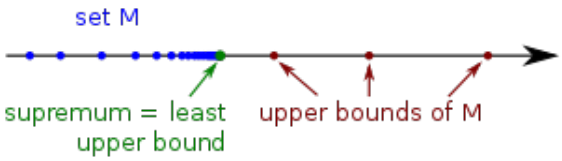
- **Support**. Sloppy definition you'll see in most places: The *set-theoretic support* of a real-valued function $f : X \mapsto \mathbb{R}$ is defined as

  $$\text{supp}(f) \triangleq \{x \in X : f(x) \neq 0\}$$

  Note that Wikipedia is surprisingly sloppy with how it defines and/or uses support in various articles. After some digging, I finally found the formal definition for probability and measure theory:

  > *If $X : \Omega \mapsto \mathbb{R}$ is a random variable on $(\Omega, \mathcal{F}, P)$, then the **support** of X is the smallest closed set $R_X \subset \mathbb{R}$ such that $P(X \in R_X) = 1$.*

- **Infimum and Supremum**. The fancy-math way of saying minimum and maximum. Yes, I recognize that these are important in certain (usually rather abstract) settings, but often in ML it is used when sup means exactly the same thing as max, but the authors want to look sophisticated. Here I'll give the formal definition for sup. You have a partially ordered set[3] P, and are for the moment checking out some subset $S \subseteq P$. Someone asks you, "hey, give me an *upper bound* of S." You just gotta find *some* $b \in P$ (the larger/global set) that is greater than or equal to every element in S. The person then comes back and says "ok no, I need the *supremum* of S." Now you need to find the *smallest* value out of all the possible upper bounds.

  Hopefully it is clear why this is only relevant in real-valued cases where the "edges" aren't well-defined.

- **Probability Measure**. Informal definition: a probability distribution[4]. Formal definition: a function $\mu : \alpha \mapsto \mathbb{R}[0, 1]$ from events to scalar values, where $\mu(\alpha) = 1$ if $\alpha = \Omega$ (the full space) and $\mu(\varnothing) = 0$. Also μ must satisfy the countable additivity property: $\mu(\cup_i \alpha_i) = \sum_i \mu(\alpha_i)$ for pairwise disjoint sets $\{\alpha\}_i$.

[3] A partially ordered set $(P, \leq)$ is a set of elements such that element i is less than or equal to element $i + i$.

[4] See this great answer detailing how the difference between "measure" and "distribution" is basically just context.

Linear Algebra. Feeling like I need a quick recap from my adv. linalg course and an area where I can ramble my interpretations. In what follows, let V and W be vector spaces over some field F.

Linear Transformation

> *A function $T : V \mapsto W$ is called a **linear transformation** from V to W if $\forall x, y \in V$ and $\forall c \in F$:*
>
> - $T(x + y) = T(x) + T(y)$.
> - $T(cx) = cT(x)$.

Now suppose that V and W have ordered bases $\beta = \{v_1, \ldots, v_n\}$ and $\gamma = \{w_1, \ldots, w_m\}$, respectively. Then for each basis vector v_j, there exist unique scalars $a_{ij} \in F$ such that

$$T(v_j) = \sum_{i=1}^{m} a_{ij} w_i \tag{15}$$

Remember that each v_j and w_i are members of a *vector* space (they are not scalars). And also be careful to not associate the representation of any vector space element with its coordinate vector relative to a specific ordered basis, which *itself is a different linear transformation from some $V \mapsto F^n$*. Keep it abstract.

Matrix Representation of a Linear Transformation

> *We call the $m \times n$ matrix A defined by $A_{ij} = a_{ij}$ the **matrix representation of T** in the ordered bases β and γ and write $A = [T]_\beta^\gamma$. If $V = W$ and $\beta = \gamma$, then $A = [T]_\beta$.*

Given this definition, I think it's wise to interpret matrix representations by the column-vector point of view. Each column vector $[T(v_j)]_\gamma$, read as "the coordinate vector of $T(v_j)$ relative to ordered basis γ," tells you how each basis vector v_j in domain V gets mapped to a [coordinate] vector in output space W [relative to a given ordered basis γ]. For some reason, my brain has always had a hard time with the fact that the matrix row indices i correspond to output space, while the column indices j represent the input space. The easiest way (I think) to help ground this the right way is to remember that $L_A(x) \triangleq Ax$, i.e. the operator view. At the same time, notice how the effect of Ax is to take successive linear combinations over each element of x.

I just realized another reason why the interpretation felt backwards to my brain: when we are taught matrix multiplication, we do the computations Ax in our heads "row by row" along A, taking inner products of the row with x, so I've been taught to think of the rows as the main "units" of the matrix. I'm not sure how to fix this, but notice that the matrix is basically just a blueprint/roadmap/whatever you want to call it for taking coordinate vectors in one basis to coordinate vectors in another basis. It's really important to remember the coefficients of A are intimately tied to the input/output bases.

AHA. I've been thinking about this all wrong. For the longest time, I've been trying to force an interpretation of matrix multiplication that "feels" like scalar multiplication. I realize now that this is going about it all the wrong way. *Matrix multiplication need only be considered from*

the lens of a linear transformation. After all, that's exactly the purpose of matrices anyway! It's so glaringly obvious from the paragraphs above, but I guess I never took them seriously enough. Matrices are simply convenient ways for us to write down linear transformations on vectors in a given [ordered] basis. The jth column of the matrix defines how the original jth basis vector is transformed. **AHA** (again). Now I see why I missed this crucial connection – *everything above focuses on the formal definition of input ordered basis β to output ordered basis γ, but 99 percent of the time in real life we have either $\beta \subset \gamma$ or $\gamma \subset \beta$ (we usually are mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$).* For example, let $\boldsymbol{A} \in M^{m\times n}$ and $\boldsymbol{x} \in \mathbb{R}^n$; the following is always true:

$$\begin{aligned}\boldsymbol{A}\boldsymbol{x} &= L_A(\boldsymbol{x}) && (16)\\ &= \begin{bmatrix} L_A(\hat{\boldsymbol{e}}_1) & L_A(\hat{\boldsymbol{e}}_2) & \cdots & L_A(\hat{\boldsymbol{e}}_n)\end{bmatrix}\boldsymbol{x} && (17)\\ &= \sum_{i=1}^{n} L_A(\hat{\boldsymbol{e}}_i)x_i && (18)\end{aligned}$$

This viewpoint is painfully obvious to me now, but I guess I hadn't thought deeply enough about the implications of the definition of a linear transformation, and I *definitely* took the **representation** of a matrix way too seriously, rather than focusing on its **sole purpose**: provide a convenient way to write down linear transformations. For example, the above is actually a direct consequence of the definition of a L.T. itself:

$$\begin{aligned}L_A(\boldsymbol{x}) &= L_A(x_1\hat{\boldsymbol{e}}_1 + \cdots + x_n\hat{\boldsymbol{e}}_n) && (19)\\ &= x_1L_A(\hat{\boldsymbol{e}}_1) + \cdots + x_nL_A(\hat{\boldsymbol{e}}_n) && (20)\end{aligned}$$

Time to really nail in the understanding. I also remember getting screwed up trying to think about *ok, so how do I conceptualize of the ith element of $\boldsymbol{x}$ after the transformation? It's just a bunch of summed up goobley-gook!*. On one hand, yes that's true, but focus on the following before/after representations of $\boldsymbol{x}$ to make your life easier:

$$\boldsymbol{x} \triangleq \sum_i^n x_i\hat{\boldsymbol{e}}_i \quad \xrightarrow{Ax} \quad L_A(\boldsymbol{x}) \triangleq \sum_i^n x_iL_A(\hat{\boldsymbol{e}}_i) \tag{21}$$

Matrix-Matrix Multiplication. Continuing with the viewpoint that a matrix is nothing more than a convenient way to represent a linear transformation, recognize that any matrix multiplication AB represents a linear transformation itself, defined as $T := T_A \circ T_B$, the composition of A and B.

Matrix Multiplication and Neural Networks. Let's use the previous interpretations in the context of neural networks. A basic feedforward network with one hidden layer will compute outputs $\boldsymbol{o}$ given inputs $\boldsymbol{x}$, each of which are vectors of possibly different dimension:

$$\boldsymbol{o}(\boldsymbol{x}) = \boldsymbol{W}^{(o)}\boldsymbol{\phi}(\boldsymbol{x}) \tag{22}$$

$$\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{W}^{(h)}\boldsymbol{x} \tag{23}$$

where $\boldsymbol{W}^{(o)}$ and $\boldsymbol{W}^{(h)}$ are the output and hidden parameter matrices, respectively. We already know that we can interpret each columns of these matrices as how the input basis vectors get mapped to the hidden or output space. However, since we usually think of the parameter matrices as representing the weighted edges of a network, we often think in terms of individual units. For example, the ith unit of the hidden layer vector $\boldsymbol{h}$ is given by $h_i = \sum_j^{n_{in}} W_{ij}x_j = \langle \boldsymbol{W}_{i,:}, \boldsymbol{x} \rangle$. One interesting interpretation is that the ith element of $\boldsymbol{h}$ is a **projection**[5] of the input $\boldsymbol{x}$ onto the ith row of $\boldsymbol{W}$. This is of course true for any linear transformation; we can always think of the elements of the transformed vector as the result of projections of the original vector along a particular direction.

Determinants. $\det A$ is the volume of space that a unit [hyper] cube is mapped to. Starting with the simplest non-trivial case, let $A \in M^{2\times 2}$, and define A s.t. it simply scales the basis vectors (zero rotation). In other words, $A_{i,j\neq i} := 0$. In this case, $\det A = a_{11}a_{22}$, which is the area enclosed by the new scaled basis vectors. Skipping straight to the general case of [necessarily square] matrix $A \in M^{n\times n}$ using Einstein summation notation and the Levi-Cevita symbol[6]:

$$\det A \triangleq \varepsilon_{i_1,\ldots i_n} a_{1,i_1} \ldots a_{n,i_n} = \varepsilon_{i_1,\ldots i_n} \prod_{j=1}^{n} a_{j,i_j} \tag{25}$$

Consider that if $\det A = 0$, then T_A "squishes" the volume of space in such a way that we essentially lose one or more dimensions. Notice how it only takes *one* lost dimension, since the volume of any region in $\mathbb{R}^n$ is zero unless there is *some* amount in all dimensions (e.g. a cube with zero width has zero volume, regardless of its height/depth). It's also interesting to consider the relationships here with the invertible matrix theorem (defined a few paragraphs below). Having the intuition that determinants can be thought of as a change-in-volume makes it much more obvious why the equivalence statements of the invertible matrix theorem are indeed equivalent.

[5] This is informally worded. See the footnotes in the dot products section to understand why the element is technically just the result of a transformation (a projection would require re-mapping the scalar back to the space that $\boldsymbol{W}_{i,:}$ lives (input space)).

[6] Recall that

$$\varepsilon_{i_1,\ldots,i_n} \triangleq \begin{cases} +1 & \text{if } (i_1,\ldots,i_n) \text{ is even perm of } (1,\ldots,n) \\ -1 & \text{if } (i_1,\ldots,i_n) \text{ is odd perm of } (1,\ldots,n) \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

Note that this implies equal-to-zero if any of the indices are equal.

Dot Products and Projections. First, recall that a projection is *defined* to be a **linear** transformation P that is idempotent ($P^n = P$ for $n \geq 1$). Also, note that what you generally think of as a projection is technically an *orthogonal projection*[7].

Here we'll show the intimate link between the dot product and [orthogonal] projection. Let $P_{\boldsymbol{u}}$ define the [orthogonal] projection onto some **unit** *vector* $\boldsymbol{u} \in \mathbb{R}^n$ (more generally, we could project onto a subspace instead of a single vector[8]). We can re-cast this as a linear transformation $T_{\boldsymbol{u}} : \mathbb{R}^n \mapsto \mathbb{R}$ (technically not a *projection*, which would require re-mapping the output scalar back to $\mathbb{R}^n$). We interpret the scalar output of $T_{\boldsymbol{u}}(\boldsymbol{x})$ as the coordinate along the line spanned by $\{\boldsymbol{u}\}$ that input vector $\boldsymbol{x}$ gets mapped to. But wait, didn't we just talk a bunch about how to represent/conceptualize of the matrix representation of a transformation? Yes, we did. Well then, what would the matrix representation of $T_{\boldsymbol{u}}$ look like? Don't forget that we've defined $||\boldsymbol{u}|| = 1$.

$$[T_{\boldsymbol{u}}]^{\mathbb{R}}_{\mathbb{R}^n} = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \tag{26}$$

$$T_{\boldsymbol{u}}(\boldsymbol{x}) = \sum_i^n u_i x_i \tag{27}$$

$$\longrightarrow \quad = \boldsymbol{x} \cdot \boldsymbol{u} \tag{28}$$

Furthermore, since linear transformations satisfy $T(c\boldsymbol{x}) = cT(\boldsymbol{x})$ by definition, the final result is true even when $\boldsymbol{u}$ is not a unit vector.

Invertibility and Isomorphisms. A function is invertible IFF it is both one-to-one and onto (i.e. bijective). Recall that $rank(T)$ is the dimensionality of the range of T, which is the subspace of W that T maps to.

> *Let $T : V \mapsto W$ be a linear transformation, where V and W are finite-dimensional spaces of equal dimension. Then T is invertible IFF $rank(T) = dim(V)$.*

For any invertible functions T and U:

- $(TU)^{-1} = U^{-1}T^{-1}$. One easy way to show this is

$$(TU)U^{-1}T^{-1}(x) = T(UU^{-1})T^{-1}(x) = TT^{-1}(x) = x \tag{29}$$

- $(T^{-1})^{-1} = T$. The inverse of T is itself invertible.

[7] The more general definition uses wording "projection of vector $\boldsymbol{x}$ *along* $\boldsymbol{k}$ onto $\boldsymbol{m}$", where the distinction is shown in italics. An orthogonal projection implicitly defines $\boldsymbol{k}$ as its null space; for any $\alpha \in \mathbb{R}$, an orthogonal projection satisfies $P(\alpha\boldsymbol{k}) = \boldsymbol{0}$

[8] And technically, you don't project onto a *vector*, but rather you project onto a *line*, which is itself technically a subspace of 1 dimension. Yada yada yada.

Inverse of a partitioned matrix

Consider a general partitioned matrix,

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{E} & \boldsymbol{F} \\ \boldsymbol{G} & \boldsymbol{H} \end{pmatrix} \tag{30}$$

where $\boldsymbol{E}$ *and* $\boldsymbol{H}$ *are invertible. Then*

$$\boldsymbol{M}^{-1} = \begin{pmatrix} (\boldsymbol{M}/\boldsymbol{H})^{-1} & -(\boldsymbol{M}/\boldsymbol{H})^{-1}\boldsymbol{F}\boldsymbol{H}^{-1} \\ \boldsymbol{E}^{-1} + \boldsymbol{E}^{-1}\boldsymbol{F}(\boldsymbol{M}/\boldsymbol{E})^{-1}\boldsymbol{G}\boldsymbol{E}^{-1} & (\boldsymbol{M}/\boldsymbol{E})^{-1} \end{pmatrix} \tag{31}$$

where $\boldsymbol{M}/\boldsymbol{H} \triangleq \boldsymbol{E} - \boldsymbol{F}\boldsymbol{H}^{-1}\boldsymbol{G}$ (32)

$$\boldsymbol{M}/\boldsymbol{E} \triangleq \boldsymbol{H} - \boldsymbol{G}\boldsymbol{E}^{-1}\boldsymbol{F} \tag{33}$$

where $\boldsymbol{M}/\boldsymbol{H}$ *denotes the* ***Schur complement*** *of* $\boldsymbol{M}$ *w.r.t.* $\boldsymbol{H}$.

Invertible Matrix Theorem

Let $\boldsymbol{A}$ *be a square* $n \times n$ *matrix over some field* K. *The following statements are equivalent (I'll group statements that are nearly identical, too):*

- *The ones I consider useful:*
  1. $\boldsymbol{A}$ *is invertible.*
  2. $\boldsymbol{A}$ *is row-equivalent (and thus column-equivalent) to* $\boldsymbol{I}_n$.
  3. $\det \boldsymbol{A} \neq 0$.
  4. *rank*$(\boldsymbol{A}) = n$.
  5. *The columns of* $\boldsymbol{A}$ *are linearly independent. They span* K^n. *Col*$(\boldsymbol{A}) = K^n$.
  6. *The transformation* $T(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ *is a bijection from* K^n *to* K^n.
- *The rest:*
  1. $\boldsymbol{A}$ *has* n *pivot positions.*
  2. $\boldsymbol{A}$ *can be expressed as a finite product of elementary matrices.*

Understanding correlation vs independence.

- Two *events* A and B are **independent** iff $P(A \cap B) = P(A)P(B)$.
- Although $\text{Indep}(X, Y) \implies \text{cov}(X, Y)=0$, the converse is *not* true. It's useful to see that statement more explicitly:

$$[\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]] \not\Longrightarrow [P(X, Y) = P(X)P(Y)] \tag{34}$$

Example: Uncorrelated $\not\Rightarrow$ Independent

To get an intuition for this, I immediately try formalizing the possible cases where this is true. It seems that symmetry is always present in such cases, and they do seem like edge cases. The simplest and by far most common example is the case where we have x,y coordinates $\{(-1,0),(0,-1),(0,1),(1,0)\}$.

It's obvious that $X * Y$ equals zero for all of these points, and also that both X and Y are symmetric about the origin, meaning that $\mathbb{E}[XY] = 0 = \mathbb{E}[X]\mathbb{E}[Y]$. In other words, they are **uncorrelated**. The **key insight** comes from understanding *why* this is so: Regardless of whether one variable is either *positive* or *negative* the other is zero. I really want to nail this insight down, because it made me realize I was thinking about correlation wrong – I was thinking about it more as independence in the first place, and so looking back it's no wonder I was confused about the difference. You simply cannot think about correlation from the perspective of a single instance. For example, when I first read this, I thought "well if I know X is 1, then I know automatically that Y is zero", and although that is technically true, *that is not what correlation is about.* Rather, correlation is about *trends* of multiple instances. A more correct thought would be "Regardless of whether X is positive or negative, Y is zero, therefore positive values of X are neither positively nor negatively correlated with values of Y."

Now that we've got the thornier part (for me at least) out of the way, recognize that although X and Y are uncorrelated, they are *not* independent. This should be fairly obvious, since given either X or Y, we can say what the other's value is with higher certainty than otherwise.

Least-Squares Regression. Note how I emphasized *least-squares*, since in this section we measure how good an estimator is based on least-squares loss. Recall that least-squares arises naturally as a result of modeling $Y \sim \mathcal{N}(f(X), \varepsilon^2)$ and conducting MLE on the log probability of the data.

- **Linear Least Squares Estimate** (LLSE). The LLSE of response Y to X, which we'll denote as $L[Y \mid X]$, is defined as

$$L[Y \mid X] \triangleq \arg\min_{\hat{Y}} \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[(Y - \hat{Y}(X))^2\right] \tag{35}$$

$$\text{where} \qquad \hat{Y}(X) := a + bX \tag{36}$$

  where finding the optimal linear function $\hat{Y}(X)$ amounts to finding the optimal coefficients (a, b) over the dataset $\mathcal{D}$. Unfortunately, it seems that the main way of "deriving" the result is actually to just proven, *given* the result, that it does indeed minimize the MSE. So, with that said, we begin with the result:

$$L[Y \mid X] = \mathbb{E}[Y] + \frac{\text{cov}(X,Y)}{\text{Var}(X)}(X - \mathbb{E}[X]) \tag{37}$$

Proof: Eq 37 Minimizes the MSE

Formally, let $\mathcal{L}(X) \triangleq \{aX + b \mid a, b \in \mathbb{R}\}$. *Prove that* $\forall \hat{Y} \in \mathcal{L}(X)$,

$$\mathbb{E}\left[(Y - L[Y \mid X])^2\right] \leq \mathbb{E}\left[(Y - \hat{Y})^2\right] \tag{38}$$

---

1. Expand the general form of

$$\begin{aligned}\mathbb{E}\left[(Y - aX - b)^2\right] &= \mathbb{E}\left[((Y - L[Y \mid X]) + (L[Y \mid X] - aX - b))^2\right] &(39)\\ &= \mathbb{E}\left[(Y - L[Y \mid X])^2\right] \\ &\quad + 2\mathbb{E}\left[(Y - L[Y \mid X])(L[Y \mid X] - aX - b)\right] &(40)\\ &\quad + \mathbb{E}\left[(L[Y \mid X] - aX - b)^2\right]\end{aligned}$$

   Our next goal is to evaluate the term in red (spoiler alert: it is zero).
2. First, it is easy to show that $\mathbb{E}[Y - L[Y \mid X]] = 0$ by simple substitution/arithmetic. We can also show that[a] $\forall aX + b \in \mathcal{L}(X)$,

$$\mathbb{E}\left[(Y - L[Y \mid X])(aX + b)\right] = 0$$

   as well.
3. Since $L[Y \mid X] \in \mathcal{L}(X)$, it is *also* true that $\forall \hat{Y} \in \mathcal{L}(X)$, we know $(L[Y \mid X] - \hat{Y}) \in \mathcal{L}(X)$, too. Therefore, the red term from step 1 equates to zero.
4. We now know that our formula from step 1 can be written

$$\mathbb{E}\left[(Y - aX - b)^2\right] = \mathbb{E}\left[(Y - L[Y \mid X])^2\right] + \mathbb{E}\left[(L[Y \mid X] - aX - b)^2\right] \tag{41}$$

   Clearly, this minimized when $aX + b = L[Y \mid X]$.

[a] Also via simple substitution and using $\text{cov}(x, y) = \mathbb{E}[xy] - \mathbb{E}[x]\,\mathbb{E}[y]$

TODO: Figure out how this formulation is equivalent to the typical multivariate expression:

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{w}}_{OLS}^T \boldsymbol{x} \tag{42}$$

$$\hat{\boldsymbol{w}}_{OLS} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y} \tag{43}$$

Questions.

- **Q:** In general, how can one tell if a matrix $\boldsymbol{A}$ has an eigenvalue decomposition? [insert more conceptual matrix-related questions here . . .]
- **Q:** Let $\boldsymbol{A}$ be real-symmetric. What can we say about $\boldsymbol{A}$?
  - Proof that eigendecomposition $A = Q\Lambda Q^T$ exists: Wow this is apparently quite hard to prove according to many online sources. Guess I don't feel so bad now that it wasn't (and still isn't) obvious.
  - Eigendecomposition not unique. This is apparently because two or more eigenvectors may have same eigenvalue.

This is the principal axis theorem: if A symmetric, then orthonorm basis of e-vects exists.

Stuff I Forget.

- Existence of eigenvalues/eigenvectors. Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$.
  - λ is an eigenvalue of A iff it satisfies $\det(\lambda \boldsymbol{I} - \boldsymbol{A}) = 0$. Why? Because it is an equivalent statement as requiring that $(\lambda \boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} = 0$ has a nonzero solution for $\boldsymbol{x}$.
  - The following statements are equivalent:
    - A is diagonalizable.
    - A has n linearly independent eigenvectors.
  - The **eigenspace** of $\boldsymbol{A}$ corresponding to λ is the solution space of the homogeneous system $(\lambda I - A)x = 0$.
  - $\boldsymbol{A}$ has at most n distinct eigenvalues.

Most info here comes from chapter 5 of your "Elementary Linear Algebra" textbook (around pg305)

- Diagonalizability notes from 5.2 of advanced linear alg. book (261). Recall that $\boldsymbol{A}$ is defined to be diagonalizable if and only if there exists and ordered basis β for the space consisting of eigenvectors of $\boldsymbol{A}$.
  - If the standard way of finding eigenvalues leads to k distinct λ_i, then the corresponding set of k eigenvectors v_i are guaranteed to be linearly independent (but might not span the full space).
  - If $\boldsymbol{A}$ has n linearly independent eigenvectors, then $\boldsymbol{A}$ is diagonalizable.
  - The characteristic polynomial of any diagonalizable linear operator splits (can be factored into product of linear factors). The **algebraic multiplicity** of an eigenvalue λ is the largest positive integer k for which $(t - \lambda)^k$ is a factor of $f(t)$.

Recall that a linear operator is a special case of a linear map where the input space is the same as the output space.

- Expectation of a random vector. Defined as

$$\mathbb{E}[\mathbf{x}] = \begin{bmatrix} \mathbb{E}[x_1] \\ \vdots \\ \mathbb{E}[x_d] \end{bmatrix} \tag{44}$$

  You can work out that it separates like that (which is not intuitive/immediately obvious imo) by considering e.g. the case where $d = 2$. You'll end up finding that

$$\mathbb{E}[\mathbf{x}] = \sum_{x_1} \sum_{x_2} \boldsymbol{x} p(\mathbf{x} = \boldsymbol{x}) \tag{45}$$

$$= \begin{bmatrix} \mathbb{E}[x_1] \\ \mathbb{E}_{x_1}\left[\mathbb{E}_{x_2 \sim p(x_2|x_1)}[x_2 \mid x_1]\right] \end{bmatrix} \tag{46}$$

  and since we know from CS70 that $\mathbb{E}[\mathbb{E}[Y \mid X]] = \mathbb{E}[Y]$, we get the desired result.

$$\sinh x = \frac{e^x - e^{-x}}{2} \tag{47}$$

$$\cosh x = \frac{e^x + e^{-x}}{2} \tag{48}$$

$$\frac{d}{dx}\sinh x = \cosh x \tag{49}$$

$$\frac{d}{dx}\cosh x = \sinh x \tag{50}$$

$$\frac{d}{dx}\tanh x = \frac{\cosh^2 x - \sinh^2 x}{\cosh^2 x} \tag{51}$$

$$= \frac{1}{\cosh^2 x} \tag{52}$$

$$= \operatorname{sech}^2 x \tag{53}$$

Determinants. For any $n \times n$ matrix $\boldsymbol{A}$ with eigenvalues λ_i:

$$\det \boldsymbol{A} = \prod_i \lambda_i \tag{54}$$

$$\det \boldsymbol{A}^{-1} = \frac{1}{\det A} \tag{55}$$

$$\det \boldsymbol{A}^n = (\det \boldsymbol{A})^n \tag{56}$$

$$\det \left(\boldsymbol{I} + \boldsymbol{u}\boldsymbol{v}^T \right) = 1 + \boldsymbol{u}^T \boldsymbol{v} \tag{57}$$

2.3.1 Derivatives

$$\left[\frac{\partial \boldsymbol{x}}{\partial y} \right]_i = \frac{\partial x_i}{\partial y} \qquad \left[\frac{\partial x}{\partial \boldsymbol{y}} \right]_i = \frac{\partial x}{\partial y_i} \qquad \left[\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} \right]_{ij} = \frac{\partial x_i}{\partial y_j} \tag{58}$$

Derivatives wrt matrix $\boldsymbol{X}$.

[Frobenius norm]
$$\frac{\partial}{\partial \boldsymbol{X}} ||\boldsymbol{X}||_F^2 = \frac{\partial}{\partial \boldsymbol{X}} \mathrm{Tr}(\boldsymbol{X}\boldsymbol{X}^T) = 2\boldsymbol{X} \tag{59}$$

[chain rule]
$$\frac{\partial g(\boldsymbol{U})}{\partial X_{ij}} = \mathrm{Tr}\left[\left(\frac{\partial g(\boldsymbol{U})}{\partial \boldsymbol{U}} \right)^T \frac{\partial \boldsymbol{U}}{\partial X_{ij}} \right] \tag{60}$$

Main Tasks in NLP

Table of Contents Local *Written by Brandon McKinzie*

Going to start compiling a list of the main tasks in NLP (alphabetized). Note that NLP-Progress, a site dedicated to this purpose, is a much more detailed. I'm going for short-and-sweet here.

Constituency Parsing.

- **Task**: Generate parse tree of a sentence. Nodes are typically labeled by parts of speech and/or chunks.
  - A constituency parse tree breaks a text into sub-phrases, or constituents. Non-terminals in the tree are types of phrases, the terminals are the words in the sentence.

Coreference Resolution.

- **Task**: clustering mentions in text that refer to the same underlying real world entities.
- **SOTA**: End-to-end Neural Coreference Resolution.

Dependency Parsing.

- **Task**: Given a sentence, generate its dependency tree (DT). A DT is a labeled directed tree whose nodes are individual words, and whose edges are directed arcs labeled with dependency types.
  - A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words which modify those heads.
- **SOTA**: Deep Biaffine Attention for Neural Dependency Parsing.
  Related Tasks:
  - Constituency parsing. See this great wiki explanation of dependency vs constituency.

Information Extraction.

- **Task**: Given a (typically long) portion of raw text, recover information about pre-specified relations, entities, events, etc.

Language Modeling.

- **Task**: Learning the probability distribution over text sequences. Often used for predicting the next word in a sequence, given the K previous words.
- **SOTA**: ELMo.

Machine Translation.

Semantic Parsing.

- **Task**: Translating natural language into a formal meaning representation *on which a machine can act*.

Semantic Role Labeling.

- **Task**: Given a sentence, extract the predicates[9] and their respective arguments.
- **Historical Approaches**.
  - **CCG Semantic Parsing**. Zettlemoyer & Collins 2005, 2007.
  - Seq2seq. Dong & Lapata, 2016.

Sentiment Analysis.

- **Task**: Determining whether a piece of text is positve, negativ, or neutral.
- **SOTA**: Biattentive classification network (BCN) from Learned in Translation: Contextualized Word Vectors (the CoVe paper) with ELMo embeddings.

Summarization.

Textual Entailment.

- **Task**: Given a premise, determine whether a proposed hypothesis is true.
- **SOTA**: Enhanced Sequenctial Inference (ESIM) model from Enhanced LSTM for Natural Language Inference with ELMo embeddings.

[9]The predicate of a sentence mostly corresponds to the main verb and any auxiliaries that accompany the main verb; whereas the arguments of that predicate (e.g. the subject and object noun phrases) are outside the predicate.

Topic Modeling.

Question Answering. Also called **machine reading comprehension**.

- **Task**: Given a paragraph of text, "read" it and then answer questions pertaining to the text.
- **Dataset**: The main benchmark dataset is the *Stanford Question Answering Dataset* (SQuAD). Each sample has the form `({question, paragraph}, answer)`, where the answer is a subsequence found somewhere in the paragraph (i.e. this is an *extractive* task, not abstractive).
- **SOTA**: Improved versions of the Bidirectional Attention Flow (BiDAF) model, with ELMo embeddings.
- **Related tasks**:

Word Sense Disambiguation.

- **Task**: Associating words in context with their most suitable entry in a pre-defined sense inventory.

2.5.1 BLEU Score

BiLingual Evaluation Understudy. For scoring machine-generated translations when we have access to one or more reference translations.

- Unigram precision: Really naive and basically useless version:

$$P = \frac{\textbf{num pred words that also appear somewhere in ref words}}{\textbf{total num pred words}} \tag{61}$$

  It's important to emphasize how ridiculous this really is. It literally means that we walk along each word in the prediction and ask "is this word somewhere in any of the reference translations?" and if that answer is "yes", we +1 to the numerator. Period.
- Modified unigram precision: actually considering how many times we've mentioned a given word w when incorporating it into the precision calculation. Now, when walking along a sentence, we add to the aforementioned question, "...and have I seen it less than N times already?" where `N = [max(count(sent, w)) for sent in refs]`. This means our numerator can now be at most N for any given word.
- Generalize to n-grams. Below is the formula for Blue score on n-grams only:

$$p_n(\hat{y}) = \frac{\sum_{\text{ngrams}\in\hat{y}} \text{Count}_{clip}(\text{ngram}, \text{refs})}{\sum_{\text{ngrams}\in\hat{y}} \text{Count}(\text{ngram})} \tag{62}$$

- Combined Blue score.

$$= \text{BP} \cdot \exp\left\{\frac{1}{4}\sum_{n=1}^{4} \log p_n\right\} \tag{63}$$

$$\text{BP} = \begin{cases} 1 & \text{len(pred)} > \text{len(ref)} \\ \exp\{1 - \text{len(pred)}/\text{len(ref)}\} & \text{otherwise} \end{cases} \tag{64}$$

  where BP is the **brevity penalty**.

2.5.2 Connectionist Temporal Classification (CTC)

Approach for mapping input sequences $X = \{x_1, \ldots, x_T\}$ to label sequences $Y = \{y_1, \ldots, y_U\}$, where the lengths may vary ($T \neq U$). First, we need to define the meaning of an **alignment** between input sequence X and label sequence Y. Most generally, an alignment is a composition of one or more functions, and accept input X and ultimately map to output Y. Take, for example, the label sequence $Y = \{h, e, l, l, o\}$ and some input sequence (e.g. raw audio) $X = \{x_1, \ldots, x_{12}\}$. CTC places the following constraints on the [first function of the] alignment sequence:

1. It must be the same length as the input sequence X.
2. It has the same vocabulary as Y, plus an additional token ϵ to denote blanks.
3. At position i, it either (a) repeats the aligned token at $i-1$, (b) assigns the empty token ϵ, or (c) assigns the next letter of the label sequence.

For our example, we could have an aligned sequence $A = \{h, h, e, \epsilon, \epsilon, l, l, l, \epsilon, l, l, o\}$. Then we apply the following two steps (can interpret as functions) to map from A to Y:

1. Merge any repeated [contiguous] characters.
2. Remove any ϵ tokens.

When you hear someone say "the CTC loss," they usually mean "MLE using a CTC posterior." In other words, there is no "CTC loss" function, but rather there is the standard maximum likelihood objective, but we use a particular form for the posterior $p(Y \mid X)$ over possible output labels Y given raw input sequence X:

$$p(Y \mid X) = \sum_{\mathcal{A} \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X) \tag{65}$$

where $\mathcal{A}$ is one of the valid alignments from $\mathcal{A}_{X,Y}$, the full set of valid alignments from X to a given output sequence Y. The per-timestep probabilities $p(a_t \mid X)$ can be given by, for example, an RNN.

Number of Valid Alignments

Given X of length T and Y of length $U \leq T$ (and no repeating letters), how many valid alignments exist?

The differences between alignments fall under two categories:

1. Indices where we transition from one label to the next.
2. Indices where we insert the blank token, ϵ.

Stated even simpler, the alignments differ first and foremost by *which elements of X are "extra" tokens*, where I'm using "extra" to mean either blank or repeat token. Given a set of T tokens, there are $\binom{T}{T-U}$ different ways to assign $T - U$ of them as "extra." The tricky part is that we can't just randomly decide to repeat or insert a blank, since a sequence of one or more blanks is *always* followed by a transition to next letter, by definition. And remember, we have defined Y to have no repeated [contiguous] labels.

Apparently, the answer is $\binom{T+U}{T-U}$ total valid alignments.

Computing forward probabilities $\alpha_t(s)$, defined as the probability of arriving at [prefix of] augmented label sequence $\boldsymbol{\ell}'_{\langle 1 \ldots s \rangle}$ given unmerged alignments up to step t. There are two cases to consider.

1. (1.1) The augmented label at step s, $\boldsymbol{\ell}'_s$ is the blank token ϵ. Remember, ϵ occurs at every other position in the augmented sequence $\boldsymbol{\ell}'$. At the previous RNN output (time $t-1$), we could've emitted either a blank token ϵ or the previous token in the augmented label sequence, $\boldsymbol{\ell}'_{s-1}$. In other words,

$$y^t_{\boldsymbol{\ell}'_s = \epsilon} \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1) \right) \tag{66}$$

2. (1.2) The augmented label at step s, $\boldsymbol{\ell}'_s$ is the same augmented label as at step $s-2$. This occurs when the [not augmented] label sequence has repeated labels next to each other.

$$y^t_{\boldsymbol{\ell}'_s = \boldsymbol{\ell}'_{s-2}} \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1) \right) \tag{67}$$

   In this situation, $\alpha_{t-1}(s)$ corresponds to us just emitting the same token as we did at $t-1$ or emitting a blank token ϵ, and $\alpha_{t-1}(s-1)$ corresponds to a transition to/from ϵ and a label.
3. (2) The augmented label at step s - 1, $\boldsymbol{\ell}'_{s-1}$ is the blank token ϵ between unique characters. In addition to the two α_{t-1} terms from before, we now also must consider the possibility that our RNN emitted $\boldsymbol{\ell}'_{s-2}$ at the previous time $(t-1)$ and then emitted $\boldsymbol{\ell}'_s$ immediately after at time t.

2.5.3 Perplexity

Per Wikipedia:

> *In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample.*

The perplexity, $\mathcal{P}$, of discrete probability distribution p over word sequences $W = \{w_1, \dots w_T\} = \boldsymbol{w}_{\langle 1 \dots T \rangle}$ of length T is defined as:

$$\mathcal{P}(p) = 2^{-\mathbb{E}_{\boldsymbol{x} \sim p}[\lg p(\boldsymbol{x})]} = 2^{H(p)} \tag{68}$$

$$= 2^{-\sum_{\boldsymbol{w}_{\langle 1 \dots T \rangle}} p(\boldsymbol{w}_{\langle 1 \dots T \rangle}) \lg p(\boldsymbol{w}_{\langle 1 \dots T \rangle})} \quad \textbf{[theory]} \tag{69}$$

$$\approx 2^{-\frac{1}{N} \sum_{\boldsymbol{w}_{\langle 1 \dots T \rangle} \in \mathcal{T}} \lg q(\boldsymbol{w}_{\langle 1 \dots T \rangle})} \quad \textbf{[empirical]} \tag{70}$$

where H is entropy (in bits). It's important to note that, in practice, we are never able to use the theoretical version since we don't know p exactly (we are usually trying to estimate it) – instead of $H(p)$ we thus usually think in terms of $H(p, q)$, the *cross entropy*[10]. The empirical definition is when we have N samples in some test set $\mathcal{T}$, and a model q that we want to approximate the true distribution p.

In NLP, it is more common to want the *per-word* perplexity of a language model. We typically do this by flattening out a sequence of words in some test set containing M words total and simply compute

$$\mathcal{P} = 2^{-\frac{1}{M} \lg p(\{w_1, \dots w_M\})} \tag{71}$$

$$= \frac{1}{p(\{w_1, \dots w_M\})^{\frac{1}{M}}} \tag{72}$$

In other words, NLP nearly always defines perplexity as the **inverse probability of the test set**, normalized by number of words. So, why is this valid? We are implicitly assuming that language sources are **ergodic**:

Ergodic

> *A random process is **ergodic** if its (asymptotic) time average is the same as its expectation value over all possible states (w.r.t the specified probability distribution).*
>
> *Informally, this means that the system eventually reaches all states, and such that the probability of observing it in state s is $p(s)$, where p is the true generating distribution.*

[10] Recall the relationship between entropy $H(p)$ and cross entropy $H(p, q)$:

$$H(p, q) = H(p) + D_{KL}(p||q)$$

In the per-word NLP case, this means we can assume that

$$\lim_{m\to\infty} \mathbb{E}\left[\lg p(\{w_1, \dots w_m\})\right] = \lim_{m\to\infty} \frac{1}{m} \lg p(\{w_1, \dots w_m\}) \tag{73}$$

where the sequence on the RHS is any sample from p[11].

Intuition. Ok, now that we've got definitions out of the way, what does it actually mean? First consider some limiting cases. If the distribution p is uniform over N possible outcomes, then $\mathcal{P}(p) = 2^{\lg N} = N$. Since the uniform distribution has the highest possible entropy, N is also the largest possible value for perplexity of a discrete distribution p over N possible outcomes.

Consider the interpretation of the cross entropy loss as the negative log-likelihood:

$$NLL(p_{data}) = -\frac{1}{M}\sum_{i=1}^{M} \log p(w{=}w_i) = \mathbb{E}_{w_i\sim p_{data}}\left[\log \frac{1}{p(w)}\right] \tag{74}$$

we see that NLL (and thus $\mathcal{P}{=}\exp(NLL)$) decreases as our model assigns higher probabilities to samples drawn from p_{data}. *Better models of p_{data} are less surprised by samples from p_{data}.* If we use the typical interpretation of entropy as the number of bits needed (on average) to represent a sample from p, then the perplexity can be interpreted as the total number of possible results (on average) when drawing a sample from p.

In the case of language modeling, this represents the total number of reasonable next-word predictions for w_{t+1} given some context $w_1, \dots w_t$. As our model assigns higher probabilities to the true samples in p_{data}, the number of bits required to specify each word, on average, becomes smaller. Therefore, you can roughly think of per-word perplexity as telling you the number of possible choices, on average, your model considers uniformly at random at a given step. For example, $\mathcal{P} = 42$ could be interpreted loosely as "to predict the next word out of some vocabulary V, my model can narrow it down on average to about 42 choices, and chooses uniformly at random from that subset", where typically $|V| >> 42$.

[11] Something feels off here. I'm synthesizing what I'm reading from wikipedia and this source from berkeley but I can't fix the sloppy tone of the wording.

2.5.4 Byte Pair Encoding

2.5.5 Grammars

In formal language theory, a **formal grammar** is a set of production rules for strings in a formal language. The rules describe how to form strings from the language's alphabet that are valid according to the language's syntax. A grammar does not describe the meaning of the strings or what can be done with them in whatever context—only their form.

- **Regular Grammar**: no rule has more than one nonterminal in its right-hand side, and each of these nonterminals is at the same end of the right-hand side. Every regular grammar corresponds directly to a nondeterministic finite automaton.
- A context-free grammar (**CFG**) is a formal grammar that consists of:
  - **Terminal symbols**: characters that appear in the strings generated by the grammar.
  - **Nonterminal symbols**: placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols.
  - **Productions**: rules for replacing (or rewriting) nonterminal symbols (on the LHS) in a string with other nonterminal or terminal symbols (on the RHS), *which can be applied regardless of context.*
  - **Start symbol**: a special nonterminal symbol that appears in the initial string generated by the grammar.

  To generate a string of terminal symbols from a CFG, we:
  1. Begin with a string consisting of the start symbol;
  2. Apply one of the productions with the start symbol on the left hand size, replacing the start symbol with the right hand side of the production;
  3. Repeat the process of selecting nonterminal symbols in the string, and replacing them with the right hand side of some corresponding production, until all nonterminals have been replaced by terminal symbols.
- A **Probabilistic CFG** extends CFGs the same way has HMMs extend regular grammars, by defining the set P of probabilities on production rules.

2.5.6 Bloom Filter

Data structure for querying whether a data point is a member of some set. It returns either "no" or "maybe". It is implemented as a bit vector. Each member of the set is passed through k hash functions. Each hash function maps an element to an integer index. For each member, we set the k output indices of the hash functions to 1 in our bit vector. To answer if some data point x is in the set, we pass x through the k hash functions, which gives us k indices. If all k indices have their bit value set to 1, the answer is "maybe", otherwise (if any bit value is 0) the answer is "no".

2.5.7 Distributed Training

Asynchronous SGD.

$$W_{i+1} = W_i - \frac{\alpha}{N_x} \sum_{j=1}^{N_x} \frac{\partial L(\boldsymbol{x}^{(j)})}{\partial W_i} \tag{75}$$

[SyncSGD]
$$W_{i+1} = W_i - \lambda \sum_{j=1}^{N_w} \sum_{k=1}^{N_x(j)} \alpha \frac{\partial L(\boldsymbol{x}^{(k)})}{\partial W_i} \tag{76}$$

where N_x is the number of data samples.

In asynchronous SGD, we just apply the gradient updates to a global version of the parameters whenever they are available. In practice, this can result in **stale gradients**, which happens when a worker takes a long time to compute some gradient step, while the master version of the parameters has been updated many times. This results is the master computing an update like $W_{t+1} = W_t - \lambda \Delta W_{t-D}$ for larger-than-desired values of D (delay in num updates).

2.5.8 Traditional Language Modeling

Some quick terminology/definitions since my brain forgets things.

- **Backoff**: when we want to estimate e.g. $p(w_1, w_2, w_3)$ but we've never seen the sequence w_1, w_2, w_3. We can instead *backoff* to bigrams if we *have* seen e.g. the sequences w_1, w_2 and w_2, w_3. More generally, if we have many N-gram LMs with different values of N, we backoff to the highest order LM that contains the N-gram we are querying for.
  - Example failure mode of backoff: we typically have to backoff for unusual sequences of words (by definition). The lower order N gram model could drastically overestimate the backoff probabilities[12].
- **Interpolation**: Using a combination of N-gram LMs with different values of N as an attempt to utilize the strengths of each and mitigate their weaknesses.
- **Kneser-Ney**: (link)

$$P_{KN}(w_t \mid w_{t-1}) = \frac{\max\left(c(w_{t-1}, w_t), 0\right)}{c(w_{t-1})} + \lambda \frac{|\{w_{t-1} : c(w_{t-1}, w_t) > 0\}|}{|\{w_{t'-1} : c(w_{t'-1}, w'_t) > 0\}|} \tag{77}$$

Click this link to see a really good overview of the terms above and more.

2.5.9 Datasets

MS COCO. (website). Bunch of images I guess?

Visual Question Answering (VQA v2). (website) Answers for natural language questions about an image.

Count statistics:

- 204,721 COCO images
- 1,105,904 questions
- 11,059,040 ground truth answers
- $\approx$ 5.4 questions per image.

Processing steps:

- Spelling correction (using Bing Speller) of question and answer strings.
- Question normalization (first char uppercase, last char '?').
- Answer normalization (all chars lowercase, no period except as decimal point, number words —> digits, strip articles (a, an the))

[12] Good example is how a unigram model assigns a decent probability for "York" but a human bigram could tell you that it is nearly certain that "New" preceded it. The backoff model would tend to overestimate p(York) since it has no contextual information

- Adding apostrophe if a contraction is missing it (e.g., convert "dont" to "don't")

It seems that everyone reports "accuracy" on VQA using the following formula?

$$acc = \min\left(\frac{\#\text{ humans that provided that answer}}{3}, 1\right) \tag{78}$$

They gather 10 answers for each question from unique workers (so 1 answer per worker, 10 workers).