repair_vs_redundancy

November 21, 2019

1 Different strategies for dealing with noise

The idea here is that there are different possible strategies of dealing with noise that are attested in natural communication systems:

- **Reduplication**: simply repeat the signal several times. This is compatible with a compositional system. It is not costly in terms of learnability, because the only extra thing that needs to be learned is a single extra rule that applies to all signals. However, it is relatively costly in terms of utterance length (it would thus not do well under a pressure for minimal effort).
- **Diversify signal**: make the individual segments that each signal consists of as distinct as possible. For example, in the language shown below, all four signals can be distinguished from each other in all cases where one character is obscured by noise. This strategy however is not compatible with compositionality, because it relies on making each of the segments as distinct from each other as possible. That means these languages are necessarily holistic, and therefore less easy to learn (so they would do less well under a pressure for learnability).
    - 02 –> 'aaaa'
    - 03 –> 'bbbb'
    - 12 –> 'abba'
    - 13 –> 'baab'
- **Repair**: This strategy could be seen as a form of redundancy across turns, instead of within a signal. However, it will be initiated only when neccesary, and should therefore fare slightly better than the reduplication strategy under a pressure for minimal effort (where effort is measured as total shared utterance length of both interlocutors across a given number of interactions).

The predictions of Vinicius & Seán (2016 Evolang abstract titled "Language adapts to signal disruption in interaction"), are that although the reduplication strategy and the repair strategy should do equally well under a pressure for learnability, adding the possibility of repair will 'lift the pressure for redundancy', such that receivers can request that speakers repeat a signal only after a problem occurs. The line of reasoning of Vinicius & Seán in this Evolang abstract is that although the reduplication strategy would do relatively well under a pressure for learnability, it has not been adopted widely in natural language. They argue that this is a result of the fact that repair was available. —> Another way of looking at this is that the repair strategy will do better than the reduplication strategy under a pressure for minimal effort (if measured in terms of shared utterance length of speaker and hearer across a fixed number of interactions). Such a pressure for minimal effort, in combination with a pressure for mutual understanding (a.k.a. expressivity) would provide a push for using repair, and once repair is used, it will 'lift the pressure for redundancy', as Vinicius & Seán put it.

1.1 Predictions under different selection pressures:

We predict that under the following assumptions: - There is a pressure for expressivity/mutual understanding - Noise regularly disrupts part of the signal (Vinicius & Seán used a 0.5 probability in their experiment) - Repair is a possibility

the following strategies will become dominant under the following combinations of the presense/absence of a pressure for learnability and a pressure for minimal effort:

	- minimal effort	+ minimal effort
- learnability	Any of the three strategies above will do	Repair + Compositional OR Holistic
+ learnability	Reduplication + Compositional OR Repair + Compositional	Repair + Compositional

Note that the prediction in the {-learnability, +minimal effort} condition above only holds if we do not distinguish between open and closed requests. Because if we do, as in the model we submitted to evolang, we'd expect the Repair + Compositional strategy to fare best in this condition (i.e. better than Repair + Holistic), without the need for a pressure for learnability.

Note also that the coding length of the Reduplication + Compositional languages *is* slightly higher than the coding length of the Repair + Compositional languages (ratio reduplication:repair = 1.09:1; see calculations in section 1.3.2), so depending on the bottleneck, we might expect the Repair + Compositional strategy to win out even in the {+learnability, -minimal effort} condition.

1.2 How to represent languages?

1.2.1 Possibility 1: Different form lengths

If we continue with Kirby et al.'s (2015) way of representing meanings and forms (which is a minimal way of creating languages that we can classify as compositional, holistic or degenerate), where meanings consist of $f = 2$ features, which can each have $v = 2$ values, we can allow for each of the language strategies specified above ('reduplication' and 'diversify signal'), by simply allowing for multiple string lengths l, while keeping the alphabet size $|\Sigma|$ at 2.

For example, where Kirby et al. (2015) only allowed for a single possible string length, and specified $f = v = l = |\Sigma| = 2$, we could minimally allow for two possible string lengths: one being equal to f (i.e. the minimum string length required to uniquely specify each meaning feature), and one being equal to $2 * f$, to enable reduplication of the signal.

That would yield the following types of languages:

Reduplication + compositional:

02 –> aaaa

03 –> abab

12 –> baba

13 –> bbbb

Diversify signal + holistic:

02 –> aaaa

03 –> bbbb

12 –> abba

13 –> baab

Repair + compositional:

02 –> aa

03 –> ab

12 –> ba

13 –> bb

In order to still make it possible for iterated learning chains to transition from a language that uses forms of length 2 into a language that uses forms of length 4 and vice versa, we need to then also allow for languages that use a mixture of form lengths (e.g. three forms of length 2, and one form of length 4). This yields the following number of possible languages:

$$(2^2 + 2^4)^4 = 160000$$

which means that compared to the Kirby et al. (2015) model (where there were $((2^2)^4 = 256$ possible languages), the hypothesis space expands by a factor of 625. That is not ideal, because if we assume that simulation run times increase linearly with the size of the hypothesis space, a simulation that took 1 hour to run in our previous model would now take almost 4 weeks to run.

However, this linear relationship between the simulation run times and the size of the hypothesis space holds when during learning, we actually loop through each hypothesis and update its posterior probability based on the data. There are a couple of ways in which this process can be optimised:

1. **memoisation**: This would require enumerating all possible data points (i.e. <meaning, form> pairs) (including all possible noisy forms), and for each of them calculating its likelihood for all possible hypotheses **once**, and caching the result. Whenever the same <meaning, form> pair is then encountered by any learner, the corresponding likelihood vector is then simply retrieved from memory and multiplied with the learner's current posterior. This should be doable given that the total number of meanings is 4, and the total number of forms (including all possible noisy variants, assuming that noise is restricted to a single character) is 56; which makes 4*56 = 224 possible <meaning, form> pairs. For each of those 224 possible datapoints, we would then calculate its likelihood for all 160,000 hypotheses, and cache these values in a 224*160,000 matrix. (That matrix thus has 224*160,000 = 35,840,000 entries.)
2. Intergenerational learning could be sped up by representing data as simple counts of <meaning, form> pairs, and simply updating the posterior probability distribution for the full data set in one step, by multiplying the prior of the hypothesis with the likelihood of the <meaning, form> pair to the power of the number of times it occurs in the data set. This should speed things up a little in intergenerational learning, but won't make a difference in *intra*generational learning, because there we assume that the hearer updates their posterior in each interaction.

3. Do not do exact inference over the full hypothesis space at all, but instead use an MCMC sampling technique (e.g. Burkett & Griffiths, 2010, and Kirby et al., 2015 use Gibbs sampling) –> This would require a bit more time to figure out, and is hopefully not necessary once optimisations 1 and 2 above have been implemented.

1.2.2 Possibility 2: Allow reduplication as grammatical rule, and increase alphabet size

If instead of allowing for multiple form lengths, we instead increase the size of the alphabet Σ from 2 to 4, that will make the diversify signal strategy possible. More concretely, that would mean that instead of there being an alphabet $[a, b]$, there would be an alphabet $[a, b, c, d]$. That would allow for the following example languages, where the bit at the end of the signal specifies whether the signal should be repeated (1) or not (0).

Reduplication + compositional:

02 –> aa1

03 –> ab1

12 –> ba1

13 –> bb1

Diversify signal + holistic:

02 –> aa0

03 –> bb0

12 –> cc0

13 –> dd0

Repair + compositional:

02 –> aa0

03 –> ab0

12 –> ba0

13 –> bb0

Choosing for this option would mean that instead of there being (22 + 24) = 20 possible forms, there would be 42 = 16 possible forms, and therefore $(4^2)^4 = 65536$ possible languages. In addition however, we'd need languages to have an extra bit that specifies whether signals are reduplicated or not (assuming there are only two options: reduplication ON versus reduplication OFF). That means that there'd be a total of $((4^2)^4) * 2 = 131072$. So compared to possibility 1, possibility 2 only reduces the size of the hypothesis space by a factor of 1.22. That is not much, but an added advantage of this way of representing languages is that it allows for a straightforward way of capturing the assumed simplicity of a reduplication rule in the coding of the languages, and therefore into the prior.

If we find that despite the optimisation strategies outlined above it is still not feasible to run simulations within a reasonable time-frame, we could consider tackling the different possible strategies for dealing with noise separately. I.e. one model where we allow for the possibility to add reduplication to signals vs. repair, and another model where we allow for diversification of signal segments

1.3 Compressibility measure for two different language representation possibilities

1.3.1 Possibility 1: Different form lengths

Rewrite rules: Reduplication + compositional:

There are in fact two different ways of reduplicating a compositional language: either reduplicating the whole signal, or reduplicating each of the segments. In both cases the length of the minimally redundant form, and therefore the language type's compressibility, will be the same however, as shown below.

Reduplicate whole signal:

Language:

02 –> aaaa

03 –> abab

12 –> baba

13 –> bbbb

Rewrite rules:

S –> ABAB

A:0 –> a

A:1 –> b

B:2 –> a

B:3 –> b

Minimally redundant form:

SABAB.A0a.A1b.B2a.B3b

Reduplicate each segment:

Language:

02 –> aaaa

03 –> aabb

12 –> bbaa

13 –> bbbb

Rewrite rules:

S –> AABB

A:0 –> a

A:1 –> b

B:2 –> a

B:3 –> b

Minimally redundant form:

SAABB.A0a.A1b.B2a.B3b

Diversify signal + holistic:

02 –> aaaa

03 –> bbbb

12 –> abba

13 –> baab

Rewrite rules:

S:02 –> aaaa

S:03 –> bbbb

S:12 –> abba

S:13 –> baab

Minimally redundant form:

S02aaaa.S03bbbb.S12abba.S13baab

Repair + compositional:

02 –> aa

03 –> ab

12 –> ba

13 –> bb

Rewrite rules:

S –> AB

A:0 –> a

A:1 –> b

B:2 –> a

B:3 –> b

Minimally redundant form:

SAB.A0a.A1b.B2a.B3b

1.3.2 Now let's calculate the actual compressibility in terms of coding length, given the strings in minimally redundant form specified above:

[1]:

```
def classify_language_four_forms(lang, forms, meaning_list):
    """
    Classify one particular language as either 0 = degenerate, 1 = holistic, 2␣
↪= hybrid, 3 = compositional, 4 = other
    (Kirby et al., 2015). NOTE that this function is specific to classifying␣
↪languages that consist of exactly 4 forms,
    where each form consists of exactly 2 characters. For a more general␣
↪version of this function, see
    classify_language_general() below.

    :param lang: a language; represented as a tuple of␣
↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param forms: list of strings corresponding to all possible␣
↪forms_without_noisy_variants
    :param meaning_list: list of strings corresponding to all possible meanings
    :returns: integer corresponding to category that language belongs to:
    0 = degenerate, 1 = holistic, 2 = hybrid, 3 = compositional, 4 = other␣
↪(here I'm following the
    ordering used in the Kirby et al., 2015 paper; NOT the ordering from␣
↪SimLang lab 21)
    """
    class_degenerate = 0
    class_holistic = 1
    class_hybrid = 2  # this is a hybrid between a holistic and a compositional␣
↪language; where *half* of the partial
    # forms is mapped consistently to partial meanings (instead of that being␣
↪the case for *all* partial forms)
    class_compositional = 3
    class_other = 4

    # First check whether some conditions are met, bc this function hasn't been␣
↪coded up in the most general way yet:
    if len(forms) != 4:
        raise ValueError(
            "This function only works for a world in which there are 4 possible␣
↪forms_without_noisy_variants"
        )
    if len(forms[0]) != 2:
        raise ValueError(
            "This function only works when each form consists of 2 elements")
    if len(lang) != len(meaning_list):
        raise ValueError("Lang should have same length as meanings")
    # lang is degenerate if it uses the same form for every meaning:
    if lang[0] == lang[1] and lang[1] == lang[2] and lang[2] == lang[3]:
        return class_degenerate

    # lang is compositional if it makes use of all possible␣
↪forms_without_noisy_variants, *and* each form element maps
    # to the same meaning element for each form:
    elif forms[0] in lang and forms[1] in lang and forms[2] in lang and forms[
        3] in lang and lang[0][0] == lang[1][0] and lang[2][0] == lang[3][0]␣
↪and lang[0][
        1] == lang[2][1] and lang[1][1] == lang[3][1]:
        return class_compositional

    # lang is holistic if it is *not* compositional, but *does* make use of all␣
↪possible forms_without_noisy_variants:
    elif forms[0] in lang and forms[1] in lang and forms[2] in lang and␣
↪forms[3] in lang:
        # within holistic languages, we can distinguish between those in which␣
↪at least one part form is mapped
        # consistently onto one part meaning. This class we will call 'hybrid'␣
↪(because for the purposes of repair, it
        # is a hybrid between a holistic and a compositional language, because␣
↪for half of the possible noisy forms that
        # a listener could receive it allows the listener to figure out *part*␣
↪of the meaning, and therefore use a
        # restricted request for repair instead of an open request.
        if lang[0][0] == lang[1][0] and lang[2][0] == lang[3][0]:
            return class_hybrid
        elif lang[0][1] == lang[2][1] and lang[1][1] == lang[3][1]:
            return class_hybrid
        else:
            return class_holistic

    # In all other cases, a language belongs to the 'other' category:
    else:
        return class_other
```

[2]:
```
from math import log2
import string


def mrf_degenerate(lang, meaning_list):
    """
    Takes a degenerate language and returns a minimally redundant form␣
↪description of the language's context free
    grammar.
    :param lang: a language; represented as a tuple of␣
 ↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param meaning_list: list of strings corresponding to all possible meanings
    :return: minimally redundant form description of the language's context␣
 ↪free grammar (string)
    """
    mrf_string = 'S'
    for i in range(len(meaning_list)):
        meaning = meaning_list[i]
        if i != len(meaning_list) - 1:
            mrf_string += str(meaning) + ','
        else:
            mrf_string += str(meaning)
    mrf_string += lang[0]
    return mrf_string


def mrf_holistic(lang, meaning_list):
    """
    Takes a holistic OR hybrid language and returns a minimally redundant form␣
 ↪description of the language's context
    free grammar.

    :param lang: a language; represented as a tuple of␣
 ↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param meaning_list: list of strings corresponding to all possible meanings
    :return: minimally redundant form description of the language's context␣
 ↪free grammar (string)
    """
    mrf_string = ''
    for i in range(len(meaning_list)):
        meaning = meaning_list[i]
        form = lang[i]
        if i != len(meaning_list) - 1:
            mrf_string += 'S' + meaning + form + '.'
        else:
            mrf_string += 'S' + meaning + form
    return mrf_string


def mrf_compositional(lang, meaning_list):
    """
    Takes a compositional language and returns a minimally redundant form␣
 ↪description of the language's context free
```

```
    grammar.

    :param lang: a language; represented as a tuple of␣
 ↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param meaning_list: list of strings corresponding to all possible meanings
    :return: minimally redundant form description of the language's context␣
 ↪free grammar (string)
    """
    n_features = len(meaning_list[0])
    non_terminals = string.ascii_uppercase[:n_features]
    mrf_string = 'S' + non_terminals
    for i in range(len(non_terminals)):
        non_terminal_symbol = non_terminals[i]
        feature_values = []
        feature_value_segments = []
        for j in range(len(meaning_list)):
            if meaning_list[j][i] not in feature_values:
                feature_values.append(meaning_list[j][i])
                feature_value_segments.append(lang[j][i])
        for k in range(len(feature_values)):
            value = feature_values[k]
            segment = feature_value_segments[k]
            mrf_string += "." + non_terminal_symbol + value + segment
    return mrf_string


def mrf_other(lang, meaning_list):
    """
    Takes a language of the 'other' category and returns a minimally redundant␣
 ↪form description of the language's
    context free grammar.

    :param lang: a language; represented as a tuple of␣
 ↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param meaning_list: list of strings corresponding to all possible meanings
    :return: minimally redundant form description of the language's context␣
 ↪free grammar (string)
    """
    mapping_dict = {}
    for i in range(len(lang)):
        mapping_dict.setdefault(lang[i], []).append(meaning_list[i])
    mrf_string = 'S'
    counter = 0
    for form in mapping_dict.keys():
        for k in range(len(mapping_dict[form])):
            meaning = mapping_dict[form][k]
            if k != len(mapping_dict[form]) - 1:
                mrf_string += meaning + ','
            else:
                mrf_string += meaning
        if counter != len(mapping_dict.keys()) - 1:
            mrf_string += form + '.S'
        else:
            mrf_string += form
        counter += 1
    return mrf_string


def minimally_redundant_form(lang, complete_forms, meaning_list):
    """
    Takes a language of any class and returns a minimally redundant form␣
 ↪description of its context free grammar.

    :param lang: a language; represented as a tuple of␣
 ↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param complete_forms: list containing all possible complete forms;␣
 ↪corresponds to global variable
    'forms_without_noise'
    :param meaning_list: list of strings corresponding to all possible meanings
    :return: minimally redundant form description of the language's context␣
 ↪free grammar (string)
    """
    lang_class = classify_language_four_forms(lang, complete_forms,␣
 ↪meaning_list)
    if lang_class == 0:  # the language is 'degenerate'
        mrf_string = mrf_degenerate(lang, meaning_list)
    elif lang_class == 1 or lang_class == 2:  # the language is 'holistic' or␣
 ↪'hybrid'
        mrf_string = mrf_holistic(lang, meaning_list)
    elif lang_class == 3:  # the language is 'compositional'
        mrf_string = mrf_compositional(lang, meaning_list)
    elif lang_class == 4:  # the language is of the 'other' category
        mrf_string = mrf_other(lang, meaning_list)
    return mrf_string


def character_probs(mrf_string):
    """
    Takes a string in minimally redundant form and generates a dictionary␣
 ↪specifying the probability of each of the
```

```
    symbols used in the string

    :param mrf_string: a string in minimally redundant form
    :return: a dictionary with the symbols as keys and their corresponding␣
 ↪probabilities as values
    """
    count_dict = {}
    for character in mrf_string:
        if character in count_dict.keys():
            count_dict[character] += 1
        else:
            count_dict[character] = 1
    prob_dict = {}
    for character in count_dict.keys():
        char_prob = count_dict[character] / len(mrf_string)
        prob_dict[character] = char_prob
    return prob_dict


def coding_length(mrf_string):
    """
    Takes a string in minimally redundant form and returns its coding length in␣
 ↪bits

    :param mrf_string: a string in minimally redundant form
    :return: coding length in bits
    """
    char_prob_dict = character_probs(mrf_string)
    coding_len = 0
    for character in mrf_string:
        coding_len += log2(char_prob_dict[character])
    return -coding_len


def prior_single_lang(lang, complete_forms, meaning_list):
    """
    Takes a language and returns its PROPORTIONAL prior probability; this still␣
 ↪needs to be normalized over all
    languages in order to give the real prior probability.

    :param lang: a language; represented as a tuple of␣
 ↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param complete_forms: list containing all possible complete forms;␣
 ↪corresponds to global variable
    'forms_without_noise'
    :param meaning_list: list of strings corresponding to all possible meanings
    :return: PROPORTIONAL prior probability (float)
    """
    mrf_string = minimally_redundant_form(lang, complete_forms, meaning_list)
    coding_len = coding_length(mrf_string)
    prior = 2 ** -coding_len
    return prior
```

[3]:

```
# First, let's check whether the functions defined above work correctly
# for the example languages given in Kirby et al. (2015):


## First some parameter settings:
meanings = ['02', '03', '12', '13']
forms_without_noisy_variants = ['aa', 'ab', 'ba', 'bb']


## Then let's specify the example languages from Kirby et al. (2015)
example_languages = [['aa', 'aa', 'aa', 'aa'],
                     ['ab', 'ab', 'ab', 'ab'],
                     ['aa', 'aa', 'aa', 'ab'],
                     ['aa', 'aa', 'aa', 'bb'],
                     ['aa', 'ab', 'ba', 'bb'],
                     ['aa', 'aa', 'ab', 'ba'],
                     ['aa', 'aa', 'ab', 'bb'],
                     ['aa', 'ab', 'bb', 'ba']]


## And now let's calculate their coding lengths:
lang_classes_text = ['degenerate', 'holistic', 'hybrid', 'compositional',␣
 ↪'other']
for i in range(len(example_languages)):
    lang = example_languages[i]
    print('')
    print(i)
    lang_class = classify_language_four_forms(lang,␣
 ↪forms_without_noisy_variants, meanings)
    lang_class_text = lang_classes_text[lang_class]
    print("lang_class_text is:")
    print(lang_class_text)
    print("lang is:")
    print(lang)
    mrf_string = minimally_redundant_form(lang, forms_without_noisy_variants,␣
 ↪meanings)
    print("mrf_string is:")
    print(mrf_string)
    coding_len = coding_length(mrf_string)
    print("coding_len is:")
```

```python
    print(round(coding_len, ndigits=2))
    prior = prior_single_lang(lang, forms_without_noisy_variants, meanings)
    print("prior this lang is:")
    print(prior)
```

```
0
lang_class_text is:
degenerate
lang is:
['aa', 'aa', 'aa', 'aa']
mrf_string is:
S02,03,12,13aa
coding_len is:
38.55
prior this lang is:
2.488119421993922e-12

1
lang_class_text is:
degenerate
lang is:
['ab', 'ab', 'ab', 'ab']
mrf_string is:
S02,03,12,13ab
coding_len is:
40.55
prior this lang is:
6.220298554984805e-13

2
lang_class_text is:
other
lang is:
['aa', 'aa', 'aa', 'ab']
mrf_string is:
S02,03,12aa.S13ab
coding_len is:
52.73
prior this lang is:
1.3368788379305763e-16

3
lang_class_text is:
other
lang is:
['aa', 'aa', 'aa', 'bb']
```

```
mrf_string is:
S02,03,12aa.S13bb
coding_len is:
53.49
prior this lang is:
7.922244965514519e-17

4
lang_class_text is:
compositional
lang is:
['aa', 'ab', 'ba', 'bb']
mrf_string is:
SAB.A0a.A1b.B2a.B3b
coding_len is:
59.2
prior this lang is:
1.5092773625944422e-18

5
lang_class_text is:
other
lang is:
['aa', 'aa', 'ab', 'ba']
mrf_string is:
S02,03aa.S12ab.S13ba
coding_len is:
61.68
prior this lang is:
2.700000000000012e-19

6
lang_class_text is:
other
lang is:
['aa', 'aa', 'ab', 'bb']
mrf_string is:
S02,03aa.S12ab.S13bb
coding_len is:
62.17
prior this lang is:
1.9221679687499936e-19

7
lang_class_text is:
hybrid
lang is:
['aa', 'ab', 'bb', 'ba']
```

```
mrf_string is:
S02aa.S03ab.S12bb.S13ba
coding_len is:
67.29
prior this lang is:
5.553712540966203e-21
```

[4]:

```python
# These are the same as in the table in Kirby et al. (2015, p. 92),
# except for the coding length of the compositional language. In
# Kirby et al. (2015) it says 55.20, when I get 59.20. But given that
# all the other values I get are the same as in the table, I suspect
# this is a typo in the actual paper.
# So now that we know that these functions are coded up correctly,
# let's have a look at the coding lengths for our example languages
# for Possibility 1: different form lengths

mrf_compositional_reduplicate_whole_signal = "SABAB.A0a.A1b.B2a.B3b"
coding_len_compositional_reduplicate_whole_signal =␣
 ↪coding_length(mrf_compositional_reduplicate_whole_signal)
print("coding_len_compositional_reduplicate_whole_signal is:")
print(round(coding_len_compositional_reduplicate_whole_signal, ndigits=2))

mrf_compositional_reduplicate_segments = "SAABB.A0a.A1b.B2a.B3b"
coding_len_compositional_reduplicate_segments =␣
 ↪coding_length(mrf_compositional_reduplicate_segments)
print('')
print("coding_len_compositional_reduplicate_segments is:")
print(round(coding_len_compositional_reduplicate_segments, ndigits=2))

mrf_holistic_diversify_signal = "S02aaaa.S03bbbb.S12abba.S13baab"
coding_len_holistic_diversify_signal =␣
 ↪coding_length(mrf_holistic_diversify_signal)
print('')
print("coding_len_holistic_diversify_signal is:")
print(round(coding_len_holistic_diversify_signal, ndigits=2))

mrf_compositional_repair = "SAB.A0a.A1b.B2a.B3b"
coding_len_compositional_repair = coding_length(mrf_compositional_repair)
print('')
print("coding_len_compositional_repair is:")
print(round(coding_len_compositional_repair, ndigits=2))


ratio_reduplication_vs_repair =␣
 ↪coding_len_compositional_reduplicate_whole_signal/
 ↪coding_len_compositional_repair
print('')
print("ratio_reduplication_vs_repair is:")
print(round(ratio_reduplication_vs_repair, ndigits=2))

ratio_diversify_signal_vs_repair = coding_len_holistic_diversify_signal/
 ↪coding_len_compositional_repair
print('')
print("ratio_diversify_signal_vs_repair is:")
print(round(ratio_diversify_signal_vs_repair, ndigits=2))
```

```
coding_len_compositional_reduplicate_whole_signal is:
64.24

coding_len_compositional_reduplicate_segments is:
64.24

coding_len_holistic_diversify_signal is:
84.83

coding_len_compositional_repair is:
59.2

ratio_reduplication_vs_repair is:
1.09

ratio_diversify_signal_vs_repair is:
1.43
```

Alright, so as we can see from the coding lengths above, possibility 1 of how to represent languages gives relative coding lengths that capture the intuitions we have about how hard it is to learn these different languages: the compositional languages with reduplication only have slightly longer coding lengths than the compositional language without it (ratio reduplication:repair = 1.09:1), whereas the holistic language resulting from the diversify signal strategy has a significantly longer coding length (ratio diversify_signal:repair = 1.43:1)

Note: What I haven't looked at yet is how these ratios scale when signal lengths are increased. For instance, if noise would regularly obscure *two* characters instead of one, the diversify signal strategy would require longer signals, which would result in even larger coding lengths.

1.4 Conclusion about how to represent languages:

Based on all the above, I'd say let's go for possibility 1 of allowing for different form lengths. That keeps our way of representing languages as close as possibility to the one used by Kirby et al. (2015); it allows us to straightforwardly calculate the coding lengths; and it will not cause languages to make use of a different number of characters, as possibility 2 would.

1.5 Should repair be learned or not?

1.5.1 There are two options here:

1. **Repair is learned:** This would mean that every possible language in the hypothesis space could in principle be combined with repair. This could be encoded as an extra bit of information added on to the language itself, so that repair can be either 'on' or 'off'.
2. **Repair is *not* learned:** This would mean that we assume that agents have the ability to asses whether their particular language requires using repair or not. So we could simply code up the communication function in such a way that if an agent's language uses neither the reduplication nor the diversify signal strategy, the agent automatically uses repair (when noise is encountered).

1.5.2 Pros and cons:

We would prefer option 1 (repair is learned) because it feels like the most complete model: we allow for all possible combinations of strategies, and no strategy is a priori excluded (such as the combination of reduplication with repair). We also know from actual development that the conversational infrastructure of things like turn-taking and repair indeed needs to be learned. However, the disadvantage of option 1 is that it would double the hypothesis space (so if we go for possibility 1 in terms of representing languages, the hypothesis space would grow from 160,000 languages to 320,000 languages.

Something that isn't particularly an advantage or a disadvantage, but simply a difference, is that adding in repair as a bit to the encoding of the language would slightly increase the coding length of the Repair + Compositionality strategy, and thereby minimise the difference in coding length between the Reduplication + Compositionality and the Repair + Compositionality strategies, which would lower the chance of the Repair + Compositionality strategy becoming dominant already in the {+learnability, -minimal effort} condition, so in the absence of the minimal effort pressure.

2 References

Burkett, D., & Griffiths, T. L. (2010). Iterated learning of multiple languages from multiple teachers. The Evolution of Language: Proceedings of the 8th International Conference (EVOLANG8), Utrecht, Netherlands, 14-17 April 2010, 58–65.

Kirby, S., Tamariz, M., Cornish, H., & Smith, K. (2015). Compression and communication in the cultural evolution of linguistic structure. Cognition, 141, 87–102. https://doi.org/10.1016/j.cognition.2015.03.016