A Candidate collection details

A.1 Part–whole relations

For multi-word object names (*pony tail*, *right arm*), appearing as wholes or parts, we first check if the whole name appears in WordNet. If so, we accept the multi-word noun as a candidate. Else, we check if the name has a descriptor using a short manually selected list of colors (e.g., *red*) and spatial descriptors (e.g., *left*), and take the second word heuristically if so. We additionally lemmatize each object name with WordNet.

A.2 Adjectives

To ensure data diversity, we use a list of common color names to limit color adjectives to one per part–whole pair. We also filter out adjectives that form a common multi-word expression when paired with the whole noun (e.g., *dutch oven*, *sick bed*).

B Non-visual annotation

We also attempted to gather annotations for triples with part–whole relations originating from other sources, such as ConceptNet (Speer et al., 2017) and Wikidata (Vrandečić and Krötzsch, 2014). However, after gathering part–whole relations and applying various filters, we still observed that about 20% of part–whole relations were invalid. Moreover, the agreement within our selected pool of crowd workers, and within the authors, was worse than that for the Visual Genome-based data and judged to be too low. As a result, in this work we focus on visual data, leaving a robust general data collection approach to future work.

C Model selection

C.1 Word embedding MLP

To choose the number of layers and the variety of word embedding, we evaluated [0, 1, 2] numbers of hidden layers, and pre-trained word embeddings from GloVe, ELMo (Peters et al., 2018), and ConceptNet Numberbatch (Speer et al., 2017) by evaluation on the validation set. We do not update the embeddings as we found that this did not improve performance on the validation set. We also tried ordinal classification approaches, but these did not outperform categorical models. This model is implemented in PyTorch 0.4[4].

[4] https://github.com/pytorch/pytorch

C.2 Language model fine-tuning

We performed light hyperparameter tuning, using the validation set, over learning rate for both models and warm-up proportion for BERT. For both models, we use open-sourced PyTorch re-implementations that match the original implementations in performance.[5][6]

D Context sentence validation

A valid context sentence contains the whole and adjective words in an *amod* relation. We identify these sentences by first filtering with simple string matching for sentences that contain the two words adjacent to each other, and then verifying that they exist in an *amod* relation using the dependency parser from spaCy[7]. We choose a distinct whole–adjective context sentence for each adjective whenever possible.

[5] https://github.com/huggingface/pytorch-pretrained-BERT

[6] https://github.com/huggingface/pytorch-openai-transformer-lm

[7] https://github.com/explosion/spaCy